1-1-2005

Distributed self-(star) minimum connected sensor cover algorithms

Rajesh Patel
University of Nevada, Las Vegas

DISTRIBUTED SELF-* MINIMUM CONNECTED SENSOR COVER ALGORITHMS

by

Rajesh Patel

Bachelor of Science
University of Southern California
1991

Bachelor of Science
University of Nevada, Las Vegas
1999

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2005

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

OCTOBER 10TH , 2005

The Thesis prepared by

RAJESH PATEL

Entitled

DISTRIBUTED SELF-*MINIMUM CONNECTED SENSOR COVER ALGORITHMS

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Co-Chair

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

ABSTRACT

Distributed Self-∗ Minimum Connected Sensor Cover Algorithms

by

Rajesh Patel

Dr. Ajoy K. Datta, Examination Committee Chair
School of Computer Science
University of Nevada, Las Vegas

Dr. Maria Gradinariu, Examination Committee Co-Chair
IRISA, Campus de Beaulieu, France

Wireless ad-hoc sensor networks are composed of a large number of tiny sensors with embedded microprocessors, that have very limited resources and yet must coordinate amongst themselves to form a connected network. Every sensor has a certain sensing radius, R_s, within which it is capable of "covering" a particular region by detecting or gathering certain data. Every sensor also has a communication radius, R_c, within which it is capable of sending or receiving data.

Given a query over a sensor network, the minimum connected sensor cover problem is to select a minimum, or nearly minimum, set of sensors, called a minimum connected sensor cover, such that the selected sensors cover the query region, and form a connected network amongst themselves. In this thesis, we use present three *fully distributed, strictly localized, scalable, self-∗* solutions to the minimum connected sensor cover problem.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to offer a special thanks to my thesis advisor, Dr. Ajoy K. Datta, for chairing my committee and advising me throughout my thesis work. His patience, support, enthusiasm, and most importantly, his confidence in my abilities, have helped me greatly throughout my graduate study. I would also like to offer a special thanks to the co-chair of my examination committee, Dr. Maria Gradinariu at IRISA/Universite Rennes 1, France, for her guidance and numerous contributions to this research. The support and advise offered by her for this thesis research was very valuable and important. I am also very grateful to Dr. John Minor, Dr. Yoohwan Kim, and Dr. Venkatesan Muthukumar for their participation in my committee.

This thesis is dedicated to my parents and family. Their support, love, and faith in my abilities were very important for completing this thesis research.

CHAPTER 1

INTRODUCTION

Recent advances in microprocessor, memory, and wireless communication technology have enabled the production of tiny networked sensors which will revolutionalize information gathering and processing in both urban environments and inhospitable terrain. These wireless ad hoc sensor networks [21] have many applications and consist of a large number of tiny sensing devices with very limited resources that must coordinate amongst themselves to gather, process, and communicate information about their environments. A research team at the University of California at Berkeley is attempting to create a networked sensor that is the size of a few cubic millimeters [42]. Once produced, hundreds of thousands of these sensors, which can be collectively referred to as "smart-dust", may be randomly deployed from an aircraft, over a certain region of interest, such as a battlefield. These DARPA smart-dust prototypes use off-the-shelf components. DARPA also supplied the funding to produce an open-source embedded platform for such wireless sensors, called the Network Embedded Systems Technology Program (NEST) [4].

Because these networked sensors are often densely deployed and have limited battery power, in a sensor network there may be some failing sensors or sensors that have merely exhausted their energy supply. However, it may be impossible or infeasible to recharge sensors once they have been deployed, especially if they have been deployed in an inhospitable or physically unreachable terrain. Therefore, since the fundamental constraint on a networked sensor is its energy consumption, only some of the sensors within a particular sensing region, or query region, should be in

an active state.

In addition to this, the topology of a sensor network may change very frequently, due to malfunctions or changes in the position or available energy of sensors. Therefore, deploying a pre-configured network of a large number of sensors is impractical. Taking these constraints into consideration, a sensor network must be self-configuring and self-maintaining or self-healing. The term self-$*$ has been used to describe properties such as self-organizing, self-configuring, self-healing, etc. In this thesis, we will present a self-stabilizing solution to the important problem of minimizing energy consumption within a sensor network. We will then show that this solution is a self-$*$ solution. In a self-stabilizing system, every computation, upon starting from an arbitrary state, eventually reaches a state where the computation satisfies the problem specification in a finite number of steps.

A sensor network can be modeled as a graph G(V, E), where every sensor in the network may be represented by a vertex in the graph. Since every sensor has a certain radius within which it can sense data with a particular confidence level, also called the sensor's sensing radius, every vertex is also associated with a disk centered at this vertex, which is called the sensor's sensing disk. A group of sensors is said to cover a certain region when the union of the sensing disks of these sensors completely cover this region. Also, since every sensor has a certain range within which it is capable of sending or receiving data, called the sensor's communication radius, every vertex is also associated with a transmission disk that is centered at this vertex. Two sensors are neighbors and are said to be connected if and only if each sensor is located within each other's transmission disk.

Within a sensor network, a query may be sent to sense certain events or data over a particular query region. Given such a query over a sensor network, the minimum connected sensor cover problem is to select a minimum, or nearly minimum, set of sensors called a minimum connected sensor cover, such that the selected sensors cover the query region, and form a connected network amongst themselves. In its general

form, this problem is known to be NP-hard [26, 36].

By definition, a dominating set is a set of vertices such that every vertex in the graph is either in the dominating set, or adjacent to a vertex in the dominating set. A connected dominating set, CDS, is a dominating set which is also a connected subgraph. This implies that every node in graph G(V,E) lies within the transmission disk of some node in the CDS. Therefore, if the communication radius of a sensor is equal to its sensing radius, and the nodes of the graph are the sensors, then for a densely populated graph, the union of the sensing radii of all nodes in a connected dominating set whose sensing radii intersect with some portion of the query region, will be sufficient to cover the entire query region. In addition to this, a minimum connected dominating set is a connected dominating set of minimal cardinality. Thus if the sensing radius of a sensor is equal to its communication radius, then the minimum connected sensor cover problem can be solved by selecting a minimum, or nearly minimum, set of sensors whose sensing ranges intersect with the query region, and that form a minimum connected dominating set. In doing so, the sensors in this set also cover the query region, can (directly or indirectly) communicate with each other, and can minimize the usage of energy.

1.1 Contributions

The topic of this thesis research is the design of an energy-efficient protocol for covering a query region in wireless sensor networks. To this end, two main areas will be discussed, the design of wireless networks and the design of self-$*$ systems. The first contribution of this research is the study of various aspects of wireless sensor networks. We examine current solutions to many important problems in this area, such as data dissemination, data aggregation, media access methods, and power awareness. The second contribution is a discussion if self-$*$ systems. Ubiquitous/pervasive computing, IBM's autonomic computing, self-repairing computers, and self-stabilizing systems will be discussed. We will also examine the link between

wireless sensor networks and self-* systems.

The third and most important contribution of this research is to design a self-* power-efficient solution to the connected sensor cover problem. This will be a localized, distributed solution to the connected sensor cover problem. In this context, a localized solution means that sensor nodes communicate only with their neighbors. Localized solutions in large networks are desirable due to their high reliability and scalability. We used the self-stabilization paradigm to implement the self-* properties of our solution. Our solution can handle different types of faults including node and link (wireless communication) failures, power level changes, and memory corruption.

