FAULT TOLERANT MISSION-CRITICAL WIRELESS SENSOR AND ACTOR NETWORKS

BY

Abdullah Alfadhly

A Dissertation Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

In

Computer Science and Engineering

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Abdullah Alfadhly** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE AND ENGINEERING.**

Dr. Uthman Baroudi
(Advisor)

Dr. Omar Al-Turki
Department Chairman

Dr. Dr. Mohamed Younis
(Co-Advisor)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Shokri Selim
(Member)

4/11/13
Date

Dr. Radwan Abdel-Aal
(Member)

Dr. Ahmad Masoud
(Member)

DEDICATION

This work is dedicated to my wife who supports me during my journey

ACKNOWLEDGMENTS

After my thanks to Allah, my first acknowledgment goes to my adviser Dr Uthman Baroudi and my co-advisor Dr Mohamed Younis for their continuous help and support. Without their guidance, I would not be able to get such achievement. My second acknowledgment goes to my committee members: Dr. Shokri Selim, Dr. Radwan Abdel-Aal, and Dr Ahmed Masoud for their guidance valuable time.

Finally, all my thanks and acknowledgments are given to KFUPM for providing me with all needed resources and facilities. Without such support, I would not reach this very important milestone in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Full Name : Abdullah Nasser Fahad Alfadhly

Thesis Title : Type Complete Thesis Title

Major Field : Computer Engineering

Date of Degree : May 2013

Wireless sensor and actor networks (WSANs) have emerged recently in many Mission Critical Applications (MCA) such as military surveillance, research and rescue, and fire extinguishing, etc. These types of applications need to be deployed on robust networks that can handle node failures in real time manner. However, WSAN usually operate in harsh environment and thus become susceptible to breakage in connectivity due to the failure of one or multiple actor nodes. Given that WSANS are deployed in remote areas, restoring connectivity through self-reconfiguring the network topology becomes the most preferred solution. In this PhD dissertation, we investigate the requirements of Critical Mission Wireless Sensor and Actor Networks in terms of robustness and connectivity and provide analytically and by simulation central and distributed approaches to handle single and multiple node failures. The central approach which serves as a lower bound to other heuristics is based on Integer Linear Programming (ILP) formulation and uses traveled distance as its objective function. While minimizing the total traveled distance is the main goal of the ILP approach, other performance metrics are considered such as the loss of coverage and the maximum traveled distance by a node. Since applying a central approach is not feasible in WSAN, we developed distributed approaches that depend on local information, and provide a restoration mechanism that can handle single and multiple node failures with minimized cost. Our first distributed approach is called Least

Distance Movement Recovery approach (LDMR) which exploits non-critical nodes in the network in order to replace failed nodes. It has been enhanced to behave adaptively based on the network topology in order to achieve better performance in sparse and dense networks. The new adaptive approach which is based on LDMR is called Adaptive Connectivity Restoration Algorithm (ACRA).

To restore network connectivity in case of multiple simultaneous failures, we developed a new approach called Simultaneous Failures Recovery Approach (SFRA). SFRA depends on constructing a recovery tree from the original network starting from a pre-assigned root. Unlike other solutions, SFRA can handle completely partitioned networks based on the current state of the network topology. We show the effectiveness and correctness of our approaches analytically and by simulations.

ملخص الرسالة

الاسم الكامل: عبد الله ناصر فهد الفضلي

عنوان الرسالة: شبكات الاستشعار التفاعلية القابلة للعيوب ذات التطبيقات الحرجة

التخصص: هندسة الحاسب الآلي

تاريخ الدرجة العلمية: مايو ٢٠١٣م

ظهرت شبكات الاستشعار التفاعلية في الآونة الأخيرة في كثير من التطبيقات الحرجة والهامة مثل المراقبة العسكرية، والبحث والإنقاذ، والإطفاء، الخ. هذه الأنواع من التطبيقات تحتاج إلى نشرها على شبكات قوية تستطيع التعامل مع حالات الفشل آنيا . تعمل شبكات الاستشعار التفاعلية في بيئة قاسية ولذلك تصبح عرضة لانقطاع الاتصال بسبب تعطل جهاز تفاعلي واحد أو أكثر. وبالنظر إلى أن هذه الشبكات تنشر في أماكن نائية ، فإن إعادة الاتصال عن طريق إعادة التكوين الذاتي لطوبولوجيا الشبكة هو الحل الأكثر تفضيلا. في أطروحة الدكتوراه هذه ، تحققنا من متطلبات شبكات الاستشعار التفاعلية ذات التطبيقات الحرجة من حيث المتانة والاتصال ، وقمنا من الناحية التحليلية و المحاكاتية بتوفير حلول مركزية و موزعة للتعامل مع فشل جهاز تفاعلي واحد أو أكثر من أجهزة هذه الشبكات. الحل المركزي والذي يمكن أن يمثل الحد الادني للحلول الأخرى قائم على البرمجة الخطية ذات العددية الصحيحة من حيث الصياغة ويستخدم مسافة التحرك كهدف وظيفي. مع أن التقليل من المسافة الإجمالية للتحرك هو الهدف الأساسي للحل المركزي ، إلا أننا اعتبرنا مقاييس الأداء الأخرى مثل فقدان التغطية والمسافة القصوى التي يمكن أن يقطعها أي جهاز تفاعلي أثناء عملية إعادة الاتصال للشبكة. وبسبب ان الحلول المركزية ليست مجدية في شبكات الاستشعار التفاعلية ، وضعنا حلول موزعة تعتمد على المعلومات المحلية لكل جهاز تفاعلي ، وتوفر آلية للترميم تستطيع التعامل مع فشل جهاز تفاعلي واحد أو أكثر بحدود أدنى للكلفة. حلنا الأول هو حل الاستعادة ذو المسافة الاقل للحركة (Least Distance Movement Recovery Approach) والذي يستغل الأجهزة التفاعلية ذات المواضع غير الحرجة (المهمة) في الشبكة من أجل استبدال الأجهزة التفاعلية المتعطلة ، وقد تم تعزيز هذا الحل بجعله متكيفا حسب طوبولوجيا الشبكة من أجل تحقيق أداء أفضل في الشبكات الكثيفة و المتفرقة. وقد أسمينا هذا الحل الجديد والذي يقوم على حلنا الأول بخوارزمية إعادة الاتصال المتكيفة (Adaptive Connectivity Restoration Algorithm).

ولاستعادة اتصال الشبكة في حالة الفشل المتزامنة لأكثر من جهاز تفاعلي، وضعنا حلا جديدا يسمى حل الاستعادة بسبب الفشل الآني (Simultaneous Failures Recovery Approach) . يعتمد هذا الحل على بناء شجرة الاستعادة من الشبكة الأصلية بدءا من جذر محدد مسبقا. وخلافا لغيره من الحلول الأخرى ، يمكن لهذا الحل التعامل مع الشبكات المنقسمة تماما استنادا للحالة الراهنة لطوبولوجيا الشبكة. وقد وضحنا صحة وفعالية هذه الحلول بالتحليل والمحاكاة.

CHAPTER 1

INTRODUCTION

1.1. Background

In this section, we will give an introduction to Wireless Sensor and Actor Networks (WSANs) and its underlying technology: Wireless Sensor Networks (WSNs) since they share many features, challenges, and applications. We will first give an overview of WSN, and then we give an overview of WSAN.

1.1.1. Wireless Sensor Networks (WSNs)

Advances in electronic design and wireless communication have enabled the development of low power devices which have the capabilities of sensing, processing, and communicating. Those small devices can then be used to deploy a self-organized network whose function is to sense the surroundings in order to detect a certain condition or event, process the data, and send information to a management station or control an actuator. Therefore, the choice of sensors is obviously controlled by the type of the deployed application. Figure 1 shows the typical components of a wireless sensor network device.

Wireless sensor networks can be used in many applications such as military, environmental, health, home, and other commercial applications. In military, they can be used in battlefield surveillance, reconnaissance of opposing forces and terrain, targeting,

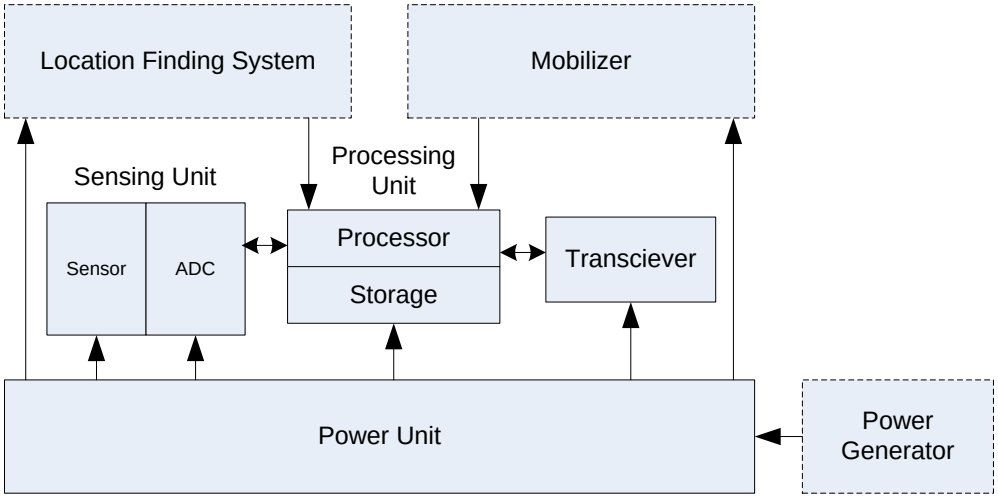


Figure 1: A typical wireless sensor network node components (redrawn from [1])

battle damage assessment, and nuclear, biological and chemical attack detection and reconnaissance. Some examples of environmental applications include tracking the movements of birds, small animals, and insects, forest fire detection, bio-complexity mapping of the environment, Flood detection, and precision agriculture. Health application include patient monitoring, drug administration in hospitals, monitoring the movements and internal processes of insects or other small animals; tele-monitoring of human physiological data, and tracking and monitoring doctors and patients inside a hospital. Home automation and smart environment are examples of Home applications. Other applications may include interactive museums, detecting and monitoring car thefts, and managing inventory control [1].

Several factors make the design of WSN protocols a challenging task. First, Sensors are constrained in energy supply, processing capability, and bandwidth capacity. Second, they are deployed in very large quantities, therefore they usually do not use global addressing because it is difficult to maintain, although there are some efforts to overcome this issue [2]–[4]. Third, in most scenarios, the flow of data in WSN is from multi-nodes (network devices) to a single node (the sink). Fourth, sensors produce redundant data which has to be aggregated so that more energy can be preserved by cutting the number of transmissions [5], [6]. Therefore, most researchers have worked to address these requirements when they designed their protocols and algorithms.

Energy efficiency, scalability, and adaptability to changes are very important features in order to design a good MAC protocol for Wireless Sensor Networks [7]. Although, there are many proposed MACs for WSNs, there is no single protocol accepted as a standard for all types of applications. The rational is that WSNs are application specific which

means different network design objectives are needed for different applications. On the other hand, routing techniques are classified into three categories based on the underlying network structure: flat, hierarchical, and location-based routing. Furthermore, these protocols can be classified into multipath-based, query-based, negotiation-based, QoS-based, and coherent based depending on the protocol operation [6].

Another influencing factor when designing a wireless sensor network is fault tolerance [1]. Fault tolerance is the ability to sustain sensor network functionalities without any interruption due to sensor node failures [8]–[10]. However, fault tolerance requirements can be relaxed if the application mission of WSN is not critical. On the other hand, On the other hand, if sensor nodes are being deployed in a battlefield for surveillance and detection, then the fault tolerance has to be high because the sensed data are critical and sensor nodes can be destroyed by hostile actions.

1.1.2. Wireless Sensor and Actor Networks (WSANs)

Wireless Sensor and actor networks have in addition to sensors more capable devices called actors. Actors are equipped with more energy and processing capability and they can communicate through longer distances. When a sensor detects an event, it first notifies a nearby actor which analyzes the data and coordinates with other actors for the required action. For example, in a forest fire detection application, sensors detect the fire and send this event to actors where they can coordinate among them and extinguish the fire before spreading to other parts of the forest. Figure 2 shows a typical articulation of a WSAN.

In order for a WSAN to carry out its application successfully, it has to satisfy two requirements [11]:

Coordination: Unlike WSN where there is a single entity (i.e. the sink) which receives all sensed information and deliver it to a central monitoring system, WSAN needs a coordination mechanism among actors to carry out the required task. The event can be detected by multiple sensors and therefore multiple actors can be notified. Reconstruction of the event and determining its characteristics is also part of actor-actor coordination [11].

Real time action: sensors have to propagate the data to actors in real time and actors have to act promptly. For example, in fire detection application, any delay yields the action later useless or impossible.

The above two requirements impose difficult challenges in order to develop and design WSAN protocols and algorithms.

WSANs are deployed usually in a hostile and harsh environment. Therefore, they are prone to frequent failures that could render the network useless if there is no self-mitigation to such failures. Moreover, these nodes are battery operated, and therefore, they may exhaust their energies any time.

1.2. Research Motivation

Wireless Sensor and Actor Networks (WSAN) have attracted a lot of interest in recent years. Their potential applications include search-and-rescue, forest fire detection and

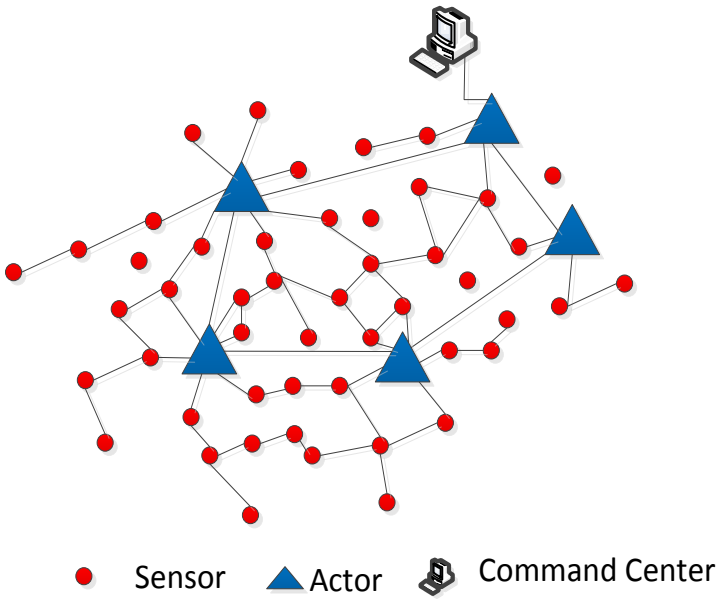


Figure 2: A typical WSAN

containment, battlefield reconnaissance, under-water surveillance, etc. Many of those applications are mission critical where robustness is very important. Given the collaborative nature of the WSAN operation and the criticality of the deployed applications, inter-actor connectivity is essential. Obviously, coordination among actors cannot be performed in a disconnected network topology. Therefore, actors strive to sustain communication links among themselves when they move. However, the failure of one or multiple actors may partition the network into disjoint sub-networks. This may happen while responding to a harsh event, e.g., a fire, and would require a rapid recovery so that the event would not get out of hand and lead to disastrous consequences. Since WSAN operate unattended and the deployment of spare actors may take time, the recovery should be performed through network self-reconfiguration using existing resources. Not only an actor failure may lead to a loss of inter-actor connectivity, but it also causes degradation in coverage in the vicinity the failed node. Having good actor coverage is very important in WSAN in order to make sure that a sensor can report its finding to an actor and the actor responds in a timely manner. Therefore, recovery from actor failure should not only restore severed connectivity but also should strive to limit the loss in actor coverage.

Although, there is some research in this area, more research is needed to model the dis-connectivity problem, develop new approaches that can recover single and multiple node failures.

1.3. Contributions

- We studied how a failure of an actor or multiple actors can be tolerated in mission-critical WSANs by maintaining the inter-actors connectivity. Unlike other costly solutions which depend on the increasing of a connectivity factor (number of connections) between actors [12], we proposed a distributed solution that can operate in real time and restore connectivity to the network with a minimized cost.

- We first modeled the problem mathematically and came up with an optimal solution. In this step, we assumed a central entity that has all information needed to reach the optimal decision. In our analysis, we considered the following metrics:

Total Traveled Distance: This metric gives the total distance traveled by all nodes in the network during the recovery restoration. This metric indicates how much energy will be consumed by the whole network due to the mechanical movements of the network actors.

Average Traveled Distance: this metric computes the average traveled distance for a node that got engaged in the recovery operation. This metric can be used to know how much energy loss is shared among nodes during the lifetime of the network.

Coverage: this metric captures the loss of coverage resulted from the node movements.

- After analyzing results from the central approach, we developed distributed approaches that achieve low distance cost, low communication messages, and can handle single and multiple node failures.

- We developed a distributed approach that takes advantage of non-critical nodes in the network in order to restore connectivity among network nodes. The approach strives to lower the total movement distance caused by the process of the recovery. Therefore, we call it Least Distance Movement Recovery Approach (LDMR). LDMR provides a detailed mechanism on searching for non-critical nodes and avoiding conflicts or network dis-connectivity during recovery.

- We enhanced LDMR by making the approach behaving adaptively based on the network topology. The new adaptive approach which is called Adaptive Connectivity Restoration Algorithm (ACRA) achieves low cost in case of sparse and dense networks. It also achieves better coverage compared to other approaches that depends on shrinking network nodes in case of failures.

- Restoring a connective to the network in a distributed manner in case of simultaneous nodes failures is very challenging. Therefore we developed an efficient and robust approach that handles multiple simultaneous failures and achieve low cost. The new approach is explained in detail and verified by an extensive simulation.

Figure 3 shows the main steps that describe our work. The mathematical representation is based on Integer Linear Programming (ILP) formulation. While message complexity is not an issue in the central approach, it is a performance metric

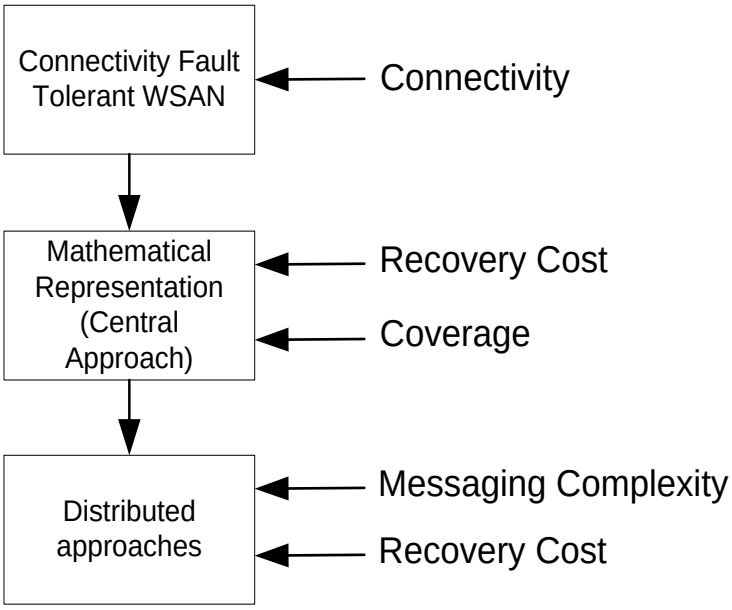


Figure 3: Dissertation Overview

in developing distributed approaches. In addition, multiple failures bring complexity to the distributed approaches because of the locality of information. A node or a group of nodes cannot be aware of all failures in the network in order to do the right action in a self-healing process.

1.4. Thesis Organization

The next chapters are organized as the followings:

Chapter 2: it provides detailed literature review on the subject of connectivity fault tolerance in WSAN. In our review, we focused on the real time approaches that use mobility as a primitive to restore connectivity to the partitioned networks**.**

Chapter 3: we explained our central approach which is based on ILP formulation. We showed the objective functions, the problem constraints, and provided a detailed example. Our work in this chapter and its simulation results is published under the following publication:

Alfadhly, A., U. Baroudi, and M. Younis. "Optimal node repositioning for tolerating node failure in wireless sensor actor network." *Communications (QBSC), 2010 25th Biennial Symposium on. IEEE, 2010.*

Chapter 4: we discussed our distributed solution for a single node failure. The solution is called Least Distance Movement Recovery Approach (LDMR). LDMR was also extended to enhance its performance. The enhanced approached is called an Adaptive Connectivity Restoration Approach (ACRA). Our work in this chapter and its simulation results is published under the following publications:

Alfadhly, Abdullah, Uthman Baroudi, and Mohamed Younis. "Least distance movement recovery approach for large scale wireless sensor and actor networks." *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International. IEEE, 2011.*

Alfadhly, Abdullah, Uthman Baroudi, and Mohamed Younis. "An adaptive connectivity restoration algorithm for wireless sensor and actor networks." *International Journal of Autonomous and Adaptive Communications Systems 6.2 (2013): 167-190.*

Chapter 5: we discussed our distribution approach for multiple node failures. The approach is called Simultaneous Failures Recovery Approach (SFRA). Our work in this chapter and its simulation results is published under the following publications:

Alfadhly, Abdullah, Uthman Baroudi, and Mohamed Younis. "An effective approach for tolerating simultaneous failures in wireless sensor and actor networks." *Proceedings of the first ACM international workshop on Mission-oriented wireless sensor networking, 2012.*

A journal version of the above publication with extra analysis and more details is also in the process of submitting.

Chapter 6: in this chapter, we presented the simulation setup, and discussed and compared the simulation results of all approaches: Central Approach, LDMR, ACRA, and SFRA.

Chapter 7: we provided our conclusions and directions for future works.

CHAPTER 2

LITERATURE REVIEW

2.1. Overview

Motion capability of nodes has been utilized in wireless sensor networks and wireless sensor and actor networks in order to enhance the performance metrics of these networks. In wireless sensor networks, movable sensors are proposed to patch coverage holes, prolong network lifetime, and restore connectivity. For example, unbalanced coverage among different regions of the deployment area can happen in cases where random deployment is used because of the hostile and harsh environments. To solve such problem, a relocation of sensors has been proposed in [13] to solve such problem or to respond to sensor failures. Redundant sensors are detected first through a Grid-Quorum search, and directed to other locations where a problem exists. A cascaded movement is used by moving sensors to balance loss of energy among sensors. For extending network lifetime, relay nodes that can move has been proposed to collect data from sensors and send it to the base station [14]. While these mobile nodes can reach isolated sensors, its movement causes latency and may not be suitable for WSAN real time applications.

Most of the published schemes on tolerating node failure in WSAN can be classified into two categories:

- Provisioned solutions which rely on the availability of redundant resources that can make up for the lost node(s). However, provisioned solutions for restoring connectivity are not suitable for WSAN since actors are typically more expensive

and hard to deploy compared to sensors and thus assuming the presence of many actors is not practical.

- Real-time solutions which rely on repositioning the healthy actors so that a strongly connected inter-actor network topology can be established.

Since our proposed approaches fall into the second category, we will focus on this category explaining approaches proposed in the literature. We shall begin with approaches that tried to tackle the single node failure at a time and then we present the existing approaches that tried to solve the simultaneous multi-failure nodes problem.

