Thesis Dissertation

Utilizing Mobile Nodes for Fault-Management in Wireless Sensor Networks and IoT Networks

Andreas Naoum

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2022

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Utilizing Mobile Nodes for Fault-Management in Wireless Sensor Networks and IoT Networks

Andreas Naoum

Supervisor

Dr. Vasos Vassiliou

A thesis submitted in partial fulfilment of the requirements for the award of a Bachelor's degree in Computer Science at the University of Cyprus

May 2022

Acknowledgements

Throughout the process of writing my thesis dissertation, I received a great deal of help and support. First and foremost, I'd want to convey my gratitude to Associate Professor Vasos Vassiliou, whose expertise was crucial in developing the research questions and methodology. His insightful feedback encouraged me to refine my thoughts and raise the quality of my work. I want to thank Ms Natalie Temene for her patient support, guidance and for all the opportunities I was given to further my research. I have significantly benefited from her wealth of knowledge and meticulous editing. Finally, I would like to thank my parents and friends for their support.

Abstract

Wireless Sensor Networks (WSN) and Internet of Things (IoT) networks have gained a lot of attention in the last decade because of their numerous applications in domains such as defense, health monitoring, environmental monitoring, and disaster management. Several problems, such as node faults, congestion, and security attacks, are the main reason that often disrupts the operation of WSNs and IoT networks.

Fault tolerance is a major challenge in WSN and IoT that attracts a lot of research attention. The main goal of fault tolerance is to ensure that the system is available for usage in the occurrence of failures. As a result, fault tolerance improves the wireless network system's availability, reliability, and dependability. One of the most common approaches for increasing a network's fault tolerance is fault management.

One technique for resolving the issue of failure is to inject extra resources into the network, such as utilizing mobile nodes. Mobile nodes are sensor nodes that can change their location in the network. Their goal is to be used and act as static nodes by either replacing faulty nodes or assisting existing nodes in performing their tasks. The primary objective of this work is to develop a strategic solution that utilizes mobile nodes efficiently and effectively in order to detect failures, reconnect and improve the lifetime of the network.

In this work, we present a Replacement Fault Management Mechanism, a Decentralized Fault Management Mechanism, a Centralized Fault Management Mechanism and finally, the MobileFM, a Fault Management Framework that utilizes mobile nodes to handle failures in Wireless Sensor Networks. All mechanisms consist of the detection phase and the recovery phase. The first phase is focused on detecting faults and investigating the affected area, and the second phase focuses on placing mobile nodes in the network to reconnect the affected areas. The algorithms are implemented in the Contiki O/S in the C programming language. Simulation results demonstrate that the proposed algorithm can significantly contribute to detecting and recovering faults in IoT and WSNs. Finally, the mechanisms are analyzed to define their major challenges, including the percentage of successfully received packets, packet loss ratio and energy consumption.

Contents

Chapter 1

Introduction

Motivation

The Internet of Things (IoT) is a new technology with many applications in everyday life. In terms of devices and communication capabilities, the IoT is a direct descendant of Wireless Sensor Networks (WSNs), a networked and resource-constrained system [13]. In fact, WSNs, which consist of many small and inexpensive devices, are used in a variety of applications (military, industry, agriculture etc.). Due to the unique characteristics of the devices, this deployment faces additional problems (small size, limited battery, limited memory, etc.).

Many issues arise due to failure, energy exhaustion, and the inability to recharge or replace batteries [13,16]. Based on the position of the failure, a whole area of the network can become disconnected. This can highlight the need for strategies to mitigate these problems and assist these networks in performing their intended functions, even in the face of faults, in order to increase fault tolerance.

A promising solution for faults and disconnections is the use of mobile nodes. The ability of mobile nodes to move around can help deal with a variety of network problems [13]. Place mobile nodes in specific locations can reconnect the network and extend the network's lifetime [1,5,6].

With little human interaction, this vision proposes that WSNs and IoT networks should monitor and reconfigure themselves in response to various events (e.g., a faulty node, congestion). In

order to accomplish this, an effective management architecture is required to provide a variety of functions. Motivated by these intentions, we started to theoretical study and practical developed a fault management framework with the aim of making WSNs and IoT networks reliable against faults and disconnections.

Objective and Contribution

The main objective of this thesis is to provide a solution that utilizes mobile nodes efficiently and effectively for fault detection, fault recovery, and assuring network connectivity in the event of a disconnection. We were particularly interested in designing and developing algorithms, as well as implementing them in real-world-like simulations and determining whether the algorithm's goal was achieved. We discussed its usefulness by performing a variety of metrics, including the percentage of successfully received packets, packet losses, and energy consumption. The end goal was to compare these measurements against the simple replacement strategy to see if they were beneficial in any specific area.

Methodology

This work extends the prior work presented by Ms Natalie Temene, Ms Nicolaou Antonia, Dr Vasos Vasiliou and Dr Chryssis Georgiou under the title "Utilizing Mobile Nodes for Congestion Control in Wireless Sensor Networks" [9]. This work presents an algorithm that utilizes mobile nodes to assist the existing nodes when congestion occurs in the network. The algorithm starts when existing congestion control algorithms fail to resolve the problem. Our extension utilizes mobile nodes in a network to deal with failures and disconnections.
Initially, we started investigating the area of interest by reading papers about mobility and fault tolerance approaches in WSNs and IoT Networks. Then, we got familiar with the COOJA Simulator of the Contiki OS [3]. COOJA simulator is a network simulator specifically designed for these networks and is a valuable tool for our work because it allows the development of algorithms and observes the nodes and network behavior. Finally, we designed and implemented algorithms in the COOJA simulator to validate the strategies and generate performance metrics.

Document Organization

In Chapter 2, we provide an overview of WSNs and IoT networks, as well as fault-tolerance approaches, mobility in WSNs and IoT networks, and related work of our thesis. In Chapter 3, we go through the clustering method and the node placement algorithms in detail (Dynamic MobileFT, Direct MobileFT). Next, in Chapter 4, we provide a detailed description of the three strategies: replacement fault management mechanism, decentralized fault management and centralized fault management, and finally present the MobileFM Framework, which uses the three strategies. In Chapter 5, the experimental environment and scenarios are presented, and a discussion on the results obtained. Finally, we conclude in Chapter 6 with a conclusion on the results, the challenges and the future work.

Chapter 2

Background and Related Work

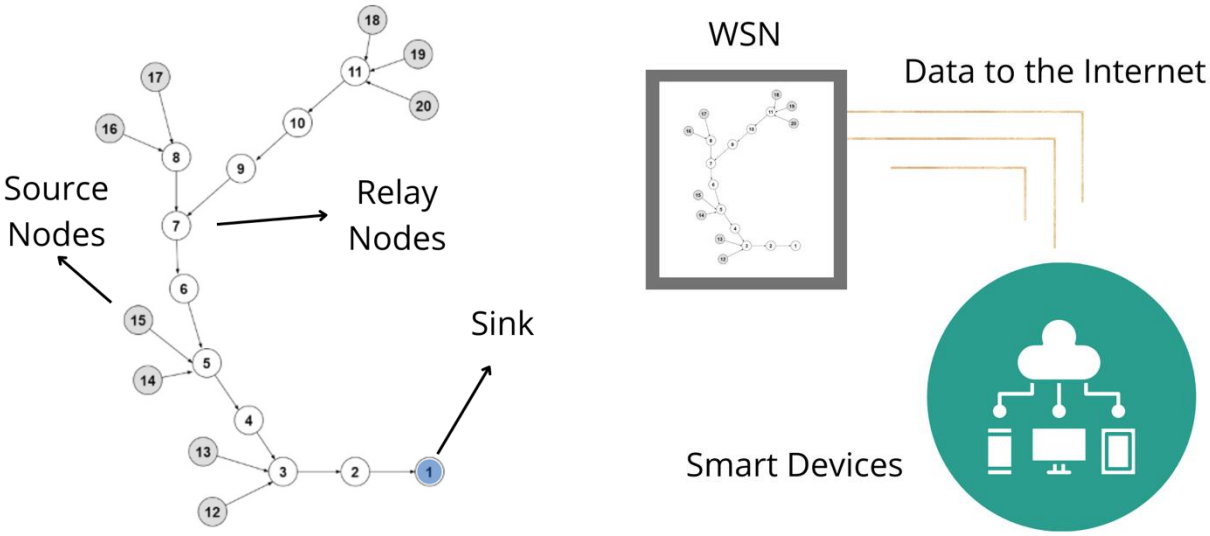
2.1 WSNs and IoT Networks

WSNs are self-configuring wireless networks made up of several small devices (nodes) with specialized sensors and wireless transceivers [16]. The main goal of WSNs is to collect data from the environment, process them and then transfer them to the base station, normally in a hop-by-hop fashion. A wireless sensor device collects data in response to a specific event or periodically. Each WSN sensor node can find a path to the selected sink, and the path is based on criteria such as the number of hops from the sink, the delay, the remaining energy, etc. WSN applications can be categorized into two categories: (a) monitoring and (b) tracking [16]. Environmental sensing, hazardous environment investigation, and health monitoring are examples of monitoring applications. Natural disaster relief efforts, tracking animals, and tracking items or individuals are all examples of tracking applications.

The IoT is a new technology that has a lot of applications in everyday life and refers to devices that are connected to the Internet. WSNs can be a subset of an IoT-based system known as IoT-enabled WSNs, in which each sensor node is defined as an IoT-enabled sensor node that will monitor the environment and gather real-time data. IoT-enabled WSNs is defined as a network of sensor nodes connected to a base station that functions as an access point and is also connected to an end-user through the Internet [13]. In this manner, the proposed mechanisms and framework in this work can be applied to IoT-enabled WSNs.

A network, that consists of many low-power sensor nodes, comes with significant limits [16]. The low-power sensor node's hardware and software capabilities in terms of computing,

memory, energy, and other factors are severely limited due to the small dimension design. The energy supply constraint is the most crucial of these constraints. The sensor nodes are expected to function independently from days to years and typically only have a limited amount of battery capacity. Since the battery life of sensor nodes is limited, the network lifetime can be affected in case of holes. These limitations are the reason that WSNs are error prone.



(a) Wireless Sensor Network (b) IoT-enabled WSN

Figure: 2.1

Several problems, such as node faults, congestion, and security attacks, are the main reason that often disrupts the operation of these networks and because of the mentioned limitations is hard to handle them . Faults in these networks is an important issue that needs to be addressed [2,8]. The hardware constraints lead sensor nodes to frequently fail. Failures in these networks can happen for different reasons, such as energy exhaustion, destruction by an external event, environmental factors, and attacks. In the case of the existence of a single path leading to the sink, a single failure can be fatal for the connectivity of the network [1]. Further, sensor nodes may be difficult to reach, or the battery could not be replaced.

2.2 Fault-Tolerance

Fault tolerance is a technique for describing a system's ability to deal with failures while maintaining its functionality [2]. Sensor node failures are divided into two categories: single node and multiple nodes. Single node failures indicate the loss of one node at a time, and multi-node failures indicate multiple node failures at the same time. Many researchers have developed fault-tolerant algorithms that can improve accuracy, energy efficiency and the network lifetime, as well as reduce failure of network components [2,8].

Fault management is one of the most prevalent methods for increasing a network's fault tolerance and is important for WSNs because of their unique features and characteristics [8]. Fault detection, diagnosis, and recovery are the three processes in a fault management framework see Fig. 2.2.

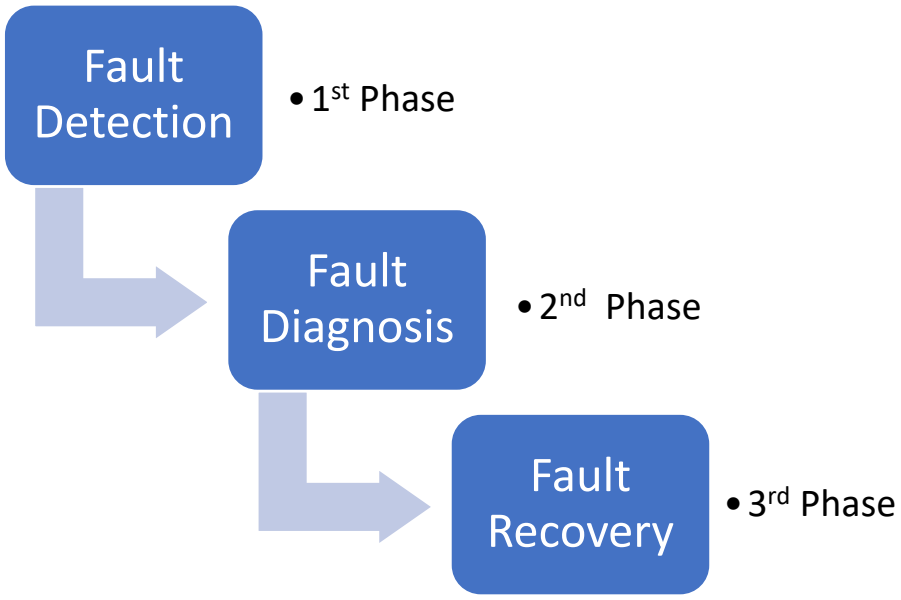


Figure 2.2: Fault Management Phases

The fault detection step is capable of identifying anything that may have an impact on the network or a node. There are three approaches to this technique: centralized, distributed, and self-managed. A centralized node, such as the sink, is responsible for managing the network and diagnosing a faulty node in the first approach. The detection is achieved by all nodes in the second approach, which makes use of neighboring nodes and clustering approaches. In the third approach, a node is in the duty of investigating, analyzing and reporting.

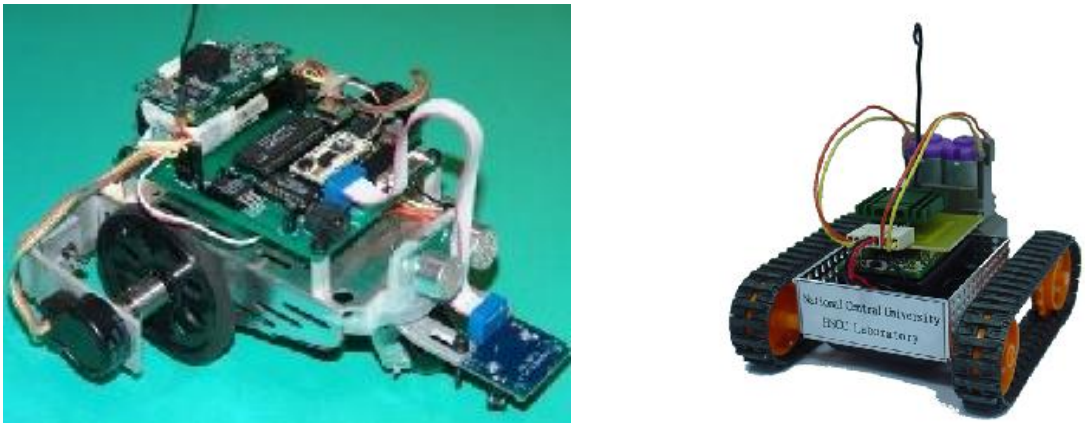
The fault diagnosis step specifies the failure in terms of type, cause, and network impact, among other things. This step is subdivided into four types of monitoring: passive, active, proactive,

and reactive. Sensor nodes are required to report their existence to a control center node (e.g., sink) with alive messages in passive monitoring. In active monitoring sensor nodes are instructed to report their existence to a control center node (e.g., sink). In proactive monitoring, every previous diagnosis is analyzed, and future events are predicted in order to keep the network running smoothly. Reactive monitoring is a management system that collects data on the state of a network in order to recognize interesting previous events and perform specific adaptive measures to reconfigure the network.

The recovery step is in control of redesigning or rebuilding the network therefore that faulty nodes don't affect its functionality or performance. This means that the network's dysfunctional state is replaced with one that is fully functional. Recovery and reconfiguration are two subsets of this method. The former mitigates the impact of the problem, whereas the latter modifies the network's topology without affecting overall output.

2.3 Mobility in WSNs and IoT Networks

Mobility in WSNs and IoT Networks is the ability of network devices to change their position [13]. Mobile devices can be robots or drones, examples of such devices are illustrated in Fig. 2.3. The use of mobile devices in a network as extra resources can be advantageous and untangle serious network problems such as congestions caused by high traffic and disconnections caused by failures in the network [1,9,14]. Moreover, bottlenecks in nodes can be created because of the multi-hop communication and as a result battery of these nodes will be exhausted. Strategies aim to improve the lifetime of the network by taking advantage of mobility.



(a) Mobile Node 1 (b) Mobile Node 2

Figure 2.3: Examples of mobile devices

In the literature, different approaches exist that include the use of mobile sink(s), mobile nodes and mobile robots. A mobile sink can move around the network and collect data from the nodes. This approach can mitigate the problem of disconnection by balancing the nodes' energy consumption and avoiding bottlenecks. Algorithms have been proposed using mobile nodes with different solutions approaches [13]. A mobile node has all characteristics of a normal sensor node with the addition of the ability to move. This approach can prevent the network from getting disconnection, reconnect the network from disconnection and ensure area coverage [13]. Mobile robots have more computational power than mobile nodes and they can achieve more complicated tasks, e.g., a mobile robot can move to the locations of failed nodes and unload functional nodes [7].

2.4 Related Work

Joshi et al. (2016) proposed a novel solution to the challenge of restoring connectivity in resource-constrained WSNs [5]. A distributed Resource-Constrained Recovery (RCR) technique that strategically repositions nodes to act as relays to rejoin a network partitioned into disjoint segments is presented. In this technique, if there are insufficient survivor relocatable nodes to build a stable inter-segment topology, some of them are used as mobile data collectors with optimized tours to reduce latency. RCR's effectiveness is proven by mathematical analysis and simulation results.

Lalouani et al (2017) presented a novel approach to tackle the topic of restoring connectivity in a highly partitioned network with the fewest number of relay nodes possible [6]. To address this issue, they present a novel Boundary-aware optimized Interconnection of Disjoint Segments technique (BIND), that attempts to restore network connectivity by establishing the least length topology in the Euclidean plane, which interconnects a subset of nodes on segment boundaries by extra Steiner points, ensuring that a path exists between each pair of segments. The simulation developed in python and the simulation findings show that BIND is effective and has an edge over rival methods.

Rao et al. (2020) presented an SDN-based strategy for reducing the number of hops in WSNs by employing mobile nodes for reliable data transmission [15]. This strategy increases the packet transmission ratio and as a result it reduces the end-to-end delay as well as the energy consumption of each sensor in the network. Simulation results show that this approach outperforms the reliability of data transmission compared with traditional methods.

Anuradha et al. (2020) presented a variety of mechanisms for restoring network connectivity [1]. The algorithm provides a faulty node detection system, which is divided into two approaches. The first approach uses periodic alert messages to detect malfunctioning nodes.

The network's nodes send an occasional alert message to the gateway containing all the node's information. The node is determined alive after the getaway receives such a message. When a message is not received the first time, the getaway waits for the second round of alarm messages to determine the node's issue. The node is considered faulty at this phase. The second approach is a residual energy-based faulty node detection method. Each node must calculate its own energy to decide whether it is faulty or not. When the node's energy level reaches the threshold, it is marked faulty, and the getaway is notified. The mobile relay node is informed of the faulty node in both detection methods and moves to the position to replace it. The proposed mechanism is tested using emulators. The results show that the proposed methods are appropriate for resource-constrained devices and perform well when restoring network connectivity.

Temene et al. (2022) proposed a Node Placement Algorithm with two variations to assist existing congestion control algorithms in facing congestion in WSNs [14]. The first variation employs mobile nodes that create locally significant alternative paths leading to the sink. The second variation employs mobile nodes that create completely individual (disjoint) paths to the sink. The results of the simulations have proven that both variations can significantly contribute to the alleviation of congestion in WSNs.

Contrarily to the bulk of existing methods using mobile nodes for the recovery phase in the literature, we propose the utilization of mobile nodes for *both* phases, fault detection and fault recovery, and we present a holistic approach that deals with failures in these networks.

Chapter 3

Clustering and Node Placement Algorithms

The upcoming mechanisms and the MobileFM framework are based on node placement and clustering algorithms that calculate positions where a mobile node can investigate suspicious faults in the detection phase and determine the placement position for the mobile node in the recovery phase. The use of these functions will be explained later in this thesis. In this chapter, we describe the clustering and node placement algorithms in detail.

3.1 Clustering

Clustering [11,12] is an unsupervised machine learning technique, which means we don't have any external knowledge to guide or supervise the process. It can be defined as the task of identifying subgroups in data so that data points within the same cluster are similar while data points within different clusters are very dissimilar. There are different clustering approaches such as partitioning, density-based, and hierarchical.

In partitioning clustering algorithms, the clusters are compact sets of points, and the parameters are usually the number *k* of clusters, the distance measure (e.g., Euclidean distance) and the points. The aim is the flat partitioning of points into *k* clusters with maximal compactness. The clustering method we developed is a variation of the *k*-means partitioning clustering algorithm with constraints to satisfy specific requirements for the upcoming strategies.

3.1.1 *k*-means Algorithm

The *k*-means algorithm [12] is an iterative technique that attempts to partition a dataset into *k* separate non-overlapping clusters, where each point belongs to only one cluster. This technique tries to make intra-cluster points as identical as possible and keep other clusters as distinct as possible. It distributes data points to clusters in such a way that the sum of the squared distances between them and the cluster's centroid, the arithmetic mean of all the points in that cluster, is as minimal as possible. In other words, this algorithm uses distance-based measurements to determine the similarity between data points.
The centroids of the *k*-means algorithm are the cluster representatives.
The centroid, μ , can be calculated based on the centroid C_i as follow:

$$\mu_{C_i} = \frac{1}{|C_i|} * \sum_{p \in C_i} p \, ,$$

The *k*-means method is one of the most popular partitioning clustering methods and is easy to implement due to its simplicity. It's efficient; however, it's not guaranteed to converge to the global optimum and often terminates at a local optimum.

3.1.2 Semisupervised Partitional Clustering

As we have already mentioned, unsupervised clustering operates without guidance or knowledge. In our mechanisms, it is necessary to incorporate existing knowledge in the clustering method. For this reason, semisupervised learning fits our needs as general rules about the concept can be satisfied.

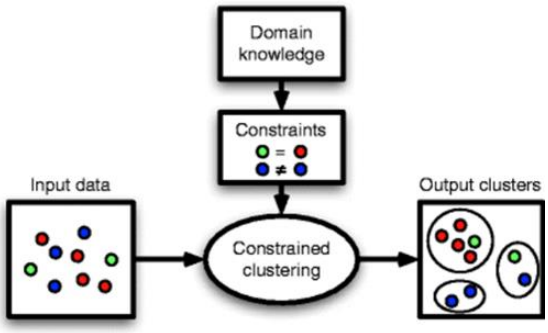


Figure 3.1: Constrained Clustering Explanation [11]

Constrained clustering [11] is a semisupervised learning technique that aims to extend the classic clustering algorithms with existing domain knowledge. This knowledge may derive from general rules, as constraints, about the application. A constrained clustering algorithm accepts the same inputs as an unsupervised clustering algorithm, as well as a set of constraints. In addition, these methods can enforce constraints in the solution (hard requirements) or/and use constraints as guidance.

Constrained clustering algorithms have now been applied to a wide variety of applications, such as web search result grouping, object identification, document clustering, etc. [11].

3.1.3 Constraints for the modified K-means Algorithm

The constrained clustering algorithm that we have developed consists of two constraints: (a) a constraint as a hard requirement and (b) a constraint as guidance.

The constraint as a hard requirement is about the coverage range of the cluster centroid. As a result, the distance between a cluster member and the centroid point of their cluster must be less than the coverage range. For the need of our algorithm, a criterion must be added as hard requirement to the k-means algorithm, the coverage criterion.

The coverage criterion can be defined as:

> The Euclidean distance between the cluster representative and every cluster member's position must be equal to or less than the constant coverage ratio R

$$Distacne(\mu_{C_i}, o) \leq R\ , o \in C_i\ ,$$

The constraint as guidance is responsible for speeding up the process of clustering. Nodes out of their cluster range and empty clusters are the two main reasons to delay the procedure. The

idea is that every time a node is within a cluster, and the distance between the node and the centroid is greater than the range, it will be assigned to an empty cluster, and all clusters are then rearranged. In this manner, solutions with nodes out of range and empty clusters are avoided. In the end, all points should be assigned to a cluster, and each cluster member should have a distance under the range from the centroid point. Thus, the need arises to use guidance for the coverage criterion in our modified *k*-means algorithm to help the processing speed up. The definition of the guidance for the coverage criterion is defined as follows:

> For every iteration in the clustering method, if a cluster member is not in the coverage area of its representative and there is an empty cluster, then this node becomes a new member of the empty cluster.

3.1.4 Modified K-means Algorithm with Constraints

The k-means algorithm must be modified because we need the algorithm to satisfy the constraints we mentioned before and return specific information. The algorithm would require the number *k* and the nodes as points in a two-dimensional space as parameters. In this work, the points in the two-dimensional space are considered as physical points (coordinates) on the plane. The *k*-means algorithm needs to return a clustering with this information: the number *k*, a list of the centroids (centroids), and each cluster's member(s).
The minimum number of clusters can be calculated because the maximum number of neighbors in these networks is known. The number *k* of clusters is unknown, and it's preferable for our purposes to find the minimum *k* number because fewer clusters will lead to fewer mobile nodes. In order to estimate the minimum number of clusters in our method, we proposed an estimation function, which can be defined as follow:
Function for the minimum number of clusters:

$$Minimum\ number\ of\ clusters = |Number\ of\ Nodes|/Maximum\ Neighbors$$

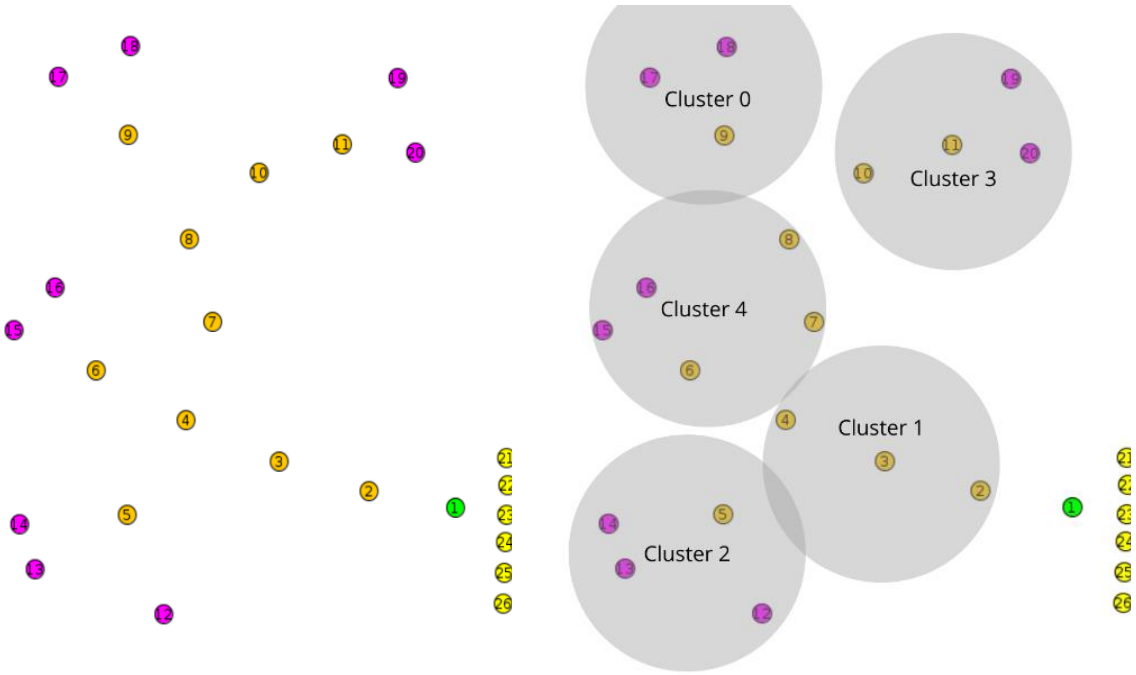
The algorithm's complexity is O(*kn*) per iteration and the number of iterations is usually in the order of 10; thus, the algorithm is efficient in comparison with other clustering algorithms.
The algorithm (see Fig.3.2), for every *k* = min, ..., n (min is estimated and n is the number of nodes), does 30 iterations of clustering with different random initial representatives (lines 2-8), guiding the process (lines 6-7), and if the coverage criterion is satisfied (lines 9-10), then the clustering is completed.

```
1. For k=min to n (number of nodes)
2.    For times = 1 to max_iterations
3.       Randomly assigned the points
4.       Do
5.          change = assign_points() // any change
6.          if (not satisfyConstraintRange())
7.              reassign_points() // constraint as guidance
8.       While (change and iter<max_iterations)
9.       If(satisfyConstraintRange)//constaint as hard requirement
10.         Return clustering
```

Figure 3.2: Pseudocode of constrained clustering algorithm

3.1.5 Examples of Clustering

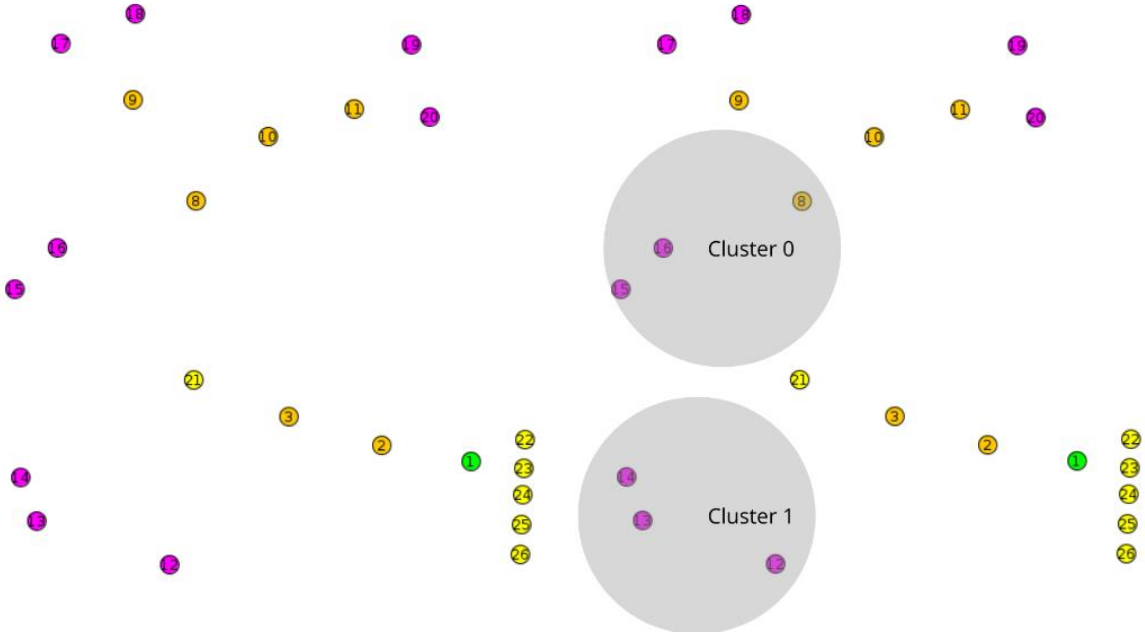
The examples below are the results of the clustering in the COOJA simulator, where the clustering algorithm is implemented. More details about the COOJA simulator will be discussed in Chapter 5.



(a) Network Topology (b) Partitioning Clustering

Figure 3.3: Example 1 of partitioning Clustering

In the example of Figure 1, the list of points consists of nodes 2-20. The algorithm divides the list into 5 different clusters. Cluster 0 consists of nodes 9,17 and 18, cluster 1 consists of nodes 2,3 and 4, cluster 2 consists of the nodes 5,12,13 and 14, and cluster 3 consists of the nodes 10,11,19 and 20 and finally cluster 4 consists of the nodes 6,7,8,15 and 16. The coverage criterion is satisfied for every cluster, which means a mobile node can move to a specific position and be able to communicate with all cluster members.



(a) Network Topology (b) Partitioning Clustering

Figure 3.4: Example 2 of partitioning clustering

Another example is presented in Figure 2, the list of points consists of nodes 8,12,13,14,15 and 16. The modified *k*-means algorithm partitioned the list into two different clusters. The first cluster consists of the nodes 8,15 and 16 and the second cluster consists of the nodes 12,13 and 14. As a result, every cluster satisfies the coverage criterion.

3.2 Node Placement Algorithms

In the expectation of carefully computation of the mobile node position to reconnect the problematic area and solve the disconnection problem, we designed and implemented two Node Placement algorithms, the Dynamic MobileFT algorithm and the Direct MobileFT algorithm.

3.2.1 Geometric Functions

The node placement algorithms are designed to solve geometric problems. They require in-depth knowledge of mathematical subjects and geometric functions to find the best node placement position. These geometric functions are based on these equations and methods: (a) Euclidean Distance, (b) Line through two points and (c) Intersection points of a line and a circle.

a) The well-known Euclidean Distance Equation is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} ,$$

where the d is the distance, (x_1,y_1) is the coordinates of the first point and (x_2,y_2) are the coordinates of the second point.

b) The equation of a line through two points:

$$y = mx + c ,$$

The straight line through two points will have an equation in the above form. Based on the coordinates (x_1,y_1) of the first point and the coordinates (x_2,y_2) of the second point, we can find the equation of the straight line between the two points using the following method:

1) We can find the value of m, the gradient of the line, by forming a right-angled triangle using the coordinates of the two points.
2) Then, we can find the value of the y-intercept, by substituting the coordinates of one point into the equation.

c) Method to find the intersection points of a line and a circle

1) A line and a circle can intersect in one of three ways: they can intersect in two points, one point, or none at all. If the line and the circle intersect, the coordinates of their intersection point (or points) simultaneously solve the line and the circle's equations. This allows us to utilize algebra to determine if a line and a circle intersect and the coordinates of their intersection points.
2) Given the equation of the line:

$$y = mx + c\ ,$$

and the equation of the circle in the standard form:

$$(x - h) + (y - k) = \mathrm{r}\ ,$$

We can substitute the expression $mx+b$ for y in the equation of the circle to obtain a quadratic equation in x in this form:

$$Ax + Bx + C = 0\ ,$$

3) The discriminant $\Delta=B-4AC$ of the quadratic $Ax+BX+C=0$ tells us about the line and circle intersections. If $\Delta>0$, the line and the circle intersect in two points. If $\Delta=0$, then the line is tangent to the circle. If $\Delta<0$, then the line and the circle are disjoint.
4) If $\Delta>0$ or $\Delta=0$, we can solve the quadratic $Ax+BX+C=0$ to find the x-coordinates of the points of intersection of the line and the circle.
5) We can then substitute these values back into the equation of the line $y=mx+b$ to find the y-coordinates of the points of intersection.