1.2 Outline of the Thesis

We start with a discussion of the design of wireless networks in Chapter 2. This includes the basic idea of mobile wireless networks such as mobile ad-hoc networks and cellular networks. We then discuss wireless sensor networks. In Chapter 3, we discuss self-* systems. We include a description of many types of fault-tolerant systems in the context of the self-* framework. In Chapter 4 we state the motivation of this research, describe some results in related areas, describe the model and program used in our contribution, and introduce the connected sensor cover problem. The main contribution of this thesis is presented in Chapters 5, 6, and 7, where we present three self-stabilizing solutions to the problem, including proofs of their correctness. The first $\mathcal{MCSC}$ algorithm is presented in Chapter 5, the second $\mathcal{MCSC}$ algorithm is presented in Chapter 6, and the third $\mathcal{MCSC}$ algorithm is presented in Chapter 7. A discussion of the complexity of the algorithms, simulation results, and other properties are included in Chapter 8. Finally, we conclude and present some ideas for future research in Chapter 9.

CHAPTER 2

WIRELESS NETWORKS

In this chapter, we will present various concepts and issues related to wireless sensor networks. However, we will first give a brief overview of wireless networks, mobile wireless networks, and ad-hoc wireless networks as background information.

A wireless network is a network of telephones or computing devices that use radio transmission as their carrier or physical layer. Examples of wireless networks are wireless LAN (local area networks), wireless PAN (personal area networks), UMTS (universal mobile telephone service), and D-AMPS (digital AMPS). All wireless networks use the transmission of radio signals to send or receive data from one device in the network to another.

2.1 Mobile Wireless Networks

The recent growth in popularity of mobile computing has led to many technological advances in this field and has resulted in the rapid development of small, inexpensive, and powerful computing devices such as mobile phones, Personal Digital Assistants (PDA's), various handheld devices, and laptop computers. The ease of mobility of these units makes it both critical and challenging to maintain communication amongst the various types of such mobile devices. However, the recent advances in wireless communication technology have enabled wireless mobile units to communicate with each other in various ways. The aim of such wireless communication is to enable users to communicate and use computing devices without being tethered to an information source. There are two main classifications of mobile wireless networks, infrastructured (cellular) and infrastructureless (ad hoc) wireless networks [7].

2.1.1 Infrastructured/Cellular Wireless Networks

An infrastructured wireless network is a wireless network in which access points are distributed along a wired backbone, and mobile devices connect to each other by communicating directly with these access points. These access points do not move and are present just to act as routers and forward packets for other nodes, thus allowing the mobile nodes to save power. Also, the access points are usually connected to the fixed network infrastructure or to the Internet. Mobile nodes that are within the coverage area of an access point are able to send and receive signals to that access point, and can thus communicate directly with that access point. However, as a mobile node moves out of the coverage area of one access point and into that of another, it must cease communication with the old access point and begin communication with the new access point. This process is called a handoff, and should be completely undetectable to the user [43]. A few examples of infrastructured wireless networks are Wireless Local Area Network (WLAN), cellular networks, Wireless Local Loop (WLL), and Global System for Mobile Communications (GSM).

Infrastructured wireless networks are typically used in locations where access points can be easily installed and connected to an existing network, such as office buildings and college campuses.

2.1.2 Infrastructureless/Ad Hoc Wireless Networks

There may be many instances in which mobile users may need to communicate with each other, and yet a fixed wired infrastructure may not be available. One example may be disaster recovery, in which the entire communication infrastructure may be destroyed, and restarting communication quickly is crucial. An infrastructure can be re-established in hours by using a mobile ad-hoc network, instead of weeks, as is required by a wired infrastructure. Such an interconnection between mobile computers does not require any pre-planned infrastructure, such as a base station, and is called an ad-hoc network.

An ad-hoc network is a network comprised solely of mobile wireless nodes. There is no wired backbone, and nodes communicate directly with one another and can also serve as relays for data packet forwarding. Such a network is often called a Mobile Ad-Hoc Network (MANET) [23, 3] and represents truly pervasive/ubiquitous computing, because in many situations, information exchange among mobile units cannot rely on any fixed network infrastructure but on the rapid configuration of wireless connections on the fly [45].

Features of MANET include:

1. Dynamic network topology. Nodes are mobile; therefore, network topology may change rapidly and unpredictably, and the connectivity among the nodes may vary with time.

2. Multi-hop routing. Routing algorithms can be single-hop and multi-hop. When delivering data packets from a source to a destination that is out of the direct wireless range of the source, packets may be forwarded via one or more intermediate nodes.

3. Fluctuating link capacity. The channel over which the nodes communicate is subject to fading, noise, and interference, and has less bandwidth than a wired network.

MANETS can be used in many types of applications, and can range from large-scale, mobile, highly dynamic networks to small, static, power-constrained networks. A few examples of such applications can be personal area network (PAN), commercial sector, military battlefield, and local level.

There are, however, several challenges that must be examined carefully before a widespread commercial deployment can be expected, including routing, security and reliability, quality of service, internetworking, and power consumption.

2.2 Wireless Sensor Networks

An overview of sensor nodes and sensor networks, as well as some key issues and concepts related to sensor networks, will be offered in this section. Rather than writing a detailed summary of related work, we will briefly describe some key issues with some references to these issues that are present in the literature.

2.2.1 Overview

Recent technological advancements have made it possible to deploy small, cheap, low-power, distributed sensing devices, which are capable of wireless communication and limited processing. These devices are called sensor nodes, and are very different from traditional desktop and server systems [30]. A collection of sensor nodes which co-ordinate amongst themselves to perform a larger sensing task is known as a sensor network. These sensor networks are composed of a large number of sensors and can measure a given aspect of their physical environment in great detail. The nodes are usually static; however, some or all nodes could be mobile.

Sensor nodes have the following constraints [54]:

1. Communication: The wireless connection between sensor nodes provides a limited quality of service due to latency with high variance, limited bandwidth, and frequently dropped packets.

2. Computation: Sensor nodes have limited computing power and memory.

3. Power consumption: Sensor nodes have a limited energy supply. Also, since sensor nodes may be deployed in inhospitable or inaccessible terrain, replacing or recharging sensors may be infeasible.

4. Uncertainty in sensor readings: Signals detected at physical sensors have an inherent uncertainty. They may contain environmental noise or may be biased due to sensor location.

5. Density: Sensor nodes are densely deployed and can range in density from a few sensor nodes to a few hundred sensor nodes in a region.

In addition to this, due to sensor node failure or movement of nodes, the topology of a sensor network may change frequently. A sensor network, therefore, should be self-healing, as well as self-organizing.

Networked sensors are both generators of data and routers. A sensor node can aggregrate such data. *Source* sensors detect critical events and are usually located where environmental events, that are of interest, occur. *Sink* nodes are connected to other networks, such as the Internet, and provide remote access to data from the sensor network. These sinks are monitoring terminals and may be mobile PDA's, laptops, or static access points.

2.2.2 Sensor Network Architecture and Applications

Each sensor node in a sensor network is equipped with a variety of sensors, including acoustic, seismic, still/motion videocamera, infrared, etc. Networked sensors can be organized in a cluster so that a locally occurring event can be detected by most, if not all, of the nodes in the cluster. Each cluster node can have enough processing power to process the data it collects, and broadcast any interpretation of this data to other nodes in the cluster. One node can act as a clusterhead, and it may also contain a longer range radio that uses a protocol such as IEEE 802.1 Bluetooth [5].