2.2. Single Failure Approaches

These approaches are based on algorithms that are designed to handle one failure at a time. If there is more than one node failing at the same time, these algorithms may not work properly. For example, DARA [15] replaces the failed node with one of its neighbors. The approach requires every node to maintain 2-hop neighbor information so that the effect of the loss of a node can be assessed, i.e., whether the failed node is highly probable a cut-vertex or not. The candidate among the neighbors of the failed node is picked based on the node degree, distance from the failed node and the node's ID respectively. The effect of moving a node triggers a cascaded relocation that ripples throughout the network to avoid breaking connectivity in another part in the network. Figure 4 shows an example illustrating how DARA works. In this example, A1 which is the failed node has four neighbors: A2, A7, A8, and A9. A8 and A2 have a degree of two where A7 has a degree of three and A9 has a degree of four. According to DARA, the node with the least degree is the winner node which is replacing the failed node.

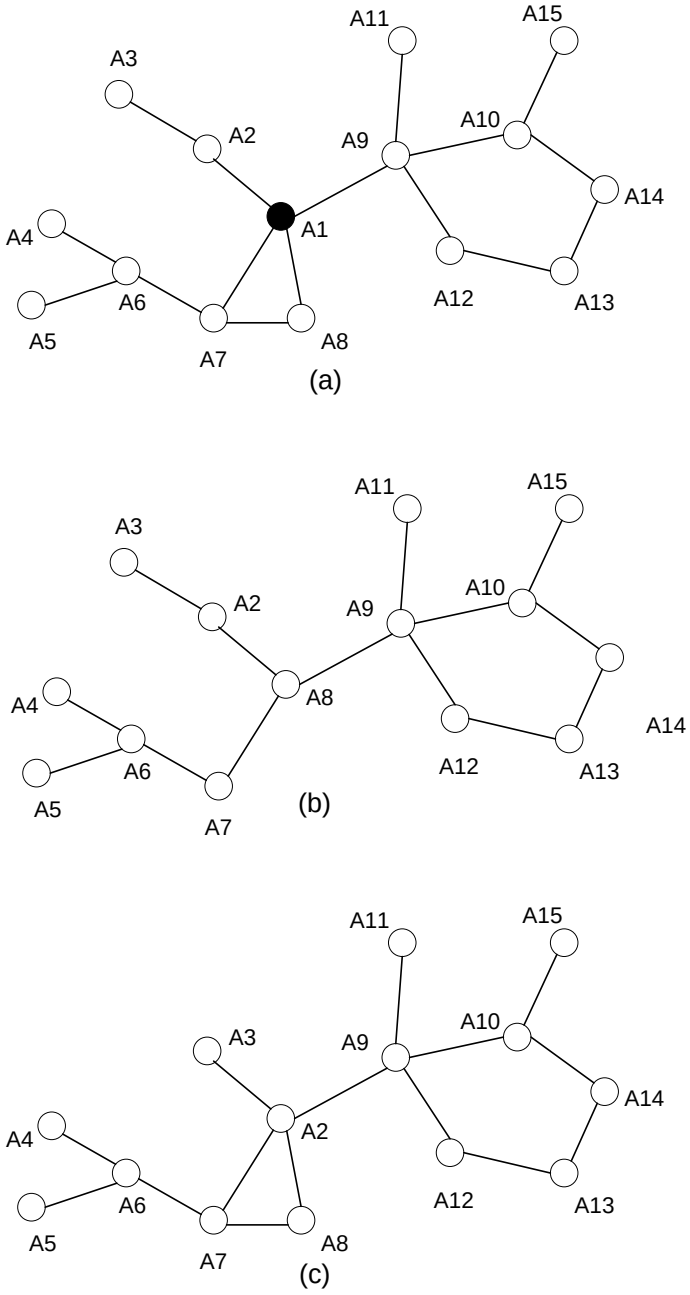


Figure 4: A DARA example: (a) the original network (b) A8 is replacing A1 (c) A2 is replacing A1

Therefore, since all nodes know the degrees of all nodes 2-hop away from them, A9 and A7 stops proceeding in the algorithm because their degrees are greater than those of A8 and A2. A2 and A8 then compare their distances to A1. If A8 has a less distance, A8 replaces A1 as shown in Figure 4-(b). Figure 4-(c) shows the network after A2 replacing A1 assuming A2 has a less distance to A1. In the latter case, A3 moves to the old position of A2 in order to connect to the rest of the network.

Another single failure approach called PADRA was proposed in [16]. It is a proactive scheme where each node assigns a fault handler (FH) for itself and sends a notification of assignment to this node. If a node fails, its fault handler starts the recovery process.

The work in [15], [16] rely on the fact that cut vertex nodes are known. Cut vertex determination can be done using a Connected Dominating sets (CDS) algorithm [17]. A set of nodes is dominating if it contains all nodes in the system or all nodes can be reached through the nodes in the set. A node is dominator if it is in the dominating set and dominatee otherwise. For example, in Figure 5, nodes 4, 8, and 7 are dominatees while nodes 1, 2,3,5,6 are dominators. Based on CDS knowledge, cut-vertex determination can be achieved as the following:

- If a node is a dominatee, it is not a cut vertex node.
- If a node is a dominator, then we have two cases:
    - If the node has a dominatee neighbor which does not have any neighbor, the node in this case is a cut vertex since it is the only source that connects the dominatee node to other parts of the network.

• If the node has dominator neighbors or dominatees with neighbors, the node in this case can be either a cut vertex or not. To be certain, the node is a cut vertex, one of the node's neighbors has to do a local Depth first Search (DFS) to look for the other neighbors of the node. If they can be found using another way, the node is not a cut vertex.

To clarify this by the example in Figure 5, nodes 4, 7, 8 are not cut-vertices since they are dominatees. Nodes 2 and 6 are cut-vertices since they are dominators which have dominatee neighbors. Nodes 1 and 5 are dominators and their neighbors are dominators also. However, if a DFS is done in node 6, there is no other way to reach node 1. Therefore, node 5 is a cut-vertex. If another DFS is done in nodes 2 or 3, there is no other way to reach node 5. Therefore, node 1 is a cut-vertex. Node 3 is a dominator but it is not a cut-vertex since we can reach 2 or 1 without passing through 3.

While [15], [16] need the detection of cut-vertices, Recovery through Inward Motion (RIM) approach [18] avoids doing this to simplify the recovery process. RIM needs only 1-hop neighbor information to function properly. After a node detects a failure of one its neighbors, it moves toward (i.e. in the direction of) the position of the failed node until it becomes *r/2* away where r is the transmission range. This first movement ensures that all neighbors of the failed node are connected. However, they may lose their connections with their neighbors. Therefore, their neighbors need to do cascaded movement but this time until they *r* away. This later process is repeated until all nodes are connected. In RIM, if a node is a neighbor of two moving nodes, it follows the node which has the highest rank (fewer hops to the failed node). If both nodes have the same rank, it moves to the closest intersection point of the two circles centered by it is two neighbors. Figure

6 is an example showing the operation of RIM. At first, nodes 2, 3, 4, and 5 detect the failure of node 1. They send notification to their neighbors and move to the position of node 1 until they are r/2 away. In the third step, although node 7 receives notifications from 5 and 4, but it moves to the node with highest rank which is 5 until it is *r* away from it. Node 6 receives notifications from two nodes with similar rank (5 and 4). Therefore, it moves to the closest intersection point of the two circles centered by 4 and 5. In the last step, node 8 moves to the position of node 7.

The approach of [19] strives to limit the scope of cascaded relocation through the identification of dominators. Basically, the dominating set is determined and only cascaded relocation is pursued when a dominator moves. Meanwhile, Basu and J. Redi [12] assume the network is bi-connected prior to the failure and propose an algorithm that moves nodes in groups in order to restore the lost bi-connectivity when a node fails. However, deploying more actors to have a bi-connected network increases the cost of the application. In addition, having this feature cannot be guaranteed for random deployment. Unlike our approach, the focus of [12], [16], [19] has been on connectivity restoration without considering coverage. Most of published schemes that consider connectivity and coverage are geared for network planning and not to tolerate a node failure [20]. The only prior effort that factors in both connectivity and coverage, to the best of our knowledge, is reported in [21]. However, the approach is based on moving the neighbors of a failed node back and forth in order to minimize the effect of a node loss. In other words, connectivity cannot be guaranteed at all times.

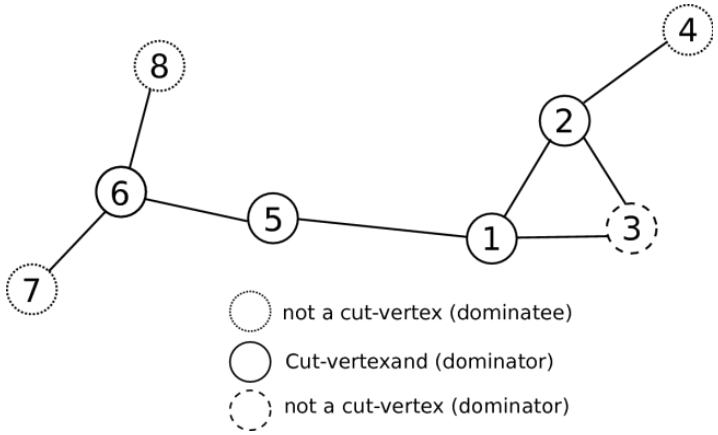


Figure 5: Cut-vertex determination

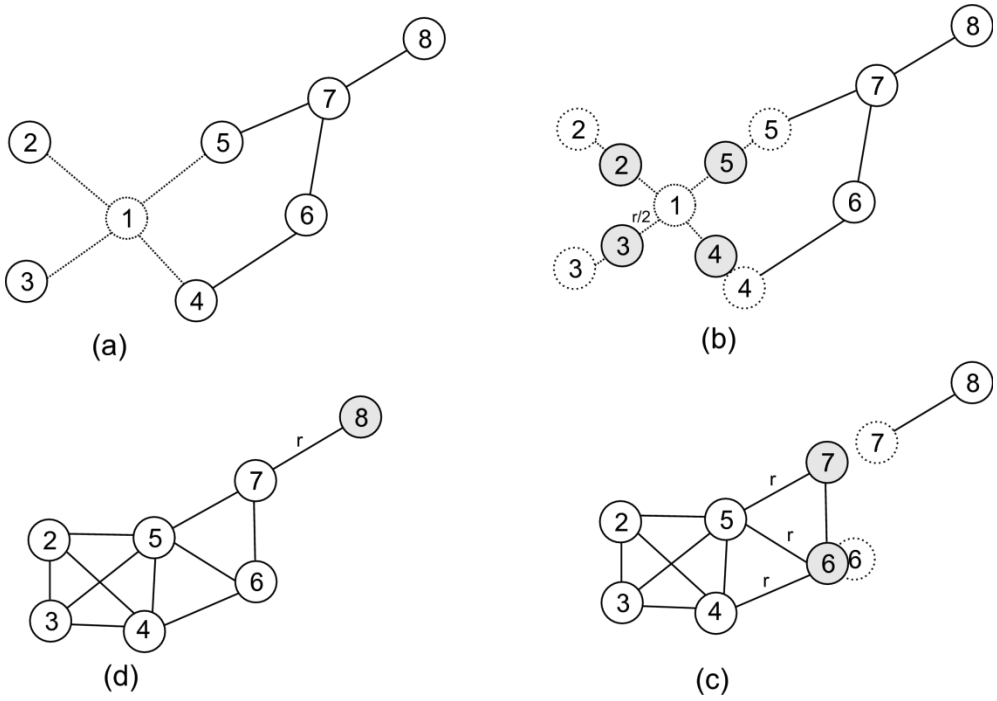


Figure 6: RIM example redrawn from [18]

2.3. Simultaneous Multiple Failures Approaches

The above approaches solved the problem of a single failed node exploiting the fact that other nodes are residing in their current positions and they are working normally. In many circumstances, such as earthquake multiple nodes will fail simultaneously that cause the network to be partitioned into multiple disconnected segments. Therefore, the single node failure approaches cannot resolve this problem and a new paradigm is needed to reestablish the network connectivity.

In [22], it was proposed to use the underlying sensors to detect other network partitions. After partitions have been detected, the closest node from each partition starts moving to each other. To maintain the connectivity in each partition, a cascaded movement is suggested

The same assumption is targeted in [23]. However, their suggested solution was to let nodes move to the center of the deployed area. An optimization based on Minimum Steiner Tree (MST) is done to minimize the number of relay nodes needed to restore the connectivity.

MPADRA was proposed in [7] as an extension to PADRA [6] to support multiple node failures. The problem of PADRA algorithm is the existence of a situation where one node is assigned to recover for two different nodes. If these two nodes fail at the same time, a race condition problem occurs and PADRA approach will not work. This problem is applicable for other proactive approach like DARA [5]. Although this problem was solved in MPADRA,

On the other hand, the idea of constructing a tree which we proposed in our simultaneous recovery approach was previously used for multicast routing in MANET [24]–[26]. However, the focus was only overcoming broken links. The loss of multiple nodes is significantly more challenging since the network gets partitioned and alternate routes will not be available between the affected nodes. Clustering is also used widely in MANET mainly for scalability reasons [27]. To the best of our knowledge the use of clustering as means for mitigating the simultaneous failure of multiple nodes has not been pursued in the literature.

CHAPTER 3

CENTRALIZED RECOVERY THROUGH NETWORK RECONFIGURATION

3.1. Problem Definition

Interconnectivity among actors in WSAN is a very important requirement for a successful deployed application. Therefore, an urgent real time restoration has to be done after a node failure. However, only cut-vertices and not all nodes in the network are essential to maintain connectivity. A failure of a cut-vertex node partitions the network into two or more disjoint segments. In Figure 7, nodes 1 and 5 are cut-vertices. To restore connectivity in this case, the failed node should be replaced by another (e.g., 6 in this example). The objective of the proposed approach is to have one or orchestrate a sequence of node movements with the least total travelled distance while not exceeding a pre-determined rate of coverage loss. We modeled the problem as an integer linear program as explained next.

3.2. Motivation

The recovery problem is modeled as Integer Linear Program (ILP) with an objective of forming a strongly connected inter-actor topology while minimizing the distance that the individual actors have to travel and minimizing the loss in coverage caused by the failure of some actors. The proposed solution handles the failure of one or multiple nodes and fits architectures in which the command center can develop the recovery plan. In addition, the proposed formulation provides a performance bound for existing schemes in

the literature, e.g. [5] which tolerates a single node failure. We also can build on this approach to develop a distributed approach that can provide a comparable performance.

3.3. Modeling

We model a WSAN as a graph $G(V,E)$. A node n_i in the network is represented with a vertex v_i in G. An edge between v_i and v_j exists if there is a communication link between the corresponding two nodes ni and nj in the WSAN. Let x_{ij} be a binary variable that equals one if a node j is located in position i. Before failure, $x_{ii} = 1$ and $x_{ij} = 0$. After a failure, this condition does not hold since a node j has to move to position i where n_i has failed. The recovery process that governs node motion is controlled by a cost function. In our approach, we tried to minimize this cost function while not violating any constraints imposed by the application. The model we used here to solve our problem is closely related to the permutation in integer programming formulation often used to solve graph related problems [28].

3.3.1. Objective Function:

If N, V, and F are three sets representing the network, cut-vertices, and failed nodes respectively, $F \subset V$ and $V \subset N$, our ILP objective function is :

$$Min \sum_{i \in N} \sum_{j \in N} x_j^i \, d_j^i \tag{3.1}$$

The objective function defines the total travelled distance resulted from the recovery operation. Our goal is to minimize this value subject to the problem constraints.

3.3.2. Constraints:

Using the *N*, *V*, and *F* sets define above, the following are the constraints of the optimizing the objective function:

$$\sum_{i \in N} x_j^i = 1, \forall j \in N, j \notin F \tag{3.2}$$

$$\sum_{j \in N, j \notin F} x_j^i = 1, \forall i \in N \tag{3.3}$$

$$\sum_{i \in V} \sum_{j \in N, j \notin F} x_j^i = M \tag{3.4}$$

Equation (3.2) ensures that a position i is not to be taken by more than one node at the same time while equation (3.3) ensures that node j can recover only one node at the same time. Assuming that there are M cut-vertices in the network, equation (3.4) guarantees that M nodes will be positioned there. Moreover, equation (3.4) ensures that the topology formed after recovery is strongly connected. To meet the coverage requirement, we added the following constraint:

$$\sum_{i \in N} \sum_{j \in N} \frac{x_j^i c_j}{cov} < l, i \neq j \tag{3.5}$$

where c_j, *cov*, and *l* are the area exclusively covered by node *j*, the total area covered by all nodes, and maximum tolerable rate of coverage loss, respectively. This constraint ensures that after concluding the recovery efforts, the relative coverage loss resulting from node moving will not exceed l. To cap the distance that a node travels, we use the

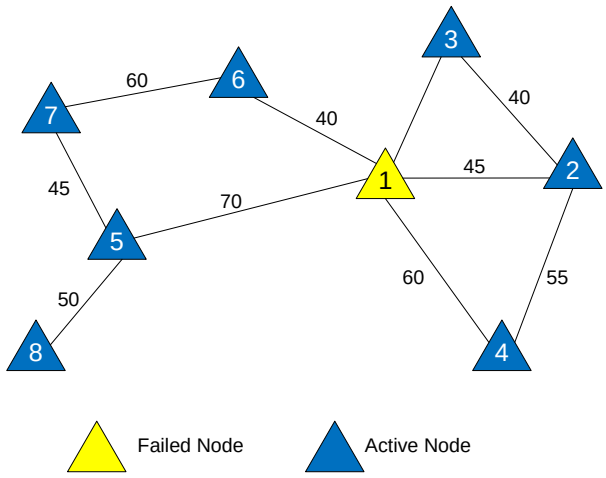


Figure 7: WSAN before recovery; the number above each link is the Euclidian distance.

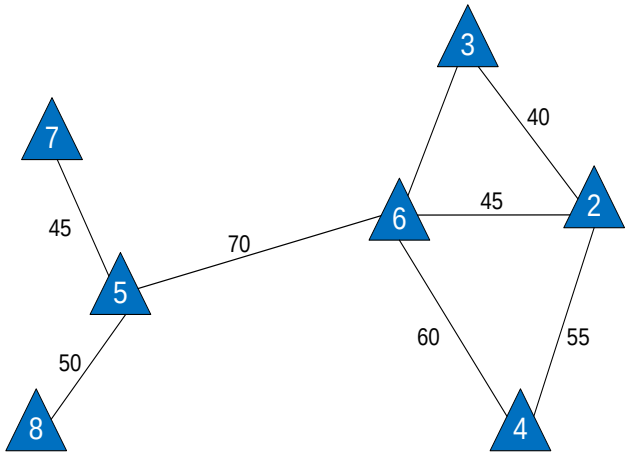


Figure 8: WSAN after recovery; the number above each link is the Euclidian distance.

following constraint:

$$x_j^i d_j^i \leq d_{max}, \forall i,j \in N \quad (3.6)$$

where d_{max} is the farthest distance that a node is allowed to move during the recovery. If the distance between any two nodes is larger than this value, none of the two nodes will move to the position of the other node during the recovery operation. It should be noted that the above formulation is able to handle the recovery of one or multiple node failures.

3.4. Detailed Example

Figure 8 shows how to apply our approach to the WSAN topology of Figure 7. Let us assume that node 1 has failed. The sets the network nodes *N*, cut-vertices *V*, and failed nodes *F* are as follows:

- *N*=1, 2,3,4,5,6,7,8,
- *V*=1,5, and
- *F*=1

Since node 1 is in the set of the failed nodes, it is considered as if it is not in its position anymore and the variable x_1^1 is ignored as a valid solution by equation (3.4). Since node 1 is a cut vertex, any valid solution to the problem has to have $x_j^1 = 1$ where $j \in N, j \notin F$. This implies that one of x_2^1, x_3^1, x_4^1, x_5^1, x_6^1, x_7^1, or x_8^1 has to be in the output solution. The goal of the objective function is to minimize the distance, and hence the node which has the least distance to the failed node, i.e., node 6, will be chosen. The same condition above is applied to node 5 because it is a cut vertex as well. However, because node 5 is

not in the failed set, x_5^5 is chosen as a valid solution since $d_5^5 = 0$. This means that node 5 is in its place and active. The solution to this example is to set x_6^1, x_2^2, x_3^3, x_4^4, x_5^5, x_7^7, and x_8^8 to 1 and set the rest variables to 0.

CHAPTER 4

DISTRIBUTED TOLERANCE OF SINGLE NODE FAILURE

4.1. Introduction

In the previous chapter, we have developed a centralized scheme which assumes a global knowledge about the network topology. The cost resulting from using this approach is optimum. When validating our centralized approach, we found that many of the failures are optimally restored by nearby non cut-vertices which can move directly to replace of the failed nodes without breaking the network connectivity (see Figure 9). These results have motivated us to develop heuristics approaches that can operate similarly in a distributed manner.

In this chapter, we present LDMR which is a distributed recovery algorithm that exploits non cut-vertex nodes in order to require the least travel distance from the engaged nodes. In LDMR, the neighbors of the failed node *F* move toward the position of *F* while they get replaced by their nearest non cut-vertex actors. The recovery process starts with the search phase where each neighbor broadcasts a message containing several entries such as failed node ID, neighbor node ID and, Time-To-Live (TTL). Each neighbor chooses the best candidate among the set of received responses based on a certain criteria (e.g. distance). The selected candidates replace the moved nodes without additional node relocation overhead. We compared our approach with RIM which depends only on cascaded movements. Extensive simulation experiments were carried out to validate the performance of LMDR. We showed that our approach outperforms RIM for larger and

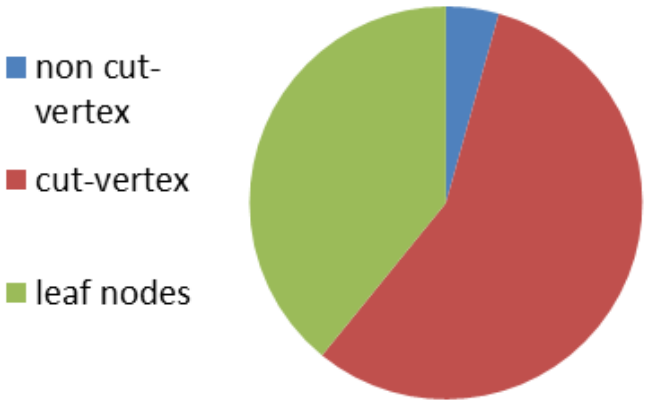


Figure 9: Optimal Solution Distribution

sparse networks. While the improvement in total travelled distance is achieved in these cases, loss of coverage after recovery operation is also comparable.

4.2. System Model and Problem statement

The WSAN network is composed of actors and sensors that are randomly deployed in an area. Actors are movable and have the capability to respond based on data collected by the sensors. All actors are assumed to have the same communication range. Since actors are more powerful than sensors, they typically have a longer communication range. After network deployment, a self-initialized phase is carried out by the whole nodes in the network. In this phase, each actor broadcasts a hello message with its identity and location. To cope with dynamic changes in the network, a heartbeat message is sent periodically by all actors. If an actor does not hear from its neighbor, a failure of that actor is assumed and the active actor has to take an immediate action.