3.2.2 Dynamic MobileFT Algorithm

The first node placement algorithm is called Dynamic MobileFT. The main idea is to find the position of a mobile node that can serve all target nodes and communicate with the destination node in case of a disconnection. The target nodes are all members of the same cluster, where a centroid point is defined. Based on this centroid point, the algorithm finds the best position that can serve all target nodes. The best position of the deployed node can be defined as the position nearest to the destination node; however, if there is no such position, the centroid position is guaranteed to communicate with all the target nodes (coverage criterion in section 3.1.3).

```
 1. line = findLine(centroid,s)
 2. For each node in targetList do
 3.    Pos = intersectionPointOfLine&Circle(line,node)
 4.    For each member in targetList do
 5.       flagRange = checkInRange(pos_member,pos_dest)
 6.       If (flagRange == true)
 7.           count++
 8.       If (count == targetList.size)
 9.           Add pos to positionList
10. If (positionList is not empty)
11.      Return nearestPos(positionList,dest)
12.  Return centroid
```

Figure 3.5: Pseudocode of Dynamic MobileFT Algorithm

The calculation of the node position is performed as follows: Firstly, the line through the centroid and the destination s is calculated (see Alg., line 1). The target list (targetList) contains all the cluster members. For each node n in the target list, the intersection point of the virtual line and the circle, which is created by the radius of the transmitting range of the node n, is calculated (see Alg., line 3). Then, the algorithm checks if the selected position is in the range of other cluster members and if that is true then this position is added to the possible positions. node (see Alg., lines 4-9) The selected position will be the one nearest the destination and that is decided by the nearestPos function (see Alg., lines 10-11). If no position is selected, then the centroid is selected to place a node (see Alg., line 12).

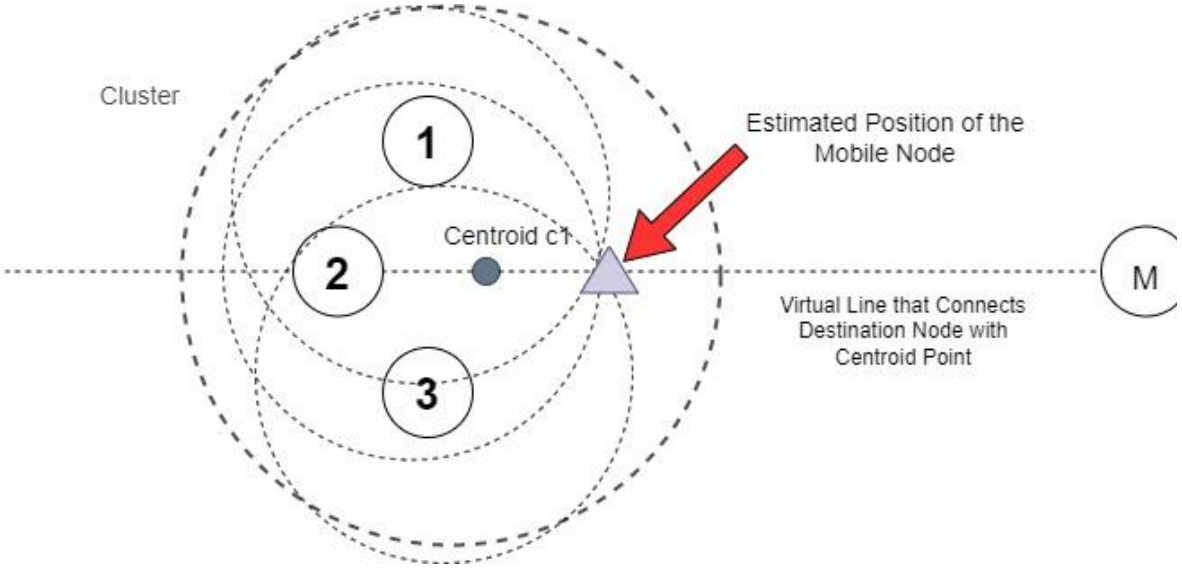


Figure 3.6: Node Placement position calculated by Dynamic MobileFT

An example of the estimated mobile node position is illustrated in Fig. 3.6. Initially, nodes 1,2,3 are all assigned to a cluster with centroid c1, and node M is the destination node to where the disconnected nodes of the cluster need to connect to. A virtual line that connects the destination node M with the centroid c1 is created. For each node in the cluster (1,2,3), the intersection point between the virtual line and the circle that is created by the radius of the transmitting range of the node is calculated. Then, a position that can serve all nodes in the cluster and is closer to the destination node is selected.

3.2.3 Direct MobileFT Algorithm

```
 1. pos = DynamicMobileFT()
 2. flagdp = false
 3. While (flagdp == false)
 4.     inRange = checkInRange(pos,dest)
 5.     If (inRange == true)
 6.         Place node to pos
 7.         Flagdp = true
 8.     Else
 9.         Place node to pos
10.         Pos = intersectionPointOfLine&Circle(pos,dest)
11. return
```

Figure 3.7: Pseudocode of constrained clustering algorithm

The Direct MobileFT algorithm is the second node placement algorithm. The main aim is to create a path of mobile nodes toward the destination in the case the selected position is not in the range of the destination. At first, the Dynamic MobileFT (see Fig. 3.7, line 1) is called to calculate the position of the first mobile node placed in the path. Then, it creates the direct line starting from the first placed mobile node and ending at the destination point. On this line, it places additional mobile nodes until one of them is in the range of the destination node and can forward packets directly to it (see Fig.3.7., line 2-10).

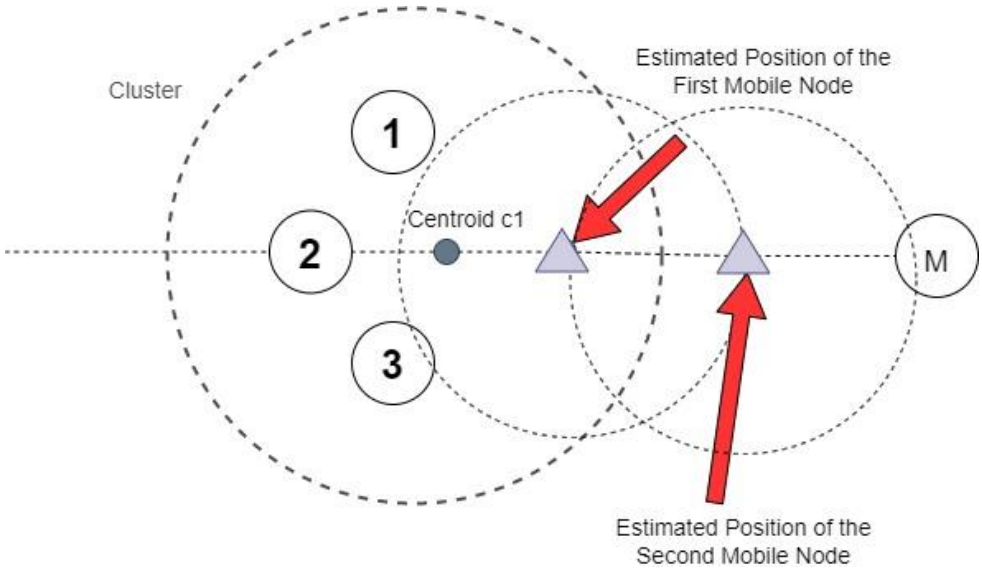


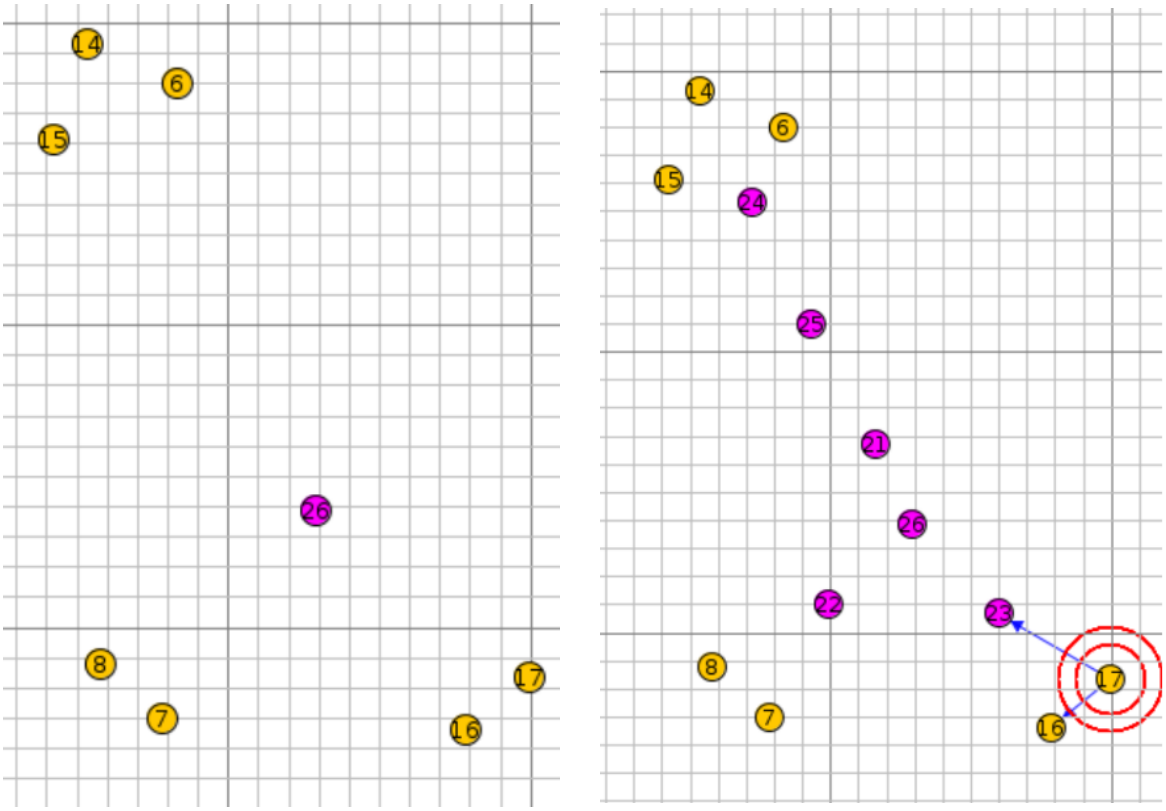
Figure 3.8: Node Placement positions calculated by the Direct MobileFT Algorithm

An example of the positions calculated by the Direct MobileFT algorithm is illustrated in Fig. 3.8. The estimated position of the first mobile node is selected by the Dynamic MobileFT algorithm and then another node, the second mobile node, is placed in the range of the previously placed mobile node. As the second mobile node is in the range of the destination node M, the procedure stops, and no more mobile nodes are placed in this path.

3.2.4 Combination of Clustering and Node Placement Algorithm

The example in the next page is the result of the combination of the clustering algorithm and the node placement algorithm in the COOJA simulator (evaluation setup), where the clustering and node placement algorithm are implemented. More details about the simulator will be discussed in Chapter 5.

The combination of the clustering algorithm and the node placement algorithm can be beneficial. Mobile nodes can be placed to specific positions in such way disconnected nodes can communicate with the destination node, that is explained further in the chapter 4.



(a) Topology (b) Mobile Nodes placement

Figure 3.9: Clustering and Node Placement algorithms

As illustrated in Fig. 3.9 static nodes 6,7,8,14,15,16 and17 were unable to communicate with the mobile node 26. A static node can be either relay or source node. The clustering algorithm (section 3.1.4) had achieved to divide the static nodes into non-overlapping subsets such that all nodes in a cluster can communicate with a node. Especially, three clusters were constructed, the first consists of nodes 16,17, the second one consists of nodes 7,8 and the third one consists of nodes 6,14,15. Then for each cluster, the node placement algorithm is performed to connect the cluster nodes (as target nodes) with the mobile node 26. Finally, it's obvious in Fig. 3.9 (b) that the static nodes were connected successfully to the destination node after the selected positions of the combination of the clustering algorithm and the node placement algorithm.

Chapter 4

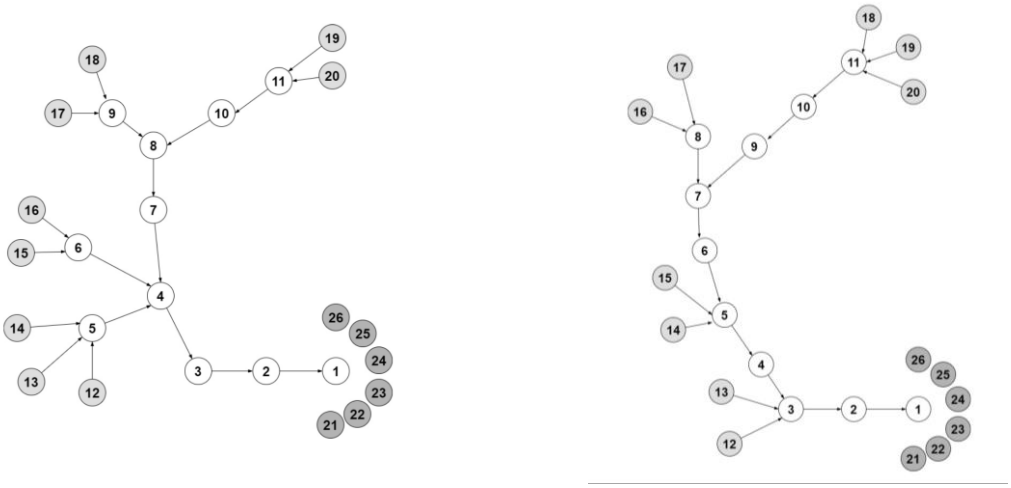
Fault Detection and Recovery Mechanisms

In this chapter, we propose solutions that make efficient and effective use of mobile nodes in order to detect faults, recover faults, and ensure network connectivity in the event of a disconnection. Initially, we present the network model and then the three mechanisms for fault management: (a) Replacement Fault Management Mechanism (b) Decentralized Fault Management Mechanism (c) Centralized Fault Management Mechanism. Finally, we present the MobileFM Framework that consists of the three previous mechanisms.

4.1 Model and Assumptions

We consider a network that consists of static nodes (source and relay) and a small number of mobile nodes that reside near the sink. We also assume the following:

- We employ a simple MAC protocol, like CSMA/CA
- The sink node knows the locations of all sensor nodes in the network, and all nodes are aware of their location in relation to the location of the sink.
- Static (relay and source) nodes and mobile nodes are sensors that collect environmental data periodically and send them to the sink node with a multi-hop communication.
- All static and mobile nodes are identical in terms of computation power, communication capabilities, sensing and transmission range, etc.
- Mobile nodes change their position only when it receives such an instruction from the sink node and can move straight to an arbitrary location in the field without considering any obstacles on their way. While the mobile node moves to its new position, its radio is turned off to prevent any interference. When it arrives at the target location, it turns its radio back on.



(a) Network Topology 1 (b) Network Topology 2

Figure 4.1: Example Network Topologies

Figure 4.1 shows two examples of different WSN topologies, where node 1 is the sink node, nodes 2-11 are the relay nodes, nodes 12-20 are the source nodes and 21-26 are the mobile nodes. The source nodes are responsible to collect the data and send it to the sink node. The relay nodes forward the received data from the source node towards the sink node. The sink node is the base of the network and is responsible to collect the data and the mobile nodes are nodes with the ability to move that are used as extra resources in case of any needs that occur in the network.

4.2 Replacement Fault Management Mechanism

The Replacement Fault Management Mechanism (RFMM) is a solution used for fault occurrence in the network. This mechanism is based on the distributed fault detection, which is performed by each node in the network. After the sink node is notified from a node about the failure, it will send one or more mobile nodes for investigation and replace the faulty node(s). The main idea of recovery is to replace each faulty node with a mobile node once the sink node is informed about the failure from a neighboring or a mobile node.

The RFMM consists of the following steps:

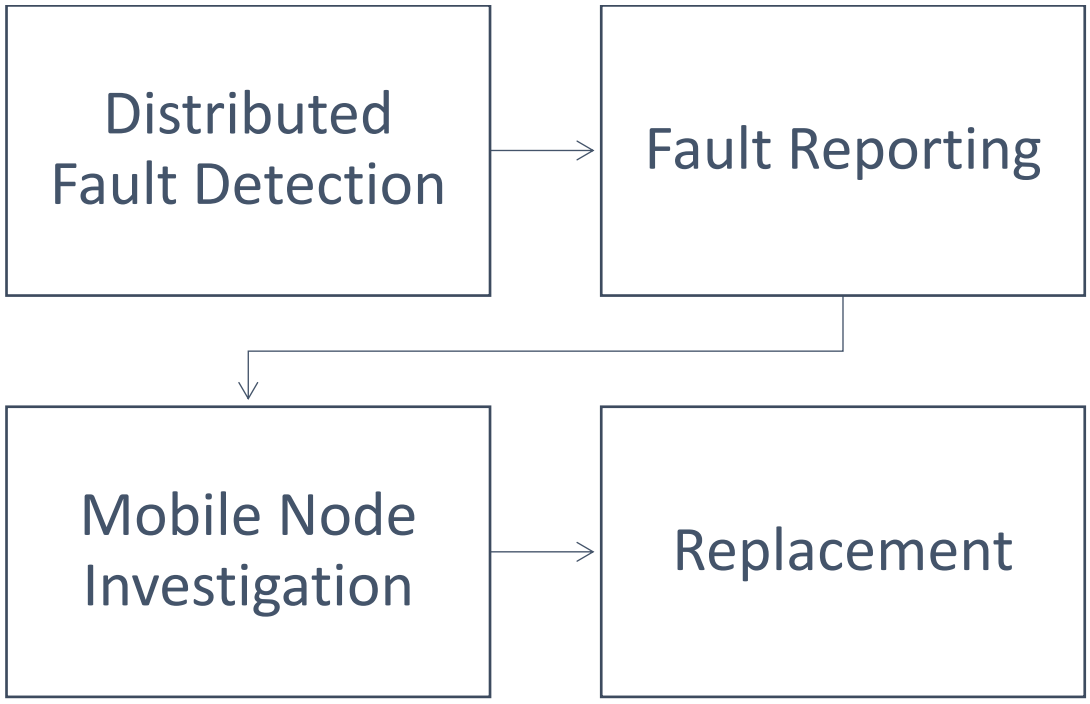


Figure 4.2: Replacement Fault Management Mechanisms Steps

The mechanism is divided into two processes: (a) detection and (b) recovery. In the detection process, the distributed fault detection, the fault reporting and the mobile node investigation are performed. Distributed fault detection aims to recognize the faults, while fault reporting aims to report the failure once a neighboring node acknowledges it. Mobile node investigation is performed by a mobile node and aims to investigate further for other faults in the affected area. In the recovery process, the replacement of the faulty node by a mobile node is performed.

```
1. Upon receive ("Faulty Notification(fn,level)") then
2.     // find the position of the faulty node
3.     pos = findPosition(fn)
4.     // number of expected neighbors
5.     neighbors = numberOfNeighbors(fn)
6.     // select an available mobile node
7.     mobile = selectMobileNode(pos)
8.     // send investigation notification to mobile node
9.     sendInvestigation(pos,level,neighbors,mobile)
10.
11.Upon receive ("Investigation Notification
   Response(aliveList)") from m then
12.   // find faulty nodes
13.   faultyList = findFaultyNodes(m,aliveList)
14.   // replace mobile nodes
15.   level =  findMobileLevel(m) + 1
16.   For each faulty in faultyList
17.       pos  findPosition(faulty)
18.       neighbors = numberOfNeighbors(faulty)
19.       mobile = selectMobileNode(pos)
20.       sendInvestigation(pos,level,neighbors,mobile)
```

Figure 4.3: Replacement Fault Management algorithm for sink node

```
1. Upon receive ("Investigation Notification
   (pos,level,expected)") from sink then
2.     move(pos) // move to position
3.     my_level = level
4.     broadcastAliveRequest()// broadcast alive request
5.     waitForTime(T) // wait for T time
6.     alive = countResponses() // count the alive nodes
7.     // send Investigation respond to sink if it s needed
8.     If responses != expected
9.        sendInvestigationResponse(neighbors)
10.    idle = false // start acting as a static node
11.    sendAnnouncementReconnect()
```

Figure 4.4: Replacement Fault Management algorithm for mobile node

```
1. // periodically checking for neighbors failure
2. Upon detect failure
3.     // battery exhaustion
4.     if (batteryExhaustion == true)
5.        sendFaultyNotification(my_ID, my_level)
6.        Return
7.     // suspicious faulty neighbor
8.     sendAliveRequest(s)
9.     waitForTime(T)
10.    // if suspicious is alive, just return
11.    if response == true
12.       return
13.    // neighbor from downstream, before it was receiving
   packets from this neighbor
14.    if (s.level > mylevel and receiving==true)
15.       sendFaultyNotification(s.ID, s.level)
16.       setFaulty(s)
17.       return
18.    // if the node does not have other upper nodes, it
   disconnected from the network
19.    if (s.level < mylevel and noUpperNode() == true)
20.       disconnection = true
21.       sendAnnouncementNoPath(mylevel)
22.
23. Upon receive ("Announcement No Path") from n then
24.    n.reachSink = false
25.    // if there is no other upper node
26.    if (noUpperNode() == true)
27.       sendAnnouncementNoPath()
28.       disconnection = true
29.
30.  Upon receive ("Announcement Reconnect") from n then
31.    n.reachSink = true
32.    if (noUpperNode() == false and disconnection == true)
33.       disconnection = false
34.       sendAnnouncementReconnect()
```

Figure 4.5: Replacement Fault Management algorithm for static node

4.2.1 Distributed Fault Detection

Each node in the network broadcasts an alive message periodically, intending to announce its healthy status to its neighbors. Additionally, each node executes the detection method periodically to identify any faults in the network.

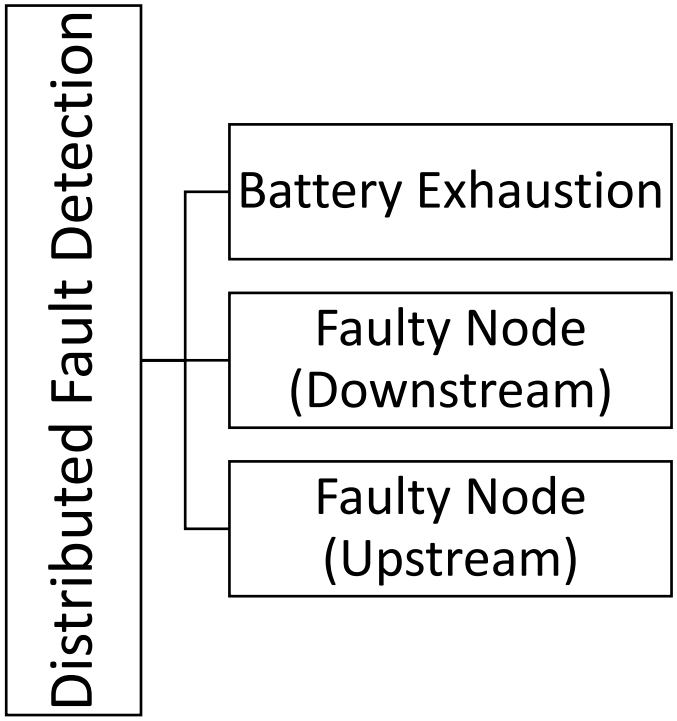


Figure 4.6: Distributed Fault Detection

Specifically, the node can recognize:

1. The different levels of its energy. If the energy consumption is lower than a given threshold, the node immediately informs the sink node about the upcoming node failure (informs about battery exhaustion).
2. A faulty node from its downstream neighboring nodes. The node can identify a suspicious faulty node from its neighboring lists when a certain amount of time has passed from the last time of their communication. Upon identifying such a situation, the node will send an alive request message to the suspicious node and wait for its reply. When the waiting timer expires, and the neighboring node does not reply, it considers it a faulty node and immediately informs the sink node.
3. A faulty node from its upstream neighboring nodes. The node considers the suspicious faulty node as a faulty node by following the process in the previous point.

4.2.2 Fault Reporting

The node, which has detected the fault from its downstream nodes or the battery exhaustion, sends a Faulty Notification Message (FNM) to the sink node. The FNM contains all the information needed, such as the nodes ID and its level. When the fault is referred to as energy exhaustion, the node sends the information of itself. In the case of a neighboring node failure, the node sends the neighbor's information, which can be found in its neighboring table.

FNM Message:

"FNM"	ID	Level

Figure 4.7: FNM Message Structure

The node, which has detected the fault from its upstream neighboring nodes, will recognize if there is no other upstream node, thus there is no path leading to the sink node. If the sink is not reachable, the node will inform its neighbors with an announcement that there is no path toward the sink and if there is no alternative path for them then the nodes will stop sending packets to not drain their energy.

4.2.3 Mobile Node Investigation

Upon receiving an FNM message, the sink node will calculate the number of expected neighboring nodes and then send a mobile node to resolve the problem. The sink node sends an Investigate Notification Message (INM), which includes the target position, the number of expected neighboring nodes, and its new level. Both position and level are the same as the faulty node.

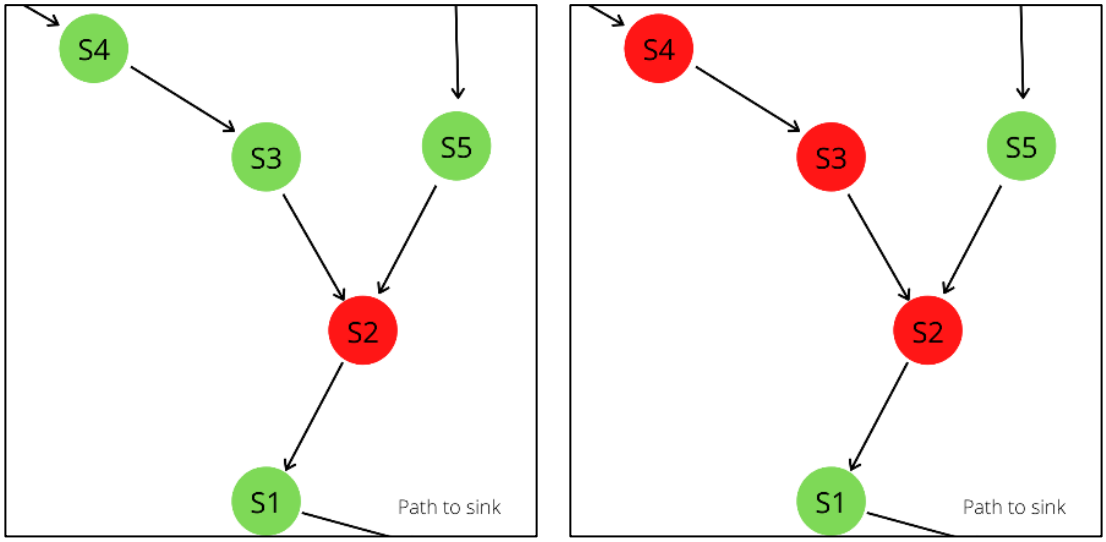
INM Message:

"INM"	Coord X	Coord Y	#Excpected Neighbors	Level

Figure 4.8: INM message structure

The investigation process starts once the mobile node arrives at its target position. The first step is to broadcast introduction messages and wait for all nodes in its range to reply. When the waiting timer expires, the mobile node starts the next step.

The following process is based on the replies received from the mobile node and is divided into two approaches: (a) single node failure and (b) multi-node failure. If the mobile node receives the expected number of replies (single node failure scenario), which means that the number of responses is equal to the number of expected neighbors; then the process stops as no other failure was detected. However, further investigation is needed in case of receiving fewer responses than expected (multi-node failure scenario).



(a) Single Node Failure (b) Multi-Node Failure

Figure 4.9: Types of Sensor Nodes Failures

In the single node failure approach, a node is the only failure in the neighborhood. In the investigation process, the mobile node will replace the faulty node by acting as a static node. The replacement method starts by introducing itself to the neighborhood and creating its neighbor list. It will recognize that the number of neighbors is the expected one; thus, there is no other failure in the neighborhood.

In the multi-node failure approach, the failure was created by more than one node in a row. The failure detection is the same as the single failure scenario. However, in this case, it will observe that its neighboring list has not the expected number of neighboring nodes, which was calculated by the sink. This indicates the existence of at least another faulty node in the network. The mobile node informs the sink node about its finding with an Investigate Results Notification Message (IRNM), including all the nodes' IDs in its neighbor table.

IRNM Message:

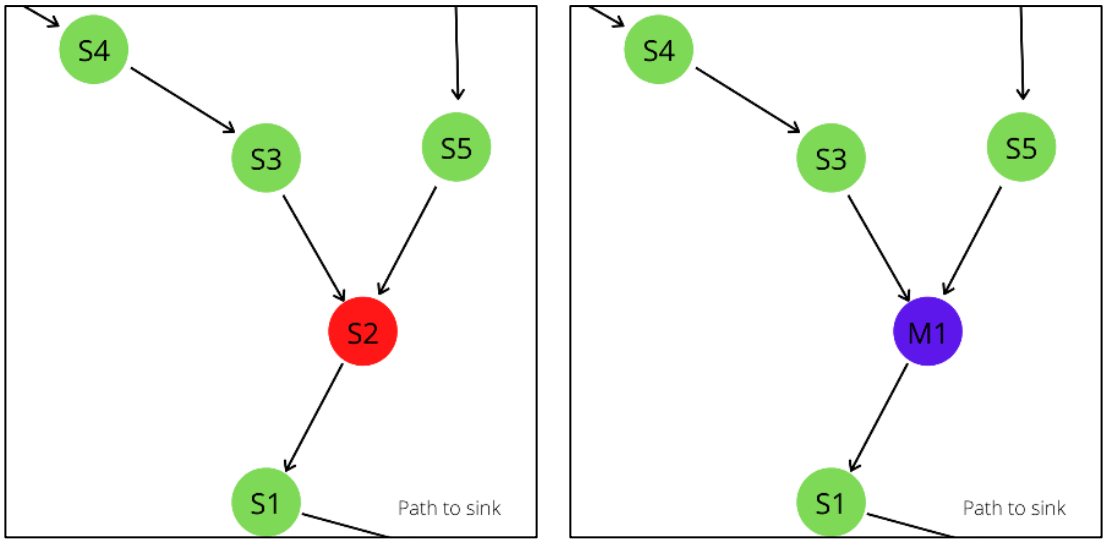
"IRNM"	ID	ID	….

Figure 4.10: IRNM Message structure

4.2.4 Replacement method

In both cases, single failure and multi-node failure, the mobile node will replace the faulty node, and depending on the role of the faulty node, the mobile node will act either as a relay or a source node. In the single failure case, the mobile node will act as a bridge between the two disconnected areas, the network will be reconnected, and the sink will be receiving all the packets once the connection is reestablished.

In the multi-node failure case, the sink will receive the IRNM message and act as follow. Based on the information received from the message, the sink will identify the missing nodes and replace each one with a new mobile node. Each mobile node will start its investigation process once it arrives to the target position, and in case of any other fault, it will inform the sink with an IRNM message.

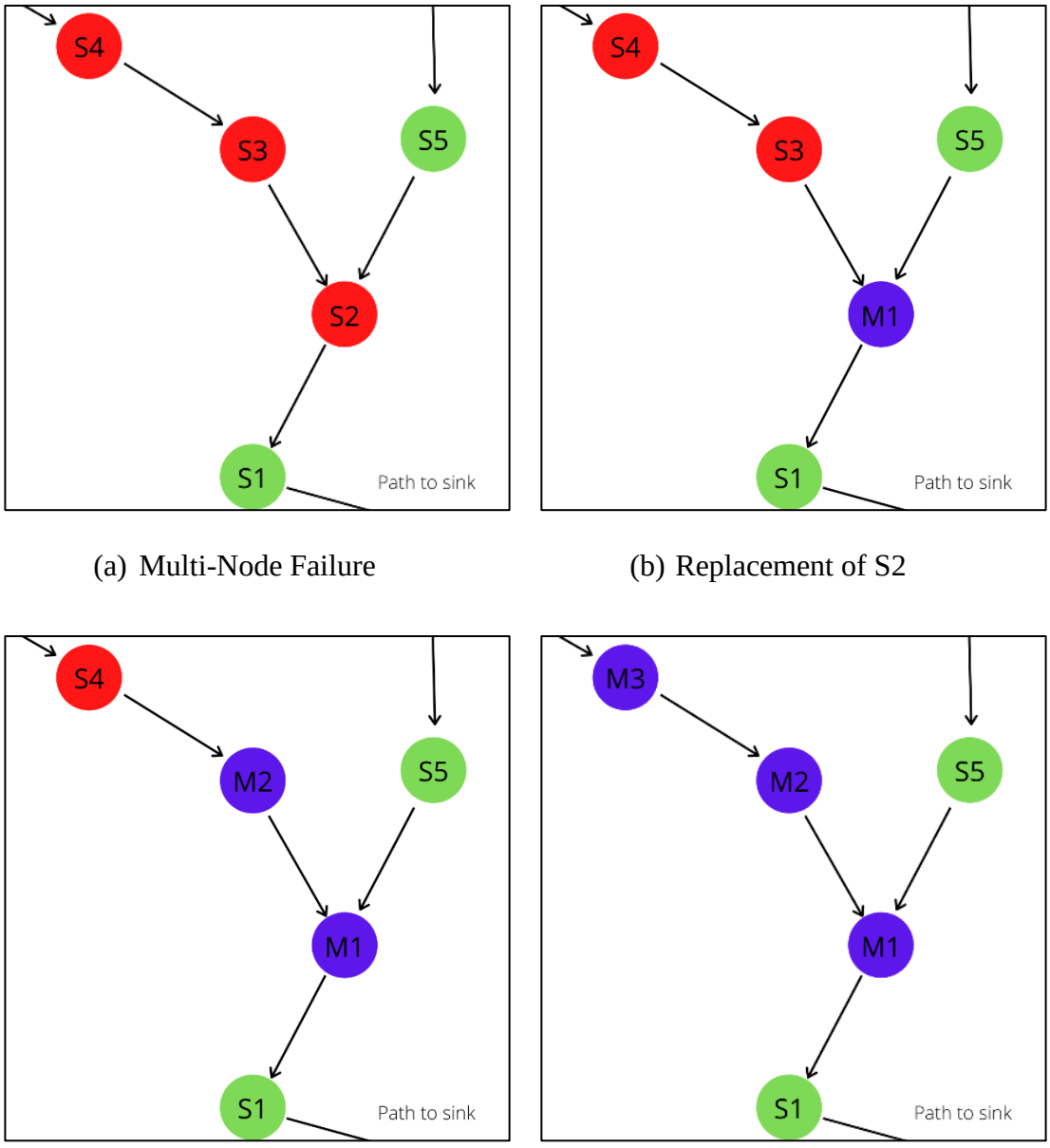


(a) Single Node Failure (b) Recovery

Figure 4.11: Recovery in Single Node Failure

Figure 4.11 shows an example of the single node failure case, where static nodes S1,S2,S3,S4 and S5 are part of the network. In the example of the Figure, sensor node S2 became faulty and disconnected a whole area of the network. S3 and S5 nodes have no node to forward their packets. Nodes S1, S3 and S5 recognizes that S2 stops functioning properly and takes action. Specifically, S3 and S5 inform their neighbor about the failure and S1 sends to the sink node an FNM message (Fault Reporting). Upon receiving the sink such a message, it sends a mobile node to the position of the faulty node. The mobile node M1 is placed in the S2 position and

starts investigating the neighborhood. Mobile node M1 finds all the expected neighbors, thus no further investigation is needed, and the network is successfully reconnected.