Many sensor network applications that change dynamically, such as battlefield and commercial inventory and distribution systems, must be controlled using adaptive methods that use real-time information gathered from integrated low-powered sensors and mobile devices deployed throughout the application. Despite dynamic changes in the topology of th sensor network, critical real-time information still must be disseminated dynamically from mobile sensor nodes through the network infrastructure to components that dynamically control the re-structuring and re-optimization of network operation based upon newly available information. In [37], three fundamental mechanisms upon which other networking and system services

may be spontaneously specified are service lookup, sensor node composition, and dynamic adaptation. A distributed implementation of these lookup servers, composition servers, and adaptation servers can be spontaneously defined in the sensor network. Different protocols for a certain service may be specified for different applications, and these protocols may interoperate through these three fundamental mechanisms provided in the sensor network architecture.

There are also three mobility-aware key system layers in the architecture of self-organizing sensor networks:

1. Application systems layer. This is the sensor information processing layer and collaborative signal processing layer.

2. Configurable distributed systems layer. This layer provides distributed services to the application systems.

3. Sensor networking and physical devices layer. This layer routes messages through the network and consists of the sensor nodes and other devices that generate the raw data.

The *Smart Dust* project at Berkeley [33, 42] exemplifies another system architecture in sensor networks. Its goal is to design a networked sensor that is limited in size and power resources. This sensor device, also known as smart dust, requires sensing, communication, and computing hardware, as well as a power supply, to occupy the space of a few cubic millimeters. The processor used is an ATMEL [2] 4MHz, 8bit micro-controller with 8 Kbytes of program memory and 512 bytes of data memory. It includes a radio with a single channel RF transceiver operating at 916 MHz and capable of transmitting at 10 Kbps using on-off-keying encoding [30, 51]. In [29], researchers introduced a tiny microthreaded OS, called Tiny OS, that provides the system software support to operate and manage this class of tiny smart devices.

Regardless of the architecture of a sensor network, there are many applications for such devices, such as healthcare, home, commercial, and military applications.

Other applications include environmental monitoring (e.g., habitat, traffic), industrial applications and diagnostics (e.g., managing inventory, product quality), and infrastructure maintenance (e.g., power grids, water distribution). One interesting application of sensor networks, given in [39], was the deployment of a sensor network on Great Duck Island in Maine, for habitat monitoring. The sensor networks deployed on this island was accessible via the Internet, used solar energy to power the sensors, and had a sensor longevity of 9 months. The sensor network was used to monitor the changes in the nesting patterns of Leach's Storm Petrel.

2.2.3 Power Awareness

Since the amount of available energy for a sensor node is limited, minimizing energy consumption in a sensor network is a critical challenge. In [12], the authors identify three main types of optimizations for reducing energy consumption in a sensor network. The first is to cover the monitoring area with the smallest subset of sensor nodes. Nodes not belonging to this set sleep and do not participate in the monitoring. Constructing a dominating node set that "monitor" other sensors within their coverage range is one example of this type of optimization. Also, the network can reselect covering nodes periodically to spread energy consumption dynamically over all nodes. The second optimization is to use energy-efficient broadcast protocols. Several protocols for minimizing retransmissions of messages sent from one sensor node to another have been proposed, including adjustable-transmission-range protocols. The third optimization is data aggregation. Aggregating measurements of sensor nodes in order to report only important information, such as average values, can also reduce energy consumption.

2.2.4 Data Dissemination

Since the energy consumption in a sensor network is dominated by the cost of transmitting and receiving messages, protocols for data dissemination are important. Data gathered from studies of popular prototypes of sensor network devices, such

as MICA2 [6], also verify the importance of reducing communication costs in sensor networks.

Various characteristics of algorithms used for the self-configuring and data dissemination of sensor networks reduce communication costs. One characteristic is that these algorithms must be data-centric (or the applications focus on the data generated by the sensors). Another characteristic is that the algorithms should be localized, meaning that the nodes communicate only with sensors that are close to their neighborhood. The nodes can achieve a global objective by using only local computations. Finally, networks can be application specific. This means that intermediate nodes can perform application-specific data aggregation and caching, or the informed forwarding of data requests.

One data-centric data dissemination paradigm is directed diffusion. In directed diffusion, data that is generated by sensor nodes is named by attribute-value pairs [32]. A sensing task is disseminated throughout the sensor network as an interest for named data. This dissemination creates gradients within the network that "draws" events (or data matching this interest). The events then start flowing towards the originators of the interests along multiple paths. One, or a small number of these paths, is reinforced by the sensor network. The intermediate nodes can cache or transform data, and can direct interests based on previously cached data.

In [27], a family of adaptive dissemination protocols, called SPIN (Sensor Protocols for Information via Negotiation), for wireless sensor networks was proposed. Meta-data negotiation and resource-adaptation is used by SPIN to overcome deficiencies in approaches such as flooding and gossiping. By assuming that all sensors can be sink nodes, SPIN focuses on the efficient dissemination of individual sensor data to all sensors in a network. In this manner, the fault tolerance of the system is increased. Also, an important piece of information can be disseminated to all the nodes. In SPIN, nodes negotiate with each other before transmitting data in order to avoid sending unnecessary data. Data is described by using meta-data in the

negotiation process, since exchanging meta-data is not as expensive as exchanging sensor data. Also, nodes poll their resources and energy before transmitting data, which allows sensors that lack energy to reduce certain activities. These characteristics of SPIN overcome problems like implosion (nodes consistently sending to their neighbors, regardless of whether or not they have already received data from another source), overlap (some nodes covering overlapping geographic areas), and resource blindness (nodes not modifying their activities based upon available energy), that are associated with simple flooding.

2.2.5 Time synchronization

A critical task in sensor networks (for various purposes such as sensor data fusion, coordinated actuation, and power-efficient duty cycling) is time synchronization. Mobile sensor devices equipped with clocks and short range radios can be deployed in the environment to measure various phenomenon. The devices can record the time during which they detect and no longer detect these phenomenon, and can communicate this information to other sensors as they pass by. The temporal ordering of these events (originating from different sensors) are used to determine the direction of the phenomenon, and difference in time between events originating from different devices are used to estimate the speed of the phenomenon. Also, time synchronization can be used to estimate the proximity of sensors by calculating the time when certain environmental phenomenon (e.g., sound or light) are sensed by different nodes. Sensor networks may also be used in many applications where accurate timekeeping is necessary. An example is the Network Time Protocol (NTP) [40] that is used to maintain Internet clocks.

Time synchronization can also be used to ensure collision-free communication in sensor networks. Collision-free communication is important because collided messages cannot be use, and collisions waste energy. In [28], the authors present a distributed TDMA slot assignment algorithm that is suitable for dynamic networks. The algorithm is self-stabilizing and uses Time Division Media Access methods to

schedule transmission in time slots to avoid collisions.

CHAPTER 3

SELF-∗ SYSTEMS

In this chapter, we will first start with an overview of self-∗ systems (Section 3.1). We will then describe many terms that are currently being used in the general area of fault-tolerant computing.

3.1 Overview

Software systems must be able to adjust to different inputs, adapt to all possible environmental changes, and handle different faults. The many concepts encapsulated in self-∗ have been introduced to detect, adjust, and recover from the above situations. We will informally describe these concepts with examples from the literature. We will also give an overview of the concept of self-stabilization in Section 3.3.