The inter-actor topology can be modeled as a graph G (N, E), where N is the number of actors and E is the number of edges. The actor's position plays a key role in the stability of the network connectivity. Actors can be classified into two types: cut-vertex and non-cut-vertex. The failure of a cut-vertex actor partitions the network into isolated islands, while when a non-cut-vertex actor fails; strong network connectivity is still maintained. For example, in Figure 10, *A21, A7* and *A6* are non-cut-vertices while *A0 and A14* are cut-vertices. Therefore, to maintain the connectivity of the network, cut-vertex determination is important to react for node failures. Determining whether a node is a cut-vertex or not can be easily done by using depth first search trees (DFS). However,

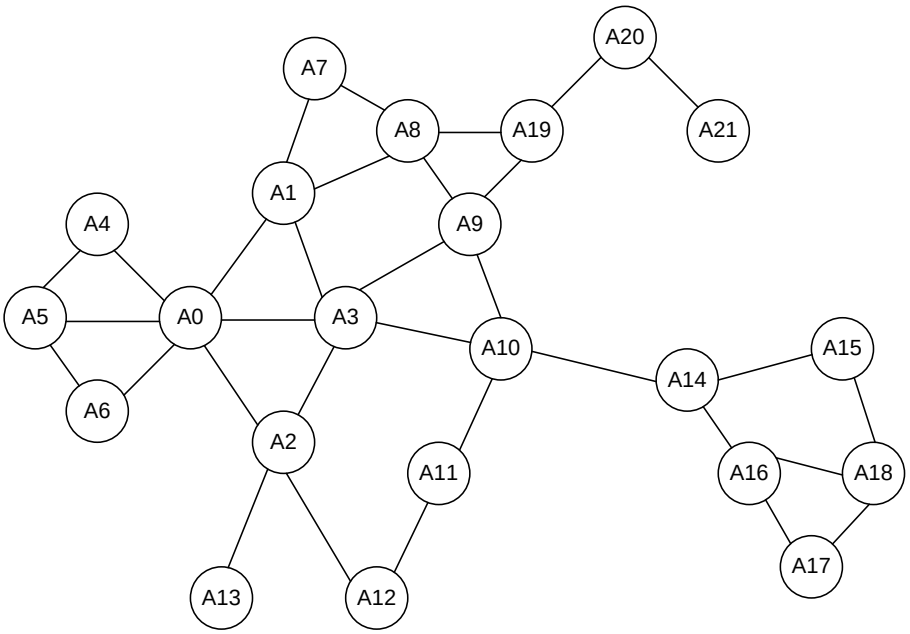


Figure 10: An example WSAN topology

this approach requires flooding the whole network and can be costly in terms of the message overhead.

Thus, LDMR uses a distributed approach for such a purpose. Our LDMR approach employs the concept of connected dominating set (CDS). As every node can reach the nodes in a CDS, the connectivity of the network can be maintained as long as CDS is connected.

We use the distributed algorithm of [17] in order to determine the CDS of a given network *G*. This identification is done only after detecting a failure.

4.3. Least Distance Movement Recovery Approach (LDMR)

LDMR exploits node mobility and the availability of non-cut-vertices in the network in order to minimize the distance that nodes collectively traveled during the recovery process. The idea is to use connectivity-uncritical nodes in restoring connectivity. The distinct feature of LDMR is the avoidance of the cascaded movement spread throughout the whole network**.**

4.3.1. Recovery Steps

The LDMR approach performs the recovery according to the following steps:

1. If an actor A_F is damaged or stops functioning, e.g. due to battery exhaustion, for example, this failure is detected by its neighbors due to the absence of the heartbeat messages which should have been sent by A_F periodically.

2. Each neighbor in step 1 and not within $r/2$ distance from A_F starts a search process looking for the nearest non cut-vertex node, where r is the communication range. This non cut-vertex is called a *candidate node* C_{ij}. The neighboring node A_{Ni} broadcasts a search message containing several entries such as failed node ID, neighbor node ID and, Time-To-Live (TTL). Then, the nearest non-cut-vertex node replies to this message with its distance to A_{Ni}. Each neighbor chooses the best candidate among received responses based on the distance D_{ij}.

3. Then, A_{Ni} sends a request message commanding C_{ij} to move to its position. Upon receiving this message, the commanded node acknowledges this message and starts moving to the specified position. This acknowledgment is necessary to avoid choosing the same non cut-vertex node by more than one neighboring node. Therefore, the commanding node A_{Ni} should wait for the acknowledgment before moving. If a node does not receive an acknowledgment, it should select the next nearest candidate and so on. It is worth mentioning that the potential candidates, including C_{ij}, will query its 1-hop and 2-hop neighbors and apply the CDS algorithm in order to know whether is not a cut-vertex, and is able to declare its candidacy and respond positively to the request.

4. Each neighbor node A_{Ni} moves toward the position of A_F until it becomes $r/2$ away of it. If one of the neighbors is within this distance, no need to move further as proven in [18]. Each candidate node C_{ij} sends movement notification message to its neighbors before sending the acknowledgment to the direct neighbor requestor A_{Ni}. This notification is essential to avoid network partitioning that may occur when multiple non cut-vertices neighbors move simultaneously. Then, if other neighbors receive similar requests to move, the nodes that believe that this movement may partition the network send panic

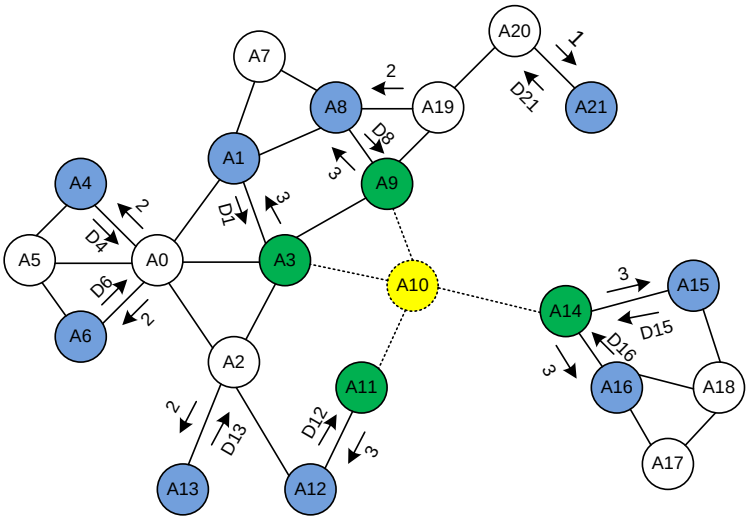
messages to prevent this movement. The next section will clarify this point more with a detailed example.

5. After the movements in step 3 and 4, the network connectivity should have been re-established.

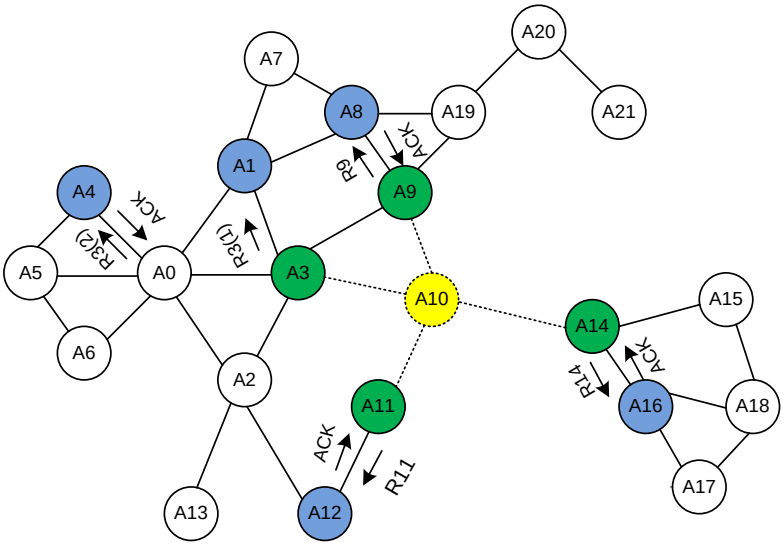
4.3.2. Detailed Example

Figure 11 and Figure 12 illustrate how the LDMR works to restore the network connectivity. Consider the network topology shown in Figure 10. As marked in Figure 11-(1), after the failure of node *A10*, its direct neighbors {*A3, A9, A11, A14*} detect the failure and start the recovery process by searching for the nearest non cut-vertex nodes. The search process may consume a lot of communication messages which is not desirable in a constrained environment such as WSAN.

Therefore, each node broadcasts a search request message and includes a Time-To-Live (TTL) parameter. In this example, we assume nodes {*A3, A9, A11, A14*} start with TTL equals 3 as shown in Figure 11-(1). Each receiving node of this message decrement the TTL value and forwards the message if the TTL is still greater than zero. If the receiving node is a non-cut-vertex, it will discard the request unless it comes from another initiator, i.e., neighbor of *A10*. In this example, nodes {*A1, A4, A6,* and *A13*} respond to the request of node *A3*, while nodes *A15* and *A16* respond to the request of *A14*. In addition, nodes *A8* and *A21* respond to the request of *A9*, and node *A12* responds to the request of *A11*. Based on the distance between the potential candidate and the neighbor node that initiates the request, the closest candidate is picked and notified. Each candidate sends an



(1)



(2)

Figure 11: LDRM steps (1) Node A10 fails and its direct neighbors begins the search process. (2) Direct neighbors send recovery requests, A3 send its second request to A4 since it does not receive ACK from A9

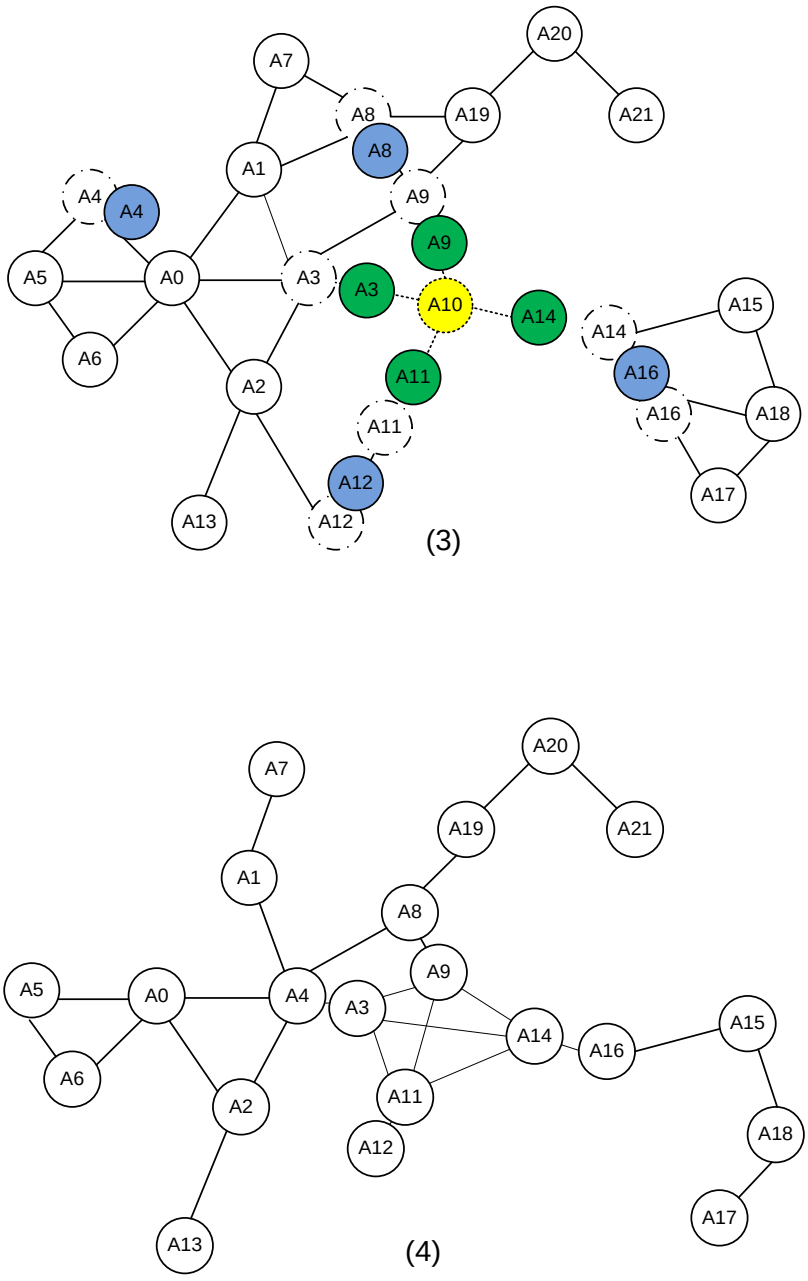


Figure 12: LDMR steps: (3) Nodes start moving (4) The final network

acknowledgment to the corresponding node and starts moving. After receiving the acknowledgments, nodes *A3, A9, A11,* and *A14* move toward the position of the failed node *A10* until they are *r*/2 away from *A10*. These movements ensure all nodes are connected to each other as shown in Figure 12.

Candidate nodes also inform their neighbors before sending the acknowledgment messages and then wait for some time to check the response of its neighbors. If no panic message is received, the candidate node C_{ij} sends the acknowledgement message to the requester. In certain scenarios, more than one neighboring non cut-vertex nodes may move simultaneously as *A8* and *A1* in this example. This situation may lead to partitioning the network again (*A7* is disconnected). Let us assume *A8* sends the notification message first, *A7* still is connected to the whole network via *A1*. Now, if *A1* sends it notification message, *A7* will send a panic message which prevents *A1* from sending the acknowledgment message. Consequently, if A_{Ni} did not receive an acknowledgment message from the nearest candidate, it picks the next nearest. In our example, it picks *A4* instead of *A1*.

Figure 13 shows the pseudo code of the LDMR approach. The recovery procedure is triggered by the missing heartbeat signal of the node neighbor. Lines 18-29 are not executed by direct neighbors since they are the ones who send search request for non-cut-vertices.

4.3.3. Algorithms Analysis

LDMR's functionality is similar to RIM [18]. Therefore, we shall compare LDMR performance to RIM. In RIM, if an actor fails (whether it is a cut-vertex or not), it's entire

```
1.      IF a node detects a failure of one of its neighbors
2.        RecovNodes=SearchNonCutVertex()
3.        While RecovNodes != Null
4.              C =min(RecovNodes)
5.              Send a Recovery command to C
6.              IF ACK is received from C
7.                 Move to the position of the failed node until
8.                 a distance of r/2
9.              ELSE
10.                delete C from RecovNodes
11.       END While
12.       IF RecoveNodes == Null
13.          Send a Recovery command to the closest neighbor
14.          Move to the position of the failed node until
15.          a distance of r/2
16.       ENDIF
17      ENDIF
18.       IF NonCutVertexRequest(ID,TTL) is received
19.          IF I am a non-cutvertex
20.             Send NonCutVertexRespnse(ID,cost)
21.          ELSE
22.             TTL = TTL – 1
23.             IF TTL != 0
24.               Forward NonCutVertexRequest(ID,TTL)
25.             ELSE
26.                Discard NonCutVertexRequest(ID,TTL)
27.             ENDIF
28.          ENDIF
29        ENDIF
30.     SearchNonCutVertex()
31.        While do
32.        Send NonCutVertexRequest(ID,TTLMAX) to each neighbor
33.        IF a Response from a NODE is received before TIMEOUT
34.          RecovNodes = NODE(S)
35.          BREAK
36.        ELSE
37.          RecovNodes = NULL
38.          BREAK
39.       END WHILE
40.     RETURN RecovNodes
```

Figure 13: LDMR pseudo code

neighbors move towards its position until they are r/2 away of each other, where r is the communication range which is assumed to be equal for all actors. This movement ensures that all neighbors are connected after they reach the new positions. Moreover, before these nodes move, they send their ranks and new positions to their neighbors. The rank of a node is the distance in hops to the failed node. For example, the direct neighbors have a rank of one. The neighbors of the direct nodes move until they are r away from those nodes. If a node is a neighbor to more than one moving node, it follows the node with higher rank, which has fewer hops to the failed node. If a node has two neighbors which have the same rank, it moves to a position where it can hear both neighbors (the intersection of two circles cantered by the two nodes and have radius of *r*).

4.3.4. Algorithm Complexity

Examining the pseudo-code shown in Figure 13, most of the running time will be spent searching for the closest non cut-vertices. The search process is $O(N.E)$ where N is the number of nodes and E is the number of edges. However, because of the node failure, this space is divided among neighbors since each neighbor will search in its partition. We also propose the use of TTL to limit the search process running time.

4.4. An Adaptive Connectivity Restoration Algorithm (ACRA)

The simulation results of LDMR have indicated its suitability for dense networks. For sparse networks, the performance advantage seems to degrade significantly. Therefore, we present an extended algorithm which is based on LDMR and performs well in both sparse and dense networks. The proposed ACRA approach is adaptive in nature. In sparse networks, ACRA simply shrinks the network inward towards the position of failed node,

similar to RIM [18], while it exploits non-cut-vertices when the network is dense. ACRA is a reactive recovery scheme that does not require any pre-failure provisioning. This eliminates the communication messages overhead when the network is healthy. The proposed approach converges faster than comparable approaches, especially for large networks.

4.4.1. ACRA Algorithm

ACRA employs node mobility to restructure the network topology and restore connectivity. To sustain network connectivity, ACRA exploits the availability of connectivity-uncritical nodes, i.e., non-cut-vertices, in the network and/or cascaded movement in order to minimize the distance that nodes collectively travel during the recovery process. The essence of the algorithm is to use connectivity-uncritical nodes in restoring connectivity if a network is dense since the node degree is high on average and some nodes can be relocated without affecting the connectivity of its neighbor nodes. If the network is sparse, most nodes become critical for strong connectivity and cascaded movement is pursued instead. In cascaded movement, each node follows its neighbor to sustain connectivity.

Upon the detection of node failure by its neighbors, each of these neighbors starts a search process to find a nearby non cut-vertex. Each neighbor then moves to the position of the failed node and commands its corresponding non-cut-vertex candidate to replace it. If a neighbor does not have a nearby non-cut-vertex, it executes cascaded movement. To simplify the presentation of ACRA, we will refer to each of these neighbors as a *searching node* and the corresponding non-cut-vertex as a *candidate node*. For example,

if node F fails in Figure 14, the neighbors of F (A_1, A_2, and A_3) move towards the position of F until they are $r/2$ away from F where r is the communication range. This movement ensures the connectivity among these nodes. To sustain connectivity, node A_5 moves to the new position of node A_1 while nodes A_4 and A_7 move to the new position of node A_3. A candidate node will stop moving when it reconnects with the corresponding searching nodes. The existence of a non-cut-vertex close to node *A3* (i.e., *A6*) will spare the branch of node A_8 from moving. After moving node A_2 to F, node A_6 will replace A_3 and therefore sustain connectivity to node A_8. In this example, cascaded movement is applied by nodes $A_1$and A_3while for node A_2 a non-cut-vertex is utilized.

Obviously, adjusting the recovery procedure based on the node density can be very subjective and cannot thus be generally applied. Therefore, we use the recovery overhead as a criterion for judging how the failure will be mitigated, as we explain next.

4.4.2. Cut-off and switch metric

As pointed out above, ACRA uses the cost of the recovery as a criterion for selecting the connectivity restoration scheme. The total traveled distance is used as a cost metric for the recovery. The cost of moving a candidate node increases when it needs to travel a long distance in order to replace the searching node. Therefore, for a certain deployment area, the number of hops between the searching node and the nearest non-cut-vertex can be used to measure the cost. For example, if A_6 does not exist in Figure 14, the nearest non-cut-vertex that will replace A_2 is A_{12}. However, since A_{12} is three hops away from A_2, the cost of moving A_{12} is high. On the other hand, as a network gets denser, it is most

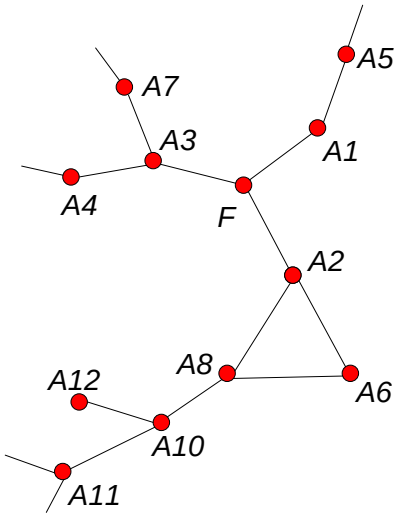


Figure 14: A WSAN example

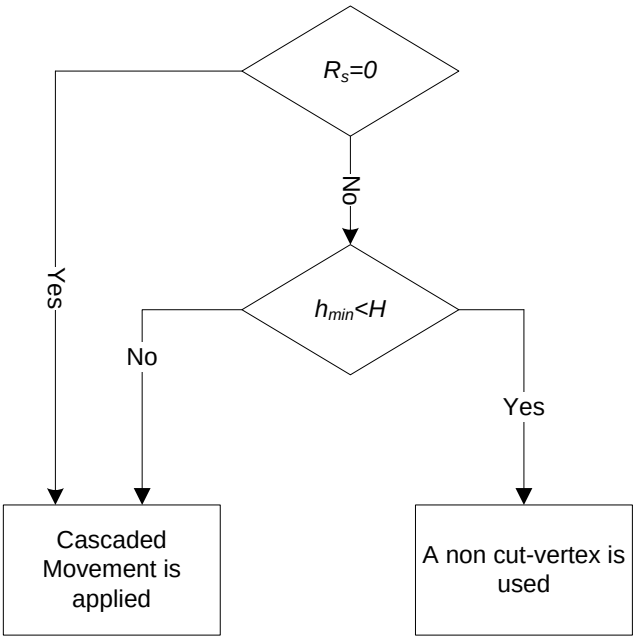


Figure 15: Algorithm selection procedure

likely that a non-cut-vertex is located in fewer numbers of hops. Therefore, the number of hops can also be used as a measure of the density and scale of the network. When the number of hops needed to find a non-cut-vertex is high, the network is likely to have few nodes, and when the number of hops is small, the network is likely to be dense. When the candidate node is far, it is better to pursue cascaded relocation in order to balance the recovery overhead cost among multiple nodes and speed up the recovery since the motion of nodes can overlap in time.

The number of hops h to the non-cut-vertex node is used to choose which approach is executed. The searching node puts out a call for help by sending a broadcast message with certain time-to-live attribute "TTL" to limit the number of hops that the message reaches. If a searching node does not receive any response from potential candidates within a predetermined time duration ($R_s=0$), cascaded movement is applied directly. If it receives multiple responses, it chooses the node with minimum hops (h_{min}). If two nodes have the same h_{min}, the node that has a shorter distance is chosen. If $h_{min} > H$, a cascaded movement is applied, where H is a predetermined threshold used to limit the number of hops. Otherwise, the node with h_{min} to the searching node and has a shorter distance is picked. Figure 15 illustrates the algorithm selection procedure.