(a) Multi-Node Failure (b) Replacement of S2

© Replacement of S3 d) Replacement of S4

Figure 4.12: Recovery in Multi-Node Failure

Figure 4.12 shows an example of the multi-node failure case, where static nodes S1,S2,S3,S4 and S5 are part of the network. In this example, sensor nodes S2,S3 and S4 became faulty and disconnected a whole area of the network (Fig. 4.12 a). Some of the nodes do not have paths available to forward their packets to upper nodes. Nodes S1,S3 and S5 recognize that S2 stops functioning properly and act accordingly. Specifically, S5 inform its neighbor about the failure

and S1 sends to the sink node a FNM message (Fault Reporting). Upon receiving the sink such a message, it sends a mobile node to the position of the faulty node. The mobile node M1 is placed to the S2 position and starts investigating the neighborhood (Fig. 4.12 b). Mobile node M1 does not find all the expected neighbors and sends an IRNM message, which contains the id of the S1 and S5 nodes, to report its findings to the sink node. The sink figures out that S3 is the missing expected neighbor, which means that it is faulty, and sends a new mobile node to investigate this area and replace the faulty node S3. Mobile node M2 is placed to the position of the node S3 and starts investigating ((Fig. 4.12 c). Mobile node M2 identifies that an expected neighboring node is missing and sends to the sink node an IRNM message, that contains only the ID of the M1 node. After that M2 starts acting as a static node. The sink finds the missing node, node S4, and again sends a new mobile node to investigate further and replace the faulty node. The mobile node M3 moves to the S4 position and during the investigation process, all expected neighbors are found. All disconnected areas are successfully reconnected to the rest of the network.

In the RFMM each faulty node is replaced by a new mobile node that takes its place in the network. This mechanism achieves to reconnect the disconnected network; however, its recovery method demands the use of many extra resources and indicate the necessity of other mechanisms.

4.3 Decentralized Fault Management Mechanism (DFMM)

The Decentralized Fault Management Mechanism is based on distributed fault detection like RFMM. The DFMM differs from the RFMM in both phases (detection and recovery), where different technique is used in the discovery step and recovery solution. In the detection process, the distributed fault detection, the fault reporting, and the investigation methods follow the exact same steps as in the RFMM. After the sink is informed about the investigation process results, if there are any other faults, mobile node discovery will be performed instead of sending another mobile node. The sink will receive the results of the discovery process, calculate positions for the mobile nodes, and place as few mobile nodes as needed to reconnect the network. In the abstract, this mechanism places mobile nodes based on the gained knowledge from the discovery process, however, this mechanism detects failures locally and has its limitation. The DFMM consists of the following steps, which will be explained further:

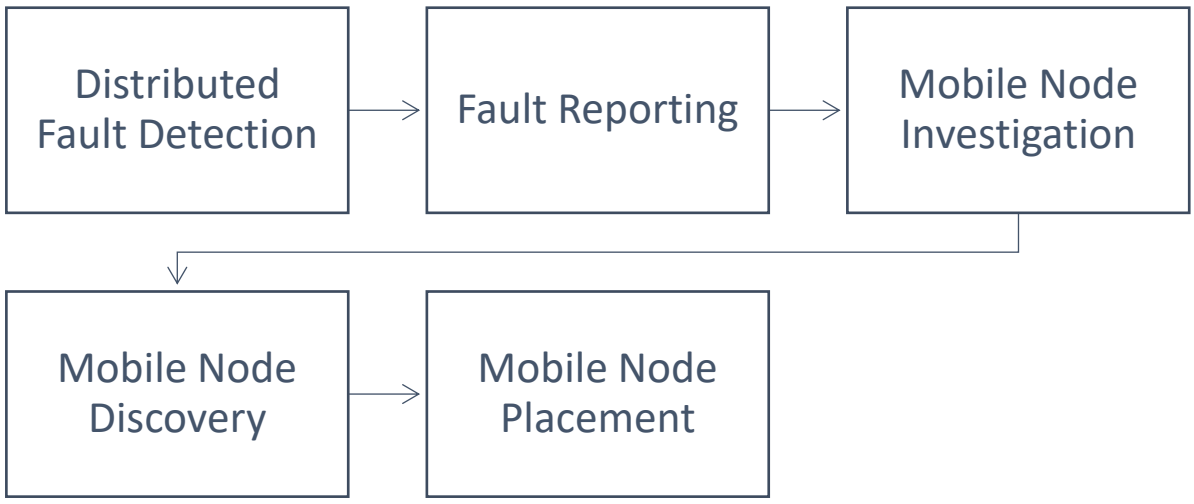


Figure 4.13: Decentralized Fault Management Mechanism

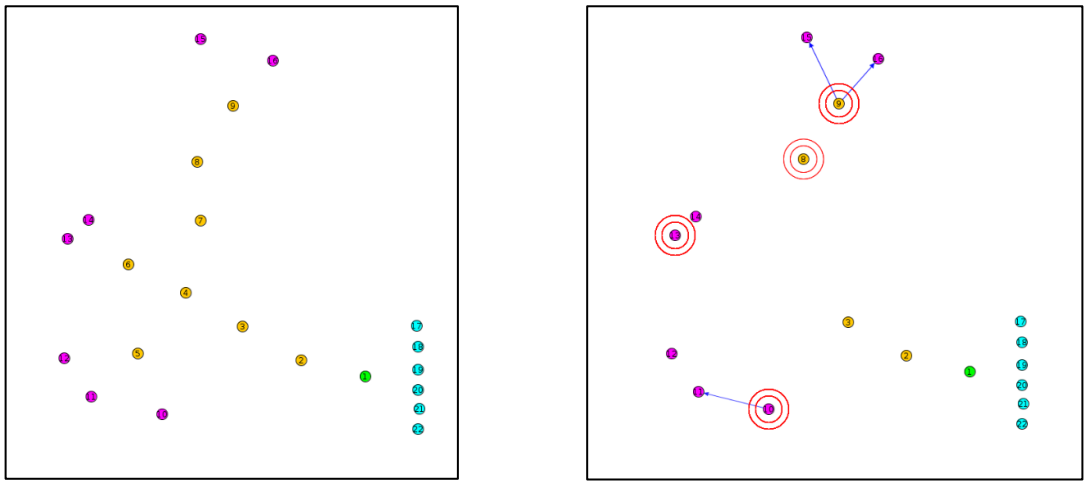
The mechanism is divided into two processes: (a) detection and (b) recovery. The detection process is responsible in identify the faults in the network and is divided into the following methods: distributed fault detection, fault reporting, mobile node investigation and mobile node discovery. Distributed fault detection aims to recognize the faults, while fault reporting aims to report the failure once a neighboring node acknowledges it. Mobile node investigation is performed by a mobile node and aims to investigate further for other faults in the affected area. The mobile node discovery step is responsible to discover failures on a deeper level. In the recovery process, the mobile nodes selected by the sink are placed at their calculated positions in order to reconnect the network. The algorithm for the static node remains the same as in RFFM. The algorithms are presented in Fig. 4.14 and 4.15:

```
1. Upon receive ("Faulty Notification(fn,level)") from n then
2.     If anyNodeStopSending() == false
3.       return
4.     // find the position of the faulty node
5.     pos = findPosition(fn)
6.     // number of expected neighbors
7.     neighbors = numberOfNeighbors(fn)
8.     // select an available mobile node
9.     mobile = selectMobileNode(pos)
10.    // send investigation notification to mobile node
11.    sendInvestigation(pos,level,neighbors,mobile)
12.
13. Upon receive ("Investigation Notification
   Response(aliveList)")  from m then
14.   // find faulty nodes
15.   faultyList = findFaultyNodes(m,aliveList)
16.   // create navigation path
17.   For each faulty in faultyList
18.       pos = findPosition(faulty)
19.       path.add(pos)
20.   // initial position of mobile node
21.   pos = findPosition(m)
22.   path.add(pos)
23.   // send Investigation Path message to m (sender of msg)
24.   sendInvestigationPath(path,m)
25.
26. Upon receive ("Investigation Path Response(aliveList)") from
   m then
27.   // constrained clustering
28.   noUpperNodesList = getNoUpper(aliveList)
29.   clustering = clustering(noUpperNodesList)
30.   For each cluster in clustering
31.      // calculate the position
32.      // target nodes are cluster members, dest is the mobile
33.      pos = DynamicMobileFT(cluster.members,m)
34.      mobile = selectMobileNode(pos)
35.      sendNewPosition(pos,mobile)
```

Figure 4.14: Decentralized Fault Management algorithm for sink node

```
1. Upon receive (“Investigation Notification
   (pos,level,expected)”) from sink then
2.     move(pos) // move to position
3.     my_level = level
4.     // broadcast alive request
5.     broadcastAliveRequest()
6.     wait for T time
7.     // count the alive nodes
8.     alive = countResponses()
9.     // send investigation respond to sink if it’s needed
10.    If responses != expected
11.       sendInvestigationResponse(neighbors,sink)
12.       idle = true // wait for discovery phase
13.    else
14.       // start acting as static node
15.       idle = false
16.
17. Upon receive (“Investigation Path(pathList)”) from sink then
18.    pos = pathList.remove()
19.    While (pathList.size() > 1)
20.       move(pos)
21.       broadcastAliveRequest()
22.       waitForTime(T)
23.       pos = pathList.remove()
24.    // inform sink node for its findings
25.    sendInvestigationPathResponse(findings,sink)
26.    // start acting as static node
27.    idle = false
28.
29. Upon receive (“New position(pos)”) from sink then
30.    move(pos)
31.    // introduce to neighbors and get the right level
32.    introduceToNeighbors()
33.    // start acting as static node
34.    Idle = false
```

Figure 4.15: Decentralized Fault Management algorithm for mobile node



(a) Healthy Network (b) Disconnected Network

Figure 4.16: Network Disconnection

An example of disconnection is shown in figure 4.16. Relay nodes 4,5,6,7 crashed and as a result, the whole network disconnected.

4.3.1 Detection, Reporting and Investigation

As we have mentioned before, the detection phase is very similar to the detection phase of the RFMM. The distributed fault detection, the fault reporting and the mobile node investigation processes remain the same. However, sink, before sending a mobile node for investigation, checks if there are nodes that stops sending packets with the same threshold was set for distributed fault detection to avoid sending mobile nodes without need.

Each node in the network broadcasts an alive message periodically, intending to announce its healthy status to its neighbors. Additionally, each node executes the detection method periodically to recognize any fault in the network (section 4.2.1). The node, which has detected the fault from its downstream neighboring nodes, sends an FNM to the sink node. The node, which has detected the fault from its upstream neighboring nodes, will recognize if there is no other upstream node and inform the other nodes about the failure (section 4.2.2). The sink node, when it receives an FNM message, checks if there are any nodes that sending packets. If there are nodes that stops sending packets, then the sink sends an Investigation Notification to a mobile node for investigating the affected area. After that, the mobile node moves to the target position and starts the investigation process and informs the sink node about its finding using an IRNM. At this point, DFMM will continue with the Mobile Node Discovery step and not the Replacement method step.

4.3.2 Mobile Node Discovery

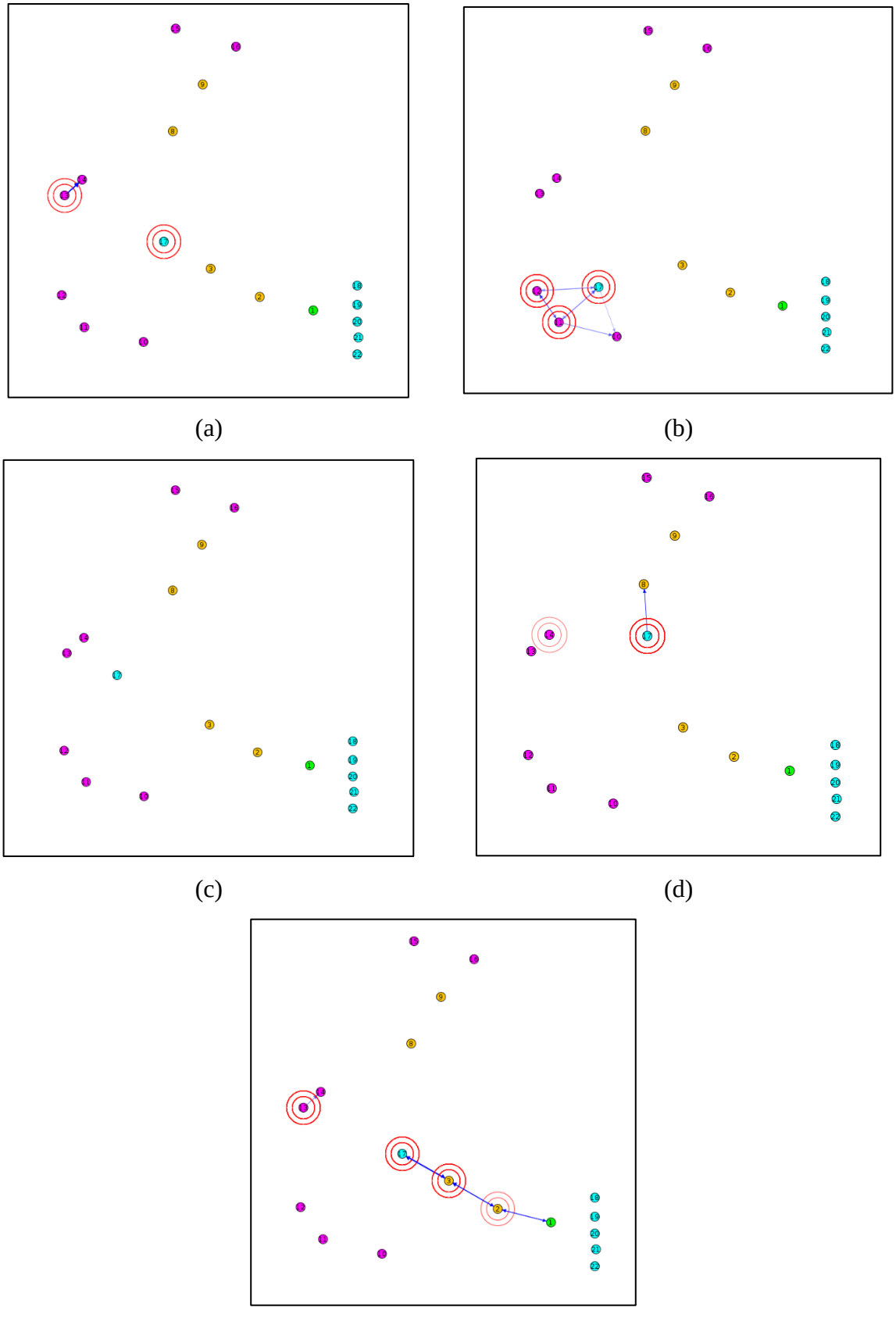
The sink upon receiving the IRNM finds which nodes did not respond to the message of the mobile node. The positions of these nodes are sent to the mobile node with the IP Message. In the one-level investigation, the mobile node receives from the sink the IP message that contains the positions of the non-responding nodes that it should visit and investigate further. When the mobile node arrives at these positions, it broadcast an “alive request” message and waits for responses. Nodes, which receive this message request, will respond with an alive response that contains if they have at least an upper node or not. At the end of this procedure, the mobile node returns to its starting position and informs the sink about its finding with an Investigation Path Response (IPR) , which includes all the nodes found alive (ID) and if they have an upper node or not (0/1). By finding all nodes without an upper node, mobile node can inform the sink about the nodes that need to be reconnected and not include the nodes that already are connected somehow in the network.

IPR Message:

“IPR”	ID	0/1	ID	0/1	….

Figure 4.17: IRP Message Structure

Figure 4.18 shows an example of the mobile node discovery. At first, mobile node 17 places to the faulty node position, investigate the area and recognize that other failures exist (Fig. 4.18a). Report its findings to sink, and then it moves to the neighbors’ positions to discover other failures (Fig. 4.18 b,c,d) At every position, it stops, sends alive requests and wait for the nodes to reply. In the end of this method, mobile node 17 moves back to its initial position and sends a message to the sink to notify it about the state of nodes in this neighborhood.

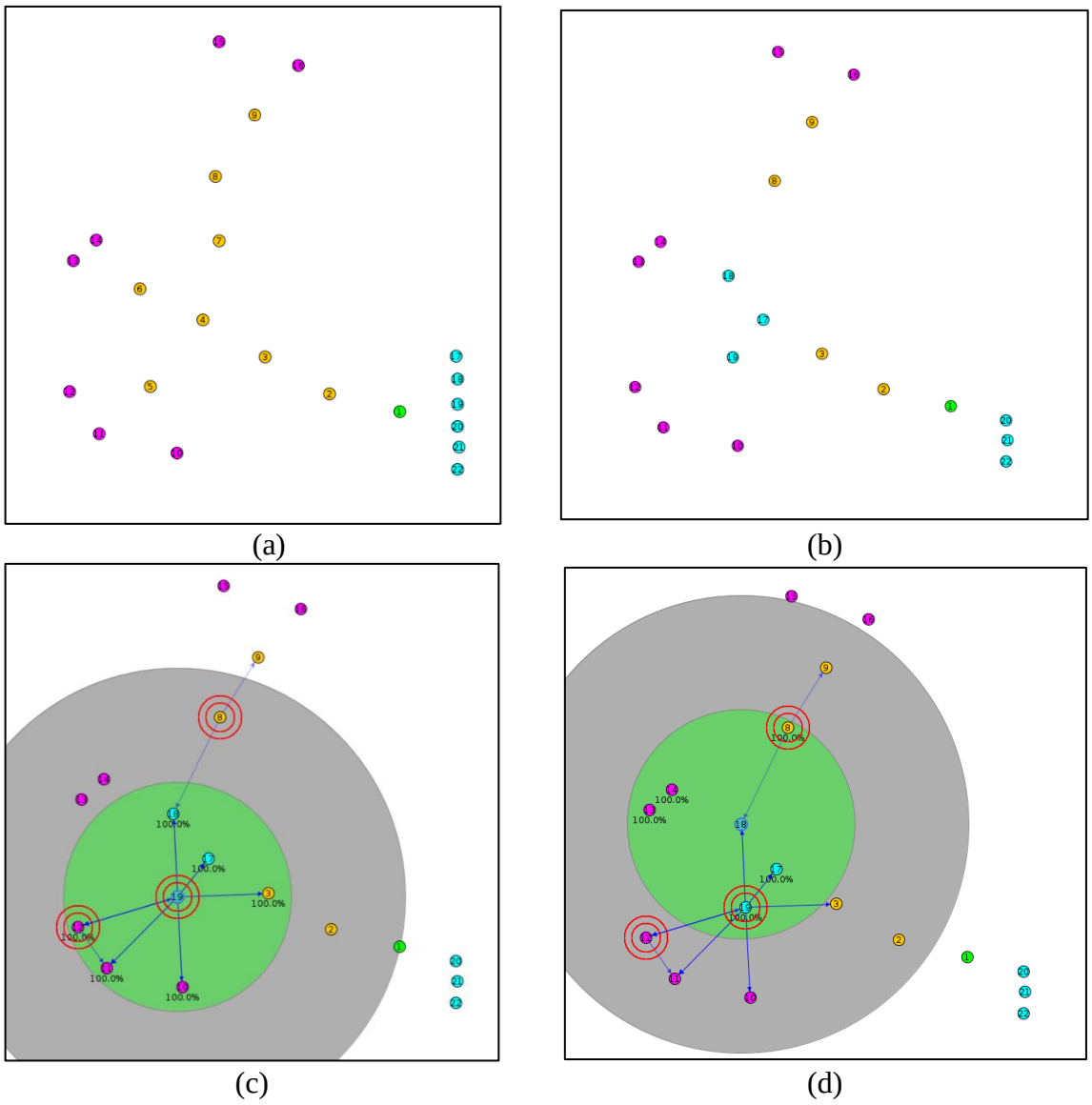


(a) (b) (c) (d) (e)

Figure 4.18: Mobile Node Investigation and Discovery

4.3.3 Mobile Node Placement

Based on the information collected by the IPR message, the sink calculates the number of mobile nodes and their position that are needed for restoring the network. This procedure is divided into two methods: (a) the constraint clustering method and (b) the positioning method. Firstly, the clustering method is used, where the sink node divides all nodes without an upper node into clusters based on their position with the modified k-means algorithm that is presented in subchapter 3.1. After that, the positioning method is used, where the Dynamic MobileFT algorithm (presented in section 3.2.3) runs to find the position of the mobile node for each cluster created. When the position is calculated, the sink selects an available mobile node and sends it to the target position. When a mobile node receives a New Position message it moves to the position, introduces itself and starts acting like a static node.



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

Figure 4.19: Recovery of Disconnected Network

we present in the figure 4.19 the recovery phase of the disconnected network (Fig. 4.18). Two additional mobile nodes placed in order to reconnect the network (Fig. 4.19 b). Specifically, mobile node 19 can communicate with static nodes 10,11 and 12 (Fig. 4.19 c) and mobile node 18 can communicate with static nodes 8, 13 and 14 (Fig. 4.19 d). Both mobile nodes 18 and 19 can forward packets to the mobile node 17.

4.3.4 Limitations

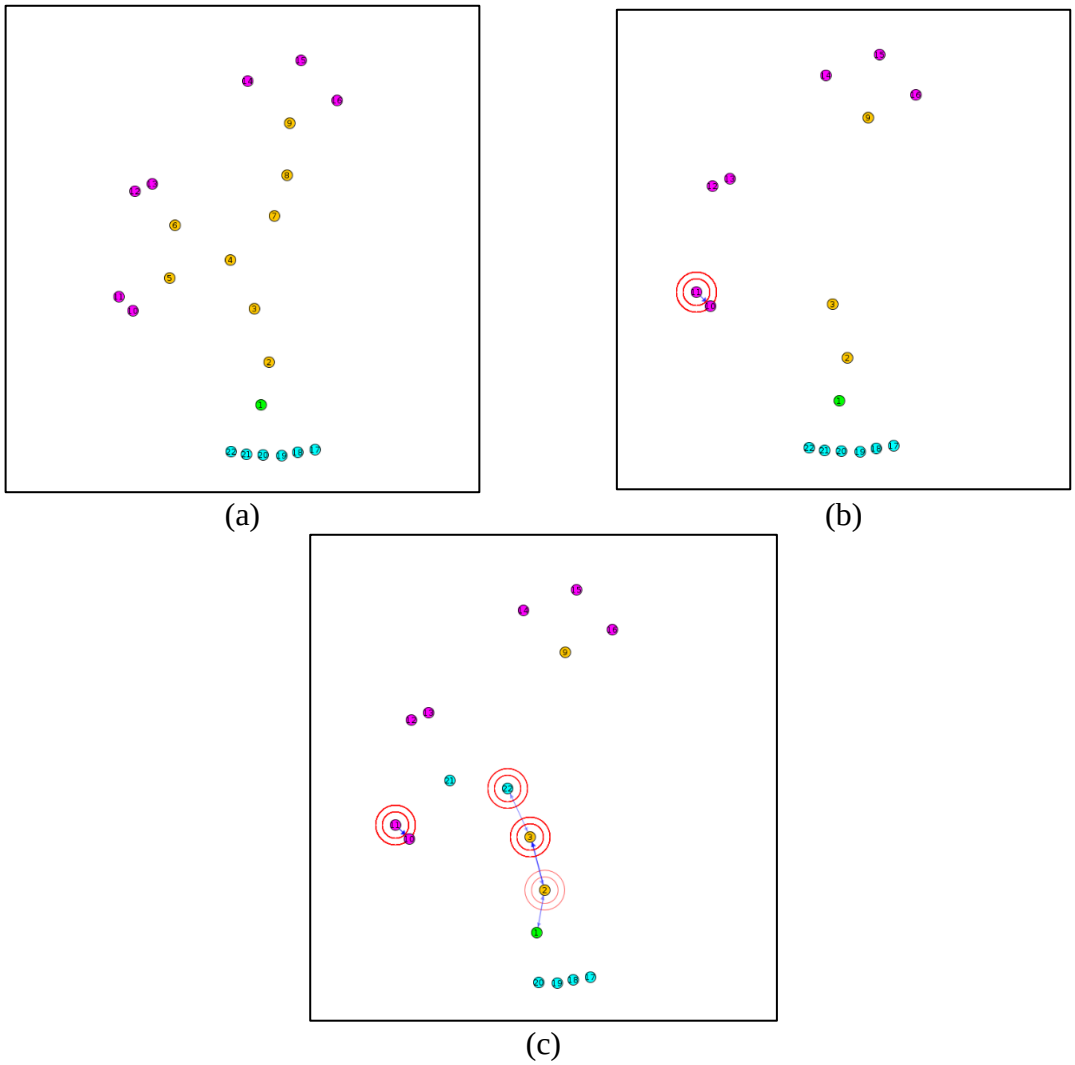


Figure 4.20: Limitations of DFMM

As we can see in Figure 4.20, the DFMM was unable to reconnect the network. That happens because in the discovery method mobile node couldn't communicate with the node 9, and thus couldn't reconnect the area of this node.

In conclusion, the DFMM can reconnect the network in case of disconnection with the minimum mobile nodes, however it cannot be used to resolve a two-level or more disconnection.

4.4 Centralized Fault Management Mechanism (CFMM)

This mechanism is based on centralized fault detection where the detection is performed by the sink node. As we mentioned before, DFMM comes with limitations, or to be more specific it can resolve only one-level disconnections. In contrast, CFMM can solve two-level or more disconnections, however it needs more time and energy in comparison with the DFMM. In this mechanism, a mobile node is sent to the affected area by the failure to discover the faults and comes back to the sink node to report its findings, thus the journey of the mobile node can be time-consuming and energy consuming. However, it can place fewer mobile nodes in comparison with the RFMM.

The CFMM consists of the following steps:

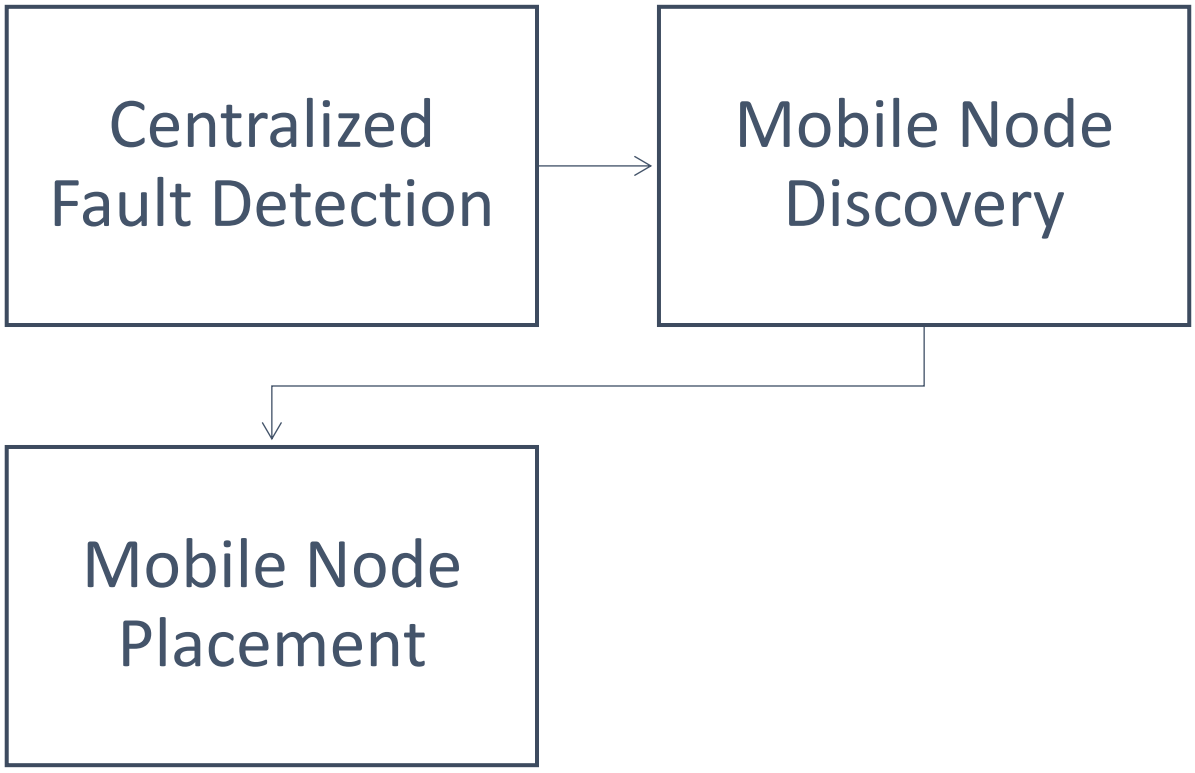


Figure 4.21: Centralized Fault Management Mechanism

The mechanism is divided into two processes: (a) detection and (b) recovery. The detection process is responsible for detecting the fault in the network and includes the centralized fault detection step and the mobile node discovery step. The recovery process is responsible for resolving the failure that occurred in the network by placing mobile node(s).

we previously presented a semi-supervised partitional clustering technique (subchapter 3.1). This mechanism uses the clustering technique for the following reasons:

1. Divide the suspicious faulty nodes (SNF-list) into non-overlapping subsets such that all nodes in a cluster can be investigated at the same time. (Mobile Node Discovery)
2. Divide the alive nodes into non-overlapping subsets such that all nodes in a cluster can communicate with a mobile node and reconnect them to the network. (Mobile Node Placement)

The algorithms for the static and mobile node remains the same as in DFFM. The centralized fault management algorithm for the sink node is presented above in Fig. 4.22.

```
1. //centralized detection based on threshold
2. Upon detect failure then
3.     // find the center based on nodes that stop sending packets
4.     center = calculateCenterOfSupsiciousArea()
5.     // a list of all nodes in suspicious area
6.     SFN_list = createSFNList(center)
7.     // clustering of SFN list and positions for mobile node
8.     navigation_path = createNavigationPath(SFN_List)
9.     // select an available mobile node
10.         mobile_node = selectMobileNode()
11.    // send Investigation Path Message to the mobile node
12.    SendIPMsg(navigation_path, mobile_node)
13.
14.Upon receive ("Investigation Path Response(nodesList)") from
   m then
15.   Clusters = clustering(nodesList)
16.   For each cluster
17.       pos = DirectMobileFT(cluster)
18.       mobile_node = selectMobileNode()
19.       sendMobile(mobile_node,pos)
```

Figure 4.22: Centralized Fault Management algorithm for sink node

4.4.1 Centralized Fault Detection

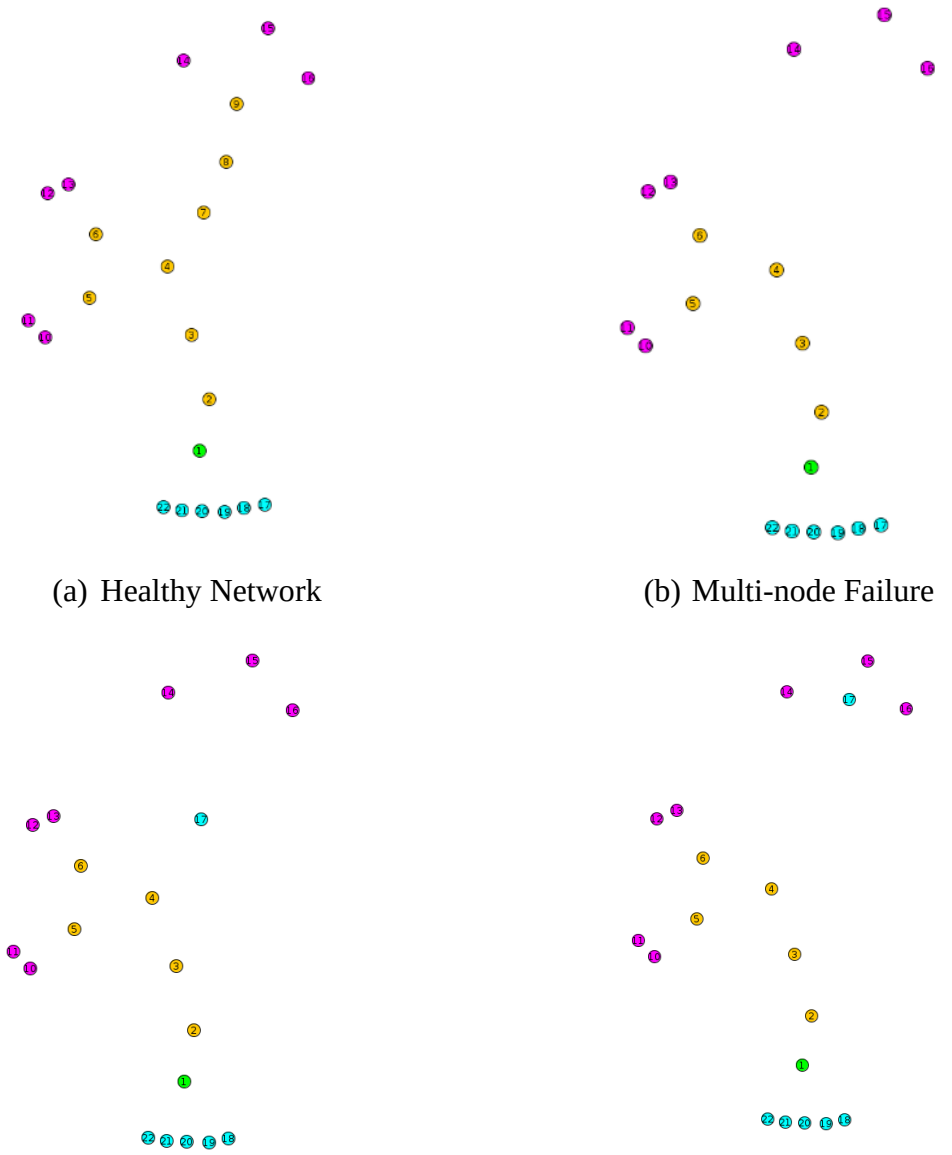
The sink node receives all collected data from the network and is aware from which node it receives data. Based on the receiving information it can identify if a node stops sending packets. The sink node once it recognizes that several nodes stop functioning correctly (sending packets) based on a threshold which is greater than the previous mechanisms, it creates a Suspicious Faulty Node list (SFN list). This list contains all the nodes that stop sending packets and nodes in their neighborhood in case of a disconnection. Then, the mobile node discovery process starts, where a mobile node should find out the alive nodes from the SFN list.