A distributed system [46] is defined as an interconnected collection of autonomous computers, processes, or processors (or nodes). In addition to this, the existence of the collection of these nodes must be transparent to the system users. The processors may also need to communicate with each other in order to coordinate their actions and achieve a reasonable level of cooperation. Many software systems being used for business-critical or other important applications are distributed systems. The term self-∗ may be applied to certain distributed systems.

A self-∗ system should be self-configuring, self-reorganizing, self-contained, self-healing, and self-managing. According to [20], "self-∗ distributed systems establish and maintain system-wide properties, e.g. properties such as being deadlock-free,

fault tolerant, or load-balanced". The authors describe self-* properties of distributed systems at the system-wide level using a method termed DRL (Distributed Reinforcement Learning).

A self-configuring system must be able to configure and reconfigure itself under varying conditions (faults). Also, a system is considered to be self-configuring if, starting from an arbitrary state and arbitrary input, the system will eventually satisfy the problem specification or start behaving properly. The term self-organizing was formally defined in [26]. In this paper, the authors apply this concept to a peer-to-peer system and define a locally self-organizing system in the context of a "p-stable" configuration.

A system is said to be self-contained if the number and location of nodes, affected by a faulty node, are minimally contained within the neighborhood of the faulty sensor. The term self-healing can refer to a system that can automatically recover form different pertubations and dynamic changes. Finally, a self-* system should be self-managing, meaning that all tasks in all phases in the life cycle of the system are automatic.

IBM's approach to solving the system management problem is called *autonomic computing* [1]. On October 15, 2001, Paul Horn, Senior Vice President of IBM Research, suggested that the solution was to "build computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies".

Another approach to building highly reliable systems is called *recovery-oriented* computing [22, 41]. Systems implementing this type of computing are called *self-repairing computers*. This concept can be applied to designing highly-dependable Internet services. A few important characteristics of recovery-oriented computing that have been identified are "system-wide support for undo", "isolation and redundancy", "integrated diagnosis support", "online verification of recovery mechanisms", "design for high modularity, measurability, and restartability", and "dependabil-

ity/availability benchmarking" [41].

3.2 Ubiquitous/Pervasive Computing

The late Mark Weiser introduced the term *ubiquitous computing* to describe an era in which many computers, that are "nearly invisible", are prevalent in large numbers in many areas of the physical environment. These computers are relatively inexpensive and are used so often by the user, that they are effectively invisible. Two key concepts of this era are *invisible computing* and *calm technology* [50]. These computers would be available and prevalent throughout the environment and would be used without the user actually having conscious recognition of their presence. In effect, the computers are "invisible" to the user. The motivation behind calm technology is to send information in a calm manner, meaning that a user's consciousness must be able to switch between peripheral (or sensory) processing and the center of processing, when using a computing or electronic device. New hardware representing the ubiquitous computing design include mobile devices, sensors, and even smart appliances.

3.3 Self-Stabilizing Systems

The concept of self-stabilization was introduced to computer science in 1973 by Dijkstra [17, 16]. A self-stabilizing system is one that can recover automatically following the occurrence of (transient) faults. A formal definition is as follows: A self-stabilizing system, starting from any arbitrary state, converges to a state that satisfies its problem specification in a finite number of steps. It can also be defined as follows: A self-stabilizing system, regardless of its initial state, reaches a state from which it starts behaving according to its specification in finite time. Two key concepts associated with self-stabilization are closure and convergence [9, 10]. Closure refers to a property in which, during all system executions, the system remains within some set of legal or desirable states unless a fault occurs. Convergence refers to a

property that requires the system to reach a legal state from any arbitrary (possibly illegal) state in finite steps. A self-stabilizing system must satisfy both the closure and convergence properties.

Many network protocols are self-stabilizing. They include protocols used in sensor networks, high-speed networks, session control, connection management, and routing. There are also many self-stabilizing distributed solutions for graph theory problems. Examples are maximal matching, finding different types of spanning trees, search structures, and graph coloring. In addition to this, there are self-stabilizing versions of many classical distributed algorithms, including mutual exclusion, token circulation, leader election, distributed reset, and propagation of information with feedback.

There are many aspects of a model that can be used for a self-stabilizing algorithm. This includes interprocess communication (shared registers and message passing), fairness (weakly fair, strongly fair, and unfair), granularity of an atomic step (composite versus read/write atomicity), and types of daemons (central and distributed). Many optimal solutions for the time complexity and space complexity of stabilizing algorithms have also been proposed.

There are two methods that have been commonly used for the proof of a self-stabilizing algorithm: the convergence stair [25] and variant function [34] methods. There are also many general methods of designing self-stabilizing programs, a few of which we will mention without description. They include silent stabilization [19], local stabilizer [8], diffusing computation [10], local checking and local correction [11, 47], counter flushing [48], self-containment [24], and snap-stabilization [14].

The protocols for setting up and organizing communication and routing infrastructures in wireless sensor networks are often based upon self-stabilizing algorithms. Self-stabilization is important for this purpose because of the dynamic nature of sensor network topology. Node and link failures, as well as the joining of new nodes in the sensor network, necessitate the use of a self-stabilizing algorithm.

CHAPTER 4

MINIMUM CONNECTED SENSOR COVER PROBLEM

After extensively researching wireless sensor networks and self-$*$ systems, we designed three local, distributed, self-$*$ protocols in order to solve the *minimum connected sensor cover* problem. We state the motivation of this research in the next section. We state how other problems mentioned in earlier chapters are related to the problem solved in this chapter. We describe some results in related areas in Section 4.2. In section 4.3, we first state the model used in writing the algorithm. We present the program that is used (including its notation) and give a formal definition of self-stabilization in that section. Finally, we give both an informal explanation and formal statement of the problem to be solved in that section.

The main results of this thesis research are reported in the next four chapters. In Chapters 5, 6, and 7, three minimum connected sensor cover algorithms (Algorithm 1 $\mathcal{MCSC}$, Algorithm 2 $\mathcal{MCSC}$, and Algorithm 3 $\mathcal{MCSC}$) are presented. In each of these three chapters, we include a detailed informal description, formal algorithm, and proof of the algorithm in that section. Simulation results and other properties of all three algorithms are given in Chapter 8.

4.1 Motivation

Sensor networks are composed of a large number of tiny sensing devices with very limited resources that must coordinate amongst themselves to achieve a larger sensing task. As mentioned in Chapters 1 and 2, these networked sensors are often

energy constrained, since a sensor's battery or energy source is small and replacing or recharging a sensor's energy supply is often infeasible. Therefore it is critical to design a robust sensor network which will allow uninterrupted operation for extended periods of time, and that is also efficient in its consumption of energy. Also, considering the size and dynamic nature of sensor networks, it is important that a sensor network be designed as a self-$*$ system (Chapter 2).

In sensor networks, queries may be sent from devices external to the network. The query needs to be broadcast to the sensor nodes within a particular region or to a particular sensor node. This would initiate the minimum connected sensor cover algorithm. Also, after the minimum connected sensor cover is computed, the data generated as a result of the query has to be reported back to the device which originated this query.

4.2 Related Work

The minimum connected sensor cover problem that is addressed in this thesis was introduced in [26]. Even though two self-organizing solutions were presented in that paper, none of the solutions were localized. Both algorithms use a greedy approach to select the best possible set of sensors in the cover set.