4.4.3. Cascaded Movement

ACRA employs a variant of the cascaded movement procedure used in [18] in order to enhance the performance and speed up the convergence of the recovery process. If a node moves, it sends a movement notification with its new position information to its neighbors. If a node receives a movement notification, it relocates to the new position of

the notifying neighbor until both nodes are connected. If a node receives a movement notification from two moving nodes, it checks the searching nodes that initiate the movement in the two requests. This information is included in the notification message. If both requests come from the same searching node, the receiving node moves to a location where it can hears both moving nodes. The new position in this case is the intersection of the two circles cantered at the two new positions of the moving nodes. However, if two notifying messages come from two different searching nodes, the receiving node will follow the node that has the closer new position. A detailed example will be given later to illustrate how this movement is performed.

As we will explain later in the detailed steps of ACRA, each searching node needs to estimate its recovery cost. In the case of cascaded movement, it is difficult for the searching node to precisely determine the exact cost of cascaded movement from local information. Therefore, in ACRA each node can estimate the cost by applying equation (4.3), where R_s is the number of responses received by the searching node and *TTL* is the initial time to live value that was used by the searching nodes. TTL also represents the maximum hops among all responses. $d_{FS}$is the distance that a searching node needs to move in order to be *r/2* away from *F*

$$Cost_{Cas} = d_{FS} * TTL * R_s \tag{4.3}$$

The resulting value by the above equation gives an estimate to the distance that may be travelled by other network nodes that follow the searching nodes in order to sustain network connectivity. If R_s is high, this indicates that the network is dense and therefore cascaded movement is not a preferred choice.

4.4.4. Detailed Steps

The following are the detailed steps that describe how ACRA restores the connectivity of a partitioned network:

1- Failure Detection*:* If an actor A_F is damaged or stops functioning, e.g., due to battery exhaustion, the transmission of heartbeat messages, which should have been sent by A_F periodically, will cease. The absence of the heartbeat messages will be is interpreted by each of A_F's neighbors, AN_i, as an indication of its failure. The recovery procedure will be executed regardless whether A_F is a critical node, i.e., a cut-vertex or not. In fact, ACRA does not assume that a node collects state information to assess the criticality of another node in the network until a failure takes place.

2- Searching for non-cut-vertices*:* Each neighbor in step 1 and not within r/2 distance from A_F starts a search process looking for the nearest non-cut-vertex node, where r is the communication range. Each searching node broadcasts a search request containing several entities such as its own ID, the failed node ID, Time-To-Live (TTL) which denotes the maximum number of hops the search spans. Each node receives such request sends a feedback if it is a non-cut-vertex node. It also decrements the TTL value and forward the request if it does not reach zero.

If two searching nodes are neighbors, they will receive requests from each other. In this case, we can let each node discard the searching request from the other node or accept and forward the request. The advantage of the first choice is to have less communication messages. However, a nearby non-cut-vertex that can be reached by other neighbors can be missed. Therefore, we apply the second option in ACRA.

It is worth mentioning that potential candidates will query their 1-hop neighbors and apply the CDS algorithm of [17] in order to know whether it is a cut-vertex, and it is able to declare its candidacy and respond positively to the request.

***3*- Receiving search responses:** After a searching node A_{Ni} receives a response from a nearby non-cut-vertex node, it stores this information to compare it with other responses. Each response is associated with a cost Dij. The cost is the distance between the searching node A_{Ni} and the non-cut-vertex node j. The number of hops (hij) between A_{Ni} and j is noted in the response packet. If the searching node A_{Ni} does not receive any response within a pre-configured time period, it goes forward to cascade relocation by sending a recovery request to the nearest neighbor and moving to the position of the failed node AF.

4- Choosing the recovery procedure and Broadcasting its Cost: Upon receiving the responses in the last step, each searching node decides to choose between cascaded movement and utilizing a nearby non-cut-vertex based on the criteria earlier and depicted in Figure 15. After a searching node estimates its recovery cost, it broadcasts such a cost to its neighbors. The cost of cascaded movement is estimated based on equation (4.3). If a node receives a cost that is lower than its own cost, it stops executing ACRA. If a node does not receive such message or it receives messages that carry a higher cost, it proceeds to the next step. We will prove later that such behavior will not affect the connectivity of the resulting network.

When the network is dense, it is expected that the average number of neighbors of each node in the network is high. Therefore, this step is very essential to lower the cost of ACRA in large networks.

5- Starting the recovery process: If utilizing a non-cut-vertex, steps 6 and 7 are executed. Otherwise, the searching node ANi sends a notifying message to its neighbors and moves to the position of AF until it is r/2 away. The neighbors of the ANi will perform a cascaded movement as explained above.

6- If utilizing a non-cut-vertex is chosen in step 4, ANi sends a request message commanding Cij (the best candidate selected in step 4) to move to its position. Upon receiving this message, the commanded node acknowledges this message and starts moving to the specified position. This acknowledgment is necessary to avoid choosing the same non cut-vertex node by two searching nodes. Therefore, the commanding node ANi should wait for the acknowledgment before moving. If a node does not receive an acknowledgment, it should select the next nearest candidate and so on).

7- Best Candidate Movement: The best candidate node Cij sends a movement notification message to its neighbors before sending the acknowledgment to the direct neighbor ANi. This notification is essential to avoid network partioning that may occur when multiple connectivity-uncritical neighbors move simultaneously. Then, if other neighbors receive similar requests to move, the nodes that believe this movement may partition the network, they send panic messages to prevent this movement. This will be illustrated with a detailed example in Section 4.4.5.1. If no panic messages are received, Cij sends an acknowledgment message and starts moving to replace A_{Ni}. After receiving

the acknowledgment, A_{Ni} moves toward the position of the failed node AF until it becomes r/2 units away. After these movements the network connectivity should have been re-established. If a searching node does not receive an acknowledgment, it should select the next nearest candidate in it responses list that has a number of hops less than H. If it does not have such a node or does not have more responses, the searching node executes the cascaded movement procedure.

4.4.5. Pseudo code and detailed example

In this section, we will explain how ACRA works through an example topology of 40 nodes. We will also summarize the algorithm using a high level pseudo code.

4.4.5.1. Detailed Example

Let us assume that node A_4 fails in the example depicted in Figure 16. The failure will be detected by the direct neighbors A_{26}, A_{38}, A_7, and A_{14}, which become *searching nodes*. Each *searching node* looks for a nearby non-cut-vertex. As shown in the same figure, nodes A_{26}, A_{38}, A_7, and A_{14} start searching by sending requests to their neighbors. To localize the search process, each request sets its time-to-live to two, meaning the request will not go beyond 2-hops. Each *searching node* gets a set of responses. Node A_{26} receives responses from nodes A_{15} and A_{24}, node A_{38} gets responses from nodes A_{37} and A_{23}, node A_7 hears back from nodes A_8 and $A37$, and node A_{14} receives responses from nodes A_{17}, A_{23} and A_{30} . Figure 16 and Figure 17 show the search process hop by hop. In these figures, S(A_{26}, 2) indicates a Search request from A_{26} and the current value of TTL is 2. R(A_{15}, A_{26}) indicates a response to the search request sent from A_{15} to A_{26}.

Let us assume in this example that each searching node orders a non-cut-vertex node if it 1-hop away. Otherwise, the searching node will pursue cascaded movement. According to the cut-off criterion for ACRA, nodes A_{26}, A_{38}, and A_{14} will utilize a nearby non-cut-vertex node while node A_7 executes cascaded movement as a next step in the recovery process. Although node A_7 receives two responses from A_8 and A_{37}, both nodes are two hops away. Both responses that have been received by node A_{38} are coming from 1-hop nodes. In this example, A_{38} will choose the nearest node which is node A_{23}.

Now, all searching nodes will broadcast their estimated moving costs to their neighbors. The moving costs in case of A38, A26, and A14 are the distance between the searching node and its selected non cut-vertex node. In case of cascaded movement, the moving cost is calculated as in (equation 4.3). If a searching node receives a message that carries less that its own cost, it will stop the process leaving the recovery to other neighbors. Based on our example, A7 has a higher cost than A38. Both A26 and A14 do not have any searching nodes which are neighbors. This step is very essential in a dense and large network to lower the cost of the adaptive approach.

The next step in the recovery is to notify the selected non-cut-vertices if any. Otherwise, the first step in cascaded motion will be executed. According to ACRA, nodes A_{38}, A_{14}, and A_{26} will send recovery requests to nodes A_{37}, A_{17}, and A_{24}, respectively. Upon receiving these requests, each non-cut-vertex node will send a movement notification to its neighbors. If no panic messages are received, each node will send an acknowledgment

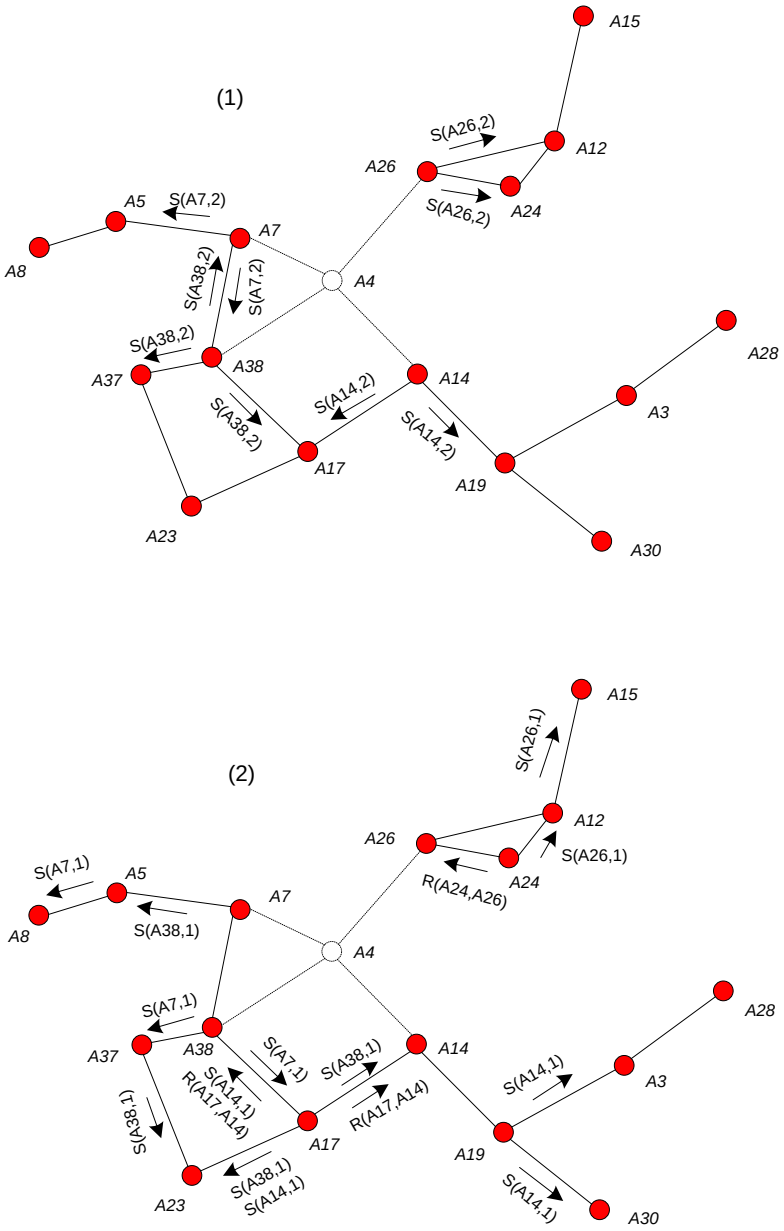


Figure 16: A detailed example of the proposed adaptive approach: Nodes A7, A38, A14, and A26 search for non-cut-vertices (1-2)

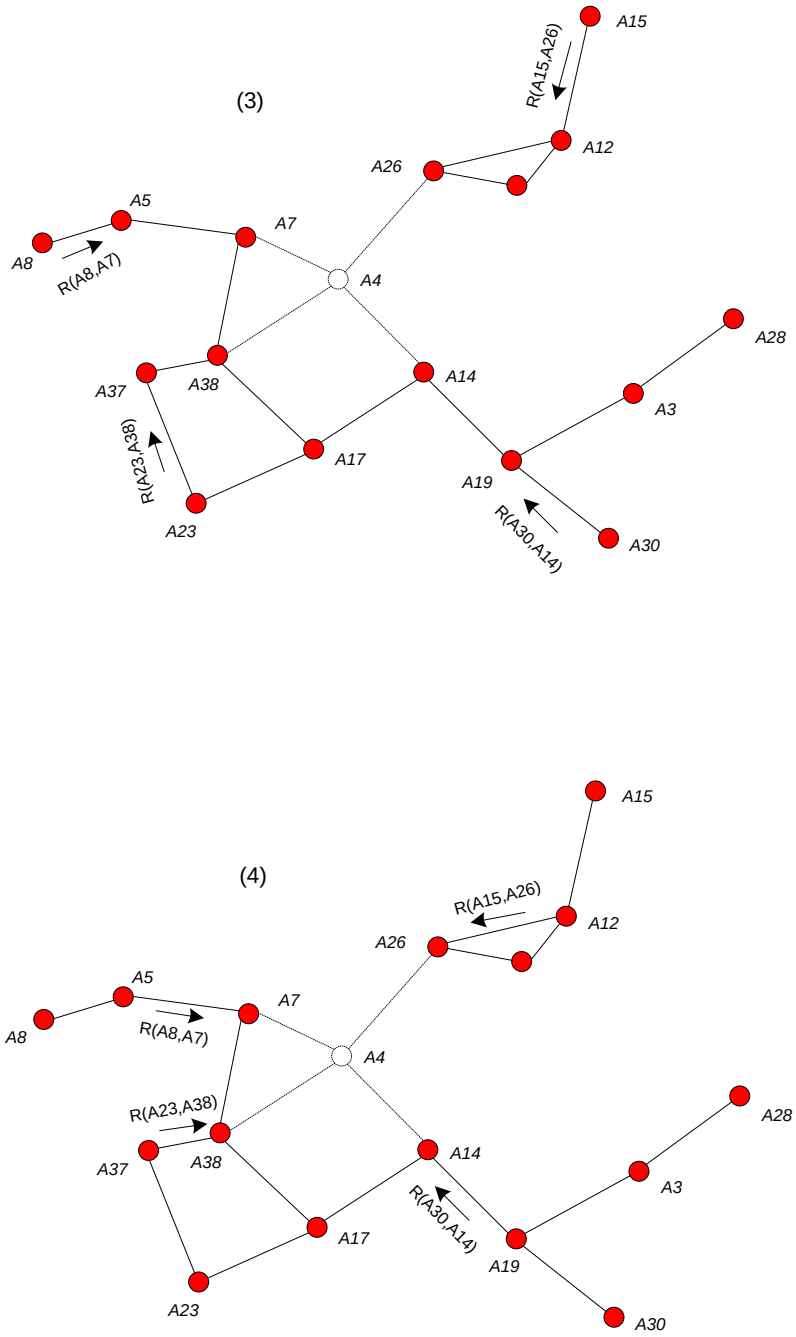


Figure 17: A detailed example of the proposed adaptive approach: Nodes A7, A38, A14, and A26 search for non-cut-vertices (3-4)

message to the corresponding *searching node*. A node sends a panic message if it is likely to be disconnected from the rest of the network. In this example, node A_{23} will receive two movement notifications from nodes A_{17} and A_{37}. Assume that it receives the notification first from node A_{37}. Since node A_{23} is still connected to the rest of the network through node A_{17}, it will not send a panic message to node A_{37}. However, node A_{23} will send a panic message to node A_{17}. Since node A_{37} does not receive a panic message, it will send acknowledgement to the searching node A_{38} informing it is ready.

Since node A_{17} will not send an acknowledgment,A_{14} will select the next nearest non-cut-vertex that has two responses from node A_{23} and A_{30}. However, both nodes are two hops away; therefore A_{14} will execute cascaded movement instead. Node A_7 is also receiving two responses from 2-hops non-cut-vertices neighbors, namely,A_8 and A_{37}. Therefore A_7 will also employ cascaded movement.

To restore the connectivity of the network, nodes A_{38}, A_{26}, and A_{14} move toward the position of the failed node A_4 until they are *r/2* units away. A_{24} will replace A_{26} and A_{37} will replace A_{38}. Since A_{14} will apply cascaded movement, A_{19} moves toward the new position of A_{14} until it is *r* units away. Both nodes A_{30} and A_3 will follow A_{19} until they are connected with it (*r units* away from the position of node A_{19}). Finally, node A_{28} will move to the new position of A_3. These recovery steps are shown in Figure 18 and Figure 19. In these figures, Re(A_{26}, A_{24}) indicates a recovery request sent by A_{26} to A_{24} where A(A_{24},A_{26}) indicates an acknowledgment sent by A_{24} to A_{26}.

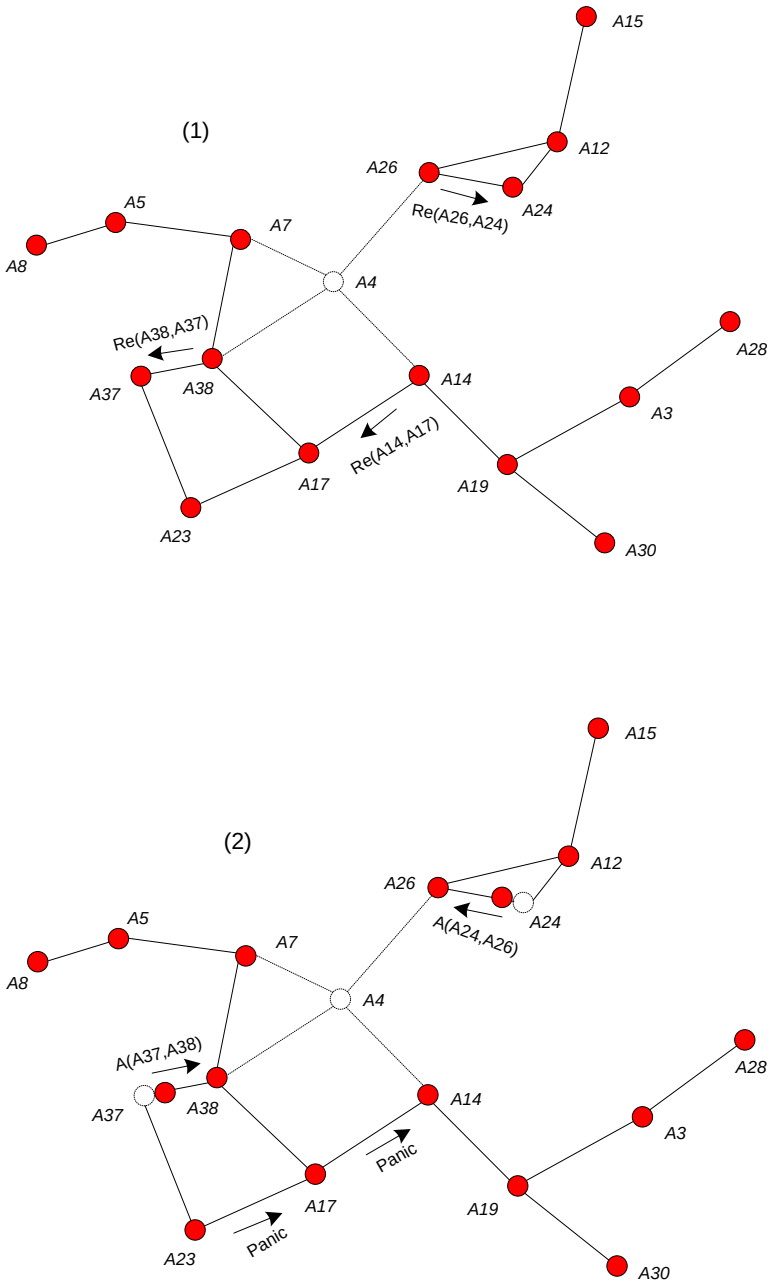


Figure 18: The searching nodes continue the recovery process ((1) and (2))

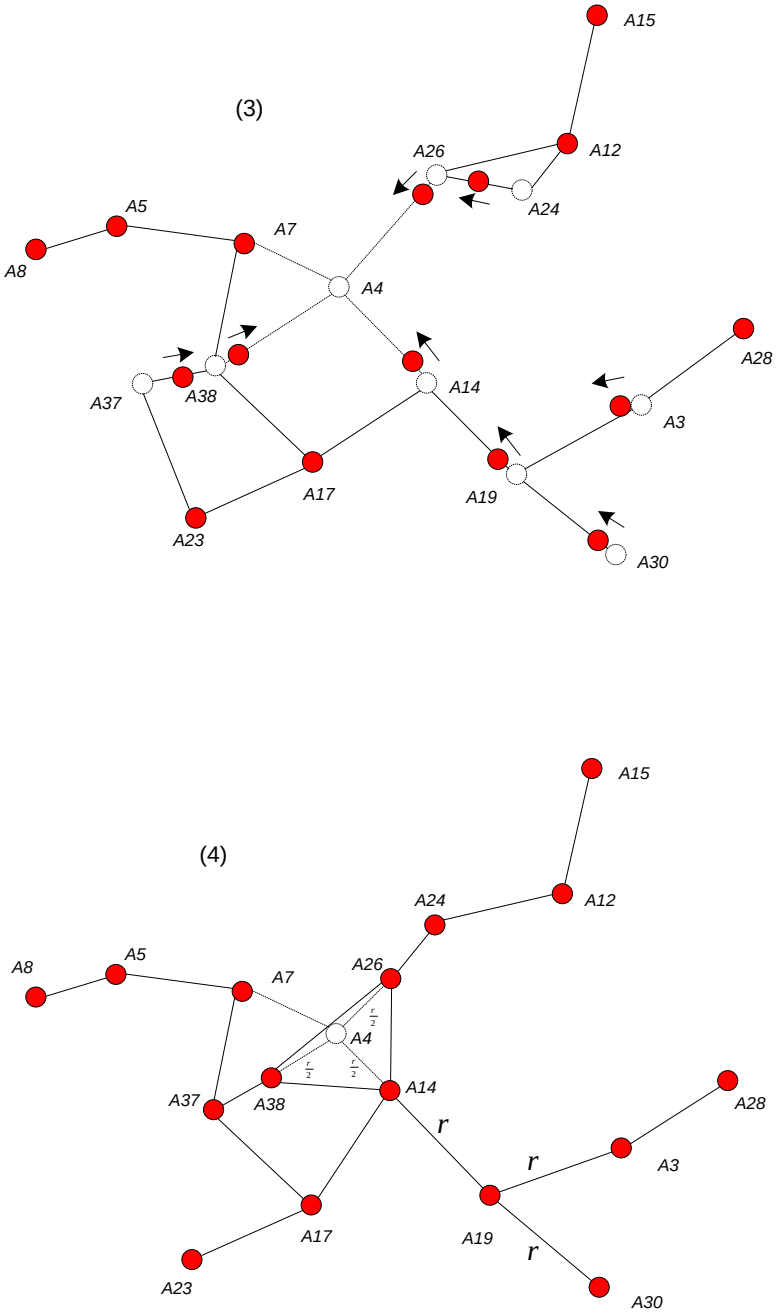


Figure 19: The searching nodes continue the recovery process ((3) and (4))

4.4.5.2. Pseudo Code

For the sake of clarity, we split the code into two figures: Figure 20 shows the steps that will be made on the searching nodes and Figure 21 shows the code that will be executed on other nodes.