4.4.2 Mobile Node Discovery

The SFN list contains all the suspicious faulty nodes, and a mobile node should check the nodes of the SFN list if they are still alive. In order to create the SFN list, the sink finds the faulty area, which can define by calculating the centroid c of the nodes that stop sending packets and then choosing the circular area with the centroid c as the center and radius r, in this model it is twice as the coverage of the sensors. The suspicious faulty nodes are all the nodes inside this faulty area. In this model, we assumed that faults could happen in a specific area only and can be covered by the faulty area.

Once the SFN list is created, the sink uses the constraint clustering method to divide the SFN list into clusters that all members can investigate at the same time by the mobile node. After that, the sink sends an Investigation Path Message (IP) that contains the positions of the target positions, which are the clusters' centroids. When the mobile node arrives at these positions, it broadcast an "alive request" message and waits for responses. Nodes, which receive this message request, will reply with an alive response that contains if they have at least an upper node or not. At the end of the mobile node discovery procedure, the mobile node returns to its starting position, near the sink, and informs the sink about its finding with an Investigation Path Response (IPR), which includes all the nodes found alive (ID) and if they have an upper node or not (0/1).

Figure 4.23 shows an example of the mobile node discovery method, where the nodes 14,15 and 16 stop sending packets and the sink node suspects failures. The mobile node should visit two positions, the first one aims to check nodes 7 and 8, and the second one aims to check nodes 9,14,15 and 16. At the end of the method, the mobile node comes back to its initial position and reports to the sink its findings. The recovery phase of this disconnection is shown in Fig. 4.24.



(a) Healthy Network (b) Multi-node Failure

(c) Discovery Position 1 (d) Discovery Position 2

Figure 4.23: Mobile Node Discovery

4.4.3 Mobile Node Placement

Based on the information collected by the IPR message, the sink calculates the number of mobile nodes and their position that are needed for restoring the network. This procedure is divided into two methods: (a) the constraint clustering method and (b) the positioning method. Firstly, the constraint clustering method is used, where the sink node divides all nodes without an upper node into clusters based on their position. After that, the positioning method is used, where the Direct MobileFT algorithm (presented in section 3.2.4) runs to find the position of

the mobile node for each cluster created. In the Direct MobileFT algorithm, the target nodes are the clusters' members, and the destination is the sink node, however, the algorithm is modified to stop when finding a node in the coverage of the last-placed node. When the positions are calculated, the sink selects available mobile nodes and sends them to the target positions. When a mobile node receives a New Position message it moves to the position, introduces itself and starts acting like a static node.

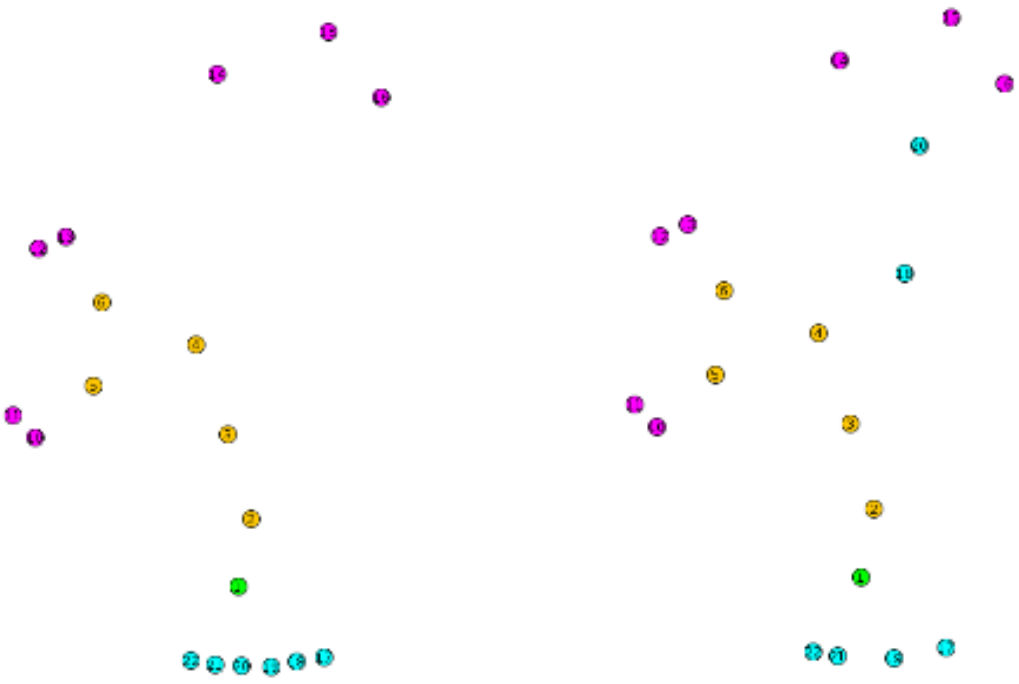


Figure 4.24: Mobile Nodes Placement

An example of the mobile node placement step is illustrated in Fig. 4.24, where two mobile nodes are placed to reconnect the static nodes 14,15,16 to the network after the discovery step.

4.5 Mobile Fault Management (MobileFM) Framework

we present the Mobile Fault-Management (MobileFM) framework that utilizes mobile nodes to reconnect the disconnected areas and handle failures of the network. The idea behind this framework is to ensure the proper functionality of the network by replacing the faulty source nodes and keeping the connectivity towards the sink node by placing mobile nodes. This framework combines the Replacement Mechanism, the Decentralized Fault Management Mechanism and the Centralized Fault Management Mechanism (Fig. 4.25).

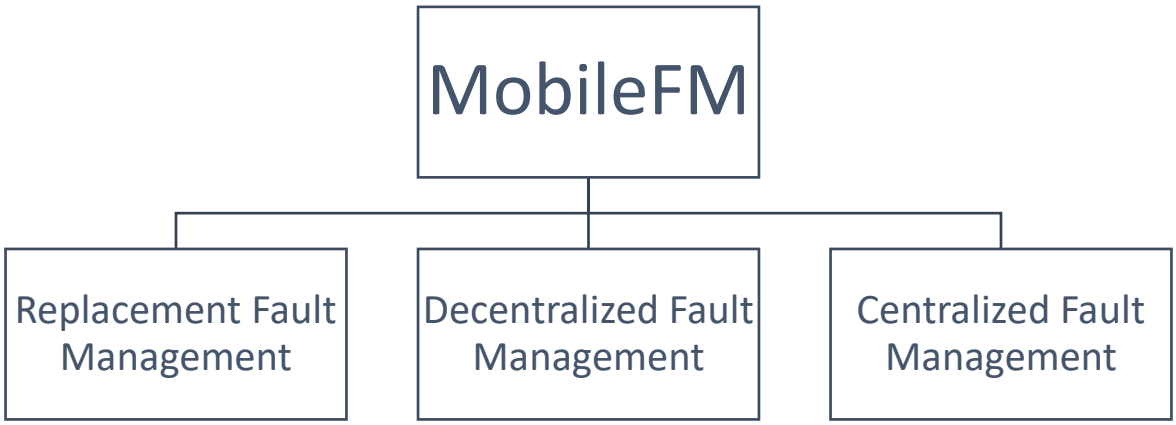
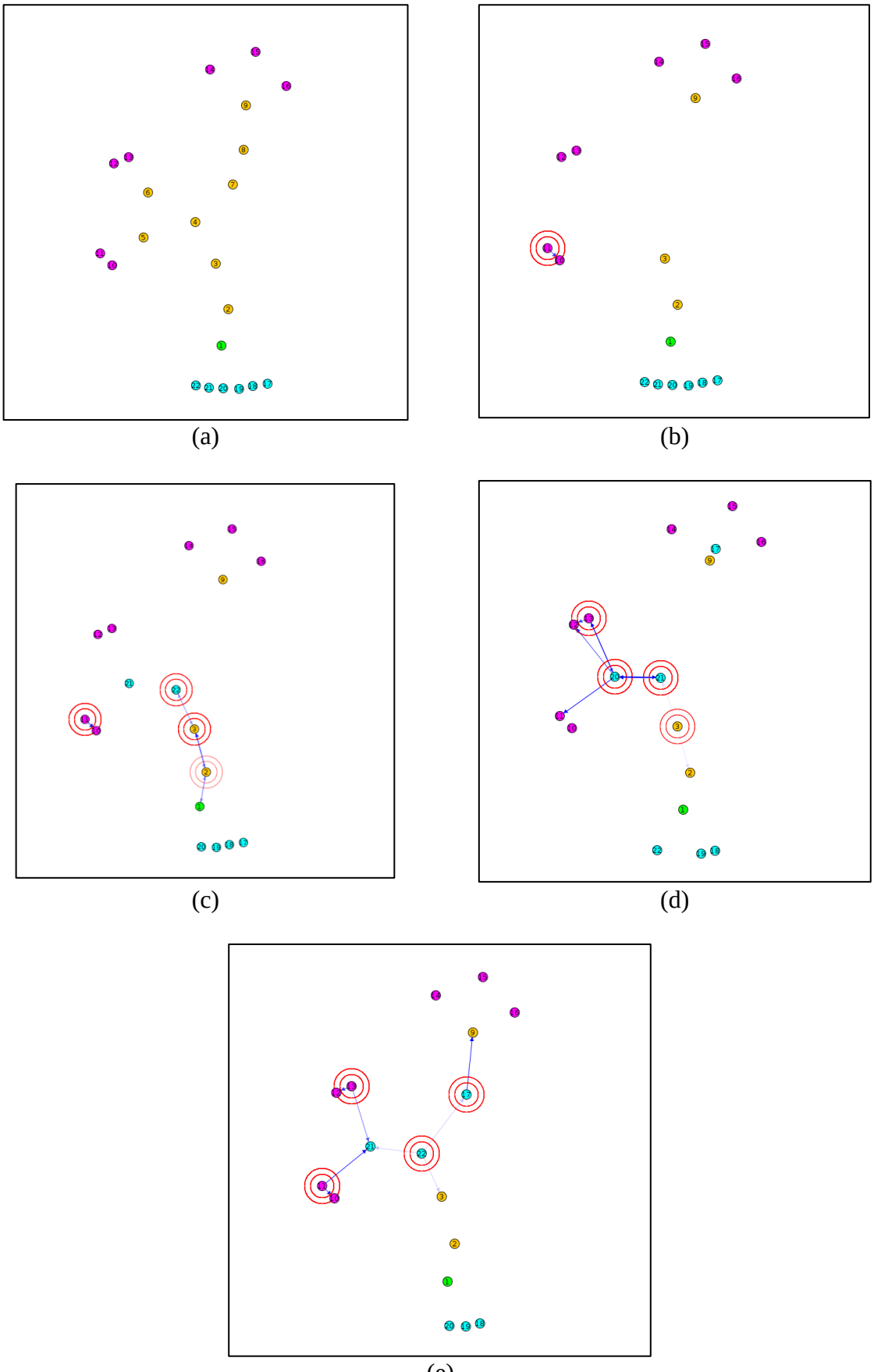


Figure 4.25: MobileFM Framework

In this framework, the fault detection has two aspects. The first aspect is a distributed detection (by itself or neighbors) where the detection method is done by each node, while in the second aspect, the centralized detection, the detection is performed by the sink node. Based on the detection a different approach to the solution is presented. Since the threshold for the centralized fault detection is higher than the threshold for the distributed fault detection, it can be expected that MobileFM tries to resolve the disconnection using the DFMM at first and then if the network is not reconnected, the CFMM is triggered to try and resolve the disconnection problem. The DFMM and CFMM should not perform simultaneously.

It is essential that the WSN is capable of reaching the desired coverage area. While the MobileFM uses the DFMM and CFMM to ensure connectivity with the minimum number of mobile nodes, it also uses the RFMM to ensure that the desired coverage area of our sensor network is reached by replacing the source nodes with mobile nodes. When a mobile node replaces a faulty source node, the mobile node investigates other source node failures.



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

Figure 4.25: Overcoming limitations of DFMM

Recalling the limitations of the DFMM back in section 4.3.4, the MobileFM could use the CFMM when the DFMM cannot solve the disconnection in order to discover the affected area and reconnect the disconnected area as in Fig. 4.25 (d). In particular, DFMM reconnects nodes 10,11,12 and 13 and recognize nodes 7 and 8 as faulty. The sink identifies that the nodes 14,15 and 16 stop sending packets and the CFMM is used to discover the affected area and reconnect the nodes 14,15 and 16 by creating a path from node 9 to node 22.

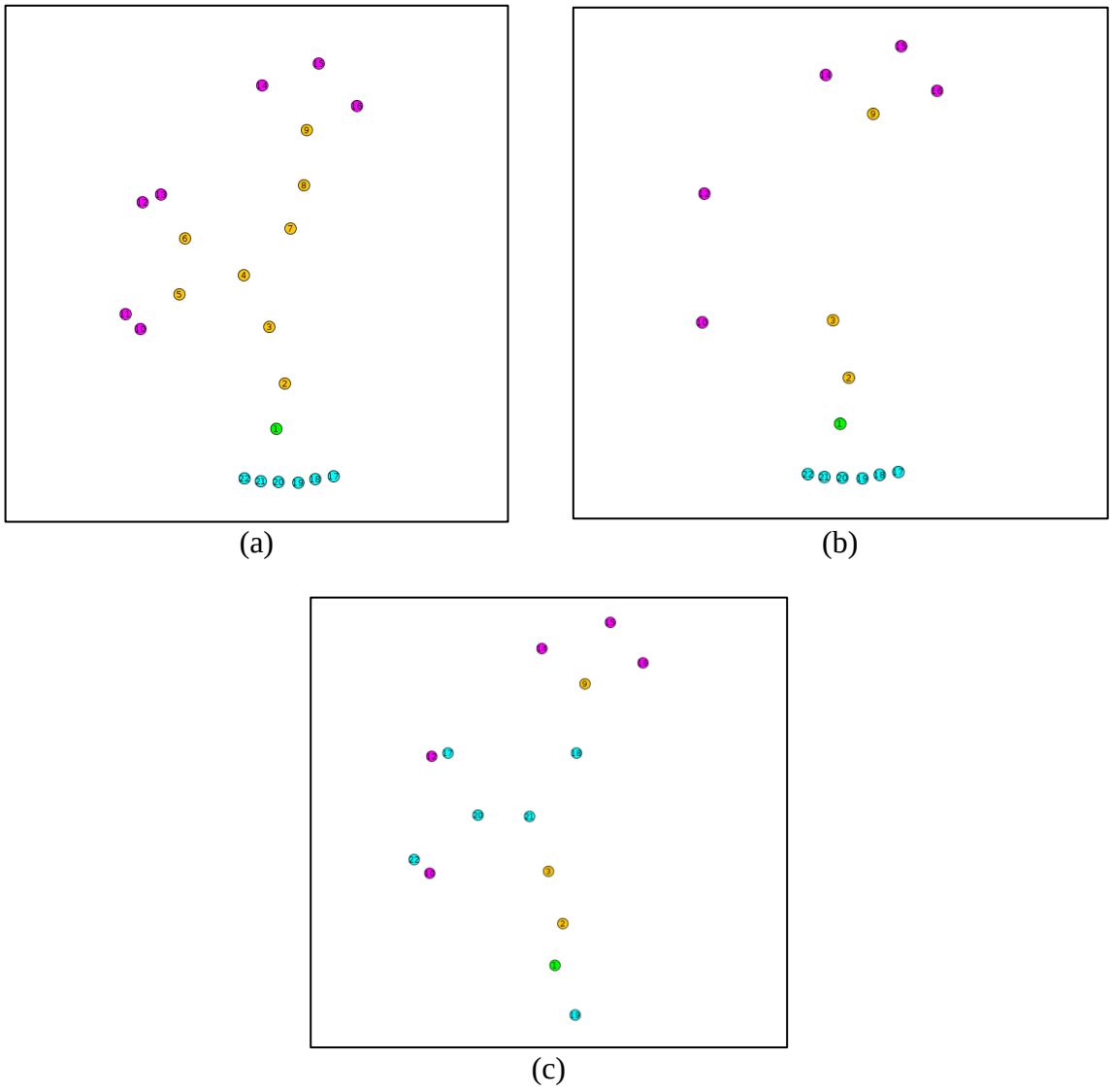


Figure 4.24: MobileFM

In Fig. 4.25 all mechanisms are used to ensure network functionality. In particular, static nodes 4,5,6,7,8,11 and 13 became faulty. Then DFMM placed the mobile nodes 20 and 21 to act as relay nodes and reconnect the static nodes 10 and 12, RFMM placed mobile nodes 17 and 22 to act as source nodes and replace source nodes 11 and 13 and finally CFMM placed the mobile node 18 to reconnect node 9 to the network.

Chapter 5

Evaluation, Results and Discussion

To prove the effectiveness of our strategies, different scenarios were implemented. For the evaluation, we focus on solving the disconnection problem and how the suggested mechanisms can be beneficial. In every scenario, the disconnection was solved by one of the mechanisms. In particular, we compared the performance of the RFMM, as the simplest strategy, between (a) the DFMM, (b) the CFMM, and (c) the MobileFM.

5.1 Evaluation Setup

The evaluation has been performed in the COOJA simulator, a dedicated simulator for Contiki OS nodes [3]. The proposed strategies were implemented within the Contiki OS. Contiki is an open-source operating system that runs on tiny low-power microcontrollers and makes it possible to develop applications that make efficient use of the hardware while providing standardized low-power wireless communication for a range of hardware platforms.
For simulating purposes, we developed a C source file for each type of node in the network. Specifically, we developed the SinkNode, SoucreNodes, RelayNodes, MobileNodes and the script file. The code is well-documented, without any warnings and known bug.

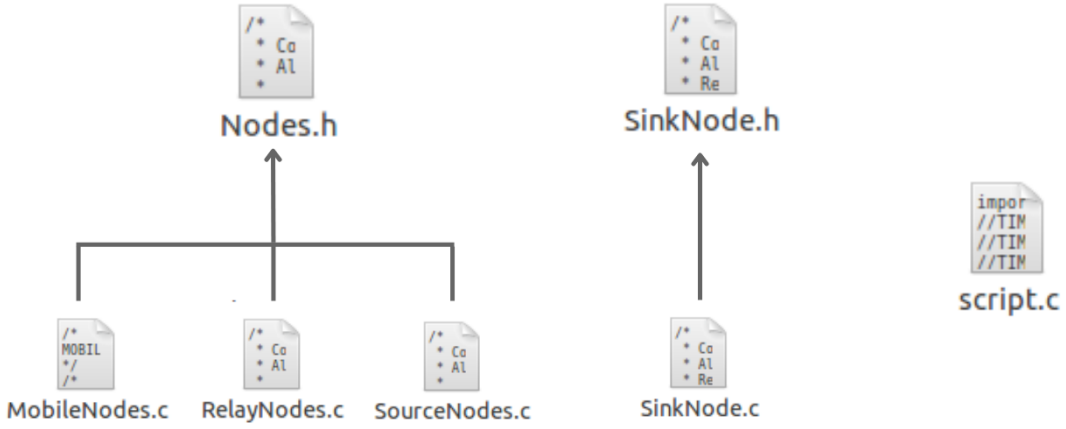


Figure 5.1: Project Structure

All Contiki programs are divided into different processes. A process is a piece of code that is executed regularly by the Contiki system. In the simulation, processes are auto started at the beginning and then each process runs when something happens, such as a timer firing. We are using the Contiki timer library for real-time task scheduling. For the purposes of our evaluation, we set three processes for each node as follows: (a) multi-hop process (b) energy process (c) fault detection. Below the processes of a source node is shown:

```
PROCESS_THREAD(energy, ev, data){
  PROCESS_BEGIN();
  printf("PROCESS ENERGY BEGINS\n");
  while(1){
    if (faulty)
      break; // for loop
    remaining_energy = (energest_type_time(ENERGEST_TYPE_TRANSMIT) *
19.5 + energest_type_time(ENERGEST_TYPE_LISTEN) *21.8 +
energest_type_time(ENERGEST_TYPE_CPU) * 1.8 +
energest_type_time(ENERGEST_TYPE_LPM) * 0.0545 ) * 3 / 4096 *8;
        printf("Time: \t %lu \t Remain energy \t
%ld.%03u\n",clock_time(),(long)remaining_energy,(unsigned)
((remaining_energy-floor(remaining_energy))*1000));
    etimer_set(&timer_energy, CLOCK_SECOND*20);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_energy));
  }
  PROCESS_END();
}
```

Figure 5.3: Energy Process

In the energy process (see Fig.5.2), the node calculates its consumed energy every twenty seconds based on energy equations that will be present in subchapter 5.3.4.

```
PROCESS_THREAD(fault_detection, ev, data){
  PROCESS_EXITHANDLER(unicast_close(&uc);)
  PROCESS_BEGIN();
  printf("PROCESS FAULT DETECTION BEGINS\n");
  etimer_set(&timer_fault, CLOCK_SECOND*65);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_fault));
  while(1){
    if (faulty)
      break; // for loop
    if (connection)
      faultdetection();
    etimer_set(&timer_fault, CLOCK_SECOND*30);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&timer_fault));
  }
  PROCESS_END();
}
```

Figure 5.3: Fault Detection Process

In fault detection process (see Fig. 5.3), the node check if there is any failure in its neighborhood every 30 seconds.

```
PROCESS_THREAD(example_multihop_process, ev, data){
  PROCESS_EXITHANDLER(multihop_close(&multihop);)
  PROCESS_EXITHANDLER(announcement_remove(&example_announcement);)
  PROCESS_BEGIN();
  printf("PROCESS MULTIHOP BEGINS\n");
  multihop_open(&multihop, CHANNEL, &multihop_call);

announcement_register(&example_announcement,CHANNEL,received_announce
ment);
  /* Allow some time for the network to settle. */
  etimer_set(&et, 60 * CLOCK_SECOND);
  PROCESS_WAIT_UNTIL(etimer_expired(&et));
  /* Loop forever, send a packet. */
  while(1) {
    if (faultyNode(linkaddr_node_addr.u8[0]))
      break; // for loop
    etimer_set(&et, CLOCK_SECOND *10);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    if (connection == true)
      sendPeriodicMessage();
    else{
      uint16_t time = clock_seconds();
      printf("Packet lost! %u\n",time);
    }
  }
  PROCESS_END();
}
```

Figure 5.4: Multi-hop Process

In the multi-hop process (see Fig.5.4), the node enables its multi-hop and announcement channels and sends a periodic message every 10 seconds (only source nodes). If a node loses its connection to the network, it stops sending packets until the connection is restored.

The following table presents the simulation parameter:

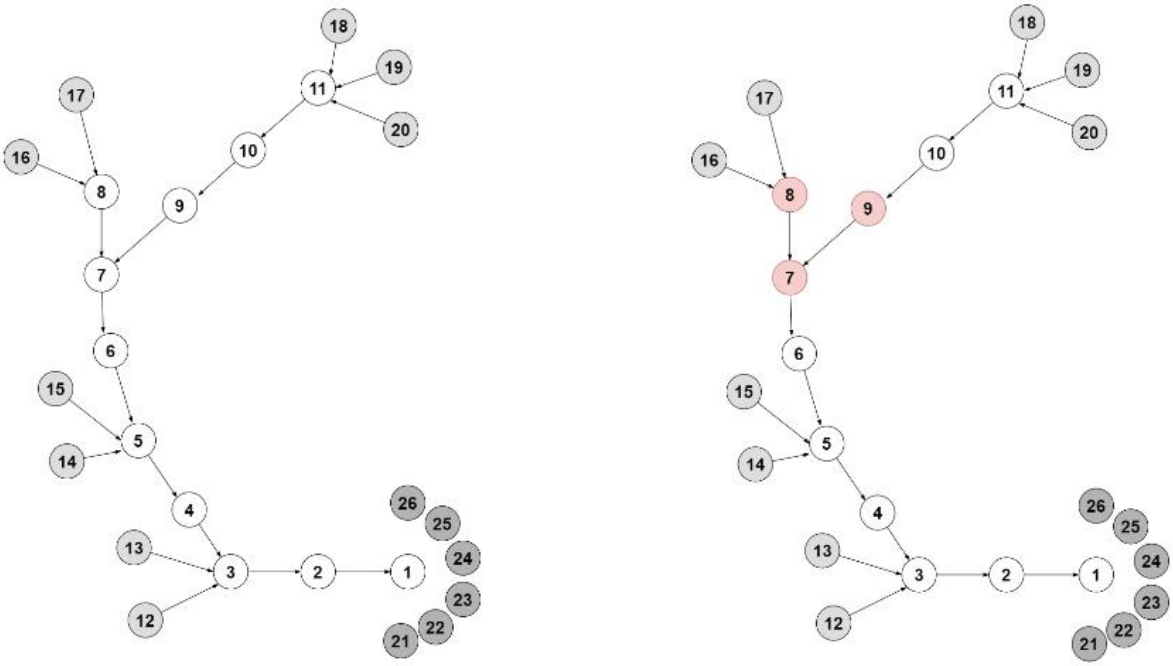
Simulator/OS	COOJA/Contiki 3.0
Protocol	Contiki Multihop/Rime
MAC	Contiki MAC/CSMA
Simulation Time	30-60 mins
Emulated Mote	Tmote sky
Number of Nodes (Sink/Fixed/Mobile)	1/16-19/6
Transmission Range (m)	50
Max Data Rate (kbps)	250
Queue Length (Pkts)	8
Packet Size (Bytes)	48
Initial Source Rate (Pkts/sec)	25
Mobile Node Speed	0.65 m/s

Table 5.1: Simulator Parameters

In the simulation, all sensors are Sky Mote nodes and have a 50m radio range. Each sensor node transmits one data packet of 48 bytes every 10 seconds. The network is set up and let to reach a steady state for one minute. Then, between the second and third minute of operation failures occur, and the mechanisms identify and resolve them.

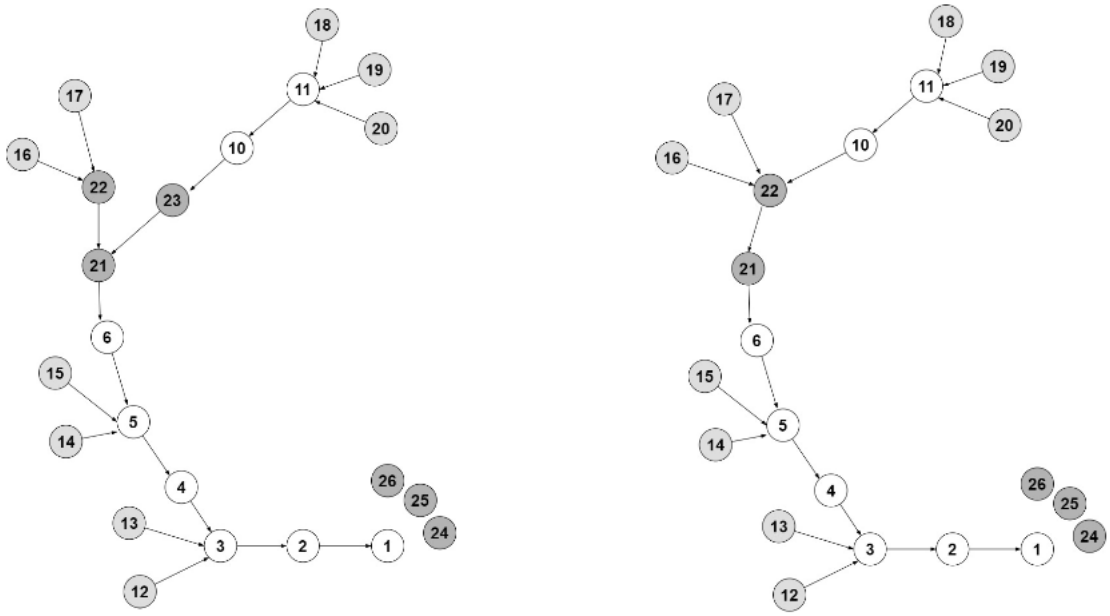
5.2 Resulting Topologies

In this section, we present the resulting topologies for the evaluation scenarios used. In each scenario, the network topology and the faults in the network are presented. Then, the recovery phase of the simple replacement mechanism RFMM and the recovery phase of the mechanism DFMM/CFMM/MobileFM are illustrated, thus the differences in the topology can be observed.



(a) Initial Topology (b) Faults in the Network

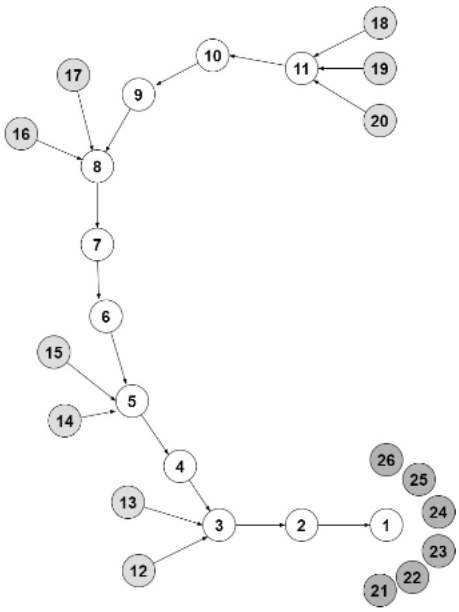
Figure 5.2: Scenario 1



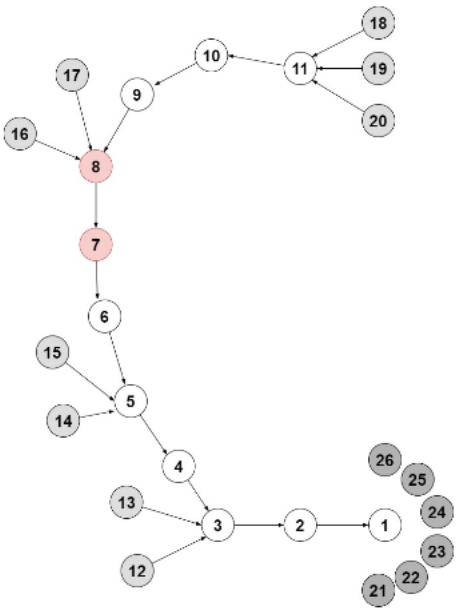
(a) RFMM (b) DFMM

Figure 5.3: Recovery in Scenario 1

As illustrated in Fig 5.2 and 5.3, nodes 7 and 8 became faulty, and two disjoint segments disconnected from the network. RFMM replaced all the faulty nodes with mobile nodes. However, the DFMM achieved to place 2 mobile nodes instead of 3 for reconnecting the disconnected segments.