In [49], the terms coverage and connectivity and the relationship between them were analyzed in a unified framework. A Coverage Configuration Protocol (CCP) that can dynamically configure networks to provide different degrees of coverage was presented in this paper. CCP was integrated with a connectivity maintenance protocol (SPAN [13]) to provide guarantees of both coverage and connectivity. The integrated coverage and connectivity problem solved in this paper is as follows: Given a coverage (or query) A and a sensor coverage degree K specified by the application, we must maximize the number of sleeping nodes such that :

1. A is at least K_S-covered (i.e., every location inside A is covered by at least K_S nodes), and

2. All active nodes are connected.

The important result of their work was that:

1. Sensing coverage implies network connectivity when $R_C \geq 2R_S$ (where R_C and R_S are the communication and sensing ranges, respectively) and

2. If $R_C \geq 2R_S$, then K_S-coverage of a convex region implies K_S-connectivity of the communication graph.

Wu and Li [53, 52] proposed a *marking process* which can determine a CDS by marking each host in a routing scheme if it has two unconnected neighbors. Two dominant pruning rules were proposed in [53] and extended in [52] to reduce the size of the CDS derived from this marking process. Rule 1 unmarks a host u if its neighbor set is covered by another marked host v and its UID is less than that of host v; that is, if all of its neighbors are neighbors of another marked host having a greater UID than its own. Rule 2 unmarks a host if its neighborhood is covered by two other directly connected marked hosts, and if its UID is less than both of these hosts. However, these pruning rules do not account for host u itself, which should also be covered by a marked node before it is unmarked. In all three algorithms presented in this paper, to ensure connectivity, a Node i must also be covered by a *chosen* node, having a greater UID than its own and for which it is not the "least UID" neighbor, before it is unmarked. Also, in both Rule 1 and Rule 2, Node u has to have the least UID of all nodes that are covering its neighbor set, before it is unmarked. This is a weaker redundancy predicate than the ones presented in this paper, since in Algorithms 2 and 3, all sensors that are neighbors of Sensor i must be neighbors of a *chosen* sensor, but Sensor i does not have to have the smallest UID of all of the nodes that are covering its neighbor set. It merely has to have a smaller UID than a *chosen* node that is covering itself. Also, in Algorithm 1, nodes that are neighbors of a *chosen* Sensor i are not considered in the redundancy predicate.

Dai and Wu [15] proposed a generic dominant pruning rule (called Rule k), which can unmark gateways covered by k other gateways, where k can be any number. Again, in this rule, Node u must have the least UID of all nodes that are covering its neighbor set, before it is unmarked. Because this rule is weaker than our redundancy predicates, Algorithms 1 and 2 produce fewer nodes in the final cover set at all query regions tested in our simulations, and Algorithm 3 produces fewer nodes in the final cover set when the query region size is less than 90 square graph units.

Carle and Simplot-Ryl [12] presented a dominating-set protocol in which the nodes that cover an "inactive" node's neighborhood have to be connected if this inactive node is to remain inactive. Our algorithms' redundancy predicates are stronger since they do not require that all *chosen* nodes that cover a marked node be connected before the *chosen* node is unmarked. Instead, our algorithms ensure connectivity by not unmarking the sensor with the greatest or the least UID within any particular *chosen* sensor's transmission disk.

The algorithm presented by Kuhn, Moscibroda, and Wattenhofer [35] relies upon sending messages on three separate channels. In this algorithm, a newly awakened node waits for messages on all three channels from existing dominators in its neighborhood. A node that has not received any message from a dominator during this waiting phase then tries to compete to become a dominator itself. This node then sends a message on the first channel with a sending probability p, which is doubled in every round. After becoming a dominator, a node then sends on the second and third channels. However, the chance of collisions on a transmission channel can cause a node to not receive a message in the waiting phase and can lead to a larger number of dominators.

Liu et al. [38] recently proposed an iterative localized algorithm for connected dominating sets, offering an improvement over [15] in terms of the size of connected dominating sets, but at the expense of additional messages between neighboring nodes. In their algorithm, each node exchanges messages with its neighbors (there

are exactly 5 messages exchanged) in order to decide whether it should be dominant, using information received from its neighbors. However, the synchronization needed to compute a dominating set make it more difficult to apply in a distributed environment. Also, beacon messages are needed for the first step to occur.

Ingelrest, Ryl, and Stojmenovic [31] proposed an algorithm which considers a node to be covered if there exists in its 2-hop neighborhood, a connected set of nodes with higher priorities which cover Node u and its 1-hop neighbors. However, this is also a weaker redundancy predicate than the ones presented in this paper, since in Algorithms 2 and 3, all sensors that are neighbors of Sensor i must be neighbors of a *chosen* sensor, but Sensor i does not have to have the smallest UID of all of the nodes that are covering its neighbor set. It merely has to have a smaller UID than a *chosen* node that is covering itself. Also, in Algorithm 1, nodes that are neighbors of a *chosen* Sensor i are not considered in the redundancy predicate.

4.3 Preliminaries

4.3.1 Model

Sensor Network. In this research, we consider *sensor networks* [26, 49] consisting of a large number of sensors (also referred to, in this paper, as *sensor nodes* or, simply as *nodes*) which are randomly distributed in a geographical region. We model the sensor network as a *directed* communication graph $G(V, E)$, where each node in V represents a sensor, and each edge $(i, j) \in E$, called *communication edge*, indicates that j is a *neighbor* of i.

For a sensor i, there is a region, called a *sensing region*, which signifies the area in which sensor i can sense a given physical phenomenon at a desired confidence level. The sensing regions are of any convex shape. For the sake of simplicity, especially, for showing examples, the sensing regions are assumed to be circular. The *sensing range* of a sensor i indicates the maximum distance between sensor i and any point p in the sensing region of sensor i. A point p is *covered* (or monitored) by a sensor

node i if the Euclidean distance between p and i is less than the sensing range of sensor i.

The *communication region* of sensor i (also called the *transmission region*) defines the area in which sensor i can communicate directly (i.e., in single hop) with other sensor nodes. The maximum distance between node i and any other node j, where j is in the communication region of i, is called the *communication range* of sensor i. Node i can communicate with node j (i.e., i can send a message to j) if the Euclidean distance between them is less than the communication range of i. Then i is called a neighbor of j, and this relation is represented by a directed edge (i, j). The set of neighbors of i is represented by N_i. Two nodes i and j can communicate directly with each other only if $i \in N_j \wedge j \in N_i$, i.e., they are neighbors of each other. If i and j are neighbors of each other, then there are two edges between them: (i, j) and (j, i).

A directed path (sequence) of sensors $i = i_1, i_2, \ldots, i_m = j$, where i_x is a neighbor of i_{x+1} for $1 \leq x \leq m-1$, is called a *communication path* from i to j. The length of the shortest (communication) path (which is the number of sensors on the shortest path) from i to j is called the *communication distance* from sensor i to sensor j.

Program. In this paper, we consider the local shared memory model of communication as used by Dijkstra [16]. The program of every processor consists of a set of *shared variables* (henceforth, referred to as variables) and a finite set of actions. Every processor (or sensor) can only write to its own variables, but can read its own variables and the variables owned by the neighboring nodes.

Each action is of the following form: $< label >:: < guard > \longrightarrow < statement >$. The guard of an action in the program of p is a boolean expression involving the variables of p and its neighbors. The statement of an action of p updates one or more variables of p. An action can be executed only if its guard evaluates to true. We assume a model of *composite atomicity*; i.e., actions are atomically executed,

or the evaluation of a guard and the execution of its corresponding statement, if executed, are done in one atomic step.

The *state* of a node is defined by the values of its variables. The *state* of a system is the product of the states of all nodes. We will refer to the state of a node and system as a (*local*) *state* and (*global*) *configuration*, respectively.