***Searching nodes*:** After detecting the failure (line 1), the searching nodes execute *SearchNonCutVertex* (line 2) to look for the nearest non-cut-vertices. The details of this function are listed on lines 31-42. The search process is primly configured to the maximum time-to-live value which prevents propagating of search messages through the whole network. After receiving all responses, *EstimateCost* is executed to determine the recovery scheme, cost of the movement, and the candidate node in case of choosing a non-cut-vertex node for recovery. Lines 44-60 list the steps for this function. A searching node broadcasts its estimated cost to its neighbors (line 4). Any searching node that receives an estimated cost that is less than its own, it abandons the recovery process (lines 5-7). Otherwise, it executes the selected approach based on the output of EstimateCost. If the chosen candidate node does not send an acknowledgment, the searching node deletes such candidate from eligible list (RecovNodes) and re-computes the cost (lines 9-22). If the acknowledgment is received, the searching node moves to the position of the failed node until it is r/2 away. The same movement is applied in case of cascaded movement and the searching node notifies its neighbors before the movement about its new position (lines 24-28).

Other nodes: Figure 21 shows the part of the algorithm that will be executed by other nodes. If a node in the network receives a request from a searching node

```
. IF a node N detects a failure of one of its neighbors F
2.      RecovNodes=SearchNonCutVertex()
3.      (A,Cost,C) = EstimateCost(RecovNodes)
4.      Node N Broadcasts Cost to its neighbors
5.      IF N receives a broadcast that has a cost that
6.      less than COST
7.              Done
8.      ELSE
9.              While RecovNodes != Null
10.                     IF A = 'NonCut'
11.                             SendRecovery(C)
12.                             IF Ack(C) is received before TIMEOUT
13.                                     Move to the position of the failed node F until
14.                                     a distance of r/2
15.                                     DONE
16.                             ELSE
17.                                     delete C from RecovNodes
18.                                     (A,Cost,C) = EstimateCost(RecovNodes)
19.                             ENDIF
20.                     ELSE
21.                             Break
22.                     ENDF
23.             END While
24.             IF A = ‘Cascade'
25.                     Notify neighbors of the new position and move to the
26.                     position of the failed node F until a distance of r/2
27.                     Done
28.             ENDIF
29.     ENDIF
30.
31. SearchNonCutVertex()
32.     While do
33.             Send NonCutVertexRequest(ID,TTL) to each neighbor
34.             IF a Response from a NODE is received before TIMEOUT
35.                     RecovNodes = NODE(S)
36.                     BREAK
37.             ELSE
38.                     RecovNodes = NULL
39.                     BREAK
40.             ENDIF
41.     END WHILE
42.     RETURN RecovNodes
43.
44. EstimateCost(RecovNodes)
45.     IF RecoveNode == NULL
46.             A = 'Cascade'
47.             D = distance(F,N)
48.             Cost = D * TTL
49.     ELSE
50.             C = min(RecovNodes);
51.     IF hops(N,C) < MaxHops
52.             A = 'NonCut'
53.             Cost = Distance(N,C)
54.     Else
55.             A = 'Cascade'
56.             D = Distance(F,N) - r
57.             Cost = D * TTL * Length(RecovNodes)
58.             C = NULL
59.     ENDIF
60.     Return A,Cost,C
```

Figure 20: The adaptive approach pseudo code (searching nodes)

```
1. IF C receives NonCutVertexRequest(ID,TTL)
2.      Do NonCutVertexCheck
3.      IF I am a non-cutvertex
4.              Send NonCutVertexRespnse(ID,cost)
5.      ELSE
6.              TTL = TTL – 1
7.              IF TTL != 0
8.                      Forward NonCutVertexRequest(ID,TTL)
9.              ELSE
10.                     Discard NonCutVertexRequest(ID,TTL)
11.             ENDIF
12.     ENDIF
13. ELSEIF C receives SendRecovery(N,C)
14.     Send NonCutRecoveryMove to its neighbors
15.     IF C do not receives a Panic message before TIMEOUT
16.             Send Ack(C,N)
17.     ENDIF
18. ELSEIF C receives NonCutRecoveryMove and its not
19. connected to other nodes
20.     Send Panic to the moving node
21. ELSEIF C receives CascadeRecoveryMove(N,L)
22.     Add (N,L) entry to the received notifications NotifyCascade
23.     IF notifications are received from all neighbors
24.             MinLevels = Nodes that has minimum L
25.             Determin the node J in MinLevels that
26.             gives the shoterst distance to
27.             its new location
28.             MinNodes = All nodes in MinLevels
29.             that has the same target of J
30.             IF length(MinNodes) == 1
31.                     Send CascadeRecoveryMove(C,L+1)        to all neighbors
32.                     ollow MinNodes until r away from it
33.             ELSEIF length(MinNodes) = 2
34.                     Move to the intersection point of
35.                     Circle(MinNodes(1),r) and
36.                     Circle(MinNodes(2),r)
37.             ELSE
38.                     Move to a point where can hear all MinNodes
39.             ENDIF
40.     ENDIF
41. ENDIF
```

Figure 21: The adaptive approach pseudo code (other nodes)

(*SearchNonCutVertex*), it first checks whether it is a non-cut-vertex. If it is, it will send a response to the searching node and forward the request to its neighbors. If it is a cut-vertex, it will only forward the search request. The forwarding to other nodes is done after checking the time-to-live value (line 1-12). If a non-cut-vertex node receives a recovery request from a searching node (*SendRecovery*), it sends a movement notification (*NonCutRecoveryMove*) to its neighbors. If the node does not receive a panic message, it will send an acknowledgment to the requested searching node (lines 13-17). A node that receives a movement notification from a non-cut-vertex neighbor will not respond if it is still connected to the network by other nodes. Otherwise, it sends a panic message to the moving node (lines 18-20). Lines (21-39) detail how a node is moving if it receives a cascaded movement notification (*CascadeRecovMove*). If the node receives one notification, it will directly move to the position of the notifying neighbor until it is r units away. If the node receives two or more notifications, it checks the searching node (N) that initiates the movement in each notification request. If both have the same initiator, it will follow both by moving to the intersection point of the two circles centered by the new positions of the two moving neighbors. If two requests come from different searching nodes, the node will follow the nearest neighbor. In all cases, the node notifies its neighbors about its new position before it moves.

4.4.6. Algorithm Analysis

In this section, we shall present several theorems to analyze the proposed adaptive algorithm.

Theorem 4.1: the total travelled distance for the associated movement algorithms is bounded by

For cascaded movement

$$d_{Cas} \leq \frac{r}{2}\sum_{i=1}^{N} h_i \tag{4.2}$$

For non-cut-vertex movement

$$d_{ncut} \leq \left(\frac{r}{2} * N\right) + \sum_{i=1}^{N} d_{ij} \tag{4.3}$$

Where r, N, h_i, d_{ij}, are the communication range, number of nodes in the network, number of hops from each neighbor to the leaf node (network diameter) assuming all neighbors have disjoint paths to leaf nodes, and the distance between the corresponding non-cut-vertex node and the position of the nominating neighbor node, respectively.

Proof: For the cascaded movement approach considers the one-dimension actor network depicted in Figure 22-a. The worst-case scenario for cascaded movement occurs when all nodes are *r* away from each other. Therefore, if A_3 (red circle) fails, all neighbors and their children move towards A_5. The total distance travelled by each node is the number of hope (h_i) multiplies by *r/2*. Hence, the total travelled distance will not exceed $\frac{r}{2}\sum_{i=1}^{N} h_i$

and its complexity is *O*(*N.E*), where *E* is the number of edges in the network. The number of hops to leaf nodes increases as the number of nodes increases. Therefore, as the network size grows, the performance of cascaded movement is expected to degrade.

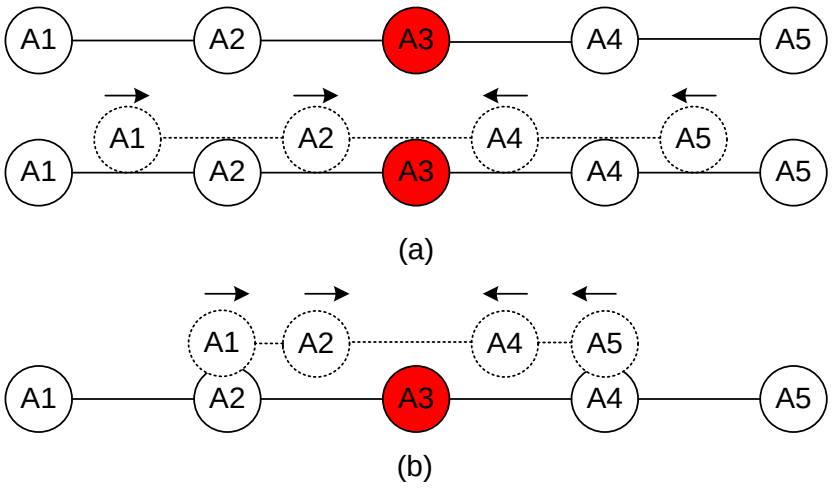


Figure 22: (a) RIM and (b) LDRM restoration scenarios that reflects the worst case travel overhead

For the non-cut-vertex based procedure, the worst-case scenario occurs when all non-cut-vertices that may replace the neighbor of the failed node are leaf nodes as illustrated in Figure 22-b. Therefore, the total travelled distance is composed of two parts: the travelled distance by the neighbors (A_{Ni}), and the travelled distance by the leaf node (C_{ij}). □

Theorem 4.2: If two or more searching nodes are neighbors, the movement of one searching node to replace the failed node is sufficient for restoring the network connectivity.

Proof: Let us take the worst-case where the distance between the two searching nodes and between each of them and the failed node is r as shown in Figure 23. After the movement of node A, the new distance between A and B is calculated as follows:

$$\overline{B_{old}A_{new}} = \sqrt{r^2 - \frac{r^2}{4}} = \frac{1}{2}\sqrt{3}r \leq r \tag{4.4}$$

Therefore, the two nodes are still connected. If there are more than two nodes, by definition the extra nodes have to be inside the triangle A_{old}-F-B_{old}s o that they are connected to F. □

Theorem 4.3: The maximum distance that can be travelled by a node in cascaded movement is $\frac{\sqrt{5}}{2}r$.

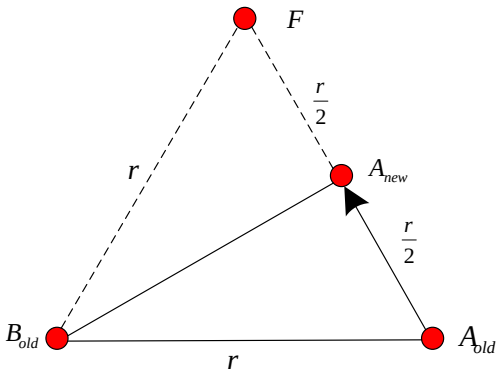


Figure 23: illustration of moving of node A is enough to maintain connectivity

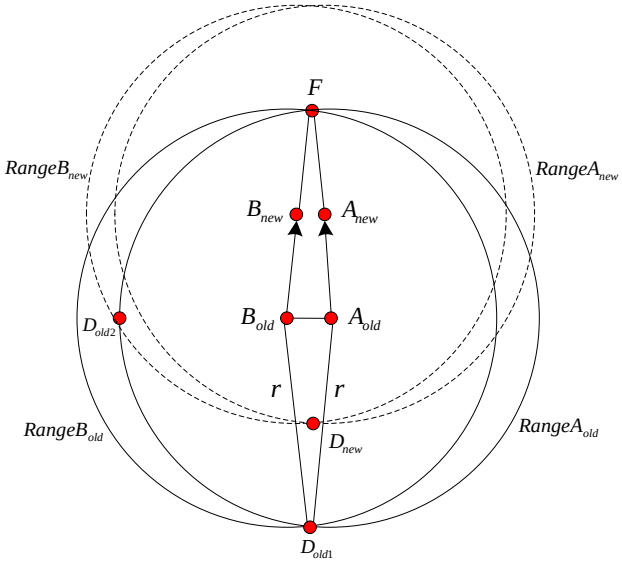


Figure 24: Cascaded movement maximum movement

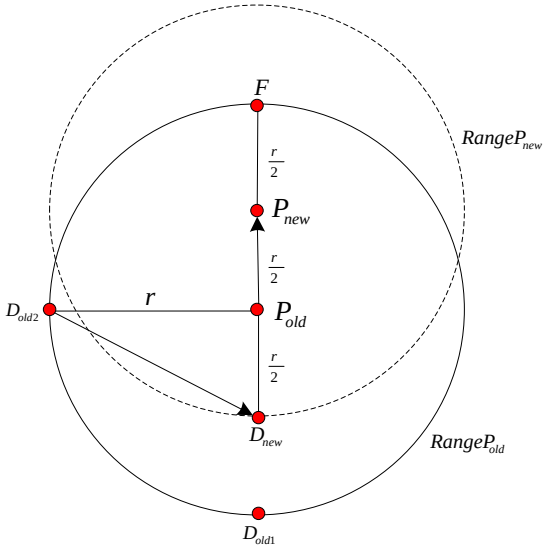


Figure 25: One point assumption to prove the maximum RIM movement

Proof: it has been proven in [18] that the maximum distance a node may travel in cascaded movement is *r/2*. This is obvious for the neighbors of the failed node (searching nodes) since the maximum distance between the failed node and any of its neighbors is *r* before the failure. After the failure, each neighbor moves to the direction of the failed node and stops at a point that *r/2* away from the position of the failed node. Therefore, the distance between the furthest point and the stopping point is *r/2*. For other nodes, the stopping point of a node is determined based on how many nodes a node will follow. If a node will follow one node, it is obvious that the maximum distance is *r/2* since this case is similar to the neighbors of the failed node.

However, if a node will follow two nodes and more, careful consideration is needed to find out what is the maximum distance a node will move. In [18], the authors consider the case shown in Figure 24 to prove that the maximum distance is *r/2* if a node follows two nodes. However, the chosen old position of node *D* (D_{old1}) is not the worst possible case scenario. As *A* and *B* become closer, the distance between D_{old2} and $D_{new}$is increased while the distance between D_{new} and D_{old1} does not change. The worst-case scenario is when *A* and *B* are collocated.

Since the distance between both nodes is very small, we ignore this distance in our calculation and consider both nodes as one point as shown in Figure 25. It has been proven in [18] that the distance between D_{new} and $D_{old1}$when *A* and *B* are collocated is *r/2*. Since the distance between P_{old} and D_{old1} is *r*, the distance between D_{new} and P_{old} is *r/2* as shown in Figure 25. The maximum movement based on the worst possible location is calculated as the following

$$\overline{D_{old2}D_{new}} = \sqrt{r^2 + \frac{r2}{4}} = \frac{\sqrt{5}r}{2} \quad (4.5)$$

Theorem 4.4: If the failed node has one or multiple groups of searching nodes (neighbors), the movement of only one node from each group will maintain connectivity.

Proof: We have two cases here. Case-I is where all searching nodes are neighbors. According to theorem 4.2, the movement of one node is enough to maintain connectivity. Case-II is where there is more than one group of nodes. In such a case, we have two scenarios depending on the moving node and whether it is a shared neighbor among all groups or not. For the first scenario (i.e. the moving node is shared among the groups), the node will be connected to all nodes in different groups as shown in Figure 26. For the second scenario, each node will be connected to its own group and it will be connected also to the other moving node since each of them will move until they are *r/2* from the failed node. In Figure 27, node *A* is the moving node from group (*A,B*) nodes, and node *C* is the moving node from group (*B,C*) nodes. According to theorem 1, both *A* and *C* are connected to *B*. For the other group of searching nodes (*D,E*) which do not have any shared nodes), it will be connected to other nodes through its moving node (*D* in this case) Therefore, the overall topology is connected after all movements. □

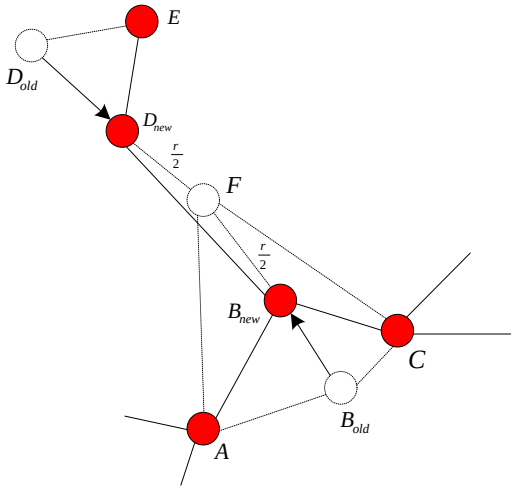


Figure 26: The moving node B is a shared node between A and C

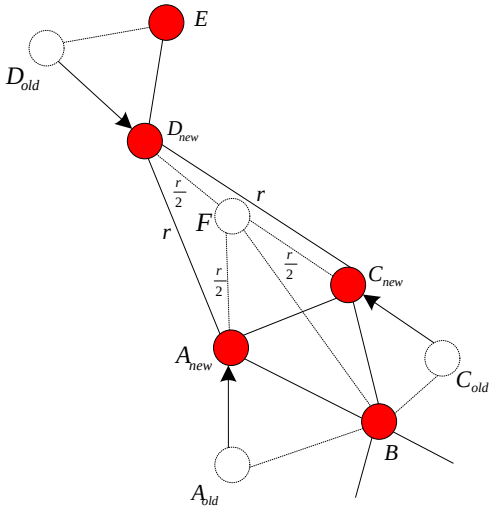


Figure 27: illustration example for theorem 4 where the moving node is not shared

CHAPTER 5

DISTRIBUTED RECOVERY FROM SIMULTANEOUS MULTI-NODE FAILURE

5.1. Overview

LDMR and ACRA are designed to restore the network connectivity for a single failure at a time and are not proven to handle simultaneous failures. Therefore, we developed a new recovery distributed scheme called Simultaneous Failures Recovery Approach (SFRA). SFRA pursues combined proactive and reactive strategies by creating a recovery tree from the original network. SFRA assumes a node is pre-assigned as a root at the time of the network deployment. A breadth first search tree is constructed and the level of the individual nodes is used for defining their role in the recovery. After a failure, one of the children of the failed parent in the recovery tree moves to the position of its parent and becomes a leader of other nodes in the sub-network in order to establish its connectivity with the rest of the network. To lower the travel overhead, each node is assigned a recovery weight based on the number of hops to its closest leaf node in the recovery tree. Moreover, to enable the connectivity of the network, some nodes are chosen to be clusters during the ranking phase. These nodes serve as gateways to nearby nodes. The leader of a disconnected sub-network moves to its cluster if it is still not connected after recovering its parent.

5.2. System Assumptions

- The network nodes are randomly deployed and we assume that nodes are connected to each other after deployment to form a connected network.
- One of the deployed nodes has to be configured as root.
- After the deployment and during the operation of the network, we assumed a major event causes multiple nodes to fail simultaneously. However, we assumed that only a small percentage of the node population fails and it makes sense to restore the network operation at a degraded level.
- All nodes have the same communication range.

5.3. Notation

- *A*, *B*, *C*, *D*, etc., are network active actors.
- N_A, C_A and S_A are A's neighbors in the original network, and *A*'s children and *A*'s sub-tree in the recovery tree, respectively.
- F_i is failed #i.
- K_A, R_A and RW_A denote the rank, the ranker (i.e., parent in the recovery tree) and the recovery weight of A, respectively.
- A_{RN} is the node with minimum recovery weight in C_A.
- Mid($\overline{AF_i}$) is the point on $\overline{AF_i}$ and has a distance of ½ *r* from F_i where *r* is the communication range.

- $D_{A,B}$ denotes the Euclidian distance between *A* and *B*.

- $RC(F_i)$ denotes a Replacement Candidate for F_i.

5.4. SFRA Algorithm

After deployment, nodes collect the information needed to work in a coordinated manner for restoring connectivity after a failure. In SFRA, we assume that one of the nodes is pre-assigned as a *root* of the network. Immediately after the deployment, the root initiates a rank assignment phase. The purpose of this phase is to assign a rank to each node and construct a *recovery tree*. The recovery tree is used to coordinate connectivity restoration as we will be explained later. The rank assignment is used to calculate the weight that determines the scope of node's participation in the recovery. The following subsections explain each phase in detail.

5.4.1. Rank Assignment

The rank of a node is the number of shortest hops to the root (*R*) of the network. In case of *R*, $K_R = 0$ and $R_R = R$. Once the network nodes are deployed and the network is ready for operation, the root of the network "*R*" runs breadth first search to assign ranks to the individual nodes, by sending a message to its neighbors which forward their reachable nodes and so on. The rank assignment message contains its ID, its rank, and the ID of the ranker (i, K_i, R_i). Let us assume that a node *j* receives this message from a node *i*. The following is executed by *j* in order to compute K_j and R_j:

```
IF Kj > Ki + 1

Kj = Ki + 1;
```

```
Rj = i;

ELSEIF rankerID = j

Add i to Cj

ENDIF
```

Since there are probably multiple paths between the root and node *j*. Node *j* may receive multiple rank assignment messages from nodes other than *i*, but it will not update its current rank unless the above condition is satisfied. If it is not satisfied and R_i is *j* itself, *j* adds the sender node *i* to the list of ranked nodes (C_j). For example, in Figure 28-(a), assuming $K_h > K_i$, *j* is ranked by *i*. Node *j* will send a rank assignment message to its neighbors. If node *i* receives this message from *j*, it adds *j* to C_i. Nodes *k* and *h* will ignore the rank assignment message from *j*. Node *m* updates its rank and sets $R_m = j$. A recovery tree is constructed at the end of this phase as shown in Figure 28-(b). The tree is used in the recovery algorithm as discussed later.

5.4.2. Weight Computing

We consider two types of weight to be computed: *clustering weight (cw)* and *recovery weight (rw)*. The clustering weight is used to decide which nodes are chosen to be cluster heads while the recovery weight determines which nodes are to move for restoring connectivity. The clustering weight of a node *v* is defined as the number of its children down in the recovery tree until the next cluster. On the other hand, the recovery weight represents the least number of hops from node *v* to any node that has a clustering weight of zero (i.e., leaf nodes in the recovery tree).