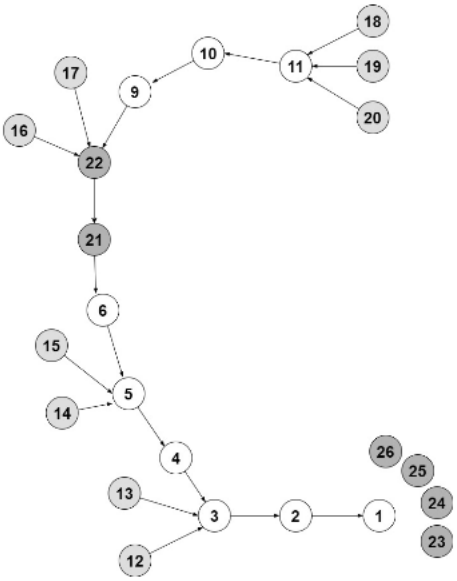


(a) Initial Topology

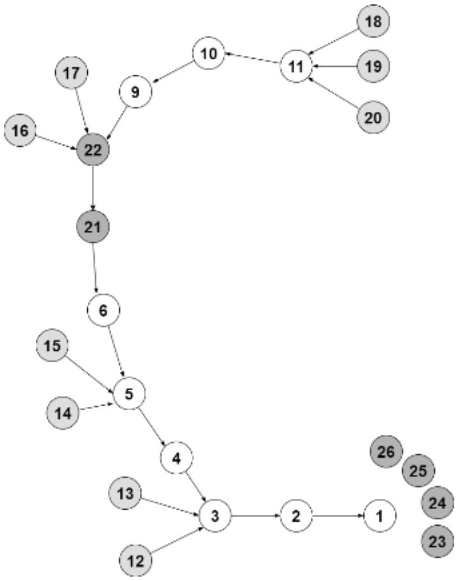


(b) Faults in the Network

Figure 5.4: Scenario 2



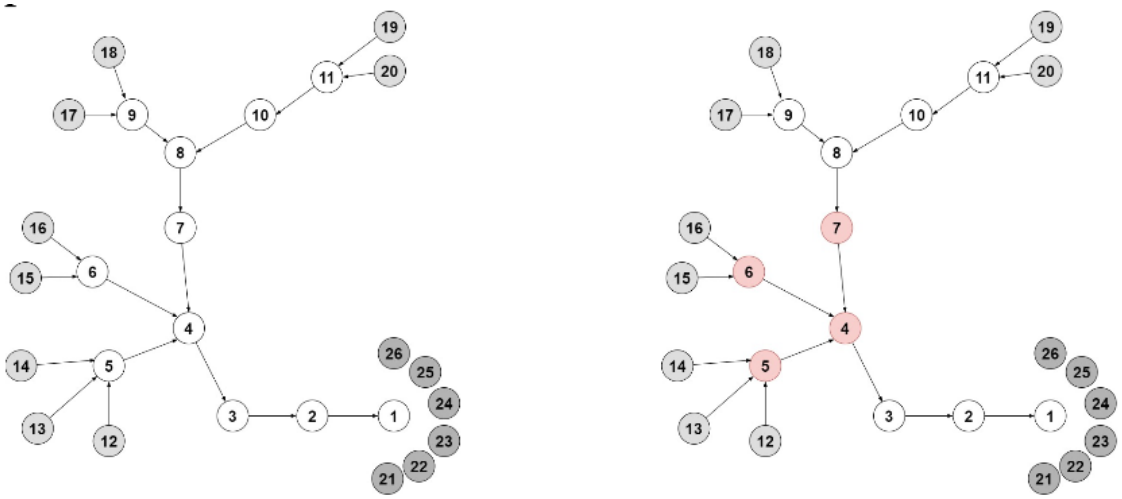
(a) RFMM



(b) DFMM

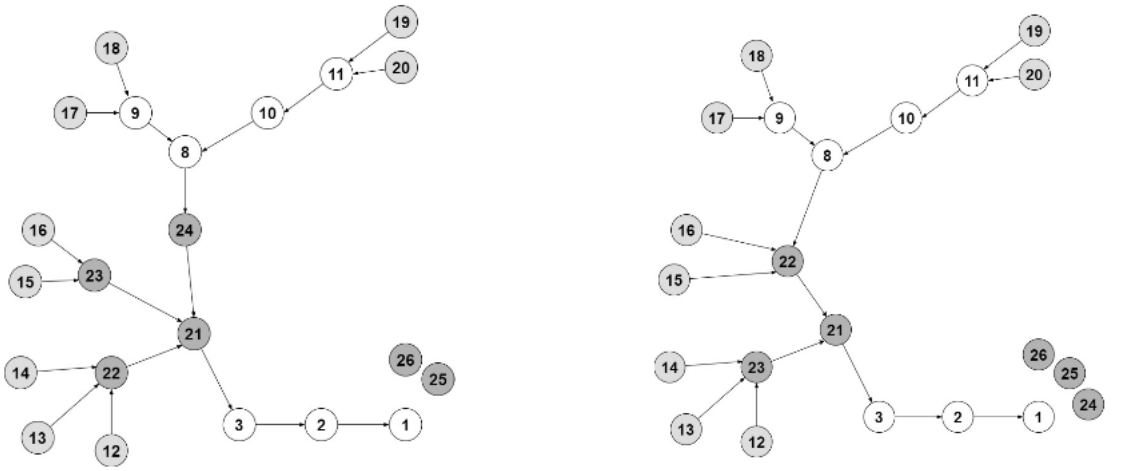
Figure 5.5: Recovery in Scenario 2

As illustrated in Fig 5.4 and 5.5, nodes 7 and 8 became faulty and a part of the network disconnected. RFMM replace all the faulty nodes with mobile nodes, and the DFMM placed 2 mobile nodes in slightly different positions.



(a) Initial Topology (b) Faults in the Network

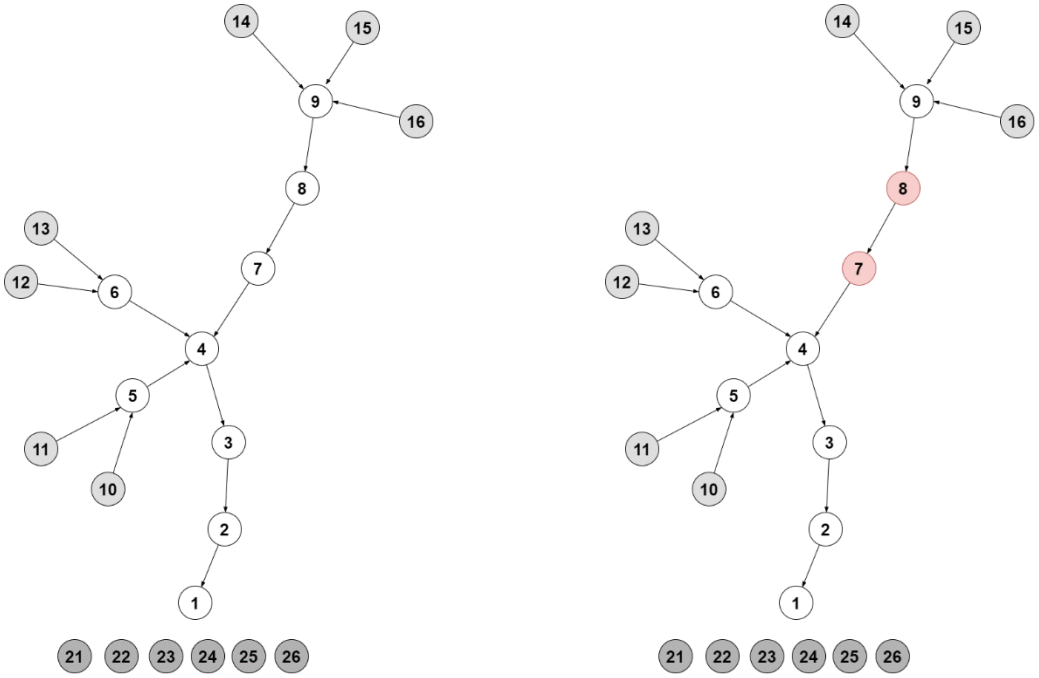
Figure 5.6: Scenario 3



(a) RFMM (b) DFMM

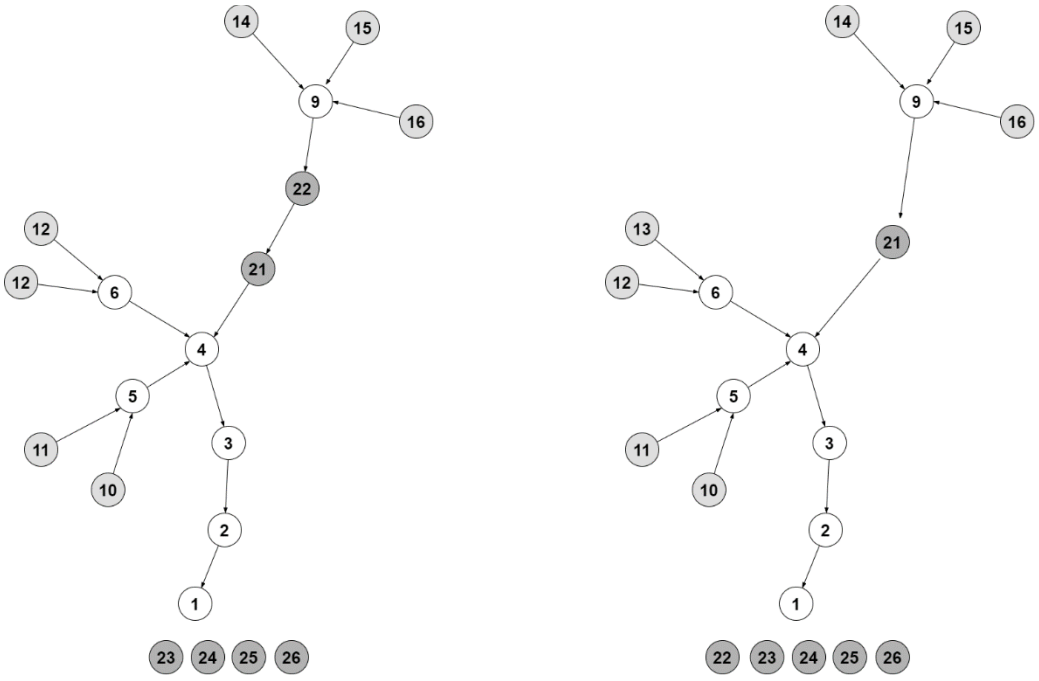
Figure 5.7: Recovery in Scenario 3

As illustrated in Fig 5.6 and 5.7, the nodes 4,5 and failed and three healthy segments disconnected from the network. RFMM replace four faulty nodes with four mobile nodes (at the same positions), however the DFMM sent 3 mobile nodes instead of 4 and reconnected successfully the disconnected segments.



(a) Initial Topology (b) Faults in the Network

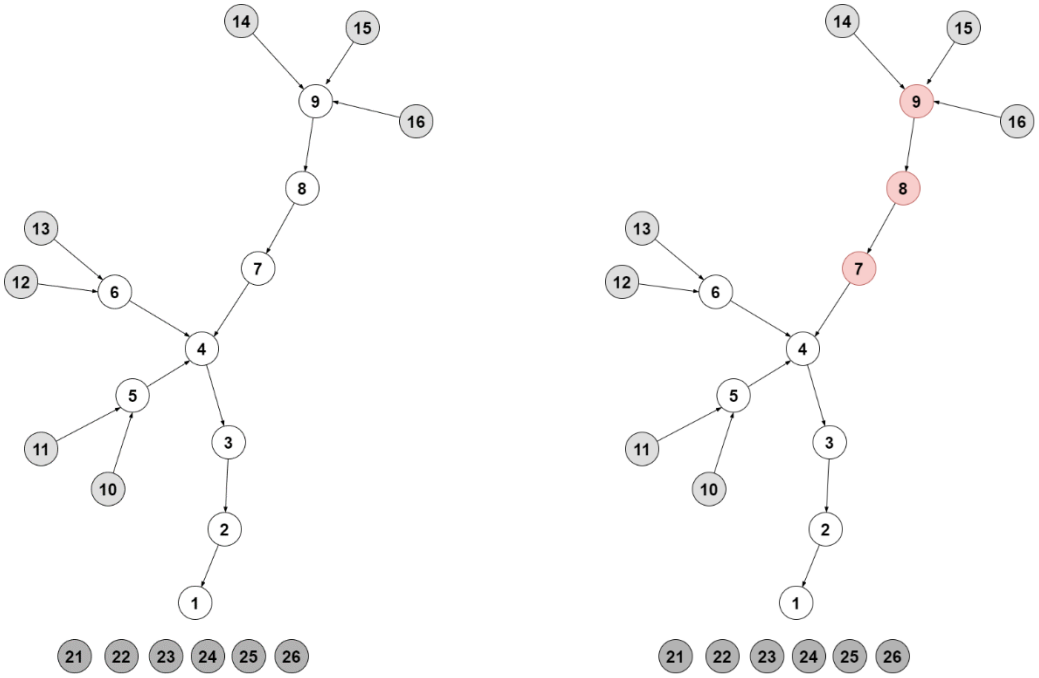
Figure 5.8: Scenario 4



(a) RFMM (b) CFMM

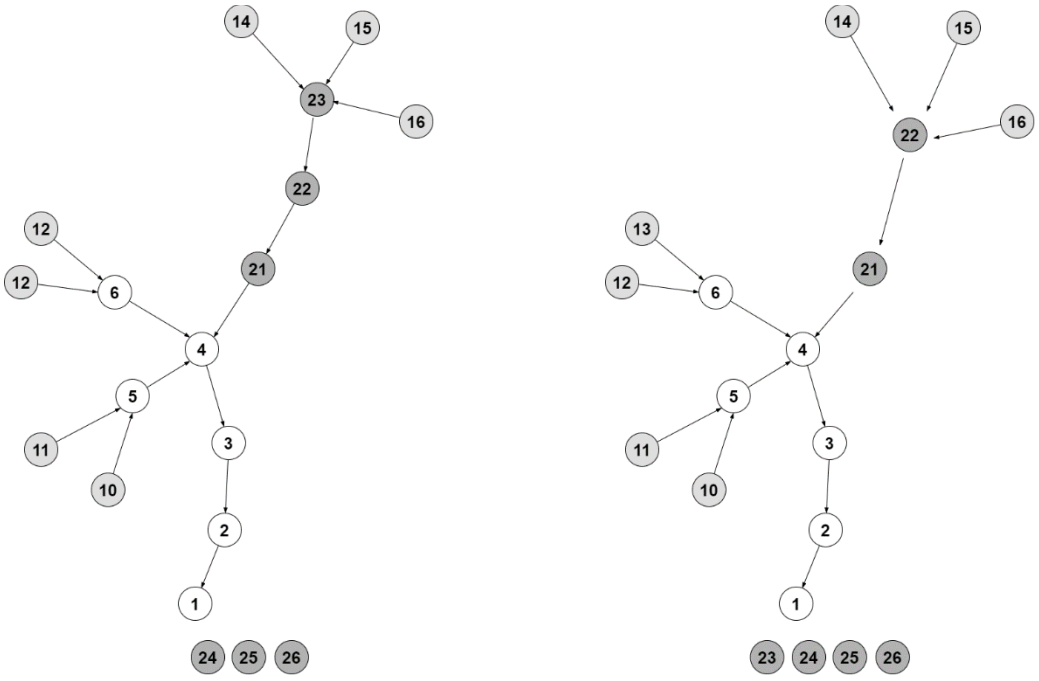
Figure 5.9: Recovery in Scenario 4

As illustrated in Fig 5.8 and 5.9, nodes 7 and 8 became faulty and an area of the network cannot communicate with the sink node anymore. RFMM replace all the faulty nodes with mobile nodes, however the CFMM positioned one mobile node instead of 2 and reconnect the disconnected area.



(a) Initial Topology (b) Faults in the Network

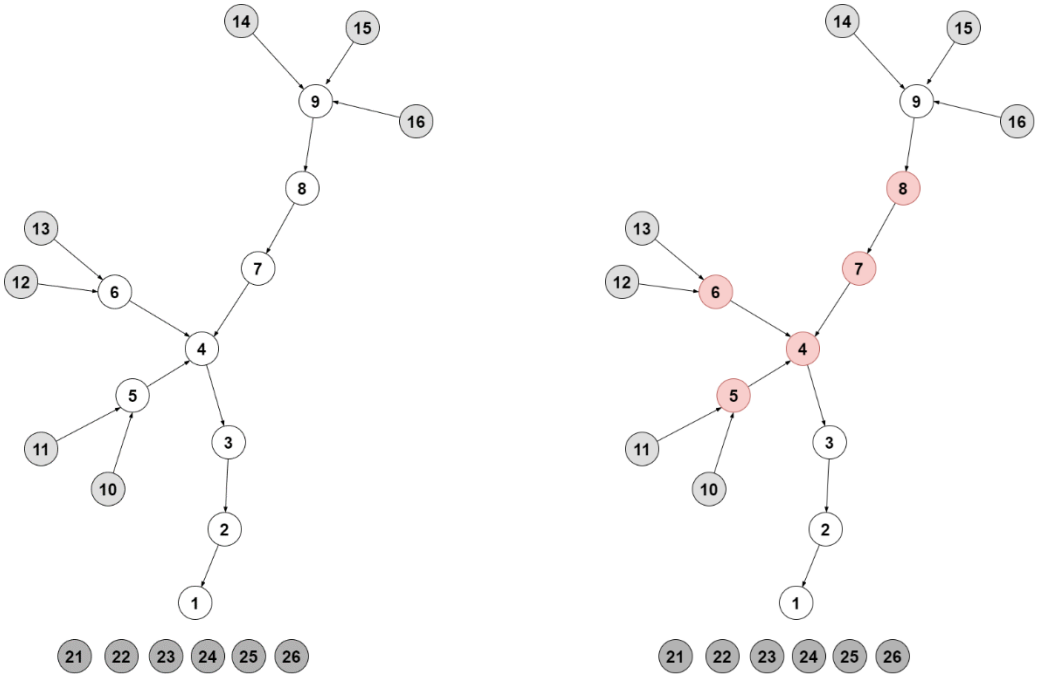
Figure 5.10: Scenario 5



(a) RFMM (b) CFMM

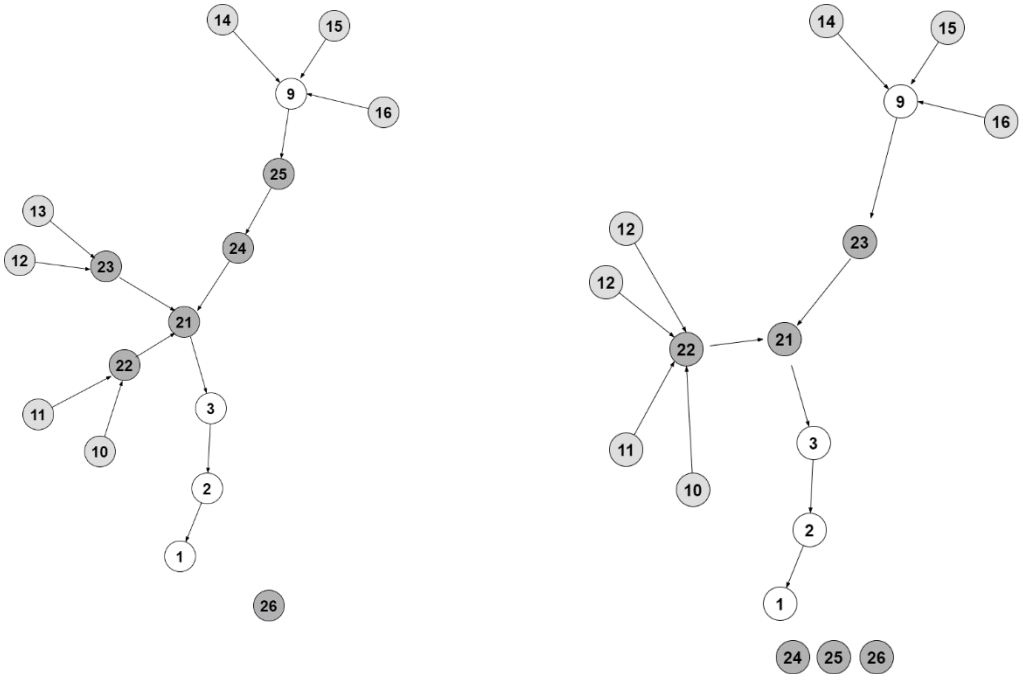
Figure 5.11: Recovery in Scenario 5

As illustrated in Fig 5.10 and 5.11, nodes 7, 8 and 9 became faulty and three source nodes (14,15,16) were unable to send packets toward the sink. RFMM replaced all the 3 faulty nodes with mobile nodes, whereas the CFMM placed only 2 mobile nodes instead of 3 and the source nodes 14,15 and 16 found a path toward the sink (by the mobile nodes).



(a) Initial Topology (b) Faults in the Network

Figure 5.12: Scenario 6



(a) RFMM (b) MobileFM

Figure 5.13: Recovery in Scenario 6

As illustrated in Fig 5.12 and 5.13, nodes 4,5,6,7 and 8 became faulty and three disjoint segments disconnected from the network. RFMM replace all the 5 faulty nodes with mobile nodes, however the DFMM utilized just 3 mobile nodes instead of 5 and reconnect the disconnected segments.

5.3 Numerical Results

For each scenario, we present the number of utilized mobile nodes, the percentage of successfully received packets, the packet loss ratio, and the network energy.

5.3.1 Mobile Nodes

Considering the limited number of mobile nodes, mechanisms that use fewer mobile nodes can be useful to solve problems in the network and extend its lifetime.

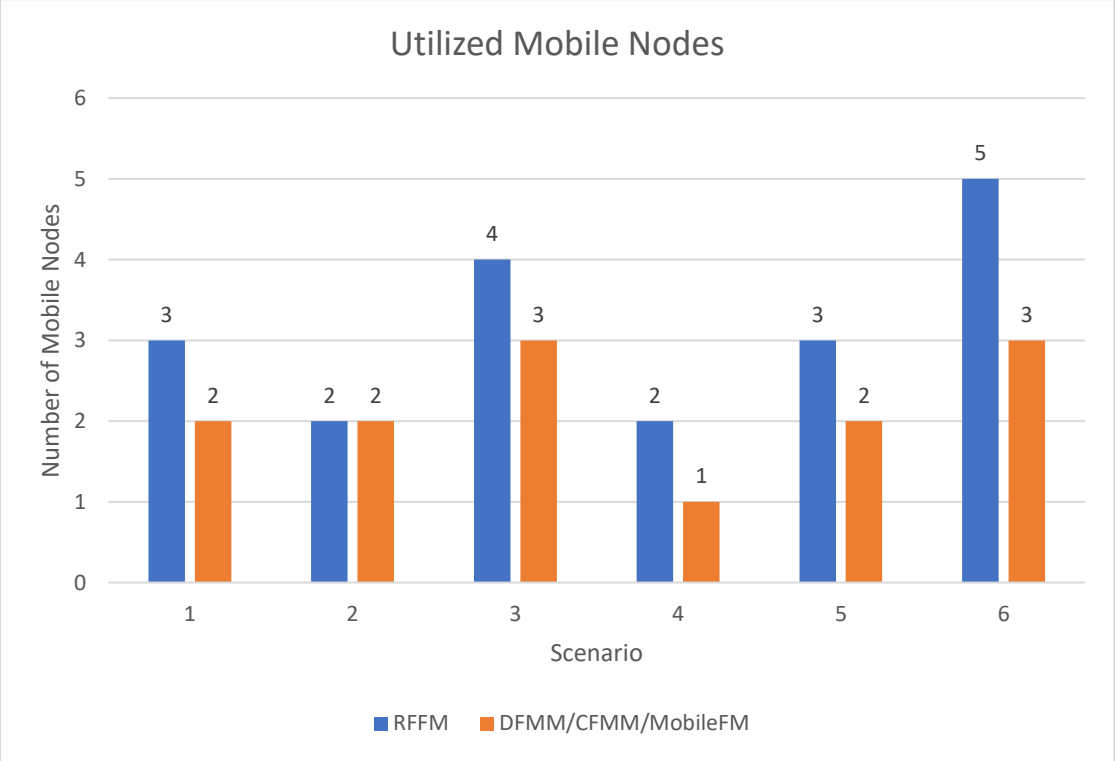


Figure 5.14 : Graph of utilized mobile nodes

The DFMM, the CFMM and the MobileFM are taking advantage of the position of the affected nodes and calculating positions for mobile nodes to serve as many as possible nodes. As a result, they require fewer mobile nodes compared to the RFMM, which just replaces the faulty nodes with mobile nodes. Because of this characteristic, these mechanisms are expected to consume less energy in the long term.

In Scenario 2, DFMM uses as many nodes as the RFMM because of the faulty node's position, thus the numerical results of scenario 2 can be interesting for the evaluation and indicate any drawback of the suggested mechanism.

5.3.2 Received Packet Ratio

The percentage of successfully received packets presents the ratio of packets received (overall packets generated by the sources during the simulation) versus the load of the network and is calculated with the equation below:

$$Received\ Packet\ Ratio = \frac{\text{Succesfully Received Packets}}{\text{Total Sent packets}}$$

In the graphs below, the DFMM, CFMM and MobileFM are compared with the RFFM (Replacement Fault Management Mechanism) as the simplest strategy. In scenarios 1,2 and 3 the simulation was performed with the DFMM, however the MobileFM would behave exactly the same with DFMM in these scenarios.

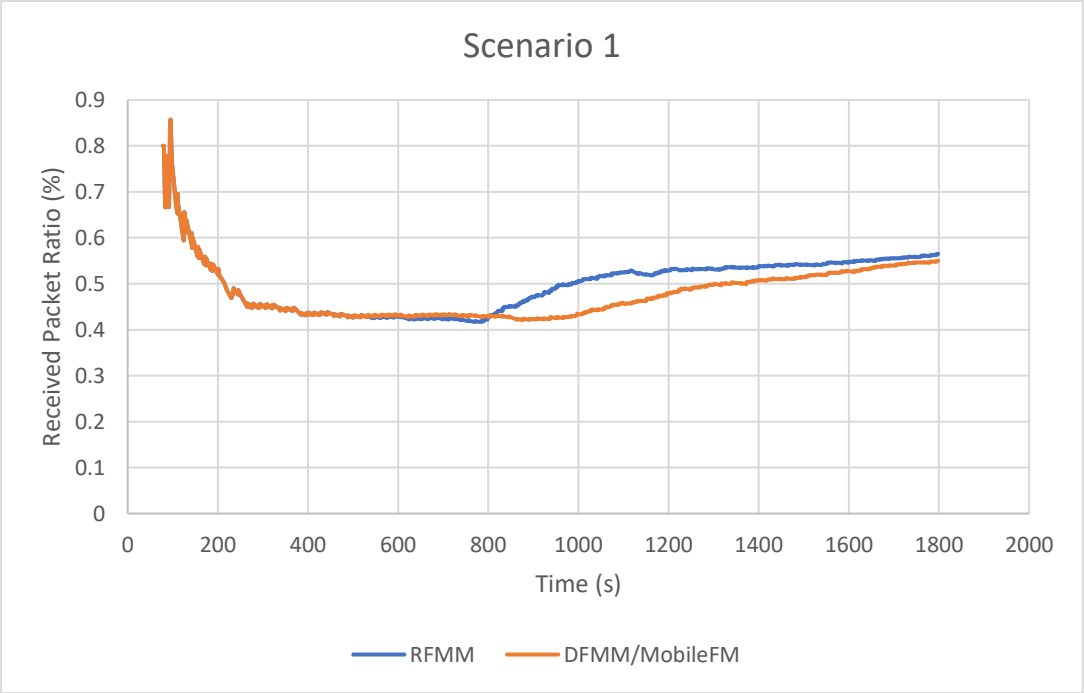


Figure 5.15

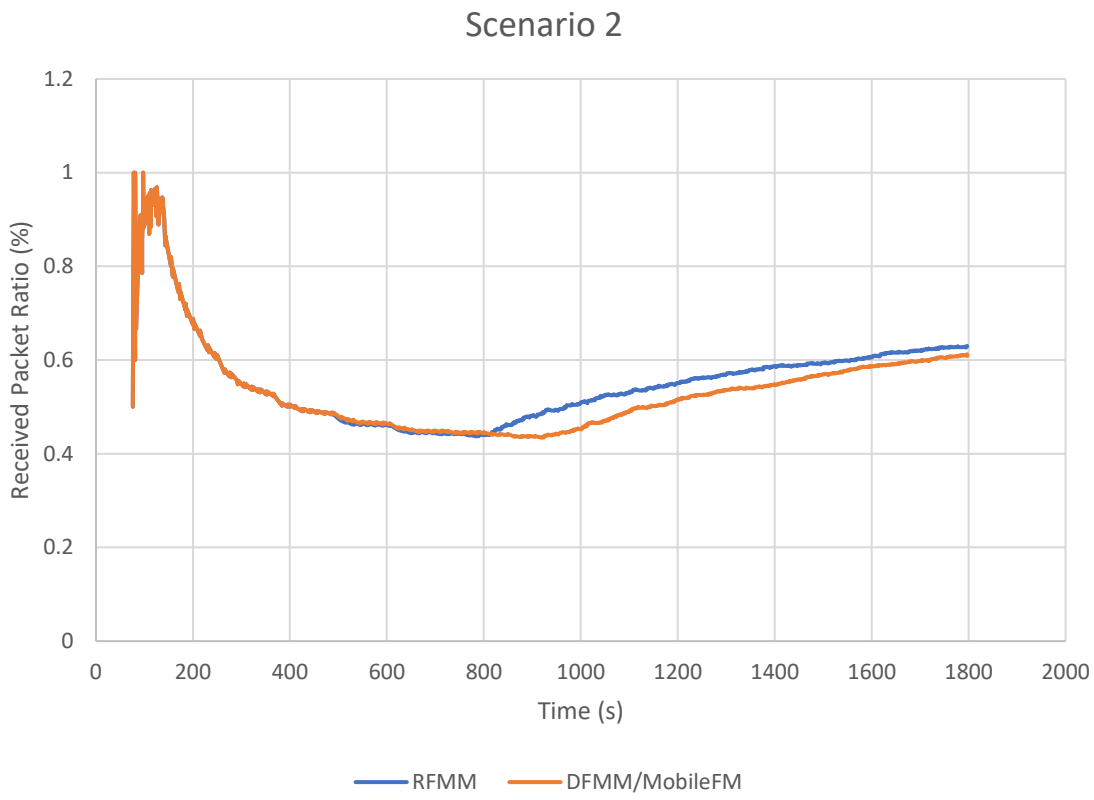


Figure 5.16

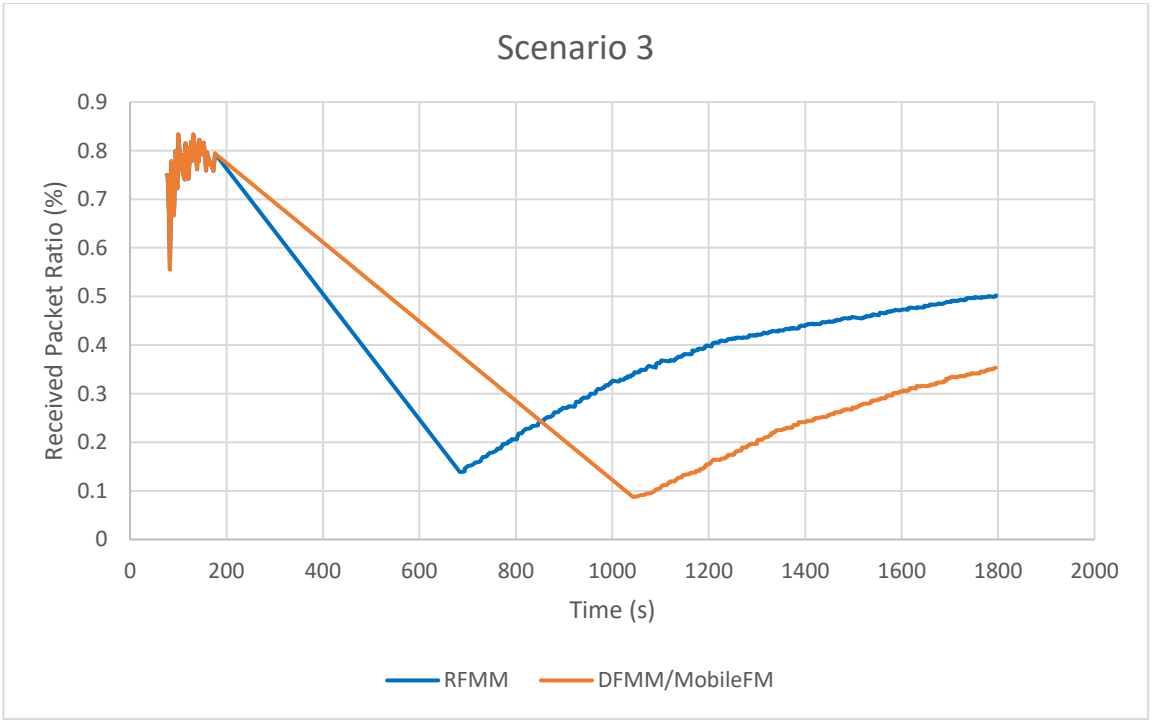


Figure 5.17

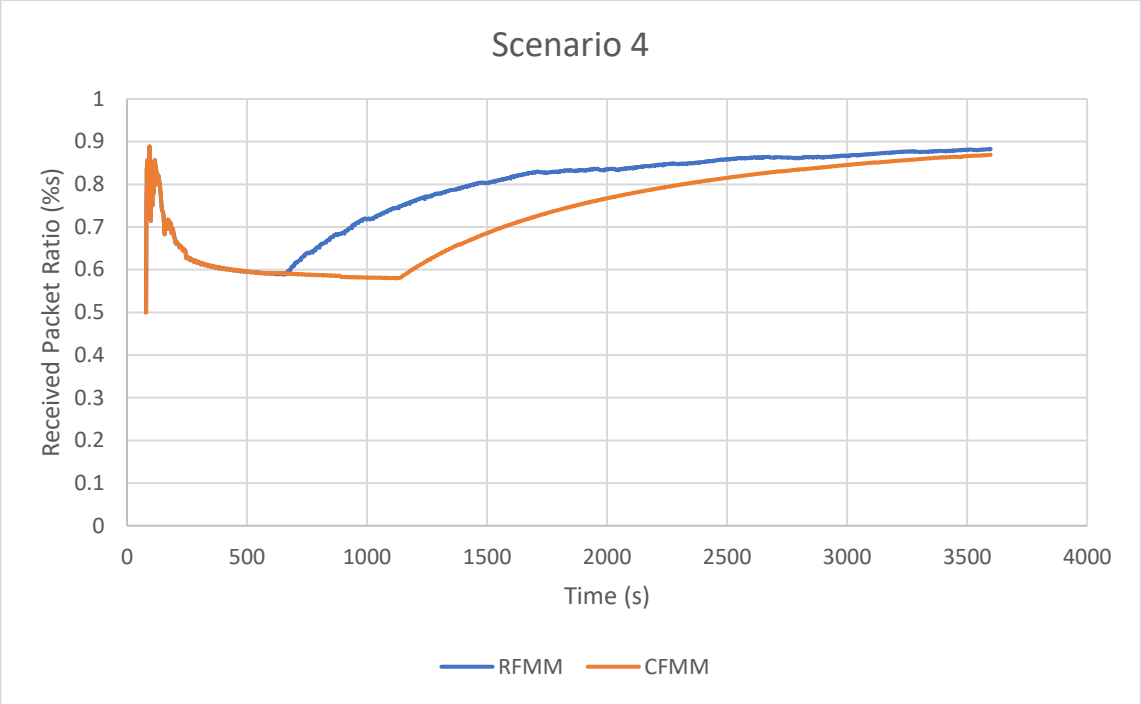


Figure 5.18

Figure 5.19

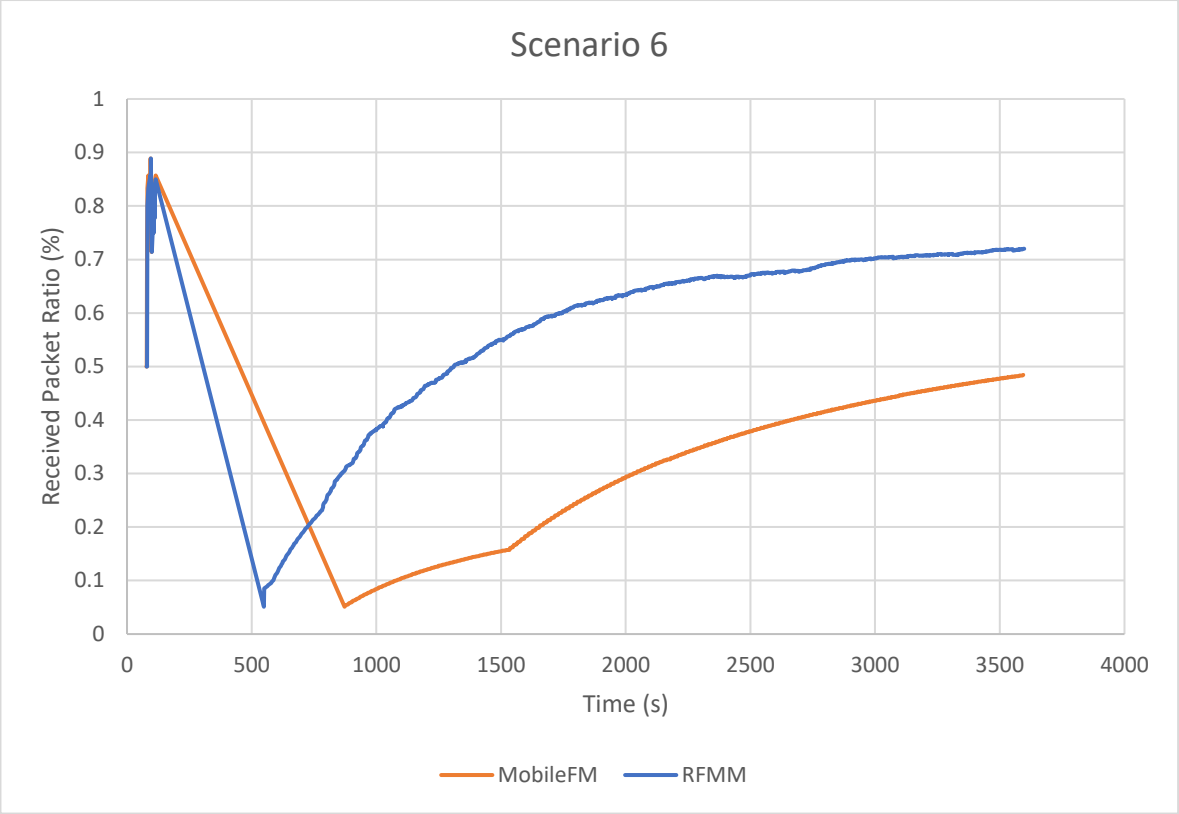


Figure 5.20

The percentage of successfully received packets is an important metric in WSNs for the Quality of Service (QoS). Firstly, the graphs show that all methods achieve to reconnect the network after a disconnection. The disconnections cause the received packet ratio to decrease remarkably, and then the mechanisms rose the received packet ratio with the deployment of the mobile nodes.