Let a distributed protocol $\mathcal{P}$ be a collection of binary transition relations denoted by $\mapsto$, on $\mathcal{C}$, the set of all possible configurations of the system. A *computation* of a protocol $\mathcal{P}$ is a *maximal* sequence of configurations $e = \gamma_0, \gamma_1, ..., \gamma_i, \gamma_{i+1}, ...$, such that for $i \geq 0, \gamma_i \mapsto \gamma_{i+1}$ (a single *computation step*) if γ_{i+1} exists, or γ_i is a terminal configuration. The *Maximality* means that the sequence is either infinite, or it is finite and no action of $\mathcal{P}$ is enabled in the final configuration. All computations considered in this paper are assumed to be maximal. The set of all possible computations of $\mathcal{P}$ in system S is denoted as $\mathcal{E}$. A node p is said to be *enabled* in γ ($\gamma \in \mathcal{C}$) if there exists an action A such that the guard of A is true in γ. We consider that any node p executed a *disable action* in the computation step $\gamma_i \mapsto \gamma_{i+1}$ if p was enabled in γ_i and not enabled in γ_{i+1}, but did not execute any action between these two configurations. (The disable action represents the following situation: At least one neighbor of p changed its state between γ_i and γ_{i+1}, and this change effectively made the guard of all actions of p false.) Similarly, an action A is said to be enabled (in γ) at p if the guard of A is true at p (in γ).

We assume a *weakly fair and distributed daemon. Weak fairness* means that if a node p is continuously enabled, then p will be eventually chosen by the daemon to execute an action. A *distributed* daemon implies that during a computation step, if one or more nodes are enabled, then the daemon chooses at least one (possibly more) of these enabled nodes to execute an action.

4.3.2 Self-stabilizing Program

Fault Model. This research deals with the following types of faults:

(i) The state or configuration of the system may be arbitrarily corrupted. However, the program (or code) of the algorithm cannot be corrupted.

(ii) Nodes may crash. That is, faults can fail-stop nodes.

(iii) Nodes may recover or join the network.

The topology (both actual and logical topologies) of the sensor network may change due to these faults. Faults may occur in any finite number, in any order, at any frequency, and at any time.

Closure: $\mathcal{R}$ is *closed* in $\mathcal{A}$ if every computation of $\mathcal{A}$ starting from a configuration satisfying $\mathcal{R}$ preserves $\mathcal{R}$.

Convergence: $\mathcal{R}$ *convergences* to $\mathcal{S}$ in $\mathcal{A}$ if the following three conditions hold:

1. $\mathcal{R}$ is *closed* in $\mathcal{A}$.
2. $\mathcal{S}$ is *closed* in $\mathcal{A}$.
3. Every computation starting from a configuration satisfying $\mathcal{R}$ contains a configuration that satisfies $\mathcal{S}$.

Self-stabilization [18]. Let $\mathcal{L}_{\mathcal{A}}$ be a non-empty *legitimacy predicate* of an algorithm $\mathcal{A}$ with respect to a specification predicate *Spec* such that every configuration satisfying $\mathcal{L}_{\mathcal{A}}$ satisfies *Spec*. Algorithm $\mathcal{A}$ is *self-stabilizing* with respect to *Spec* iff the following two conditions hold:

(i) Every computation of $\mathcal{A}$ starting from a configuration satisfying $\mathcal{L}_{\mathcal{A}}$ preserves $\mathcal{L}_{\mathcal{A}}$ (*closure*).

(ii) Every computation of $\mathcal{A}$ starting from an arbitrary configuration contains a configuration that satisfies $\mathcal{L}_{\mathcal{A}}$ (*convergence*).

4.3.3 Problem Specification

Specification 0.0.1 (Connected Sensor Coverage Problem). *Given a sensor network and a query Q over the network, the connected sensor coverage problem is to find the smallest connected sensor cover (we will call it $\mathcal{MCSC}_Q$). Additionally, we require the algorithm (solving the above problem) to be self-organizing, self-healing, and self-stabilizing.*

CHAPTER 5

FIRST $\mathcal{MCSC}$ ALGORITHM

5.1 Description of First $\mathcal{MCSC}$ Algorithm and Data Structures Used

In this algorithm, the following strategy is taken to compute the minimum connected sensor cover $MCSC_Q$:

1. Algorithm 1 finds an $MCDS$ (Minimum Connected Dominating Set) for all nodes whose sensing range intersect with the query region. The $MCDS$ that is calculated does not include another $MCDS$, but is not minimal in the number of nodes in the set. However, the sensing range of all the nodes in the $MCDS$ will cover the query region. The $MCSC$ that is formed from all sensors in this $MCDS$ is minimum such that another connected sensor cover set is not included in this set.

The following assumptions are made for this algorithm:

Assumption 0.0.1.

(i) The communication radius equals the sensing radius for the sensors.

(ii) The sensing radii, and hence the communication radii, of all sensors are equal.

(iii) There always exist a sufficient number of sensors in the network with sufficient density to cover the query region if all of them are deployed.

(iv) There exist a lot of redundant sensors which are either boundary or interior sensors with respect to the query region.

The algorithm uses three shared variables, S_i, UID_i, and $Status_i$. S_i represents the sensing region of Sensor i. UID_i is the unique identifier (UID) of Sensor i, which is a positive integer. Finally, $Status_i$ represents the status of a sensor. The status of a sensor may be *unchosen*, *undecided*, or *chosen*.

5.2 Predicates Used in First $\mathcal{MCSC}$ Algorithm

The predicate $QryRgnIntrsctn(i)$ evaluates to true if the sensing disk of Sensor i intersects with **some** portion of the query region. $NoIntrsctn(i, j)$ evaluates to true if Sensor i has a status of *unchosen*, there are no *chosen* sensors within the transmission disk of Sensor i, and if the sensing disks of Sensor i and any *chosen* Sensor j do not intersect. $NgbrOfChsn(i)$ evaluates to true if Sensor i is a neighbor of a *chosen* sensor. $HasChsnNgbr(x)$ evaluates to true if Sensor x has a *chosen* neighbor. The predicate, $IsLeastUIDNgbr(i, x)$, evaluates to true if Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID. $LessNotLeastNgbrOfChsn(i)$ evaluates to true if Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID. The predicate $NotOrLeastUIDNgbrOfChsn(i)$ evaluates to true if Sensor i is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID.

$MISNode(i)$ evaluates to true if the status of Sensor i is *unchosen*, and the sensing disk of Sensor i intersects with some portion of the query region, but does not intersect with the sensing disk of a *chosen* sensor. $BridgeNode(i)$ evaluates to true if the status of Sensor i is *unchosen*, the sensing disk of Sensor i intersects with some portion of the query region, Sensor i is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID, or if part of the transmission disk of Sensor i is not covered by a *chosen* sensor. The predicate $FillNode(i)$ evaluates to true if the status of Sensor i is *undecided*, and there are no *undecided* sensors within the transmission disk of Sensor i whose UID is greater

than that of Sensor i, or Sensor i is the neighbor of an *undecided* sensor having the least UID.

$Redundant_1(i)$ evaluates to true if the status of Sensor i is *undecided*, there is an *undecided* sensor within the transmission disk of Sensor i whose UID is greater than that of Sensor i, and Sensor i is not the neighbor of this *undecided* sensor having the least UID. Finally, $Redundant_2(i)$ evaluates to true if the status of Sensor i is *chosen*, Sensor i has a smaller UID than another *chosen* Sensor j that is within its transmission disk, but Sensor i does not have the smallest UID out of all the neighbors of Sensor j.