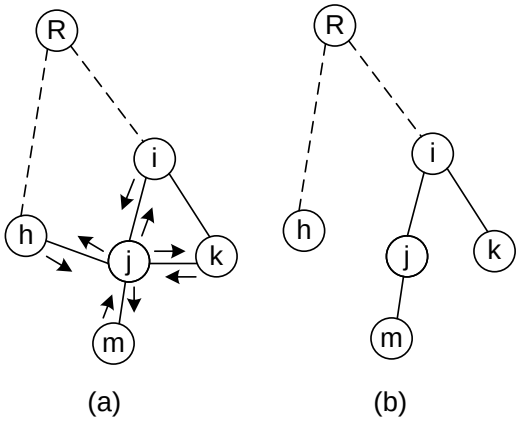


Figure 28: (a) Rank Assignment phase (b) Recovery tree

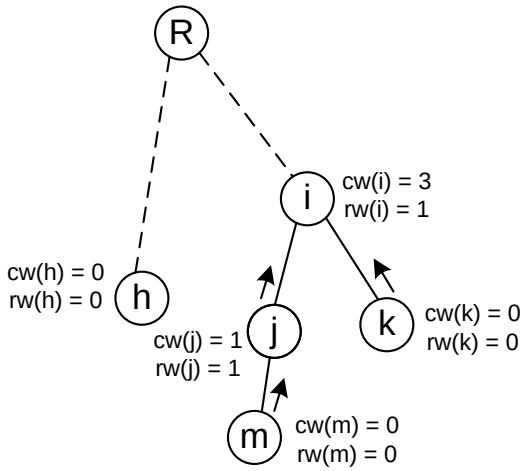
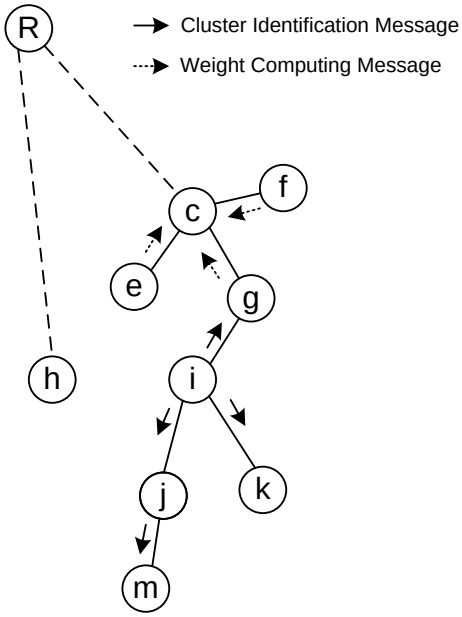


Figure 29: Illustrating weight computation for node i



Node	cw	rw	Cluster
c	3	1	c
e	0	0	c
f	0	0	c
g	0	2	c
i	3	1	i
j	1	1	i
k	0	0	i
m	0	0	i

Figure 30: Clustering phase and weight computing results

This phase is initiated by the nodes that are not nominated as rankers by any of their neighbors during the rank assignment phase. Those nodes represent leaf nodes in the recovery tree. For example, in Figure 28, nodes *m*, *k*, and *h* will start the weight computation phase.

All initial weights are set to zero. Each node sends the computed weight to its ranker. The message contains the sender ID and the clustering and recovery weights incremented by one, i.e., (i, CW_i+1, RW_i+1). For example in Figure 29, *k* shares its weights with its ranker *i*. Node *m* sends a similar message to *j* and *j* send its own to *i*. When node *i* receives these messages, it updates CW_i and RW_i as follows:

$$CW_i = CW_i + CW_j \tag{5.1}$$

$$RW_i = RW_j \tag{5.2}$$

However, RW_i is not updated by *i* unless the received recovery weight is smaller than the current recovery weight RW_i. It is obvious that the first message that is received by *i* (i.e., when RW_i is zero) is an exception for this rule. After receiving the two weight messages from *j* and *k*, $CW_i = 3$ and $RW_i = 1$.

One important step in the weight computation phase is to check the clustering weight (*cw*) against a predefine *cluster size*. If that weight exceeds this value, the node identifies itself as a cluster head and sends cluster identification to its neighbors. Otherwise, it sends a compute weight message to its ranker.

5.4.3. Cluster Identification

Let us assume for simplicity a cluster size equals three in Figure 29. After nodes *j* and *k* send compute weight messages to *i*, it updates CW_i to three. Since CW_i equals the pre-defined cluster size, node *i* identifies itself as a cluster head and it sends a cluster identification message to its neighbors. This message contains the sender ID, its cluster, cluster location, and recovery weight of that node (*i*, CHi, loc(CH_i), RW_i). To show how other nodes act upon receiving this message, let us assume there is a node *g* up in the hierarchy of the recovery tree to *R*, where $K_g = K_i\text{-}1$ and $R_i=g$, as shown in Figure 30. The message will be received by all neighbors of *i* (i.e. *j*, *k*, and *g*). The nodes that are ranked by *i* (i.e. *j* and *k*) will see the message coming from their ranker and therefore they set their cluster to *i*, save its location, and forward the message to their neighbors (if any). Node *m* will receive the message from *j* and do the same. Node *g* also will hear this message from *i* but it will act differently since the message is coming from one of its ranked nodes. Node *g* updates its recovery weight RW_g according to equation 5.1, sets its clustering weight CW_g to zero, and sends a compute weight message to $R_g=c$. When *c* receives the weight computing message from *g*, it updates CW_c. RW_c may not be updated by *c* since it achieves lower recovery weight through nodes *e* or *f*. Figure 30 shows the clustering weights, recovery weights, and cluster assignments for all nodes. To avoid confusion, a node does not send its compute message to its ranker until it computes its correct clustering weight by waiting for all children messages. For example, if node *i* receives a weight message from *k* before *j*, it waits until hearing from *j*. If for any reason, node *i* does not hear from *j*, it can send its current weight and update its ranker *g*. While,

if that happens, more clustering messages are needed, the process will yield correct clustering assignments.

5.4.4. State Diagram Description of SFRA

Figure 31 is a UML (Unified Modeling Language) behavioral state diagram showing the different states of an actor "*A*" when executing SFRA after the failure of its neighbor F_1. In addition to *A*, other healthy nodes such as *B*, *C*, *D*, and *E* are also engaged in the recovery. These nodes may be at the same time working on restoring connectivity after the failure of another node F_2, for example, that is not neighbor of *A*. The following is an explanation of the various states:

- **Normal**: After node *A* is deployed, it performs its tasks normally. The node also returns to this state when it thinks that its role in the recovery phase ends.

- **Finding** $RC(F_1)$**:** When node *A* detects the failure of a neighboring node F_1, it transits to this state in order to find $RC(F_i)$. However, the transition is guarded by the condition $[F_1=R_A]$, which means that node *A* will not start the recovery procedure unless F_1 is its ranker. All nodes that are ranked by F_1 will exchange their recovery weights. Therefore, node *A* starts this state by sending `SENDRW(`RW_A`,` R_A`)` message and waits. While waiting, if "*A*" receives another `SENDRW` message from another node, e.g. *B*, node *A* checks if "*B*" is ranked by the same node (i.e. $R_A=R_B$). If this is the case, "*A*" stores (*B*, RW_B). Before "*A*" leaves this state, it computes $RC(F_1)$ by finding the node which has the minimum recovery weight based on the `SENDRW` that it has received. If two nodes have

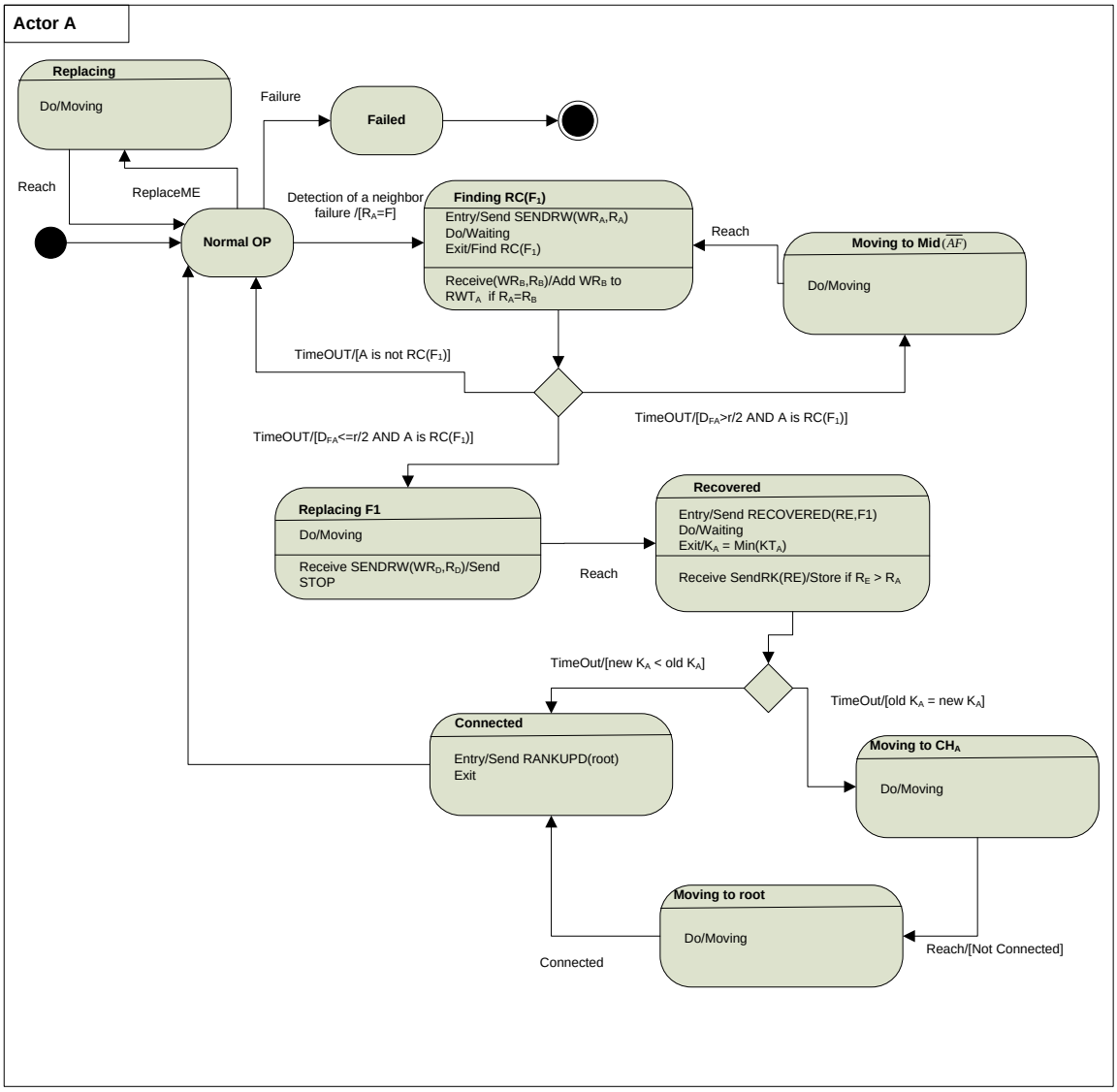


Figure 31: SFRA UML based State Diagram Representation

- the same minimum weight, "*A*" breaks the tie by choosing the node with smaller ID.

- **Replacing F_1:** *A* transits to this state from the previous state if the following two conditions are satisfied:

    - $A = RC(F_1)$, in the previous state, and

    - $D_{A,F1} \leq ½\ r$, where *r* is the communication range.

Since the movement of *A* may break its connectivity with its neighbors, as soon as *"A"* transits to this state, it sends *ReplaceMe* request to A_{RN} to replace it. In this state, *"A"* is continuously moving to the position of F_1. While in motion, if *"A"* receives a *SendRW* request from another node *"B"*, it notifies *"B"* that F_1 is already handled. This state is also reached by Finding $RC(F_1)$ state as explained later.

- **Moving to Mid($\overline{AF_1}$):** *A* transits to this state from "Finding $RC(F_1)$"if the following two conditions are satisfied:

    - $A = RC(F_1)$

    - $D_{A,F1} > ½\ r$

In this state, *A* is continuously moving to Mid($\overline{AF_1}$) position. It leaves the state when it reaches that position and returns back to Finding $RC(F_1)$.

- **Recovered:** let us assume $A= RC(F_1)$ for now. When "*A*" reaches the position of F_1, it enters the recovered state by sending *Recovered(A,F1)* and waits. Other nodes

receiving this message respond by sending their ranks to "A". While waiting, "*A*" stores all ranks that are smaller than K_A (i.e. have fewer hops to root). Before "*A*" leaves this state, it identifies the node(s) with the smallest rank among the nodes it heard from.

- **Moving to CH**: Node "*A*" transits to this state if it could not find another node, say *E*, for which $K_E < K_A$. In this state, "*A*" moves to the position of CH_A. While moving, "*A*" is continuously trying to establish a communication link to another node that has fewer hops to the root (i.e. has a lower rank).

- **Moving to root**: If "*A*" could not find CH_A or a node with a lower rank, it moves towards the root of the network. Again, while moving, *"A"* will try to establish a communication link with a node with a lower rank.

- **Connected**: *A* is connected after finding a node with a lower rank. This could happen immediately after replacing F_1 or during its movement to CH_A or root. Node *A* exits this state and return to normal operation by sending a ranking update request to the root.

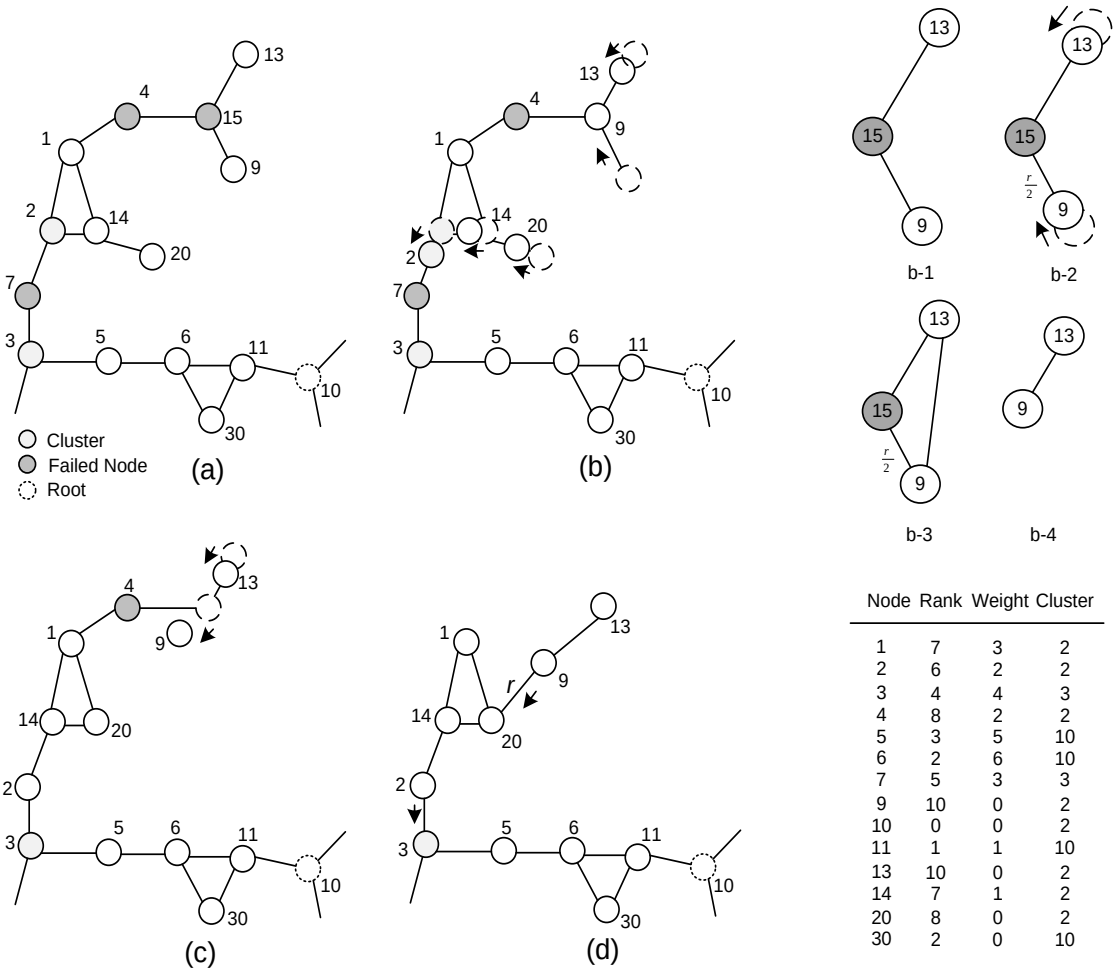
- **Replacing**: While being in the normal state, if "*A*" receives a *ReplaceMe* request from R_A, node *A* transits to this state and move to the position of R_A, and switches back to the normal state upon reaching the target position.

5.4.5. Detailed Example

Figure 32 shows a part of a topology which has 40 nodes. In this example, we assume that nodes 4, 15, and 7 fail. Although all neighbors of the failed nodes will detect their

failures, only nodes which are in their children sets are responsible for recovery. For example, the failure of node 7 is detected by both nodes 3 and 2. However, node 3 will simply ignore the failure since it is not in C_7. The first step in the recovery process is to replace each failed node with the child that has the smallest weight. To accomplish that goal, node 9, 13, and 2 send *SendWeight* request and wait. If they receive a similar request with smaller weight, they will stop leaving other nodes to participate in the recovery process. However, since node 2 is the only node in C_7, it starts moving to recover the failure of 7. Before node 2 moves, it sends a *ReplaceMe* request to the node with the smallest weight among C_2 which is node 14 in our example. In response node 14 moves and requests node 20 to replace it (Figure 32-b). When node 2 reaches the position that is r/2 to node 7, it sends a *RequestWeight* request to reach other possible nodes that are in C_7 and not heard by 2 in its original location. Node 2 will continue moving to the position of F when it does not hear any other nodes.

Node 15 has two children, 13 and 9 which are not connected to each other. Therefore, when they first detect the failure and send *SendWeigth* requests, they would not hear each other. Both nodes will move first to the position where they are r/2 from the position of the failed node 15. We assume that node 9 is closer to 15 and reaches that position first. It sends *RequestWeight* and waits. When node 13 receives the request, it compares the received weight with its own weight and send another request if it has a lower weight.



Node	Rank	Weight	Cluster
1	7	3	2
2	6	2	2
3	4	4	3
4	8	2	2
5	3	5	10
6	2	6	10
7	5	3	3
9	10	0	2
10	0	0	2
11	1	1	10
13	10	0	2
14	7	1	2
20	8	0	2
30	2	0	10

Figure 32: Detailed example to illustrate how SFRA algorithm restores connectivity after multiple nodes fail

Since node 13 has the same weight of one, it will not send any response letting node 9 handle the recovery (see b-1 to b-4 in Figure 32).

After node 9 reaches the position of node 15, it sends a Recovered message informing other possible nodes in C_{15}. When node 13 receives that message, it changes its ranker to node 9. If a node that has a higher rank receives the recovered message, it sends a *ResponseRecovered* message allowing the sender node (i.e. 9) to join its children and connect to the network. In this example, there is no such node with a higher rank since the parent of node 15 (node 14) has failed. After a certain waiting time, node 9 starts moving to the position of the head of its cluster (node 2) hoping to find some nodes along the way or ultimately connect to node 2. It also sends a *ReplaceMe* message to node 13. Before reaching the position of node 2, node 9 establishes a connection with node 20 and sends a *StopRequest* message to node 13. Figure 32-(d) shows the final network topology after all recovery operations end. The network now needs to be updated for the latest rank information. Since node 9 and 2 have changed their position, they will send update request to the root of the network. Other moving nodes 13, 14, and 20 will not send such update request because they are connected to nodes with higher ranks. When Node 20 receives a request sent by 9, it will ignore it in order to prevent duplicated update requests. When the root (node 10) receives the update request from node 11, it sends *RankAssignment* message only to the requested node (node 11). The rationale of this is to limit the extent of the rank update based on the scope of the failure in order to lower the messaging overhead cost.

5.4.6. Algorithm Analysis

5.4.6.1. Basic Analysis

Lemma 5.1: Considering the recovery tree, if there are L leaf nodes (i.e., have no children), the following equation is true for a network of N nodes, where $|C_i|$ is the number of nodes in C_i (i.e., children of node i)

$$\sum_{i=1}^{N-L} |C_i|_i = N - 1 \tag{5.3}$$

Proof: This is easily proven by noting that from the graph properties of a tree, each node in the recovery tree has only one parent except the root of the tree which has no parent. Therefore, the left hand side provides the summation of all nodes in all children sets minus one node which is the root.

Lemma-5.2: For a failed node F_1**,** the number of nodes that transit from the "Finding RC(F_1)" to "Moving to Mid($\overline{AF_1}$)**"** is less than six.

Proof: Since nodes in C_F exchange their recovery weights, only nodes with minimum recovery weights move in order to compete for replacing the failed node. Let us assume there are X nodes in C_{F1}, we can show that the maximum number of nodes that can move is constant regardless of X. If we assume there are two nodes A and B where $D_{A,F1}$, $D_{B,F1}$, and $D_{A,B}$ are all equal r, we will have an equilateral triangle (A,B,F_1) as shown in Figure 33. The maximum number of nodes that can be located on the boarder of the circle centered at F_1 without hearing each other is less than 360/60 = 6. This is true regardless of the number of nodes surrounding F_1. Now if we put additional node in any point inside

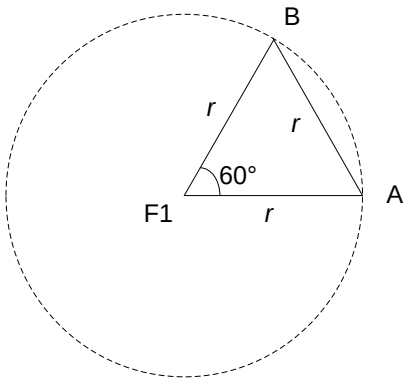


Figure 33: Illustrating the motion scenario to replace the failed node F.

the circle of *F1*, it has to be neighbor to one of these six nodes and therefore only the node with minimum recovery weight will move, not both ■.

***Theorem 5.1*:** For a failed node F_1, only one node replaces F_1 (i.e. the number of nodes that transit from the "Finding $RC(F_1)$" to the "Replacing F_1" state is one)

Proof: Let us assume that a failed node F_1 has two neighbors *A* and *B* where $RW_A < RW_B$. Now, if *B* is a neighbor of *A*, *B* will receive a *SendRW* message from *A*, and since $RW_A < RW_B$, *B* returns to normal operation . If *B* is not a neighbor of *A*, then, we have three scenarios:

- $D_{A,F1} < r/2$ and $D_{B,F1} > r/2$, then, *A* transits to the "replacing F_1" state first and sends *STOP* message to *B* when receiving *SendRW* from *B*.
- $D_{A,F1} < r/2$ and $D_{B,F1} < r/2$, then, *B* replaces *F1* and sends *STOP* message to *A* in the second scenario or
- $D_{A,F1} > r/2$ and $D_{B,F1} > r/2$, then the node which is closer to F_1 will replace F_1. The other node will stop ■

5.4.6.2. Complexity Analysis

Theorem 5.2: The messaging cost of clustering a network of size *N* is *O*(*N*).

Proof: During clustering, an actor *A* sends two or three messages depending on its computed clustering weight CW_A. If *CWA* > cluster size, *"A"* sends only two messages: one message for ranking and another message for clustering identification. If $CW_A <$ cluster size, *"A"* sends additional message for weight computing. Therefore, *"A"* sends

two messages if it is assigned as a cluster-head and three messages if it is not. The maximum cost of clustering occurs in the extreme case when only one cluster is formed, i.e., all N nodes become part of one cluster, making the worst case number of sent messages to be:

$$2 + 3(N - 1) = 3N - 1 = O(N) \tag{5.4}$$

The first part is sent by the root where the second part is sent by the other nodes. The least cost corresponds to having all nodes as cluster-heads, making the number of messages to be $2N$, which is also $O(N)$. ■.

Theorem 5.3: The messaging cost of applying SFRA to restore the network connectivity after the failure of *m* nodes is *O(mN)*, i.e., linear in the number of failure incidents.