The main reason for the fall of the received packet ratio is the position of the failure, since some nodes can be the only path toward the sink and when the failure occurs, a whole area or even the whole network can be disconnected. The phenomenon of the whole network being disconnected is illustrated in Scenarios 3 and 6, where all source nodes were unable to send packets to the sink node, and there was a dramatic fall in the received packet ratio. After the utilization of the mobile nodes, the network recovers.

It can be observed that DFMM, CFMM and MobileFM need more time to reconnect and return to a stable state than the RFMM because the discovery method requires extra time. However, the received packet ratio in these mechanisms is increasing at a higher pace than the RFFM.

DFMM seems to have very similar results with the RFMM in Scenario 2 (Fig. 5.16), where the same number of mobile nodes needed to reconnect the network.

It must be address that the DFMM, CFMM and MobileFM placed the mobile node closer to the destination node in comparison to RFMM, and that can be the reason of the better recovery rate in these mechanisms.

5.3.3 Packet Loss Ratio

Packet loss ratio is another important metric in WSNs for the QoS. The packet loss ratio is the ratio between the number of lost packets to the total number of packets sent within a specific amount of time.

$$Packet\ Loss\ Ratio = \frac{\text{Packets Lost}}{\text{Total Packets}}\ x\ 100\%$$

In the graphs below, the DFMM, CFMM and MobileFM are compared with the RFFM (Replacement Fault Management Mechanism) as the simplest strategy. In scenarios 1,2 and 3 the simulation was performed with the DFMM, however the MobileFM would behave exactly the same with DFMM in these scenarios.

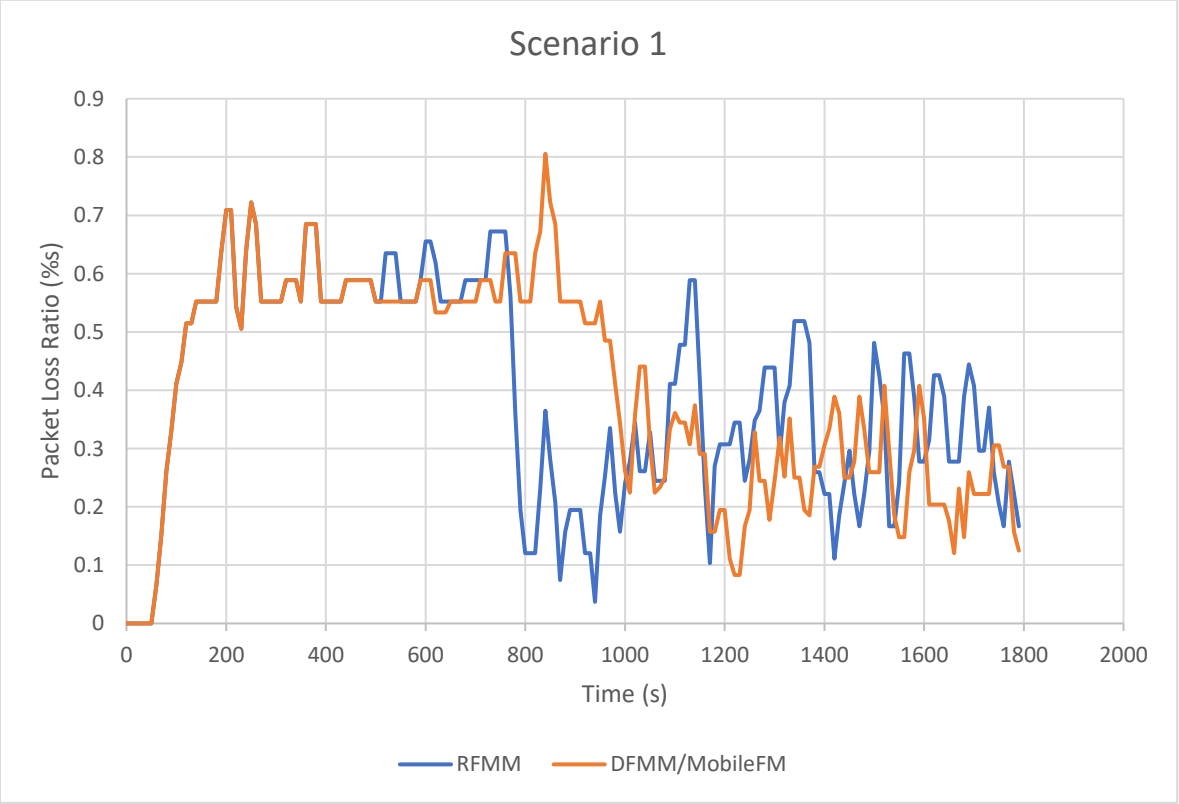


Figure 5.21

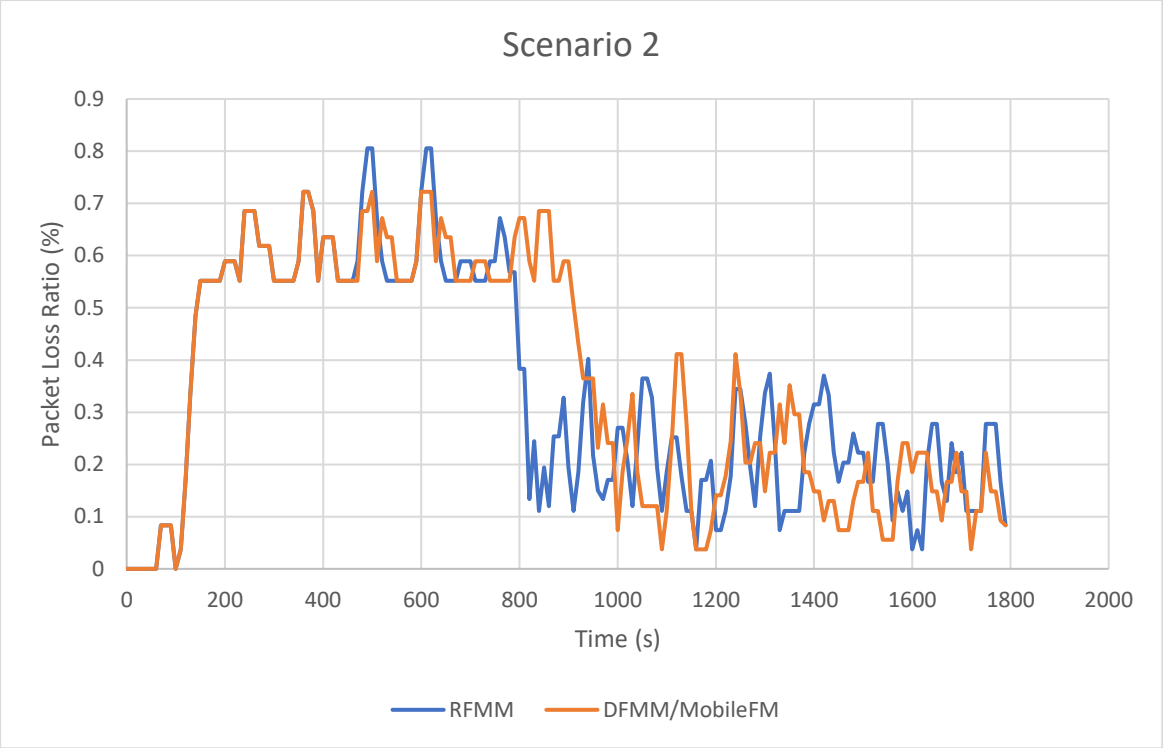


Figure 5.22

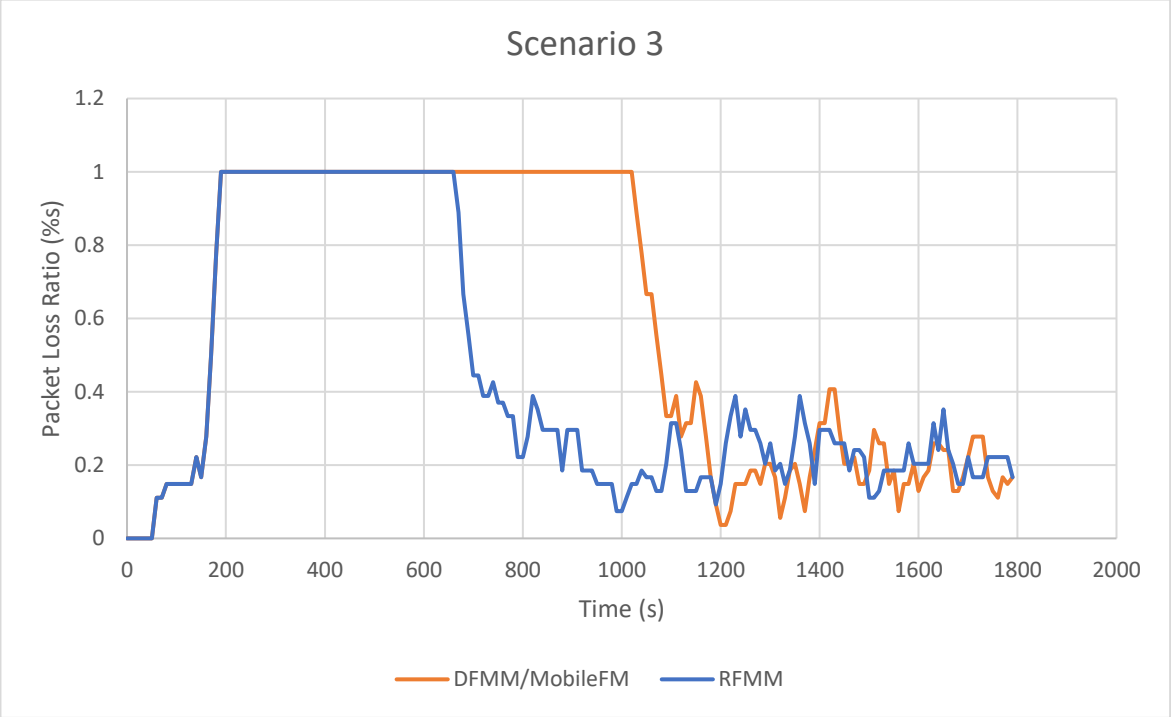


Figure 5.23

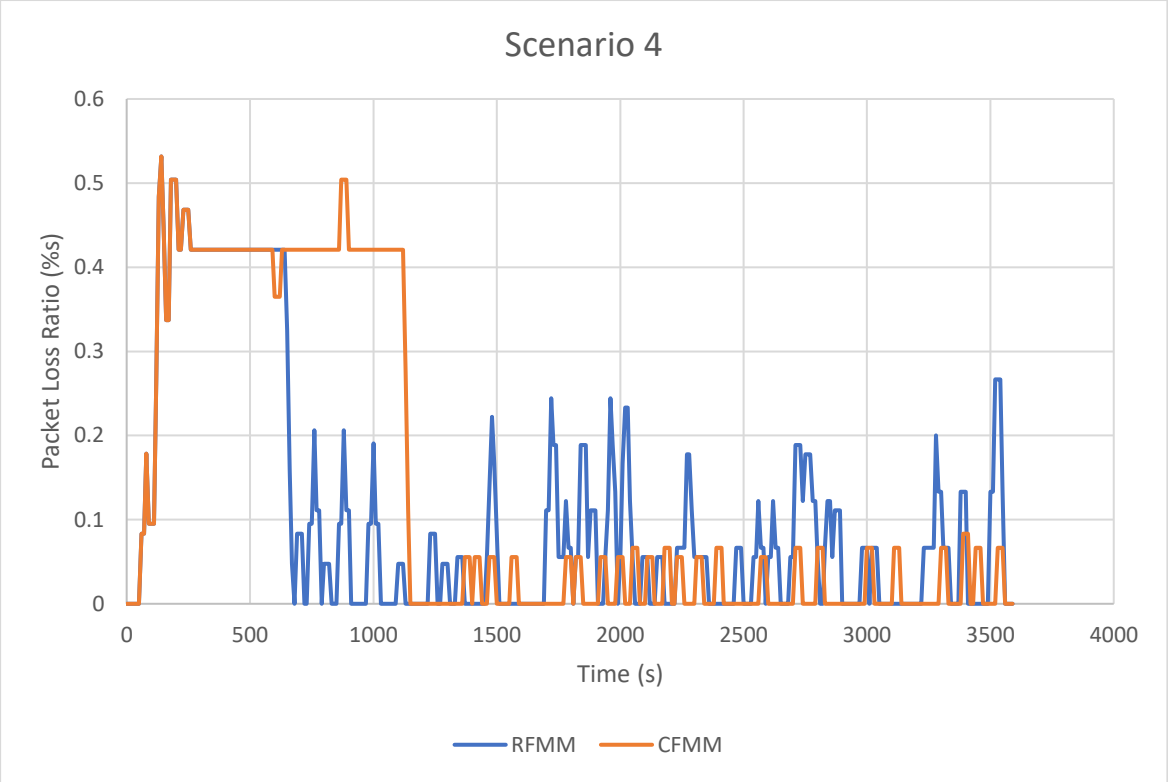

Scenario 4
Packet Loss Ratio (%s)
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
3000
3500
4000
Time (s)
RFMM
CFMM

Figure 5.24

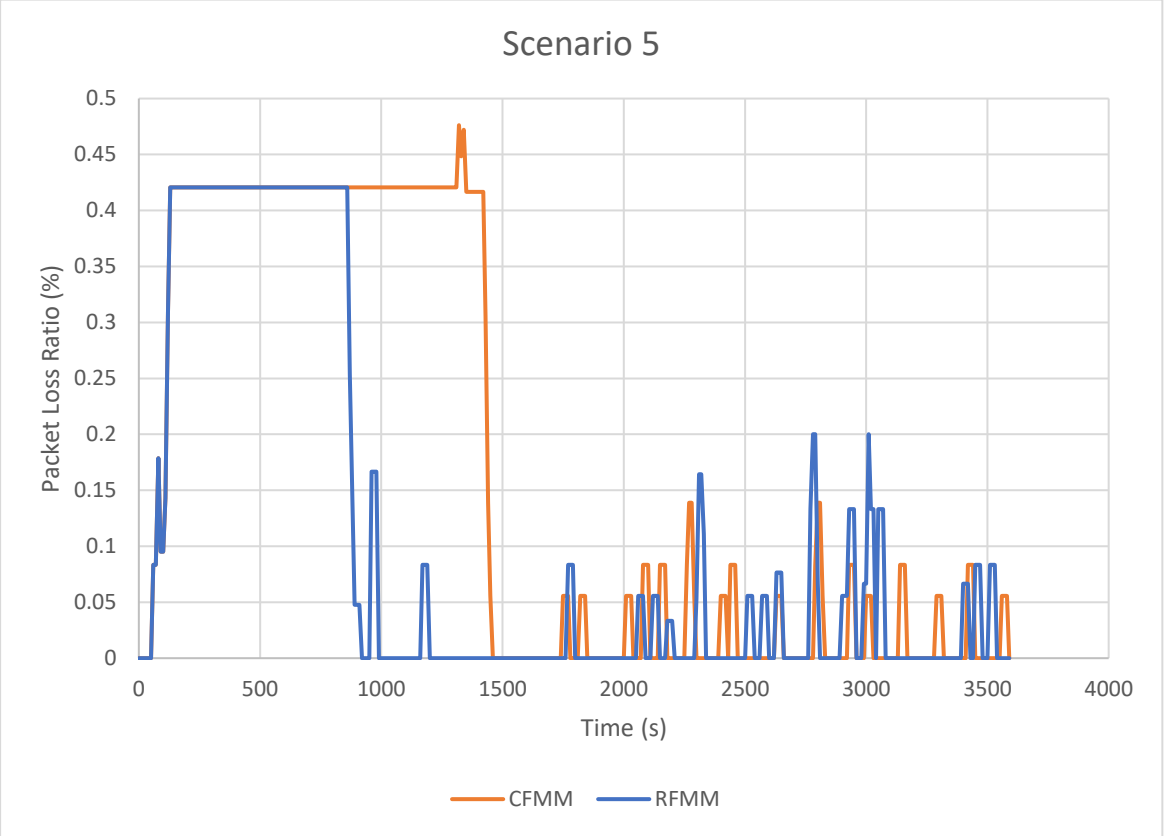

Scenario 5
Packet Loss Ratio (%)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
500
1000
1500
2000
2500
3000
3500
4000
Time (s)
CFMM
RFMM

Figure 5.25

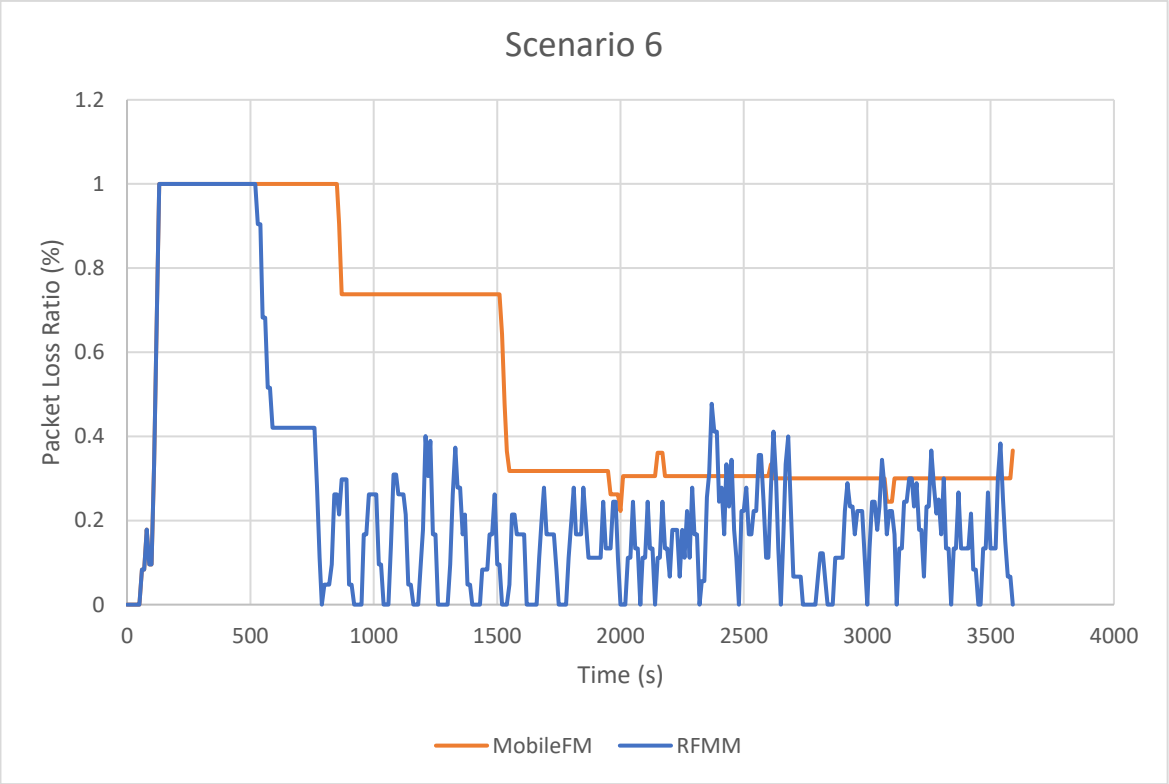


Figure 5.26

Packet losses happen mainly because of congestion or failures in these networks. Congestion is responsible for a small packet loss ratio in all the simulation time. The graphs show that a disconnection happened around 100-200 seconds and the packet losses rose sharply. In every scenario, a whole area of the network was disconnected. Despite rising, the packet loss ratio in all scenarios then falls after the placement of the mobile nodes. In Scenarios 3 and 6, it is noticeable that the packet loss ratio reaches 100% because the whole network was disconnected. Lastly, it is shown that the ratio after the recovery phase is slightly better in DFMM and CFMM in comparison with the RFMM. The mobile nodes are placed in different position by the DFMM, CFMM and MobileFM than in RFMM. These positions can be the reason of the differences in rate of the average throughput and in the packet loss ratio.

5.3.4 Energy

The network energy is the total energy consumed in the network, measured in mJ, during the operation of the network. To measure this metric, we calculated the energy consumed by each node (energy_i) with the equation below:

$$Energy_i = Operation\ Energy_i + Move\ Enery_i ,$$

where Operation-Energy is the computational energy usage and Move-Energy is the energy usage of moving. The Move-Energy of static nodes is zero.

The Operation-Energy of each node is calculated with the equation:

$$Operation\ Energy = (transmit * 19.5mA + listen * 21.8mA + CPU * 1.8\,mA + LPM * 0.0545\,mA)\ x\ 3V/4096 * 8 ,$$

where transmit 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 [4].

The Move-Energy of each mobile node is calculated with the equation below:

$$Move\ Energy = P_u\ x\ \frac{s}{u},$$

where P_u is the power consumption of a given speed u and s is the total travelling distance [10].

The total energy consumed by the network is calculated by the equation below:

$$Network\ Energy = \sum_{i=1}^{n} \text{Energy}_i ,$$

where *n* is the number of nodes.