5.3 Normal Execution of First $\mathcal{MCSC}$ Algorithm

The steps of the algorithm are as follows:

1. The algorithm attempts to form an initial pattern of coverage of the query region that is composed of the union of the sensing radii of sensors whose status is *chosen*. These sensing regions also form a disjoint set, in the sense that no two sensing disks within this set intersect. To this end, it changes the status of all *unchosen* sensors whose sensing regions intersect with the query region, and whose sensing regions do not intersect with the sensing region of a *chosen* sensor, to *chosen*. Thus, an initial pattern of non-overlapping sensing disks, whose sensors are marked as *chosen*, is formed to cover the query region.

2. The uncovered regions between the sensing radii of all *chosen* sensors is then covered as follows:

   (a) If the status of Sensor i is *unchosen*, the sensing disk of Sensor i intersects with some portion of the query region, and Sensor i is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID, or if part of the transmission disk of Sensor i is not covered by a *chosen*

sensor, then the *unchosen* sensor's status is changed to *undecided.* The reasoning used is that all sensors that lie within the uncovered "gap" regions between the sensing radii of all *chosen* sensors that were marked by $MISNode(i)$, will have part of their sensing disks not covered by the sensing disks of all sensors chosen by $MISNode(i)$. In addition to this, all sensors that have the least UID, within a particular *chosen* node's neighborhood, are needed to ensure connectivity, and also have their status changed to *undecided.*

(b) To ensure that only the most suitable of these sensors, located within **each** uncovered region, are marked as *undecided*, if any sensor's status is *undecided*, and it has another *undecided* sensor within its transmission (and hence its sensing) disk, whose UID is greater than that of it's own, or if this sensor is the neighbor of an *undecided* sensor and does not have the least UID of all neighbors of this *undecided* sensor, then it's status is changed to *unchosen.*

(c) All sensors with an *undecided* status, that do not have another *undecided* sensor with a UID greater than their own, within their transmission (and hence sensing) disks, and that are not the neighbors of an *undecided* sensor and that also have the least UID of all neighbors of this *undecided* sensor, have their status changed to *chosen.*

3. $Redundant_2(i)$ is used to eliminate any redundant *chosen* sensor that has a smaller UID than another *chosen* Sensor j that is within its transmission disk, but that does not have the smallest UID out of all the neighbors of Sensor j.

4. Finally, action $\mathcal{A}_1$ ensures that any redundant sensor or any sensor whose sensing disk does not intersect with the query region, has its status changed to *unchosen.*

5. All *chosen* sensors are in the final MCDS.

5.4 Faults and Recovery of First $\mathcal{MCSC}$ Algorithm

In this section, we focus on the fault handling features of the proposed algorithm (Algorithm 1 $MCSC$). There are three variables used in the solution: S_i, UID_i, and $Status_i$ for a Sensor i. So, we need to show that our solution can cope with all possible corruptions associated with these three variables. In the following, we will make an attempt to list all or most of the important types of faults, and show how they are dealt with in Algorithm 1 $MCSC$. **(1) Wrong initialization of the $Status_i$ variable.** As discussed in the previous subsection, all sensors, if properly initialized, start as *unchosen*. *(a) Sensor i is initialized to* ***undecided*** . Assume that Sensor i is initialized to *undecided*. If i is not a redundant node, then i remains *undecided*, and subsequently changes to *chosen*. (see Actions $\mathcal{A}_2$ and $\mathcal{A}_3$). That is, no correction is necessary. If i is redundant, then it will satisfy the predicate $Redundant_1(i)$ and will change to *unchosen*. *(b) Sensor i is initialized to* ***chosen*** . If the sensing disk of Sensor i does not intersect with the query region, then, by executing $\mathcal{A}_1$, Sensor i will change to *unchosen*. So, no correction is necessary. If Sensor i is redundant, then then it will satisfy the predicate $Redundant_2(i)$, and will change to *unchosen*. If it is nonredundant then Sensor i is necessary, either to ensure coverage or connectivity, and should not be unmarked. **(2) Wrong initialization of the UID_i variable.** *(a) Sensor i is initialized to a UID that is used to identify another Sensor*. If Sensor i is redundant, then any other Sensor within the transmission disk of Sensor i, that has a larger UID than Sensor i, will cause Sensor i to evaluate $Redundant(i)$ as true and to become unmarked. If it is nonredundant, then Sensor i is needed in the final cover set, and should not be unmarked. **(3) Weakening or Failure of sensors, both in terms of communication and sensing ability.** The weakening or failure of sensors will affect the sensing and communication range of the sensors. In other words, the constant set R_S or R_C will change. Change of R_S or R_C may change the values of $Redundant(i)$, $MISNode(i)$, $BridgeNode(i)$, or $FillNode(i)$. All these changes will be reflected in the change of values of the guards

of the corresponding actions. So, eventually, the status of the affected nodes will change due to the execution of these actions. However, these changes will not affect the execution of these actions by the neighbors of the affected nodes. Therefore, any changes in the $Status_i$ variable of the affected nodes will be handled as mentioned earlier.

Algorithm 1 Connected Sensor Cover Algorithm (Algorithm 1 $\mathcal{MCSC}$) for Sensor i.

Constants:

R_Q:: Query region;

R_C:: Radius of communication of a sensor in the network;

N_i:: Set of sensors within the communication range of Sensor i;

Shared Variables:

S_i:: Sensing region of Sensor i;

UID_i:: Unique user identification number of Sensor i;

$Status_i \in \{unchosen, undecided, chosen\}$:: Status of Sensor i;

Predicates:

$QryRgnIntrsctn(i) \equiv S_i \cap R_Q \neq \emptyset$;
$\equiv$ sensing disk of Sensor i intersects with some portion of query region;

$NoIntrsctn(i,j) \equiv Status_i = unchosen \wedge (\forall j \in N_i : Status_j \neq chosen) \wedge (\forall j : Status_j = chosen \wedge \forall x \in N_i \wedge \forall y \in N_j : Pos_x \notin (S_i \cap S_j) \wedge Pos_y \notin (S_i \cap S_j))$;
$\equiv$ status of Sensor i is *unchosen*, there are no *chosen* sensors within the transmission disk of Sensor i, and sensing disks of Sensor i and Sensor j do not intersect;

$NgbrOfChsn(i) \equiv (\exists j : i \in N_j \wedge Status_j = chosen)$;
$\equiv$ Sensor i is a neighbor of a chosen sensor;

$IsLeastUIDNgbr(i,x) \equiv i \in N_x \wedge (\forall j \in N_x : j \neq i \wedge UID_i < UID_j)$;
$\equiv$ Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID;

$LessNotLeastNgbrOfChsn(i) \equiv (\exists j : i \in N_j \wedge Status_j = chosen \wedge UID_i < UID_j \wedge \neg IsLeastUIDNgbr(i,j))$;
$\equiv$ Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID;

$MISNode(i) \equiv QryRgnIntrsctn(i) \wedge NoIntrsctn(i,j)$;
$\equiv$ status of Sensor i is *unchosen*, and the sensing disk of Sensor i intersects with some portion of the query region, but does not intersect with the sensing disk of a *chosen* sensor;

$NotOrLeastUIDNgbrOfChsn(i) \equiv \forall j : i \in N_j : (Status_j \neq chosen \vee LeastUIDNgbr(i,j))$;
$\equiv$ Sensor i is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID;