Proof: In the proof of Lemma-2, we have shown that for a failed node *F*, less than 6 nodes will reach the second phase of finding the replacing node candidate. Each one of these nodes needs to send one message in that phase before the node with the highest weight relocates to the position of "*F*". If we assume that the number of failed nodes is *m*, then the messaging complexity is:

$$\sum_{i=1}^{m} C_i + 6m + m + Nm = N - 1 + 7m + (Nm - m^2) \tag{5.5}$$

The first part represents *SendRW* messages sent by all children of the *m* failed nodes. The number of *SendRW* messages sent in the "Finding *RC(F)*" state is *6m* assuming there are

maximum of 6 nodes neighbors for each failed node . The nodes that move to the position of the failure send *m* recovered messages. A maximum of *N-m* messages will be transmitted in response to the *Recovered* message. Using Lemma-5.1, the first part is less than (*N-1*). Therefore, the most messaging cost comes from the last part which is $O(mN)$ ■.

5.4.6.3. Cost Analysis

***Theorem 5.4*:** The total travel distance by the nodes participating in restoring connectivity using SFRA is limited by *(N-m-1)r*, where *m* is the number of failed nodes and *r* is the communication range.

Proof: the worst case scenario occurs when all network nodes are connected as a linear chain and each node is *r* from its neighbors and the root is on the edge of the network as shown in Figure 34. Since SFRA depends on restoring connectivity towards the root of the network, all healthy nodes shall move to the position of the root in a cascaded manner. In this particular case, each of these nodes travels a distance of $r \times m$. Therefore, the total distance resulted from the movement of all nodes is *(N-m-1)r*, where *(N-m-1)* represents the number of healthy nodes other than the root which sticks to its position ■.

Collary 5.1: During the recovery, the maximum distance a node can move is *rm*, where *m* is the number of failed nodes and *r* is the communication range.

Proof: This is can be observed from the worst case scenario shown in Figure 34. If we assume there is one node connected to the last failed node (i.e. F_m), while the other *N-m-3*

active nodes are connected to root. Therefore, they do not move during the recovery. This case is shown in Figure 35■.

Theorem 5.5: In large networks, a node can shorten its travel distance up to *(0.75N-2)r* by moving first to the position of the head of its cluster.

Proof: Let us consider the case shown in Figure 36(a). After the failures of F_1 and F_2, *A* will move *2r* in order to connect with CH_A. Now, let us assume that *A* moves directly to root as shown in Figure 36(b). In this case, the distance is maximized when no nodes with smaller ranks are located at a distance less than *r* while *"A"* is moving to the root. Therefore, we need to calculate the distance between *"A"* and the root. In this particular figure, there are two similar triangles (*A,C,*CH_A*)* and (*A,B,root)*. Therefore, for large *N:*

$$\frac{2r}{r} = \frac{d_{AB}}{d_{Broot}} \tag{5.6}$$

$$d_{AB} = 2d_{Broot} \approx \frac{2}{3}Nr \tag{5.7}$$

Since $A\widehat{CH}_AC$ is a right angle:

$$d_{Aroot} = \sqrt{(\frac{2}{3}Nr)^2 + (\frac{1}{3}Nr)^2} = \frac{\sqrt{5}}{3}Nr \approx 0.75Nr \tag{5.8}$$

0.75Nr represents the distance that A needs to travel to connect to the root of the network assuming that *"A"* does not move to CH_A first. If *"A"* moves to CH_A, it needs a distance of *2r* in order to connect. Therefore, moving to CH_A first saves a distance of *0.75Nr-2r* in the best case■.

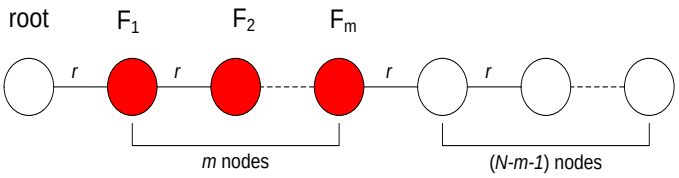


Figure 34: The failure scenario that illustrates the most travel overhead during the recovery phase.

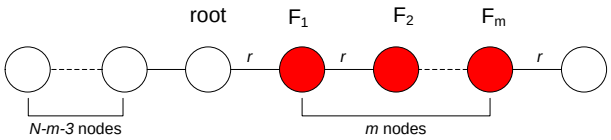


Figure 35: rm is the maximum distance a node can move

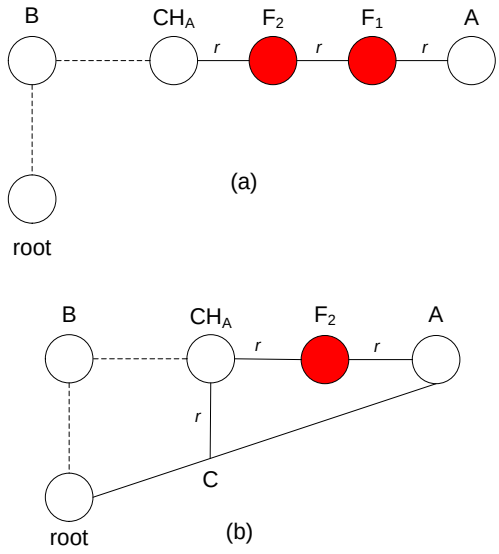


Figure 36: The best case of Clustering

CHAPTER 6

SIMULATION RESULTS AND DISCUSSION

6.1. Simulation Setup

Table 1 shows the simulation tools that we have used.

Table 1: Simulation Tools

Software	Platform	Tasks
GNU GLPK	Linux	Formulate and Solve the ILP problem
MATLAB	Windows, Linux	-Create Topologies, data, and module files for GNU GLPK - Simulate DARA, RIM, LDMR, ACRA, SFRA, and produce numerical results
MS EXCEL	Windows	-Plotting

For solving the ILP model, we used the GNU Linear Programming Kit (GLPK) under Linux [24]. Parameters needed for ILP computation are computed by MATLAB and are fed to the GLPK solver as shown in Figure 37.

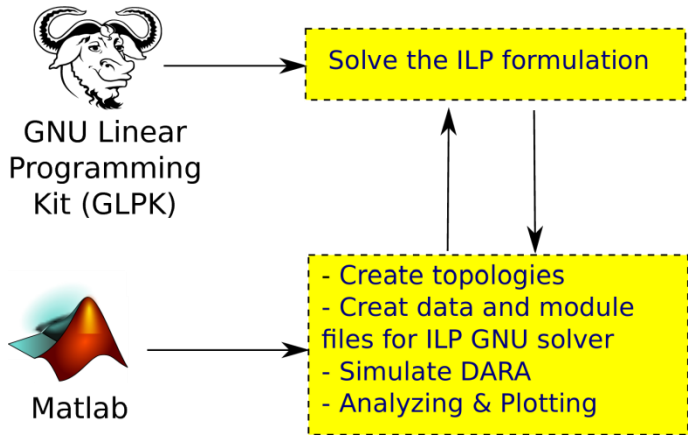


Figure 37: Simulation tools for executing the optimization approach.

The deployment area is 1000x1000 m^2 and the node's transmission range is set to 100m. To obtain statistically significant results, we have generated 20 random topologies for each set of actor nodes. For each topology, we let each cut vertex node to fail and performance metrics are computed accordingly. Finally, the results are averaged over all randomly generated topologies. These averaged samples ensure that all results are subjected to 90% confidence interval analysis and stays within 10% of the sample mean.

6.2. Performance Metrics

To evaluate the proposed approaches, we have computed the following performance metrics:

- **Total Travelled Distance**: This metric gives the total distance travelled by all nodes in the network during the recovery restoration. This metric indicates how much energy will be consumed by the whole network due to the mechanical movements of the network actors.

- **Average Travelled Distance**: This metric computes the average travelled distance for a node that got engaged in the recovery operation. This metric can be used to know how much energy loss is shared among nodes during the lifetime of the network.

- **Coverage loss rate**: This metric captures the loss of coverage resulted from the node movements.

- **Coverage Loss:** The area covered by a node is a circle with a radius of the acting range of the actor. If two or more nodes overlap, they share an area that can be

covered by any of them. Overlapping among nodes decreases the total area covered by a network.

$$Cov_{Loss} = Cov_1 - Cov_2 \tag{6.1}$$

$$Cov_{LossRate} = 1 - \frac{Cov_2}{Cov_1} \tag{6.2}$$

Where Cov_1 and Cov_2 are the total area covered by network actors before and after the failure, respectively.

- **Number of Relocated Nodes**: This metric shows also how many nodes in the network had to move to restore the network connectivity. This metric is important for mission-critical applications that strive to move the least number of actors.

- **Number of messages:** This metric captures the number of communication messages that are sent by all nodes to perform the connectivity restoration process.

6.3. Simulation Results

6.3.1. Central Approach

We compared our Central approach (ILP approach) to DARA [18] based on the total travelled distance and the average travelled distance metrics. DARA – as explained in chapter 2- is a distributed approach and therefore is different from ILP which solves the problem based on a central point of knowledge. Each node in DARA keeps a table of its

1 and 2-hop. If a node does not hear from its 1-hop neighbor for a certain amount of time, it assumes that neighbor has failed and a selection scheme is executed on each of the failed node's direct neighbors. Since each node has 2-hop information, all nodes executing DARA have all the information needed to reach a decision. The best candidate for moving is the one which has the least degree. If more than one node is having the same degree, the best candidate is the one with the shortest distance. The final criterion used to choose the best candidate in case of having two candidates is the node id.

To investigate the effect of maximum allowed travelled distance on the above metrics, we compared the ILP model with the ILP model constrained with d_{max}. If d_{max}=100m, the maximum allowed travelled distance equals to the actor communication range (i.e. 100m) and it is called optimal cascading approach. Also, we compare ILP with d_{max}=200m and it is called ILP-200.

Figure 38 compares ILP approach with DARA and optimal cascading. As expected, the central approach based on ILP has the least travelled distance since it has global information compared to DARA which is based on 2-hop information. The actors following DARA are expected to consume twice the energy consumed by ILP approach. Furthermore, ILP provides a lower bound to other heuristic approaches. However, considering the averaged travelled distance per node, we can observe that DARA shows the least consumed energy per moving actor as shown in Figure 39. This implies that DARA involves more nodes in the recovery process compared with other approaches. This finding shows that DARA is better than ILP approach in collaboratively recovering the failed node.

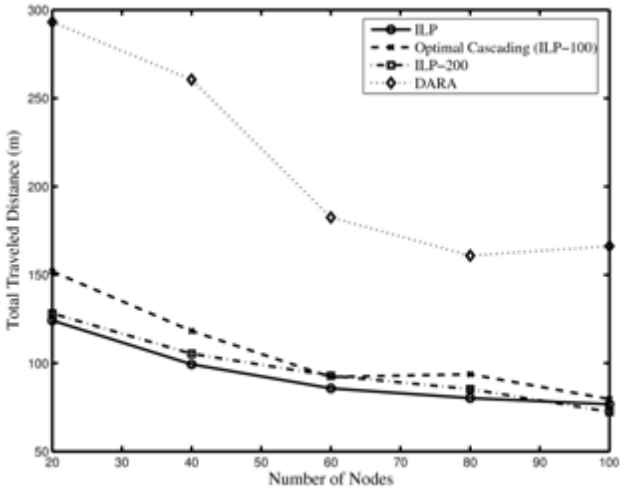


Figure 38: ILP average total travelled distance

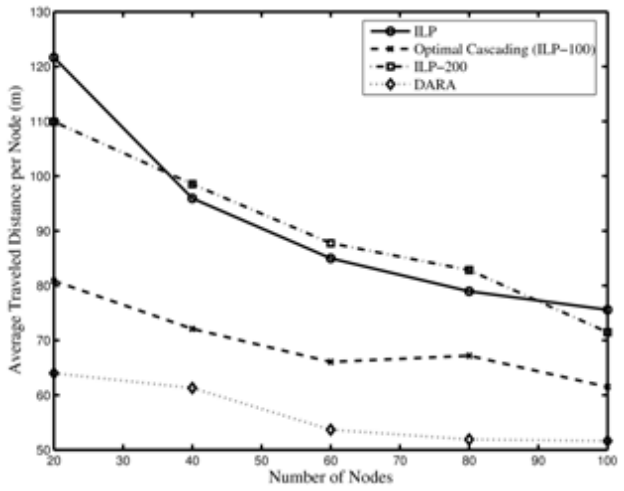


Figure 39: ILP average travelled distance per moving node.

A long travelled distance leads to excessive loss in the energy due to mechanical movements of the actor during the recovery process. The ILP-200 curve show the total travelled distance after putting a constraint of 200m on the maximum travelled distance during recovery. Based on our simulation, a constraint of 200m which is double the transmission range has a very comparable result to the general case where no restriction is imposed.

One of the additional features that our approach is taking into account during recovery is the coverage loss. For this purpose, we constrained the maximum coverage loss due to node movement after the failure. As shown in Figure 40, the higher the accepted coverage loss gets, the lower the travelled distance becomes. In fact, when we restrict the coverage loss, more nodes need to move during the recovery and hence the total travelled distance increases.

Now, let's examine the case of multiple failures. Our ILP approach formulation is able to handle multiple node failures at the same time. Figure 41 illustrates the effects of number of failed nodes on the total travelled distance for different network sizes. Small networks incur the most relocation overhead when many nodes fail simultaneously, because the node density is low and the actor nodes need to move long distances to re-establish the connectivity. Figure 42 shows a linear relation between the total travelled distance and the number of failed nodes for different values of coverage loss thresholds. The relationship is leaner since the recovery of every additional failed node adds a relatively similar cost for the same network size.

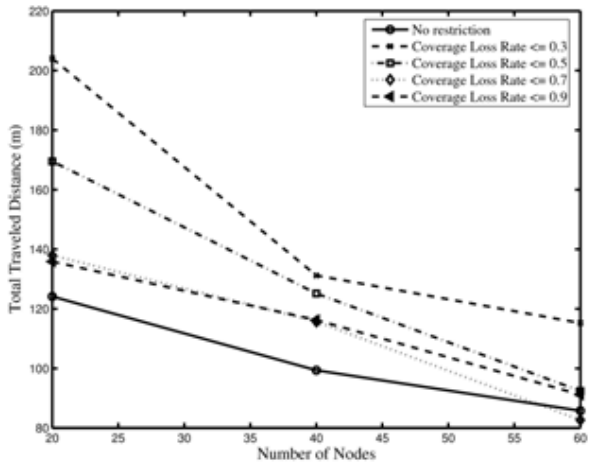


Figure 40: ILP coverage effects on the average total travelled distance.

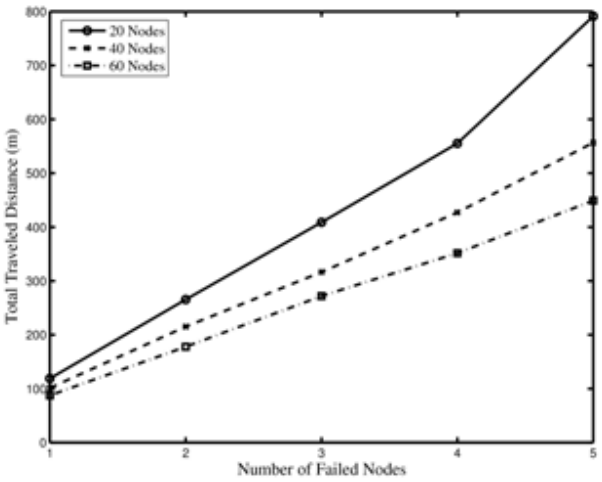


Figure 41: ILP averaged total travelled distance versus the number of failed nodes.

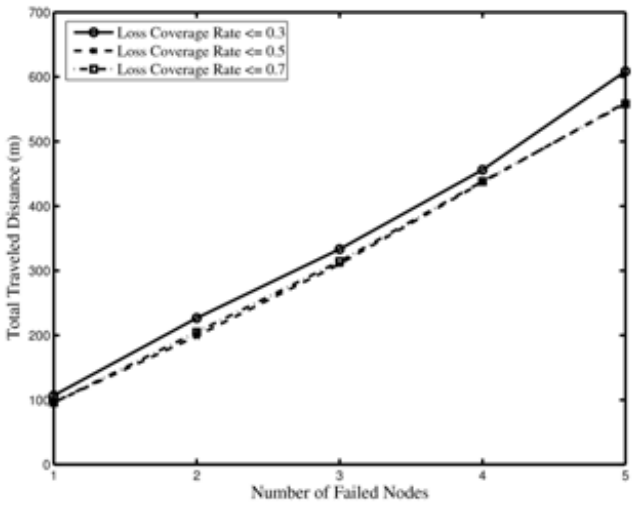


Figure 42: ILP coverage effects on total travelled distance in case of multiple failures (Nodes = 60)

.

6.3.2. LDMR Simulation Results

In this section, we compare LDMR to RIM approach which is based on shrinking network topology toward the failed node in order to restore connectivity of the network as explained in chapter 2. The transmission range used in our simulation is set to 150m.

Figure 43 shows the total travelled distance for the two approaches. RIM performs better when the number of nodes is small. As the number of nodes increases, LDMR outperforms RIM. In our simulation, this happened when the number of nodes exceeds 70 nodes. In the case of RIM approach, the number of moving nodes increases as the network becomes larger. The total travelled distance resulted using RIM is almost doubled (N>100) compared to LDMR. Furthermore, in LDMR, the probability of finding a closer non cut-vertex node increases when the number of nodes increases. Therefore, when the network becomes larger, the total distance travelled under RIM increases while it decreases in the case of the LDMR.

Figure 44 shows the average coverage loss rate resulting after applying each approach. RIM also shows a better result when the number of nodes are small. However, when the number of nodes increases, the network starts to lose coverage because more overlapping is resulted due to larger number of nodes movement. In the case of the LDMR, the nodes which replace the positions of the direct neighbors of the failed node become very close to each other and therefore more overlapping is resulted which degrades the total

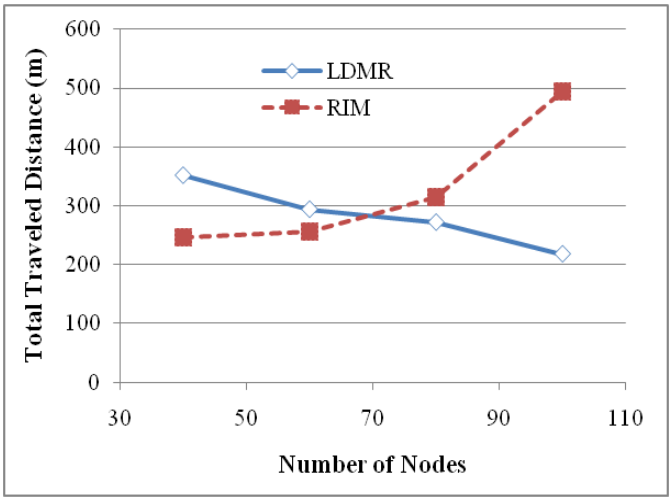


Figure 43: The total distance (LDMR vs. RIM) travelled by the involved nodes during the recovery (r=150).

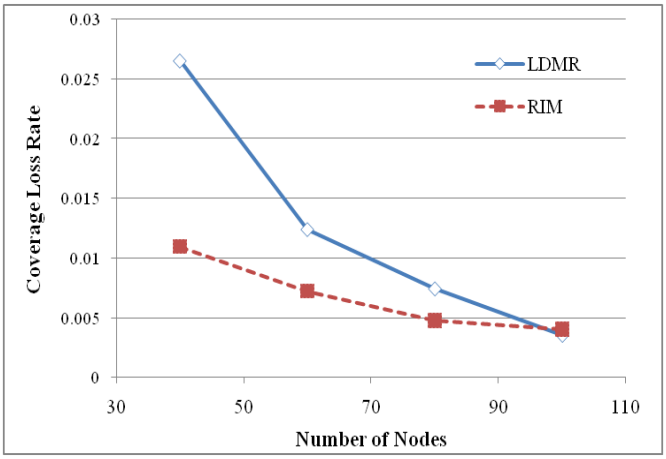


Figure 44: The loss coverage rate (LDMR vs. RIM) during the recovery (r=150).

coverage compared to RIM. However, as the network becomes larger the total coverage for the two approaches becomes comparable. As clearly indicated in Figure 44, the loss coverage rate becomes relatively very small as the number of nodes increases. When the network is large, this value becomes very small compared to the total covered area.

6.3.3. ACRA Simulation Results

In the section, we compare ACRA to RIM which depends only on cascaded movement. We described how the movement used in ACRA is different from RIM in section 4.4.1. We also compared ACRA to approaches that depend on 2-hop information such as DARA [18] in terms of communication messages.

Total Travelled Distance: Figure 45 illustrates the robustness of the adaptive approach where the total travelled distance is lower than RIM for the whole range of network sizes. According to the simulation results, ACRA has a very comparable cost for networks with less than 40 nodes. As networks become larger than 40 nodes, ACRA starts outperforming RIM because it avoids cascaded movement when nearby non-cut-vertices can be used to bring back the connectivity to the network with lower cost.

Number of relocated nodes: The Results depicted in Figure 46 show that ACRA tries to move fewer nodes than RIM. This is a very interesting and desirable feature for mission-critical networks. Furthermore, the results show that ACRA is less sensitive to the network size which means the proposed approach is capable of selecting the right node and move it to the right location. This feature can also be observed in Figure 45 and Figure 51.