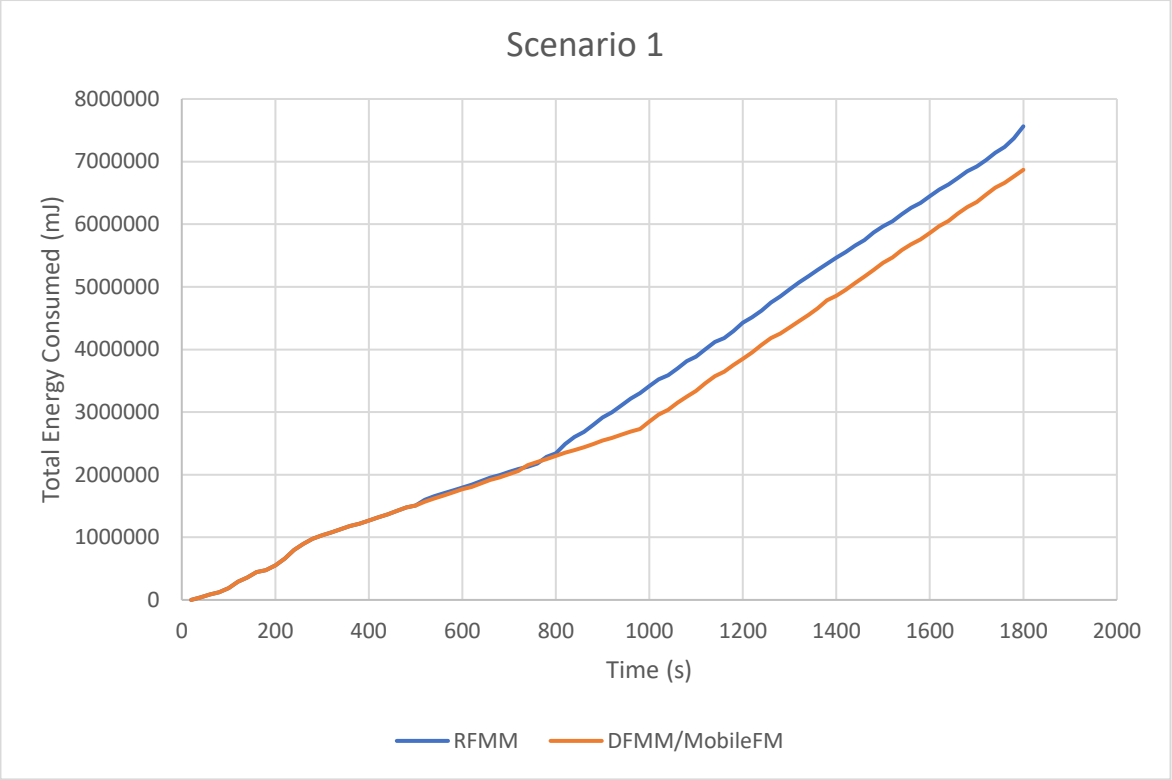


Figure 5.27

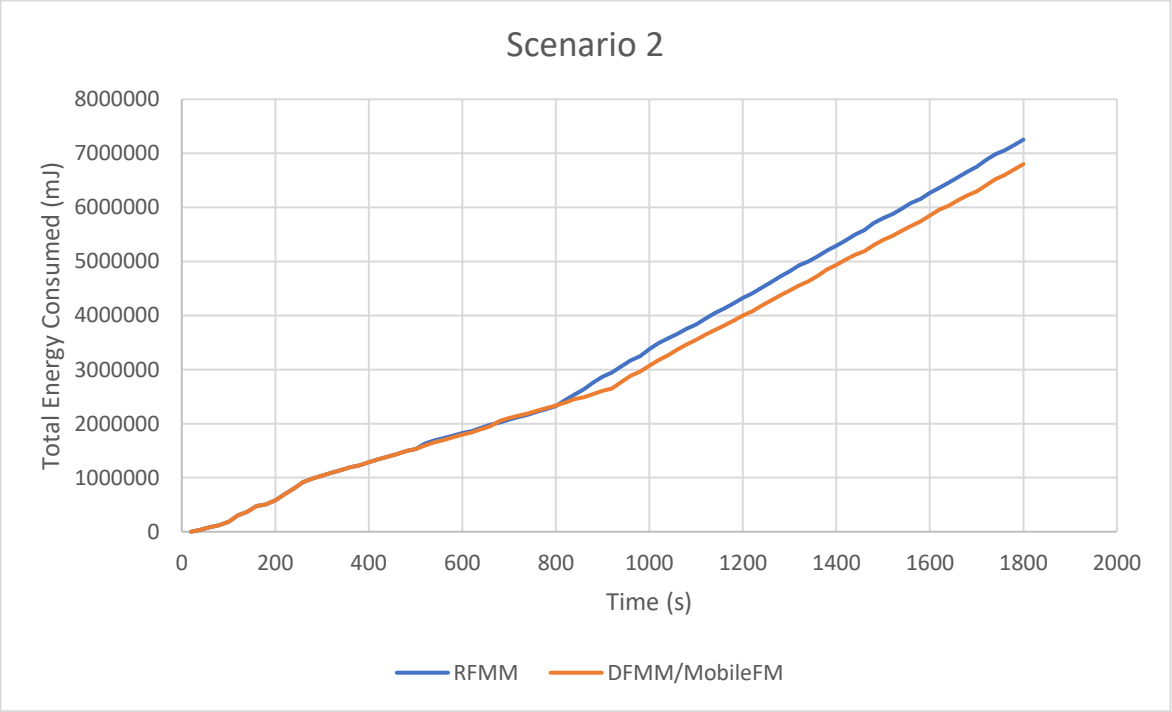


Figure 5.28

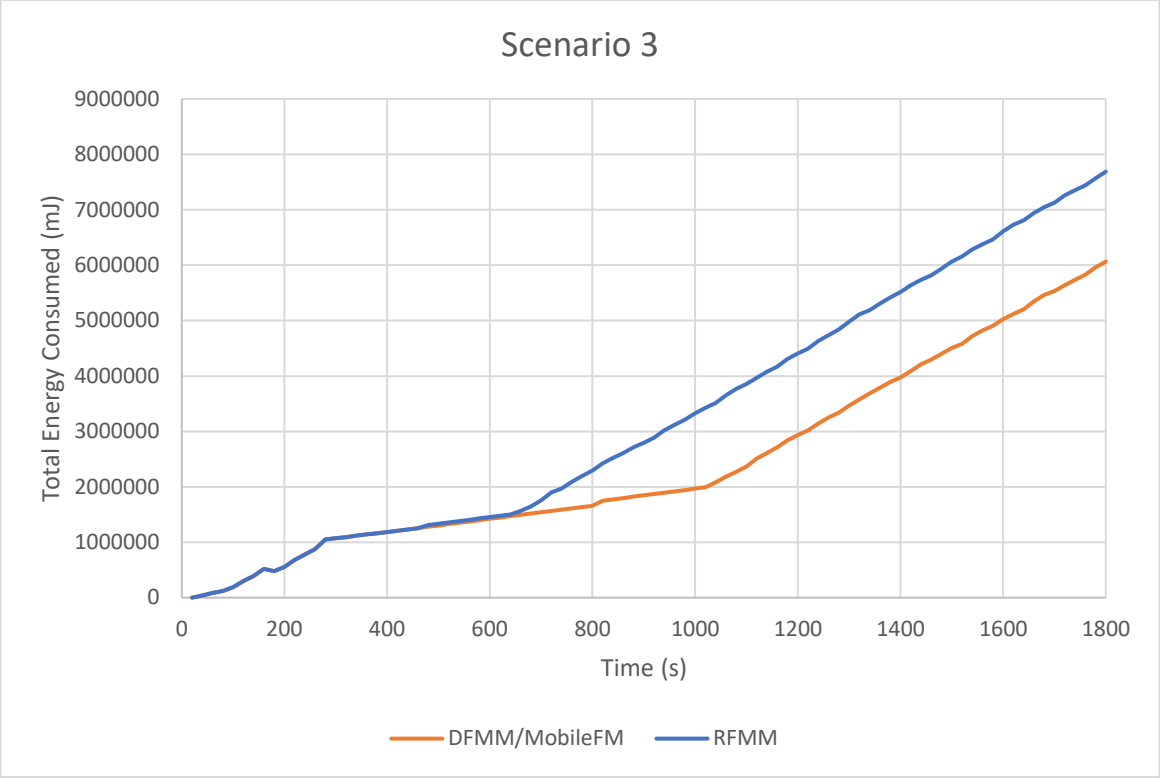


Figure 5.29

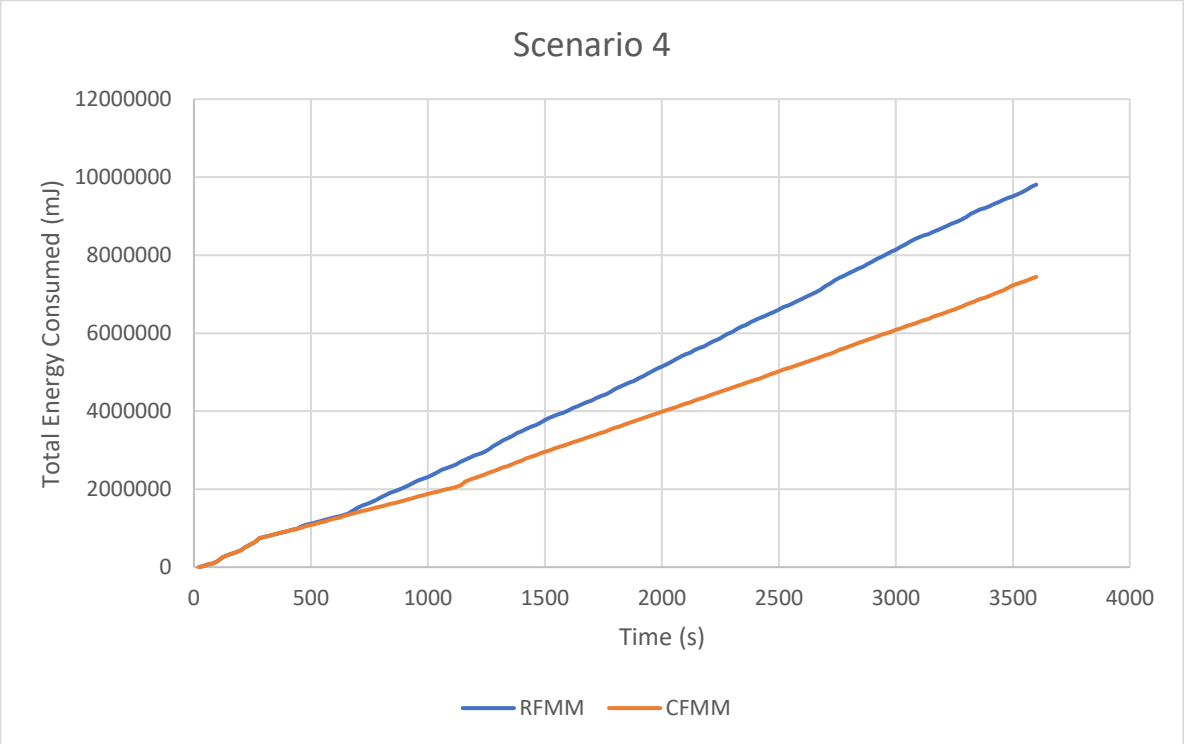


Figure 5.30

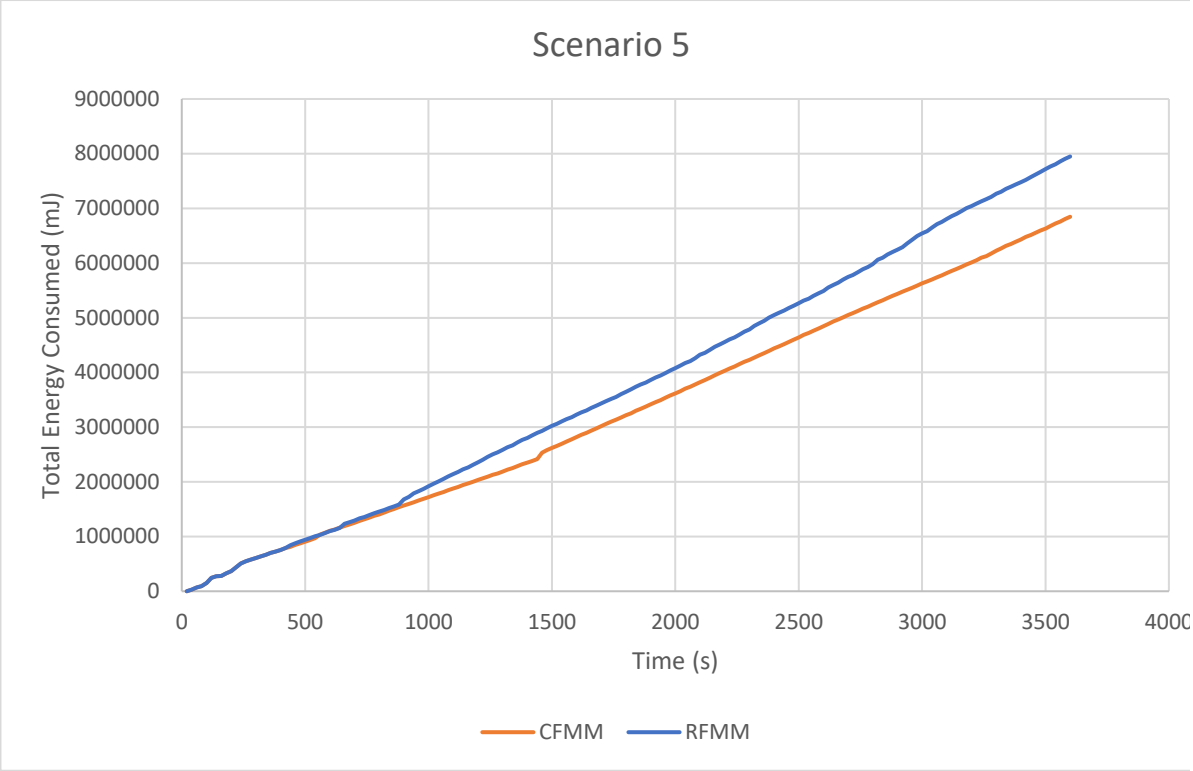


Figure 5.31

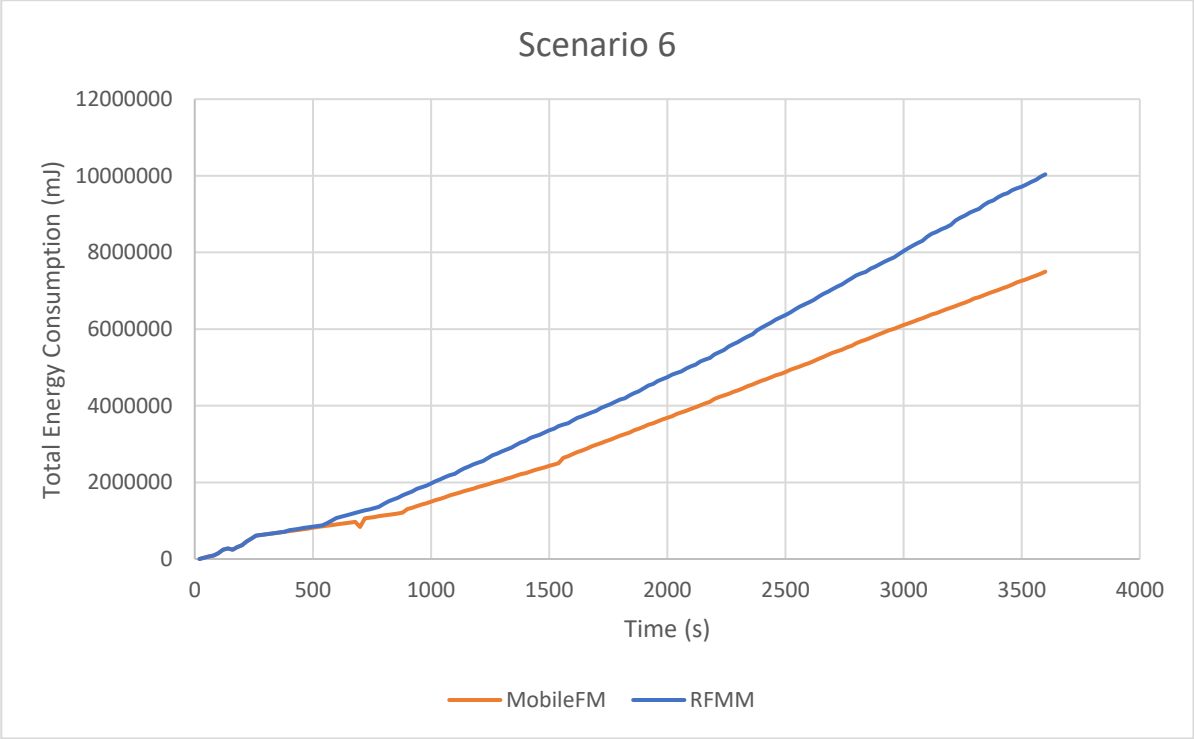


Figure 5.32

Energy consumption is critical for the lifetime of the network. The plots show the total energy consumed in the network. The RFMM reconnects the network faster than the other mechanisms, thus the disconnected area starts to operate and consume energy earlier than the other mechanisms. It must be pointed out that the total energy consumed by the RFMM in every scenario is rising at a rapid rate, and this rate is higher than the other mechanisms. Following this, it is expected to increase faster and has a significant difference from the other mechanisms after hours or days.

Therefore, DFMM and CFMM can deal with disconnections and consumed less energy than the RFMM. Having mechanisms that solve the disconnection problem with fewer resources is very crucial in these networks. This shows how much it is beneficial to use the MobileFM, which combines the DFMM and the CFMM to solve disconnections, compared to a simple replacement strategy. In addition, MobileFM uses the replacement strategy only for the source nodes, assuming that their positions are important for the coverage.

5.4 Random Faults

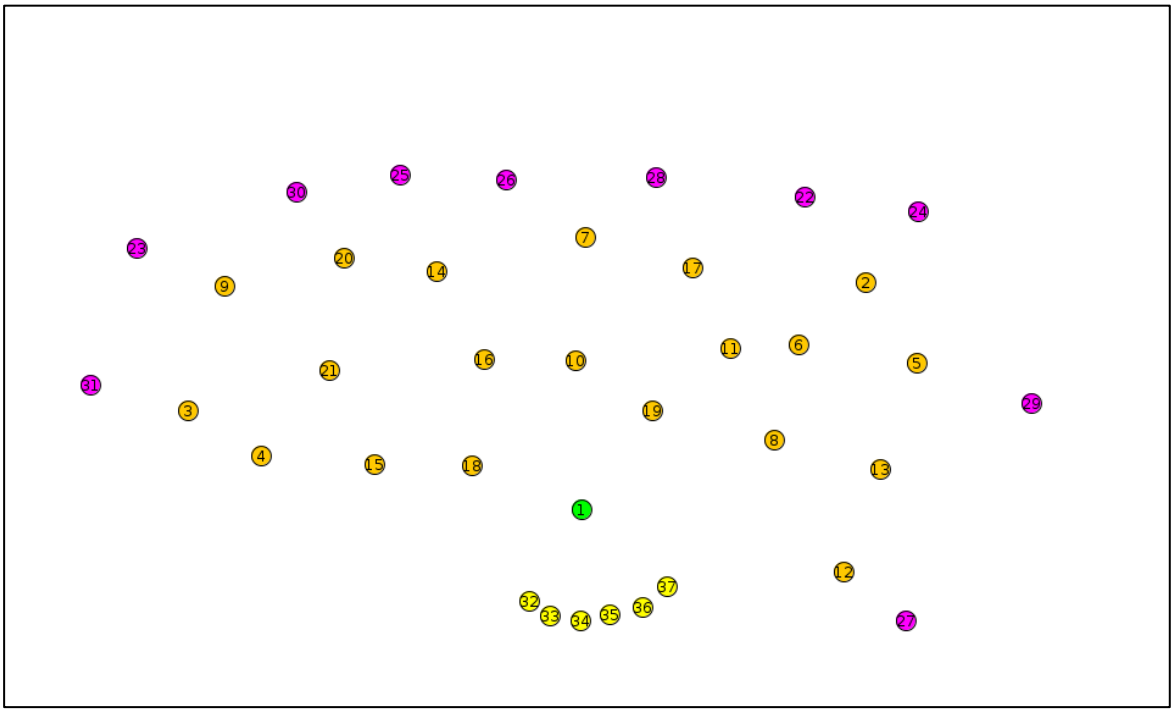
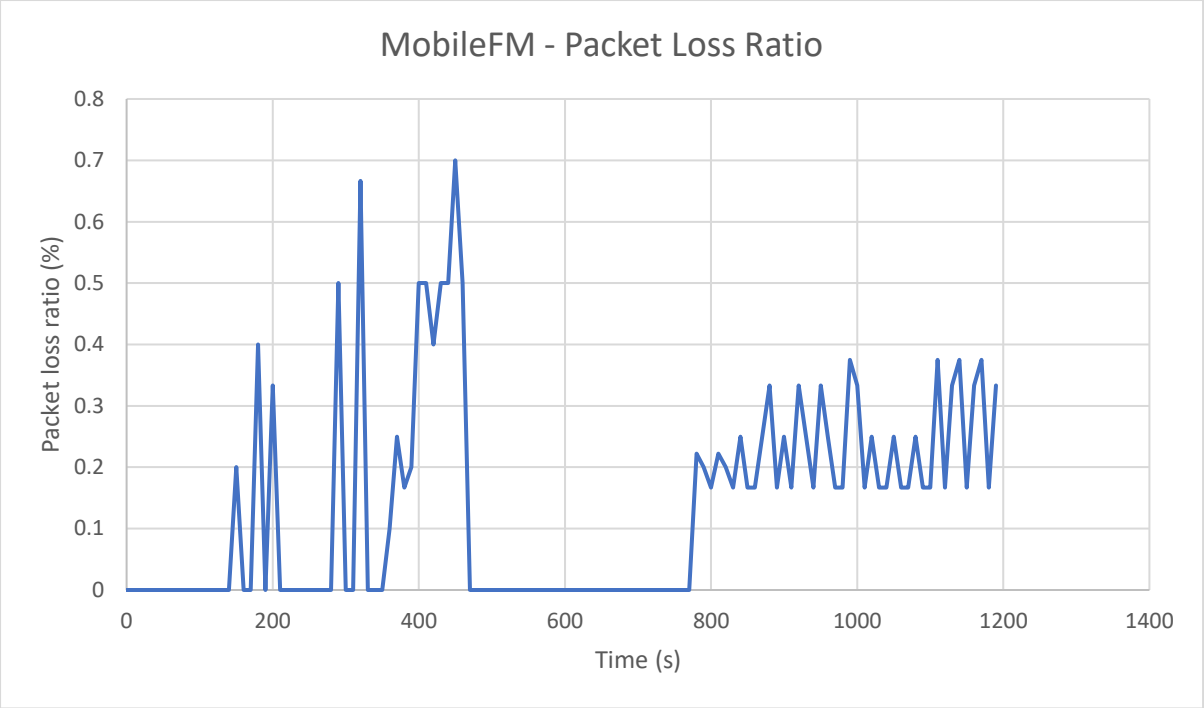
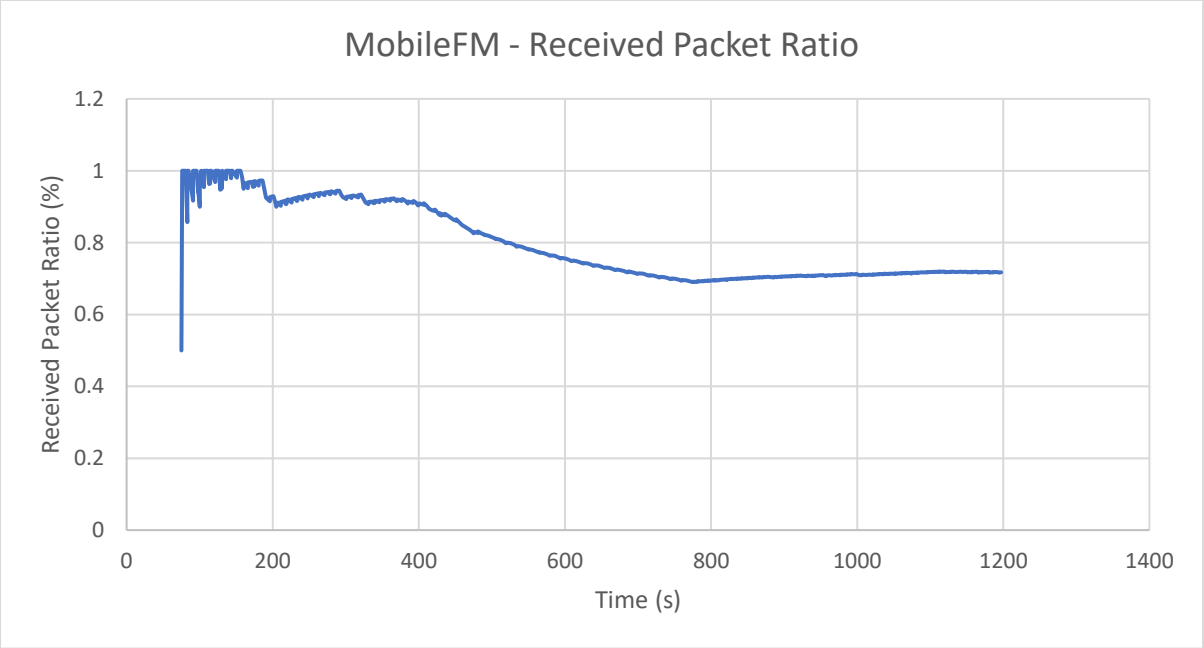


Figure 5.33 : Scenario Random Faults





When a failure occurred, the packet loss ratio increased, and the received packet ratio declined. It can be obsessed that after every failure, the network recovered successfully because the packet loss ratio decreased importantly, and the received packet ratio increased.

The MobileFM expected to maintain the network functionality as long as extra resources are available. If there is no mobile node, the network cannot reconnect the disconnected area and the packet loss ratio will not drop.

Chapter 6

Conclusion

6.1 Summary

This thesis focuses on designing strategies and developing algorithms for utilizing mobile nodes for fault detection and fault recovery in WSNs and IoT networks. We present novel algorithms for partitioning the nodes in the network in such a way that they can be investigated or connected by one node and find positions to place the mobile nodes to minimize the number of utilized mobile nodes. Based on these algorithms, we present mechanisms that deal with failures and disconnections. we combine all the mechanisms to design a framework. The proposed mechanisms are implemented in a simulation and successfully tackled failures and disconnections in these networks. The evaluation results demonstrate that DFMM, CFMM and MobileFM can consume less energy than a simple replacement strategy, which can be crucial for the network lifetime and the management of the extra resources of the network.

6.2 Challenges

One of the most challenging parts of this thesis was the implementation of the simulation. The documentation for the Contiki OS is available online and is very helpful. However, the simulation demands code in C programming language for all the types of nodes (sink, mobile, relay, source) and each file is 1000-2500 lines of code. Once we got familiar with the environment, we were able to test and develop the algorithms and then came up with better ideas about the algorithms. Moreover, many challenges arose in the simulator, and we needed to think creatively. Few of them are getting the position of the nodes dynamically (using the

script), the mobility of the nodes, the failure of nodes, the use of the processes for the evaluation in Contiki OS and developing a java program to get the results (measure packet loss ratio, sum energies, etc.).

6.3 Future Work

For future work, the current mechanisms can be evaluated by using random faults and different topologies and also compared with different fault management mechanisms.

Additionally, the clustering and node placement algorithms can be used to tackle other problems in WSNs, such as congestion or attacks. In particular, a mobile node can navigate to the problematic area, get information and act accordingly.

Finally, another future work could be the employment of the RPL protocol, that reconstructs the topology when a failure happens, in the MobileFM.

Bibliography

[1] Anuradha, M., Swetha, A., & Doraipandian, M. (2020, April). Fault Node Detection and Connectivity Restoration with Mobile Relay Node in Wireless Sensor Networks. *Journal of Computer Science*, 551–558. https://doi.org/10.3844/jcssp.2020.551.558

[2] Chouikhi, S., el Korbi, I., Ghamri-Doudane, Y., & Azouz Saidane, L. (2015, September). A survey on fault tolerance in small and large scale wireless sensor networks. *Computer Communications*, 22–37. https://doi.org/10.1016/j.comcom.2015.05.007

[3] *Contiki*. (2018). [The open source os for the internet of things]. http://www.contiki-os.org/

[4] Hou, L., Zhang, L., & Kim, J. (2018, December). Energy Modeling and Power Measurement for Mobile Robots. *Energies*, 27. https://doi.org/10.3390/en12010027

[5] Joshi, Y. K., & Younis, M. (2016, May). Restoring connectivity in a resource constrained WSN. *Journal of Network and Computer Applications*, 151–165. https://doi.org/10.1016/j.jnca.2016.03.009

[6] Lalouani, W., Younis, M., & Badache, N. (2017, December). Optimized repair of a partitioned network topology. *Computer Networks,* 63–77. https://doi.org/10.1016/j.comnet.2017.02.003

[7] Mei, Y., Xian, C., Das, S., Hu, Y. C., & Lu, Y. H. (2007, September). Sensor replacement using mobile robots. *Computer Communications*, 2615–2626. https://doi.org/10.1016/j.comcom.2007.05.047

[8] Moridi, E., Haghparast, M., Hosseinzadeh, M., & Jassbi, S. J. (2020, April). Fault management frameworks in wireless sensor networks: A survey. *Computer Communications*, 205–226. https://doi.org/10.1016/j.comcom.2020.03.011

[9] Nicolaou, A., Temene, N., Sergiou, C., Georgiou, C., & Vassiliou, V. (2019). *Utilizing Mobile Nodes for Congestion Control in Wireless Sensor Networks*. IEEE PIMRC.

[10] Raza, S., Wallgren, L., & Voigt, T. (2013, November). SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks*, 2661–2674. https://doi.org/10.1016/j.adhoc.2013.04.014

[11] Sammut, C., & Webb, G., I. (2011). *Encyclopedia of Machine Learning* (2010th ed.). Springer.

[12] Tan Et Al, P. (2022). *Introduction to Data Mining: Global Edition*. PEARSON.

[13] Temene, N., Sergiou, C., Georgiou, C., & Vassiliou, V. (2022, February). A Survey on Mobility in Wireless Sensor Networks. *Ad Hoc Networks*, 102726. https://doi.org/10.1016/j.adhoc.2021.102726

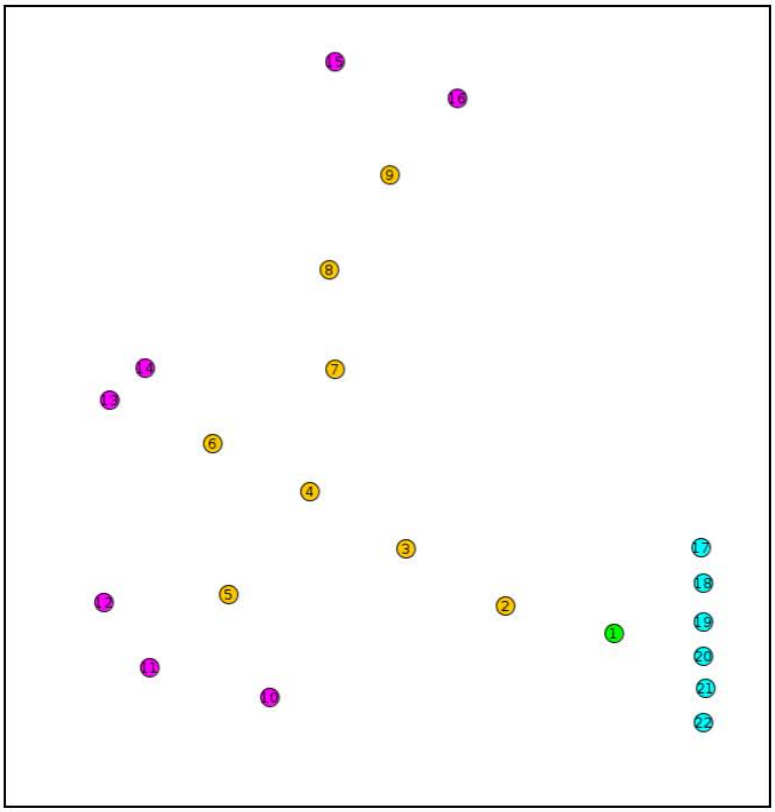
[14] Temene, N., Sergiou, C., Ioannou, C., Georgiou, C., & Vassiliou, V. (2022). A Node Placement Algorithm Utilizing Mobile Nodes in WSN and IoT Networks. *Telecom*, *3*(1), 17–51. https://doi.org/10.3390/telecom3010002

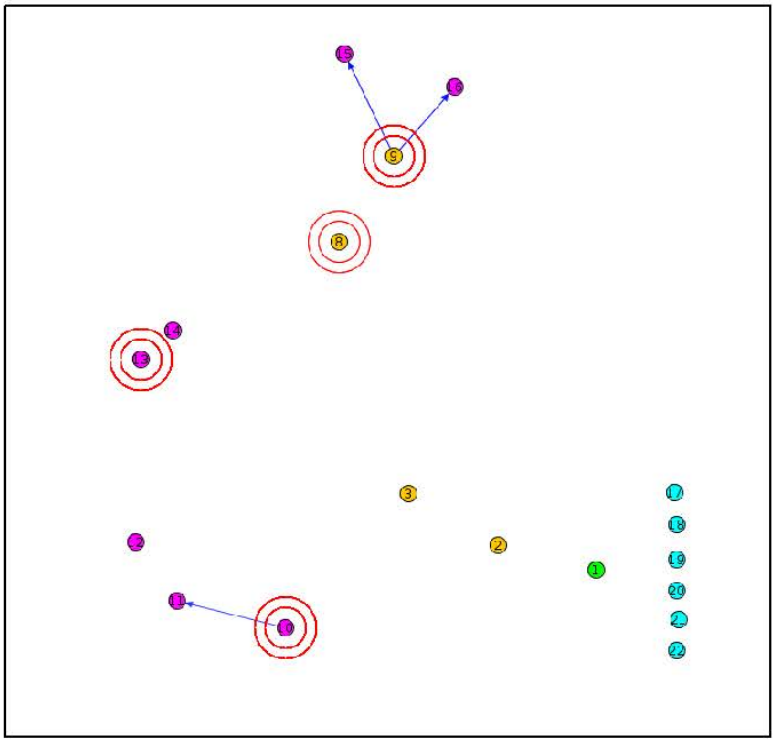
[15] Rao, V.S., Dakshayini, M., 2020. An sdn-based strategy for reliable data transmission in mobile wireless sensor networks, in: EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing, Springer. pp. 87{96. doi:10.1007/978-3-030-19562-5 9.

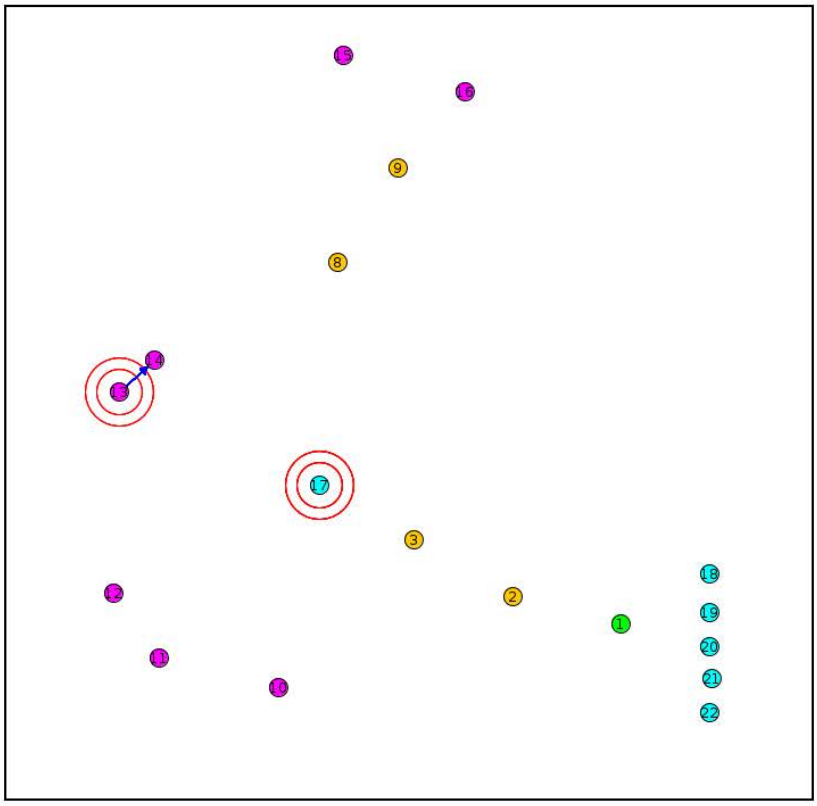
[16] Zhao, F., & Guibas, L. (2004). *Wireless Sensor Networks: An Information Processing Approach (The Morgan Kaufmann Series in Networking)* (1st ed.). Morgan Kaufmann.

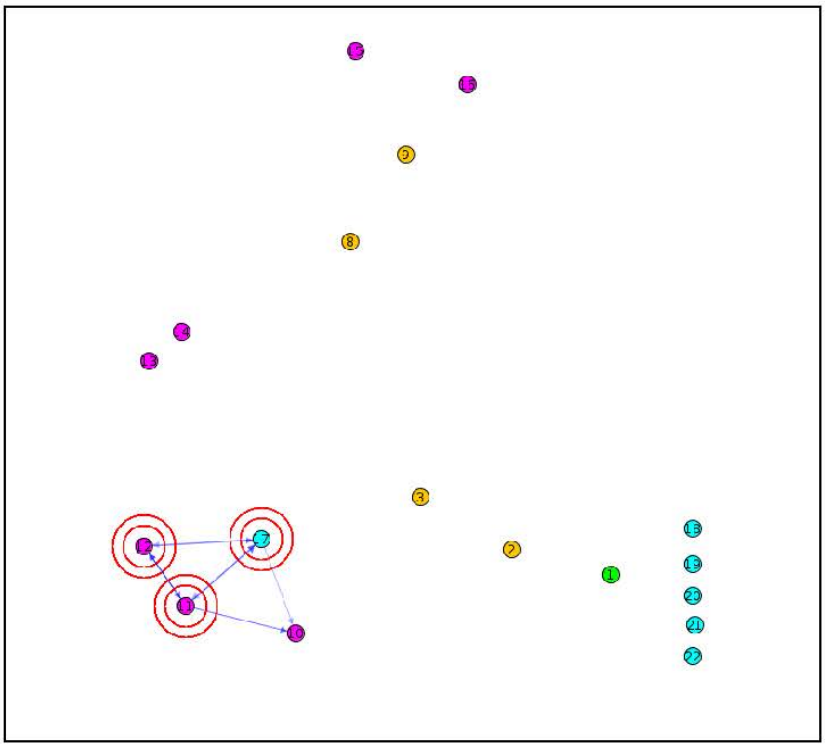
Appendix A

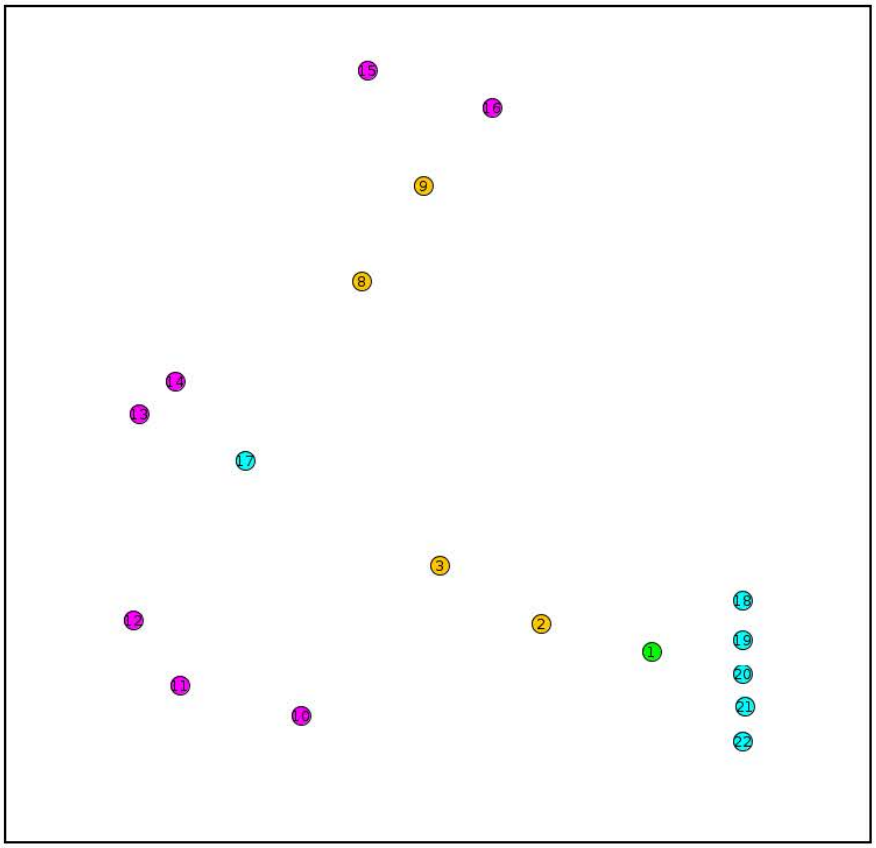
Scenario A

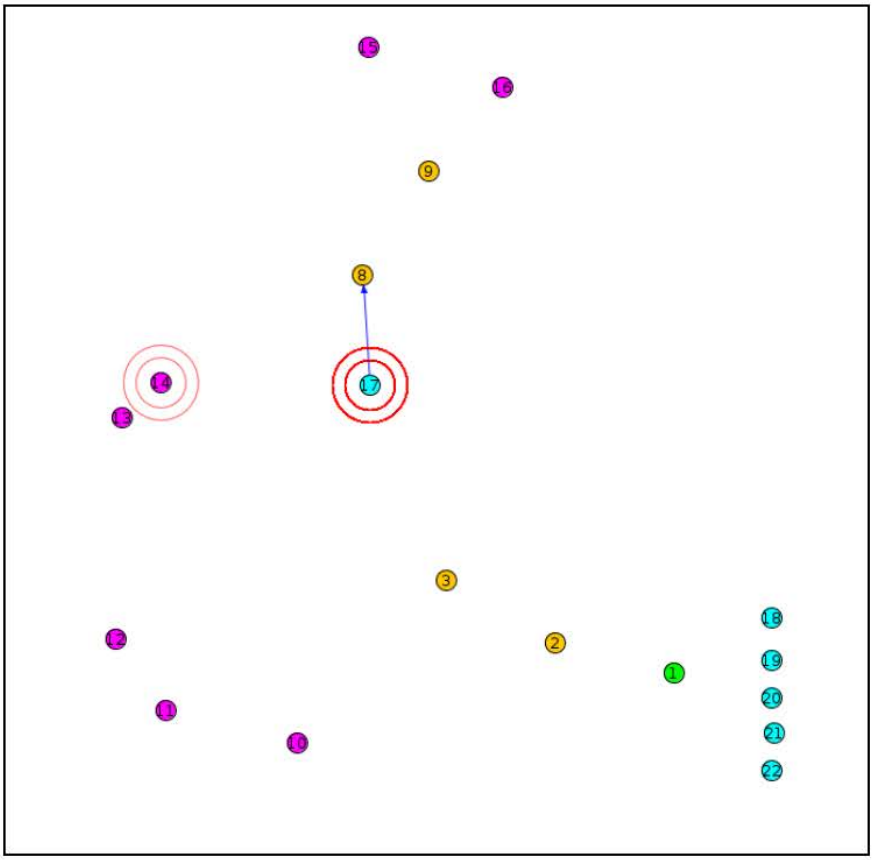


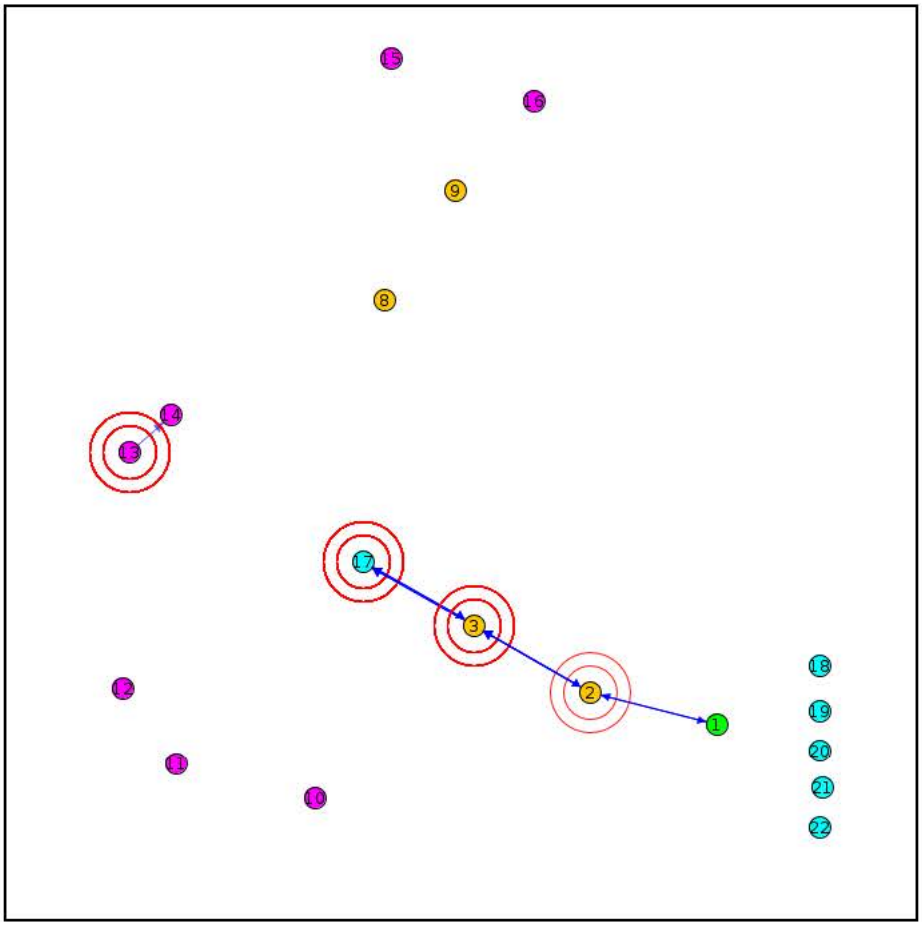


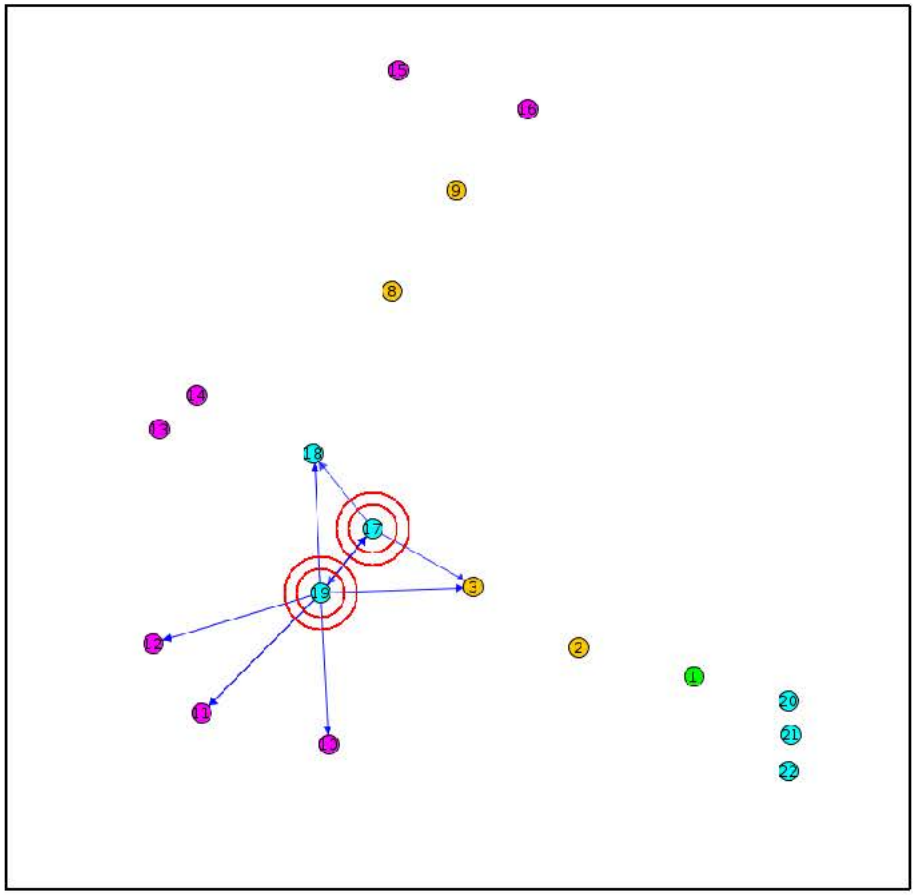

15
16
9
8
14
5
17
3
18
12
2
19
1
20
11
21
10
22

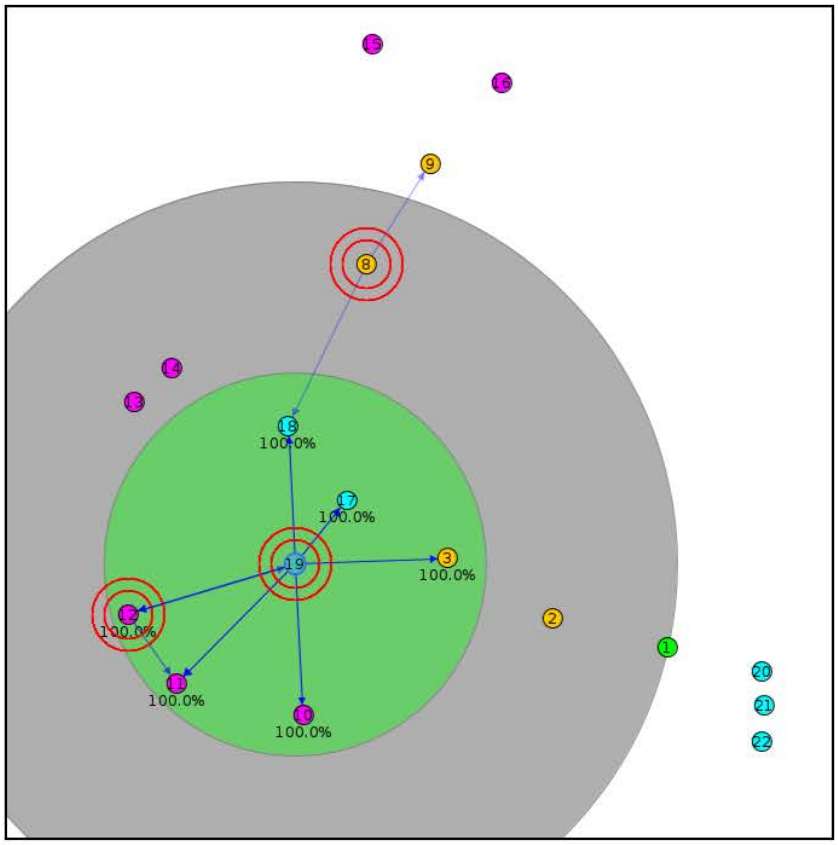

15
16
9
8
4
14
3
18
2
19
1
20
7
5
21
11
10
22

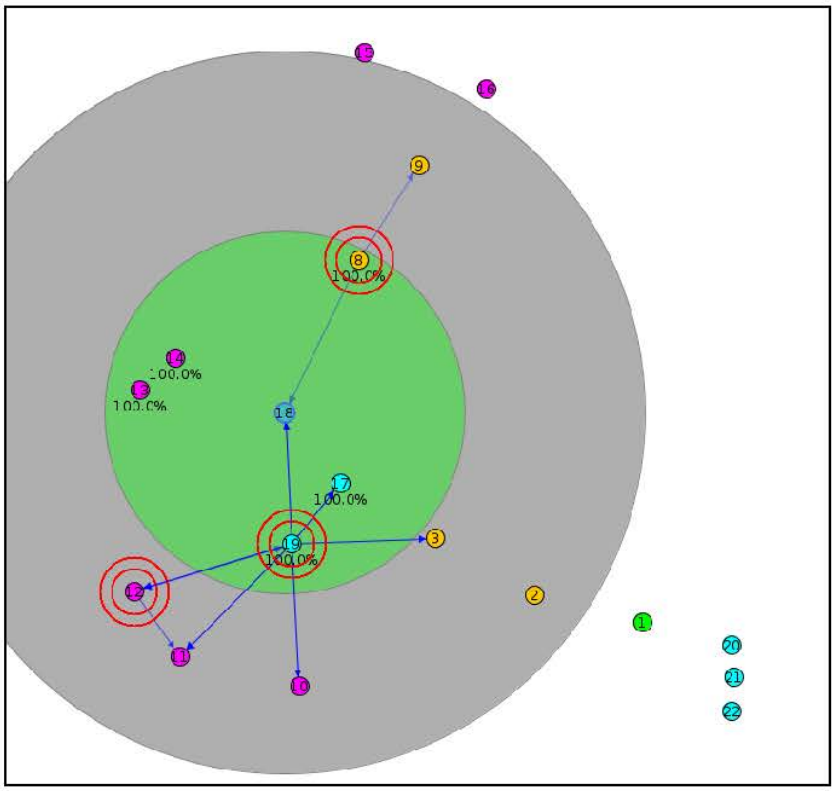

15
16
9
8
14
13
17
3
12
2
1
18
19
20
21
22
11
10

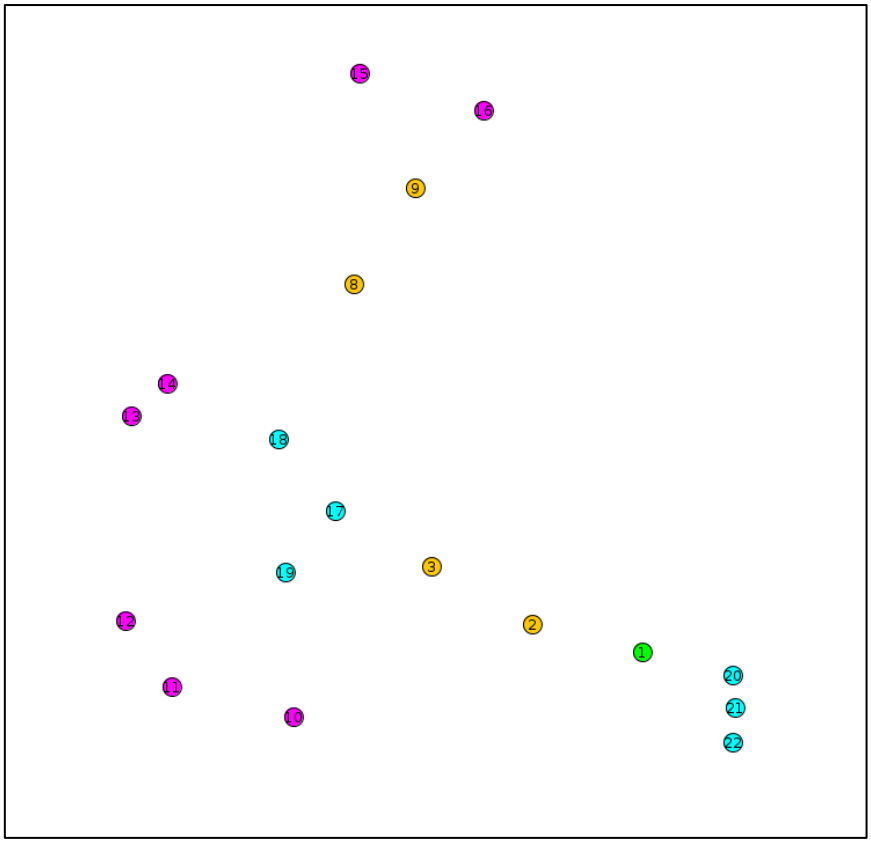

15
16
9
8
14
17
13
3
12
2
1
18
19
20
21
22
11
10


15
16
9
8
14
13
17
3
2
1
18
19
20
21
22
12
11
10

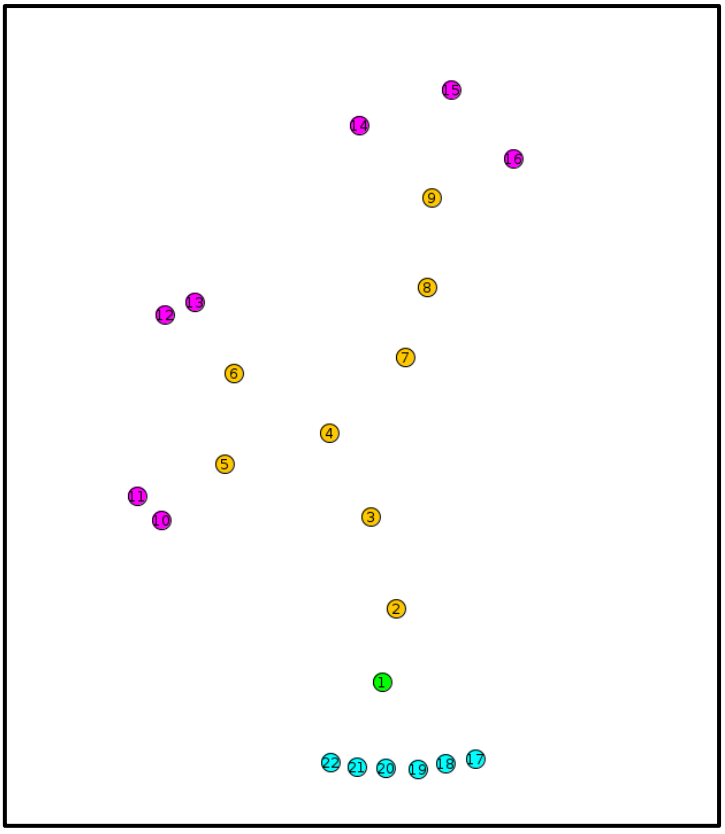

15
16
9
8
14
13
18
17
3
12
2
1
11
10
20
21
22

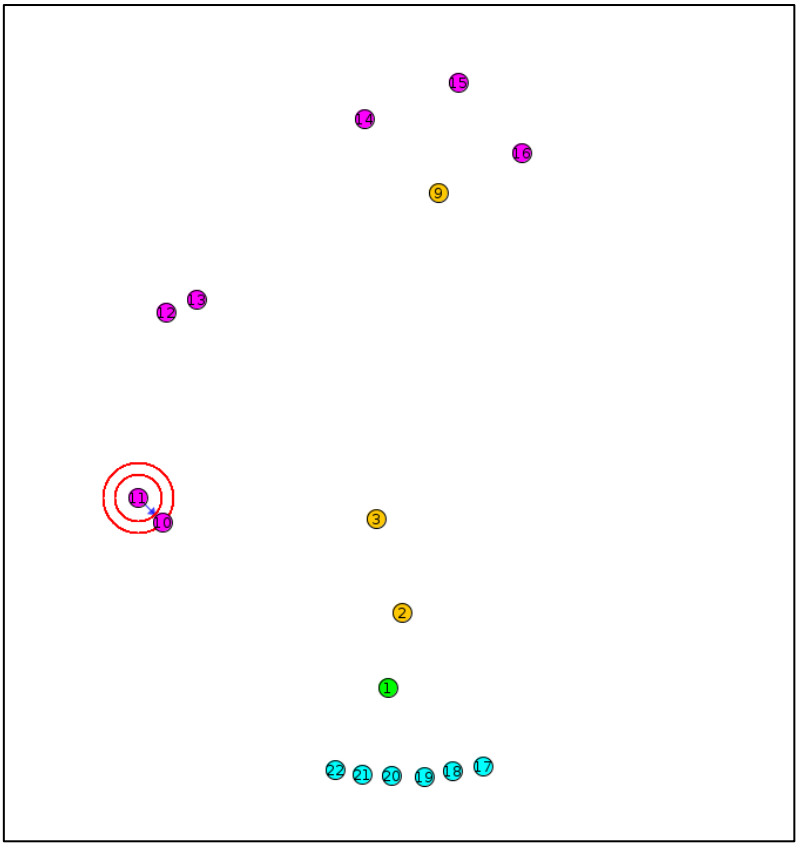

15
16
9
8
14
13
18
100.0%
17
100.0%
19
3
100.0%
12
100.0%
2
1
11
100.0%
10
100.0%
20
21
22

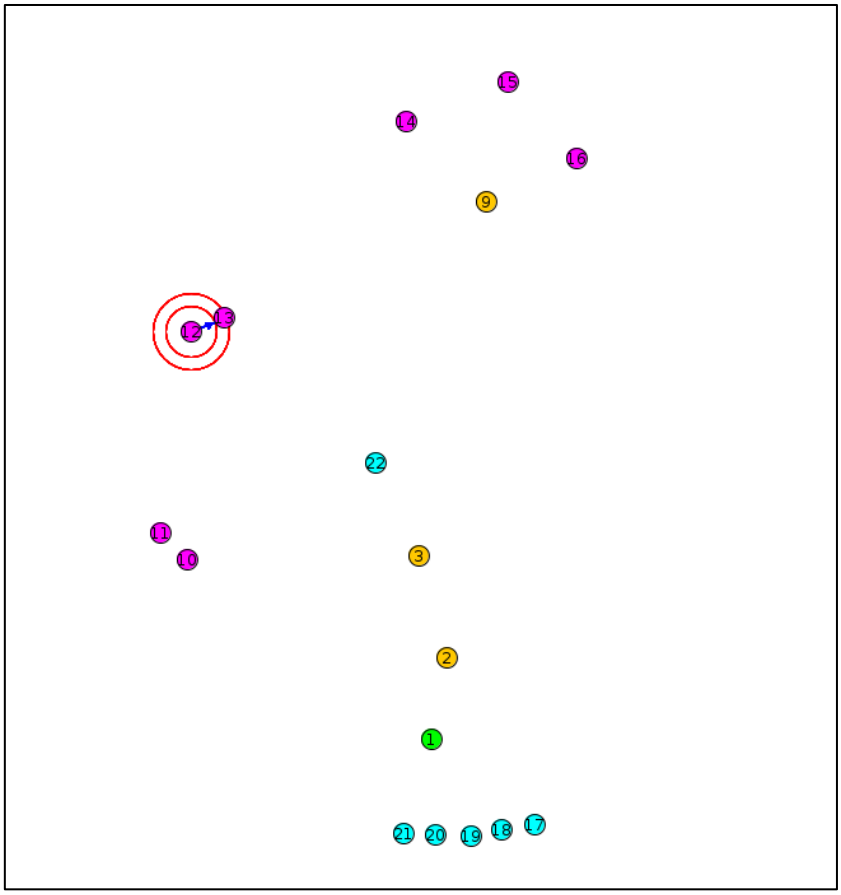

15
16
9
8
100.0%
14
100.0%
13
100.0%
18
17
100.0%
19
100.0%
3
12
2
1
11
10
20
21
22

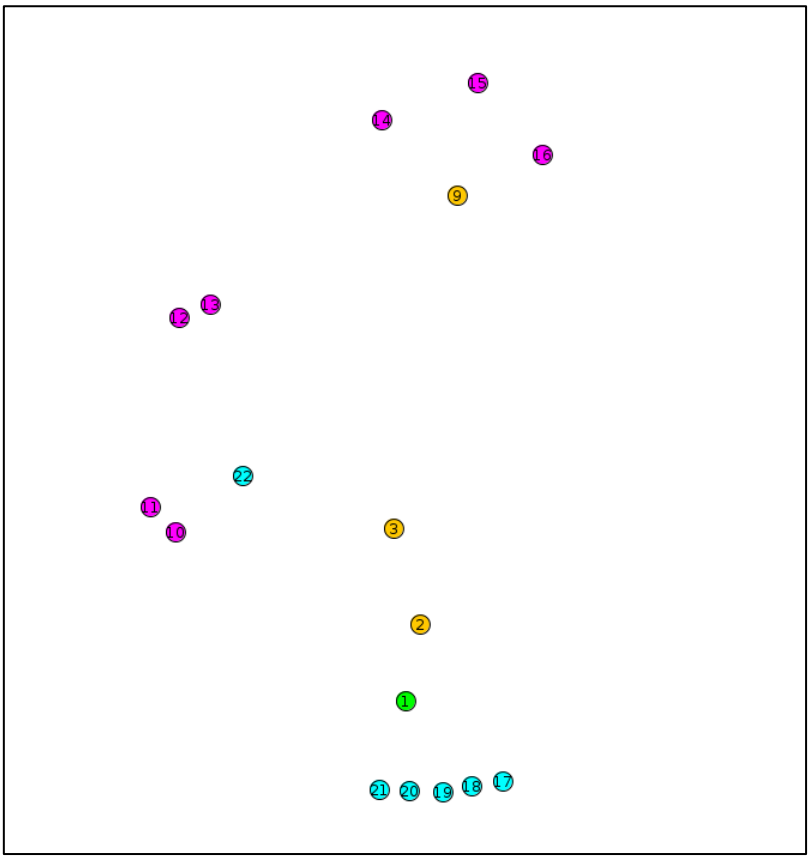

15
16
9
8
14
13
18
17
19
3
12
2
1
20
21
22
11
10

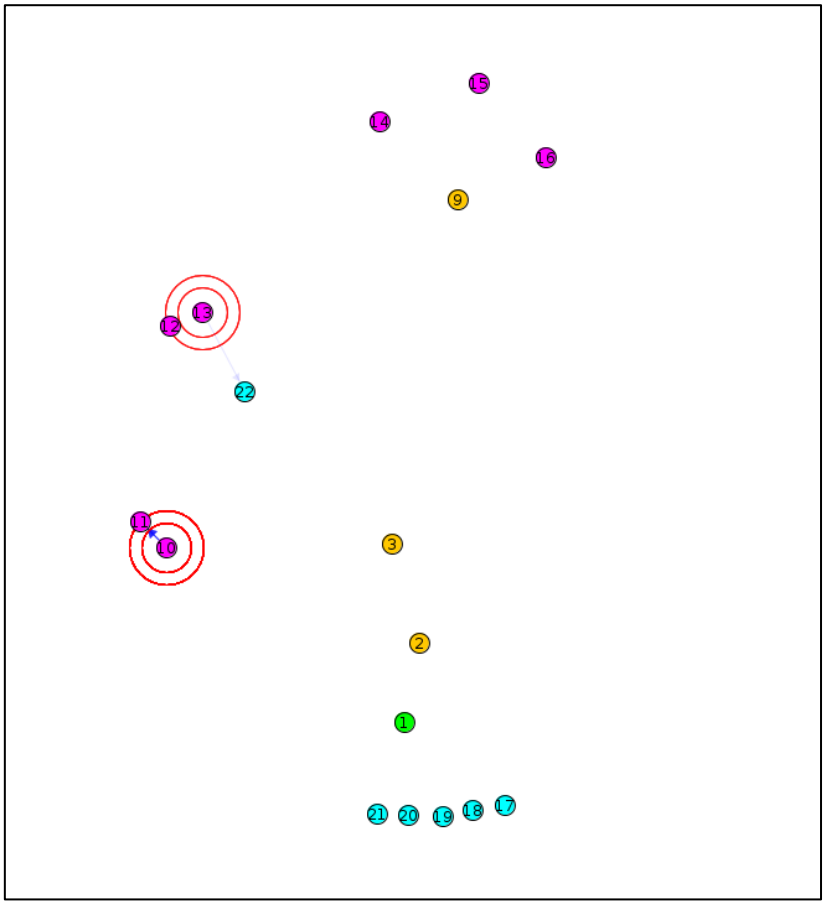
Scenario B

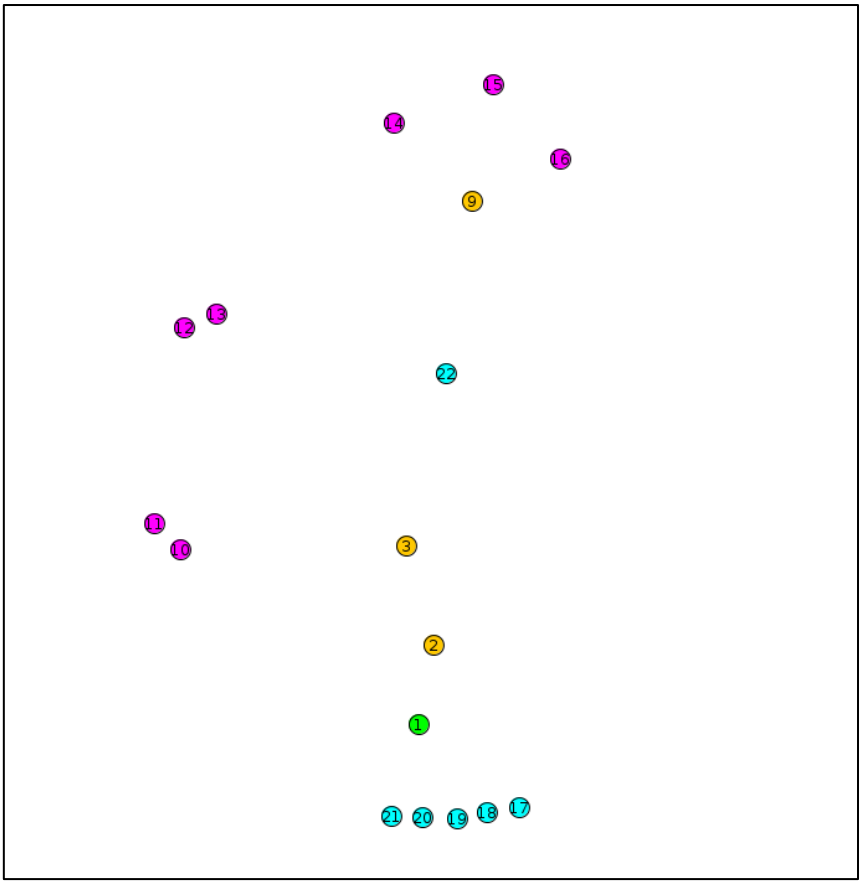

15
14
16
9
8
12
13
7
6
4
5
11
10
3
2
1
22
21
20
19
18
17

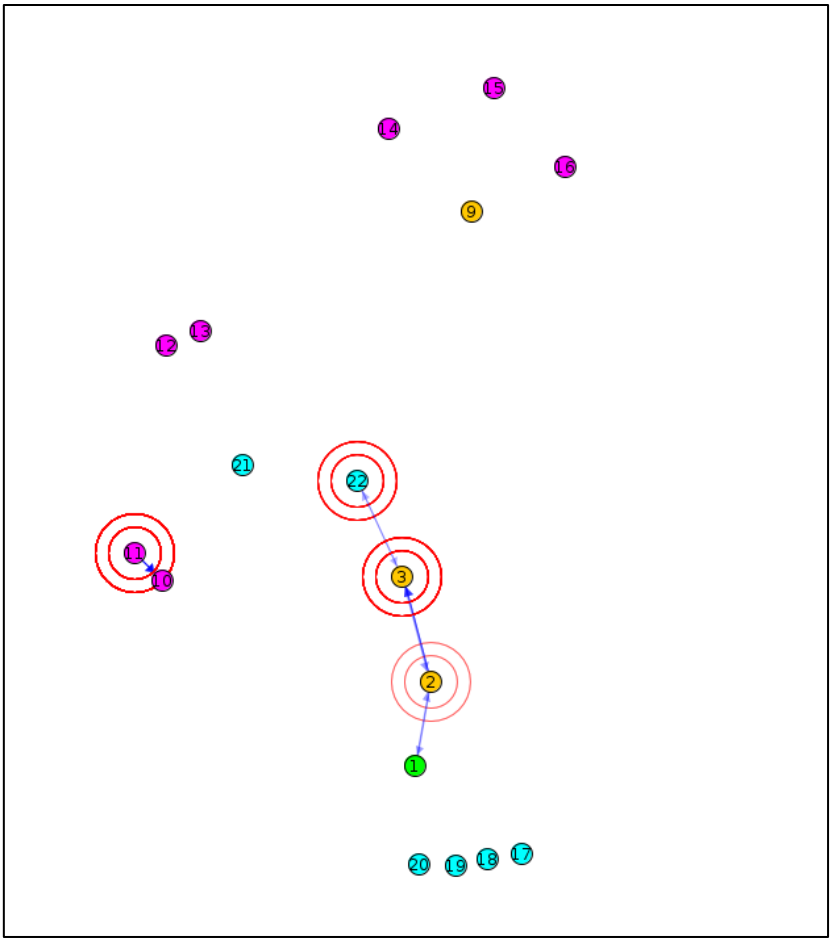

L5
L4
L6
9
L2
L3
L1
L0
3
2
1
22
21
20
L9
L8
L7

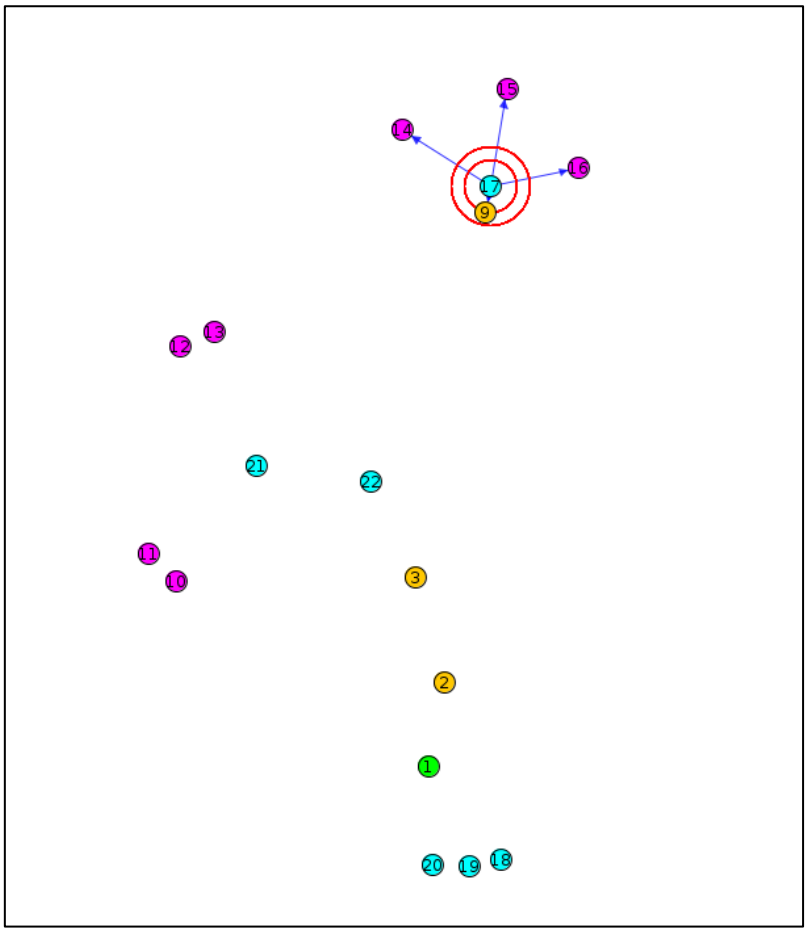

L5
L4
L6
9
L2
L3
22
L1
L0
3
2
1
21
20
L9
L8
L7

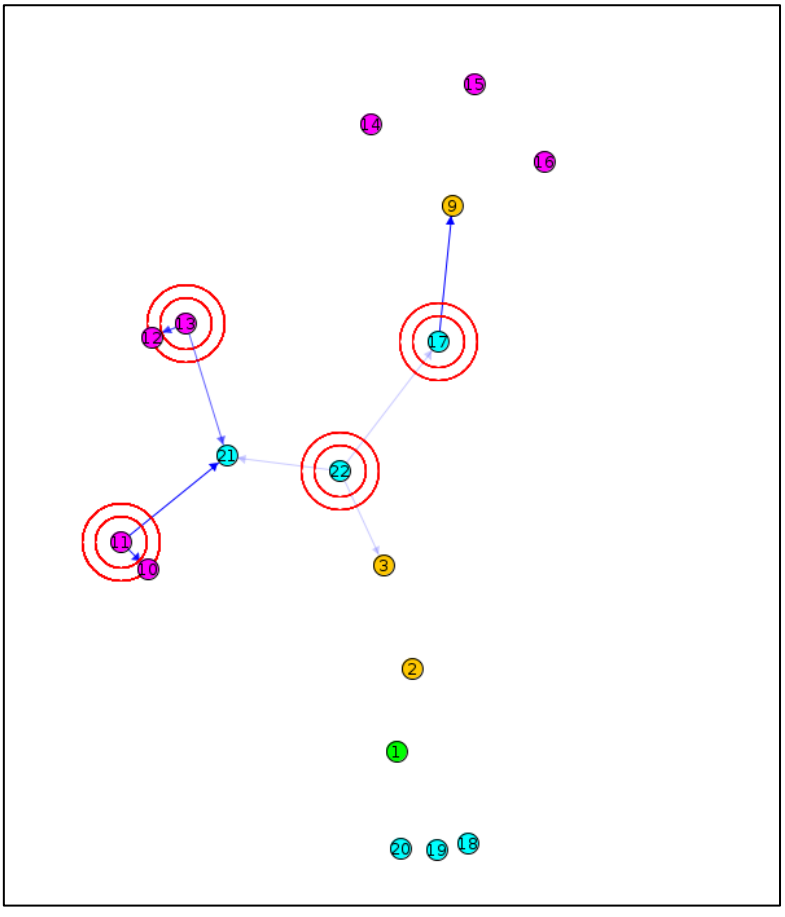

15
14
16
9
13
12
22
11
10
3
2
1
21
20
19
18
17


15
14
16
9
13
12
22
11
10
3
2
1
21
20
19
18
17


L5
L4
L6
9
L2
L3
22
L1
L0
3
2
1
21
20
19
L8
L7


L5
L4
L6
9
L2
L3
21
22
L1
L0
3
2
1
20
19
L8
L7


l5
l4
l6
l7
9
l3
l2
21
22
l1
l0
3
2
1
20
l9
l8


l5
l4
l6
9
l3
l2
l7
21
22
l1
l0
3
2
1
20
l9
l8