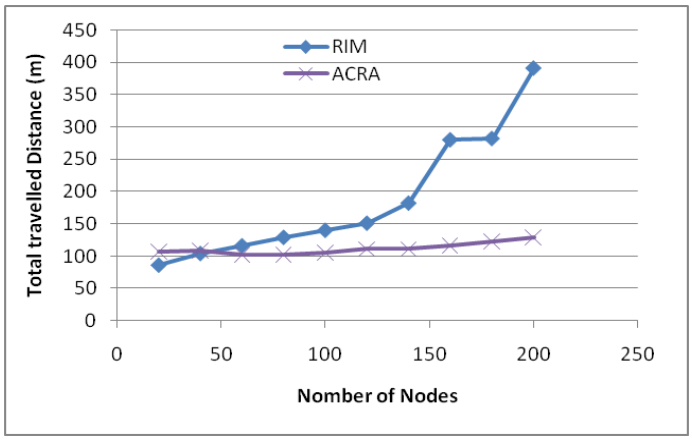


Figure 45: Total Travelled Distance (ACRA vs. RIM) (r=100)

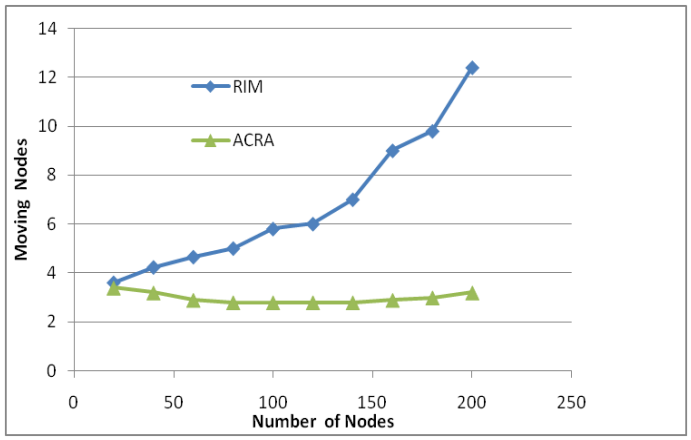


Figure 46: Figure 46: Number of moving nodes (ACRA vs. RIM) (r=100)

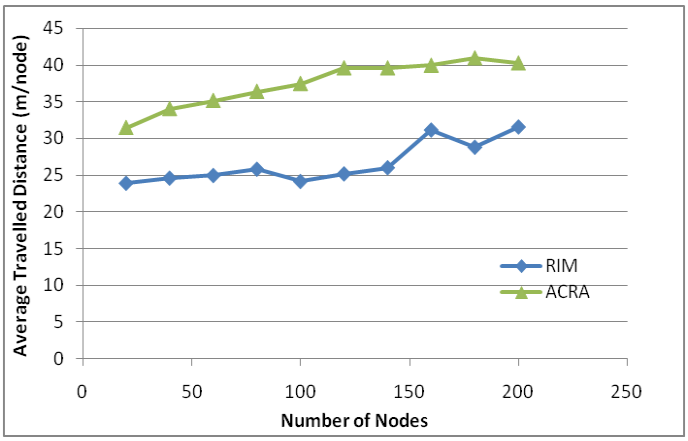


Figure 47: Average Travelled Distance per node (ACRA vs. RIM) (r=100)

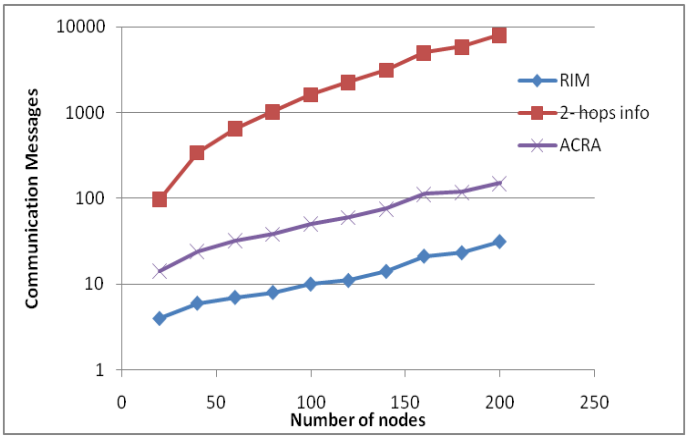


Figure 48: Number of communication messages (ACRA vs. RIM vs. 2-hop info (DARA))

Average Travelled Distance per node: RIM has a lower average travelled distance per node than ACRA because RIM moves larger number of nodes. However, since RIM starts to perform very bad in large networks, the gap becomes smaller in large networks (see Figure 47).

Communication Messages: Approaches that need 2hop information such as DARA show more overhead and communication messages than RIM and ACRA. RIM shows the least message overhead compared with ACRA and DARA. This result is expected as ACRA involves search requests, responses, and other type of messages needed to properly execute the recovery process as we explained in section 4.4 (see Figure 48).

Maximum number of hops: Figure 49 shows that when the maximum number of hops (i.e., H) increases, we can see a small enhancement in the performance if repositioning non-cut-vertices is the only option. This is because moving a node that is two hops away from the searching node is better than commanding the closest node to recover and do the search again. However, the enhancement is minimal and demands extra overhead. To include the nodes that are two hops away, the time-to-live has to be at least two hops. The effect of the maximum number of hops on ACRA is shown in Figure 50. As the number of hops increases, the total travelled distance increases in smaller networks. However, this effect starts to diminish as the network size increases. Such a trend is expected since using a non-cut-vertex is more frequently executed if we increase the cut-off threshold (number of maximum hops). Non cut-vertex utilization has a lower cost in larger networks. By comparing Figure 49 and Figure 50, ACRA shows a better performance for small networks (less than 100) since ACRA executes RIM in case of not finding a non-cut-vertex candidate. RIM has a better performance than LDMR in small networks.

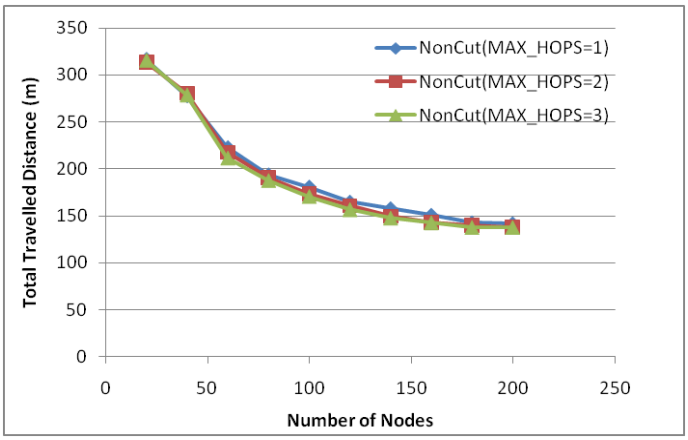


Figure 49: The effects of maximum hops on the total travelled distance using (ACRA) (r=150)

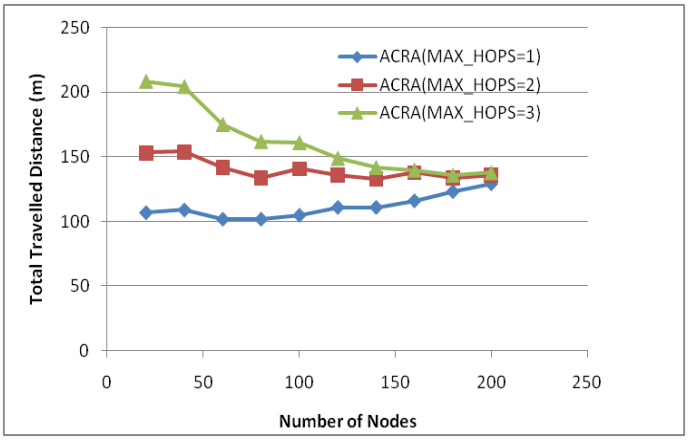


Figure 50: The effects of maximum hops on the total travelled distance of ACRA (r=150)

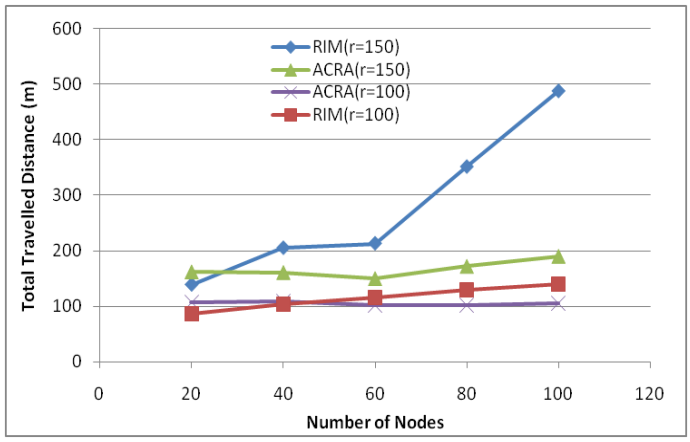


Figure 51: The effect of communication range on the average travelled distance for ACRA and RIM.

Communication range: Both approaches (RIM and ACRA) are affected by increasing the communication range. However, RIM is the most affected approach since its cost increases rapidly when the network size grows. ACRA is still proving to be the best by having the least travelled distance cost for all range of network sizes as shown in Figure 51.

Coverage Loss Rate: Results shown in Figure 52 are based on equation (6.2). Since ACRA behaves adaptively, the coverage loss rate is between RIM and LDMR approaches. It achieves better coverage in small networks than LDMR because it execute cascaded movement (RIM) in small networks more often than exploiting non-cut-vertices which cause more overlapping with neighbors after moving. On the other hand, ACRA achieves the same performance as LDMR in large networks.

6.3.4. SFRA Simulation Results

In this section, we studied SFRA via simulation. We used a uniform random probability of failure (PF) to pick the failed nodes among the cut-vertices. For each network size, we compute the average number of failed nodes based on this probability. It is worth to mention that non cut-vertices are leaf nodes in the recovery tree and do not partition the network if they fail.

Figure 53 shows the total travelled distance cost of SFRA for different probabilities of failure for each multiple network sizes. It is obvious from the plot that for a certain network size, the recovery cost grows when with the increase in the failed node count (higher PF). Having many failed nodes in the recovery tree means more children need to move and therefore a longer distance to be travelled. For a fixed probability of failure, the

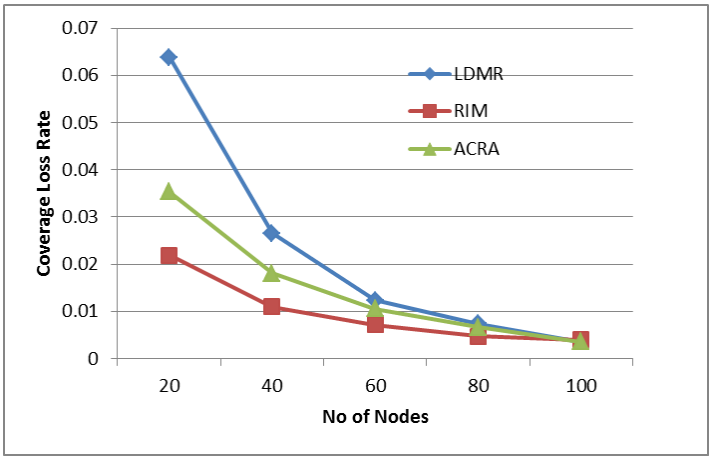


Figure 52: Coverage Loss Rate of ACRA compared to RIM and LDMR

cost of recovery increases when the network size (number of nodes) increases. This is because the increased node count implies the involvement of more nodes in the recovery process and thus the total and average distance grows.

To assess the overhead imposed by the recovery process executed by SFRA, we first examine SFRA for a single node failure and hence, we compared the total travelled distance per failed node to RIM and LDMR. RIM is a recovery approach that moves healthy nodes inward toward the failed node and achieves good results in small networks. On the other hand, LDMR depends on non-critical nodes to recover from a node failure and performs well in large and dense networks. Figure 54 shows that SFRA yields the smallest travelled distance per failed node compared to RIM and LDMR. This indicates the robustness and efficiency of SFRA for various network sizes.

SFRA operation depends on the per-defined cluster size. To explore the impact of cluster size on the performance of SFRA, we fixed the failure probability and compared the total travelled distance for different cluster sizes and different network sizes. We observed that assigning each node to a nearby cluster head really helps in lowering the travel overhead. Figure 55 shows that for networks with 40 and 60 nodes, the recovery cost grows slightly when the cluster size increases. This is very much intuitive since the relocating nodes have to travel further to the position of their cluster heads, when deemed necessary. However, for a network of 100 nodes, clustering does not show significant improvement since the number of healthy nodes is large, and many leader nodes can be identified to replace their parents without the need to reach the cluster heads. This result shows that clustering always improve the performance compared to the case of letting leader nodes go to the root directly (cluster size equals one). In addition, using a smaller cluster size

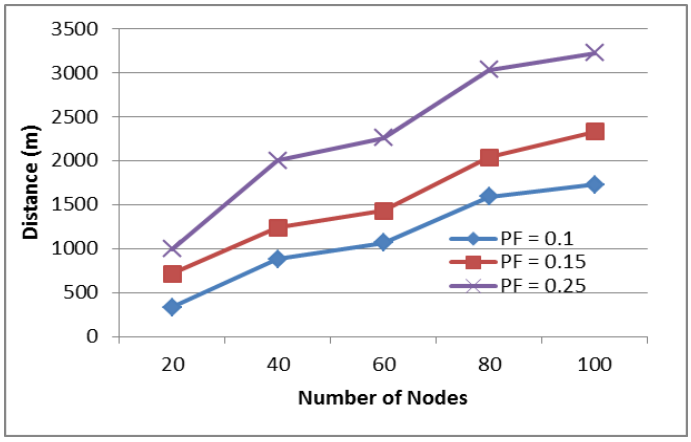


Figure 53: Total travelled distance of SFRA for different probability failure (PF)

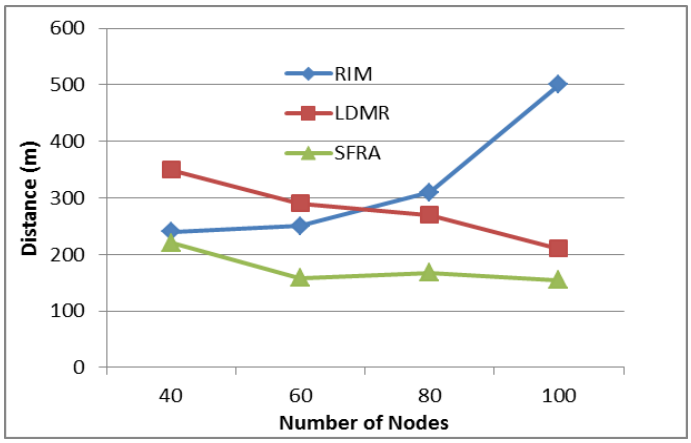


Figure 54: Avg. travelled distance per failed node of SFRA compared to single failure approaches

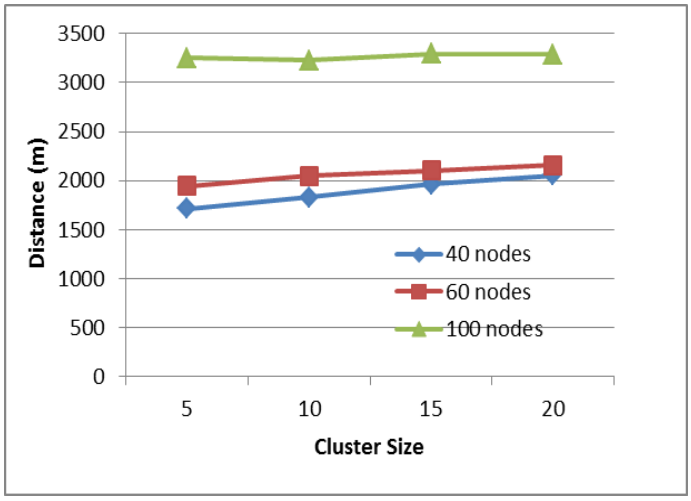


Figure 55: Total traveled distance of SFRA for different cluster sizes (PF = 0.25)

achieves a slightly fewer number of communication messages during weight computing and clustering phases. Non cluster nodes will send three different messages: rank assignment, weight computing, and clustering messages. Cluster nodes will send only rank assignment and clustering messages. Increasing the cluster size will increase non cluster nodes and decreases cluster count for the same network size which lead to a slightly more communication messages. Table 2 shows that there is a about %4.5 increase in number of sent messages when the cluster size is increased from 5 to 15. The percentage of increase stays almost unchanged for different network sizes.

Table 2: Number of sent messages during rank, weight computing, and clustering phases of SFRA

# nodes	Cluster size = 5	Cluster size = 15	% increase
40	4647	4853	%4.43
60	7847	8201	%4.51
100	1602	1674	%4.49

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In conclusion, we provided the followings through our work:

- The problem of optimal self-healing of a partitioned wireless sensor and actor network is formulated as an Integer Linear Program (ILP). The total distance travelled by relocating nodes is minimized while not exceeding a predefined coverage loss rate. Extensive simulation experiments have been conducted to validate our approach. We have used DARA, a distributed connectivity restoration heuristic, as a baseline for performance comparison. The simulation results have confirmed the superiority of our approach. Our ILP formulation can be viewed as providing a lower bound on achievable total travel distance for node-relocation-based connectivity restoration schemes.

- A distributed approach that restores the connectivity of WSAN has been presented. The Least distance Movement Recovery (LDMR) exploits non cut-vertices actors to replace other nodes in the recovery process. We have compared LDMR approach to RIM and shown via extensive simulation experiments and analysis that LDMR imposes less travelled distance overhead in larger networks.

- An enhancement to LDMR is given by presenting an adaptive connectivity restoration approach (ACRA) for efficient and autonomous repair of partitioned wireless sensor and actor networks that is caused by the failure of a critical (cut-vertex) actor. ACRA replaces the failed actor with one of the existing nodes and

adaptively selects the failure recovery strategy in order to minimize the overhead. Non cut-vertices are preferred if they are available in the vicinity of the failed node, otherwise multiple nodes are sequentially moved in a cascaded manner. Due to the adaptive nature of ACRA, it can be implemented over a wide range of network sizes. ACRA is validated analytically and via simulation. Extensive simulation experiments have confirmed the effectiveness and correctness of ACRA and demonstrated that it imposes less motion overhead and engages fewer nodes than contemporary recovery schemes found in the literature.

- A new approach for recovery from multiple simultaneous node failures in wireless sensor and actor networks (SFRA) has been presented. In SFRA, each node has a rank based on the number of hops to a pre-designated root node in the network. Some nodes are identified as cluster heads based on the number of their children in the recovery tree. Each node is assigned a recovery weight and a nearby cluster node which serves as a gateway to other nodes that belong to that cluster. The recovery weight is used to decide which node is better to move in order to achieve lower recovery cost. The simulation results have demonstrated that SFRA can achieve low recovery cost per failed node in small and large networks. The results have also shown that clustering leads to lower recovery cost if the sub-network needs to re-establish links with the rest of the network.

For future work, we could do the followings:

- Improving our centralized scheme by adding new constraints representing additional factors to the problem such type of task, status of the node, and other QoS metrics.

- Since our distributed schemes depend on searching, we could extend the search to support different node capabilities. For example, the searching nodes in LDMR and ACRA look for a node having the same capability of the failed node.

- Integrating ACRA and SFRA in one unified approach.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[2] E. Ould-Ahmed-Vall, D. M. Blough, B. H. Ferri, and G. F. Riley, "Distributed global ID assignment for wireless sensor networks," *Ad Hoc Networks*, vol. 7, no. 6, pp. 1194–1216, Aug. 2009.

[3] H. Zhou, M. W. Mutka, and L. M. Ni, "Reactive ID Assignment for Wireless Sensor Networks," *Int J Wireless Inf Networks*, vol. 13, no. 4, pp. 317–328, Oct. 2006.

[4] Z.-G. Du, D.-P. Qian, and Y. Liu, "Addressing Protocols for Wireless Sensor Networks," *Journal of Software*, vol. 20, no. 10, pp. 2787–2798, Nov. 2009.

[5] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, May 2005.

[6] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6 – 28, Dec. 2004.

[7] I. Demirkol, C. Ersoy, and F. Alagoz, "MAC protocols for wireless sensor networks: a survey," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 115 – 121, Apr. 2006.

[8] G. Hoblos, M. Staroswiecki, and A. Aitouche, "Optimal design of fault tolerant sensor networks," in *Proceedings of the 2000 IEEE International Conference on Control Applications, 2000*, 2000, pp. 467–472.

[9] D. Nadig, S. S. Iyengar, and D. N. Jayasimha, "A new architecture for distributed sensor integration," in *, IEEE Southeastcon '93, Proceedings*, 1993, p. 8 p.–.

[10] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor information networking architecture and applications," *IEEE Personal Communications*, vol. 8, no. 4, pp. 52–59, 2001.

[11] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "Communication and Coordination in Wireless Sensor and Actor Networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 10, pp. 1116 –1129, Oct. 2007.

[12] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36 – 44, Aug. 2004.

[13] G. Wang, G. Cao, T. La Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005, vol. 4, pp. 2302–2312 vol. 4.

[14] W. Wang, V. Srinivasan, and K.-C. Chua, "Using mobile relays to prolong the lifetime of wireless sensor networks," in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2005, pp. 270–283.

[15] A. A. Abbasi, M. Younis, and K. Akkaya, "Movement-Assisted Connectivity Restoration in Wireless Sensor and Actor Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 9, pp. 1366–1379, 2009.

[16] K. Akkaya, F. Senel, A. Thimmapuram, and S. Uludag, "Distributed Recovery from Network Partitioning in Movable Sensor/Actor Networks via Controlled Mobility," *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 258–271, 2010.

[17] F. Dai and J. Wu, “An extended localized algorithm for connected dominating set formation in ad hoc wireless networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 10, pp. 908–920, 2004.
[18] M. Younis, S. Lee, and A. A. Abbasi, “A Localized Algorithm for Restoring Internode Connectivity in Networks of Moveable Sensors,” *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1669–1682, 2010.
[19] K. Akkaya and M. Younis, “C2AP: Coverage-aware and Connectivity-constrained Actor Positioning in Wireless Sensor and Actor Networks,” in *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa*, 2007, pp. 281–288.
[20] M. Younis and K. Akkaya, “Strategies and techniques for node placement in wireless sensor networks: A survey,” *Ad Hoc Networks*, vol. 6, no. 4, pp. 621–655, Jun. 2008.
[21] N. Tamboli and M. Younis, “Coverage-aware connectivity restoration in mobile sensor networks,” *Journal of Network and Computer Applications*, vol. 33, no. 4, pp. 363–374, Jul. 2010.
[22] K. Akkaya and F. Senel, “Detecting and connecting disjoint sub-networks in wireless sensor and actor networks,” *Ad Hoc Networks*, vol. 7, no. 7, pp. 1330–1346, Sep. 2009.
[23] S. Lee and M. Younis, “Recovery from multiple simultaneous failures in wireless sensor networks using minimum Steiner tree,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 5, pp. 525–536, May 2010.
[24] P. Sinha, R. Sivakumar, and V. Bharghavan, “MCEDAR: multicast core-extraction distributed ad hoc routing,” in *1999 IEEE Wireless Communications and Networking Conference, 1999. WCNC*, 1999, pp. 1313–1317 vol.3.
[25] S. K. Das, B. S. Manoj, and C. S. R. Murthy, “Weight based multicast routing protocol for ad hoc wireless networks,” in *IEEE Global Telecommunications Conference, 2002. GLOBECOM ’02*, 2002, vol. 1, pp. 117–121 vol.1.
[26] R. S. Sisodia, I. Karthigeyan, B. S. Manoj, and C. S. R. Murthy, “A preferred link based multicast protocol for wireless mobile ad hoc networks,” in *IEEE International Conference on Communications, 2003. ICC ’03*, 2003, vol. 3, pp. 2213–2217 vol.3.
[27] A. A. Abbasi and M. Younis, “A survey on clustering algorithms for wireless sensor networks,” *Computer Communications*, vol. 30, no. 14–15, pp. 2826–2841, Oct. 2007.
[28] J. Luttamaguzi, M. Pelsmajer, Z. Shen, and B. Yang, “Integer programming solutions for several optimization problems in graph theory,” in *20th International Conference on Computers and Their Applications (CATA 2005). Also as a DIMACS technical report*, 2005.

Vitae

Name :Abdullah Alfadhly

Nationality :Saudi

Date of Birth :1/11/1970

Email :ab.fadhly@gmail.com

Address :P.O Box 6086. Riyadh 11442

Academic Background :I received the B.S degree in Computer Science and Engineering from King Saud University, Riyadh, Saudi Arabia in 1994. In 1995, I joined King Abdulaziz City for Science and Technology (KACST). I received the M.S degree in Computer Engineering from Case Western Reserve University (CWRU), Cleveland, US in 2001. My research interests include Wireless Sensor Networks, Cognitive Radio, and 4G networks.