$BridgeNode(i) \equiv Status_i = unchosen \wedge QryRgnIntrsctn(i) \wedge (NotOrLeastUIDNgbrOfChsn(i) \vee (\exists j \in N_i : \neg NgbrOfChsn(j)))$;
$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of the query region, Sensor i is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID, or part of the transmission disk of Sensor i is not covered by a *chosen* sensor;

Algorithm 1 Connected Sensor Cover Algorithm (Algorithm 1 $\mathcal{MCSC}$) for Sensor i (Continued)

$FillNode(i) \equiv Status_i = undecided \wedge (\forall j \in N_i : Status_j \neq undecided \vee UID_i > UID_j \vee LeastUIDNgbr(i,j))$;

$\equiv$ status of Sensor i is *undecided*, and there are no *undecided* sensors within the transmission disk of Sensor i whose UID is greater than that of Sensor i, or Sensor i is the neighbor of an *undecided* sensor having the least UID;

$Redundant_1(i) \equiv Status_i = undecided \wedge (\exists j \in N_i : Status_j = undecided \wedge UID_i < UID_j \wedge \neg LeastUIDNgbr(i,j))$;

$\equiv$ status of Sensor i is *undecided*, there is an *undecided* sensor within the transmission disk of Sensor i whose UID is greater than that of Sensor i, and Sensor i is not the neighbor of this *undecided* sensor having the least UID;

$Redundant_2(i) \equiv Status_i = chosen \wedge LessNotLeastNgbrOfChsn(i)$;

$\equiv$ status of Sensor i is *chosen*, Sensor i has a smaller UID than another *chosen* Sensor j that is within its transmission disk, but Sensor i does not have the smallest UID out of all the neighbors of Sensor j.

$Redundant(i) \equiv Redundant_1(i) \vee Redundant_2(i)$;

Actions:

$\mathcal{A}_1 :: \neg QryRgnIntrsctn(i) \vee Redundant(i)$

$\longrightarrow Status_i = unchosen$;

$\mathcal{A}_2 :: BridgeNode(i)$

$\longrightarrow Status_i = undecided$;

$\mathcal{A}_3 :: MISNode(i) \vee FillNode(i)$

$\longrightarrow Status_i = chosen$;

5.6 Correctness of First $\mathcal{MCSC}$ Algorithm

Definition 0.0.1. *The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$) if the following conditions are true with respect to a query region:*

i) All non-redundant sensors are marked chosen.

ii) All redundant sensors are marked unchosen.

5.6.1 Proof of Closure

Lemma 0.0.1 (Coverage). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm 1 $\mathcal{MCSC}$ completely covers the query region R_Q.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 1 do not completely cover the query region.

$\Rightarrow$ Since the sensing disks of the sensors chosen by $BridgeNode(i)$ and $FillNode(i)$ cover the uncovered regions between the sensing disks of sensors chosen by $MISNode(i)$ that form the initial Maximal Independent Set of Coverage, there exists a region between the sensing disks of the sensors chosen by $MISNode(i)$ that is not covered by the sensing disk(s) of one or more sensors that should be chosen by the $BridgeNode(i)$ and $FillNode(i)$ predicate.

$\Rightarrow$ Within the query region, there is no *unchosen* sensor that is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID, or that has part of its transmission disk not covered by a *chosen* sensor.

$\Rightarrow$ Within the query region, all *unchosen* sensors are neighbors (that may not have the least UID) of a *chosen* sensor and also have all parts of their transmission disk covered by *chosen* sensors.

$\Rightarrow$ Since all sensors are initially *unchosen*, the query region is completely covered by the sensing disks of *chosen* sensors.

Hence we arrive at a contradiction.

Alternatively, there is a sensor that is not the neighbor of a *chosen* sensor unless it is the neighbor of a *chosen* sensor having the least UID, or that has part of its transmission disk not covered by a *chosen* sensor, but this sensor was marked *undecided* and then marked *unchosen* by the $Redundant(i)$ predicate or was not marked *chosen* by the $FillNode(i)$ predicate.

Case 1:

The sensors in the Maximal Independent Set chosen by the $MISNode(i)$ predicate formed an initial pattern of coverage in which there are two uncovered regions between the sensing disks of four of these sensors. Figure 6.1 is an illustration of this case.

$\Rightarrow$ Since the graph is densely populated, we can find two sensors in both of these uncovered regions, let's name them Sensor A and Sensor B, such that Sensor A has a lesser UID than Sensor B, but Sensor A does not have the least UID of all neighbors of Sensor B, and both Sensor A and Sensor B have no other *undecided* sensors within their transmission disks

$\Rightarrow$ Since both nodes are not the neighbors of *chosen* sensors, both nodes must have been marked *undecided* and either node or both nodes were marked *unchosen* by $Redundant_1(i)$ or were not marked *chosen* by $FillNode(i)$.

$\Rightarrow$ Since Sensor A and Sensor B are both *undecided*, Sensor A has a lesser UID than Sensor B, and Sensor A is not the least UID neighbor of Sensor B, Sensor A

and Sensor B must be neighbors.

$\Rightarrow$ Sensor A and Sensor B are located within each other's communication disk.

$\Rightarrow$ The distance between Sensor A and Sensor B is less than or equal to the radius of communication.

$\Rightarrow$ If we let $R_C = 1$, in Figure 6.2, $\overline{AB} < 1$.

$\Rightarrow$ Since $\overline{CE} = \overline{CF} = \overline{EF} = 2$, then $\triangle CEF$ is an equilateral triangle.

$\Rightarrow$ If we bisect $\angle FCE$, $\triangle CGF$ is a 30-60-90 triangle.

$\Rightarrow cos\ 30° = \frac{\overline{CG}}{2r}$

$\Rightarrow \frac{\sqrt{3}}{2} = \frac{\overline{CG}}{2r} \Rightarrow 2\overline{CG} = 2\sqrt{3} \Rightarrow \overline{CG} = \sqrt{3} \Rightarrow \overline{CD} = 2\sqrt{3}$

Similarly, $\triangle CAH$ is a 30-60-90 triangle

$\Rightarrow cos\ 30° = \frac{r}{r+\overline{IA}}$

$\Rightarrow \frac{\sqrt{3}}{2} = \frac{r}{r+\overline{IA}}$

$\Rightarrow cos\ 30° = \frac{r}{r+\overline{IA}}$

$\Rightarrow \frac{\sqrt{3}}{2} = \frac{r}{r+\overline{IA}}$

$\Rightarrow \sqrt{3}r + \sqrt{3}\ \overline{IA} = 2r$

$\Rightarrow \sqrt{3}\ \overline{IA} = 2 - \sqrt{3}$

$\Rightarrow \overline{IA} = \frac{2-\sqrt{3}}{\sqrt{3}}$

$\Rightarrow \overline{IA} = \frac{2}{\sqrt{3}} - 1$

$\Rightarrow \overline{CA} = 1 + (\frac{2}{\sqrt{3}} - 1)$

$\Rightarrow \overline{CA} = \frac{2}{\sqrt{3}} = \overline{CA} = \frac{2\sqrt{3}}{3}$

Since $\overline{BD} = \overline{CA}$

$\Rightarrow \overline{AB} = \overline{CD}$ - $2\overline{CA} = 2\sqrt{3} - 2(\frac{2\sqrt{3}}{3}) = 2\sqrt{3} - \frac{4\sqrt{3}}{3} = \frac{2\sqrt{3}}{3}$

$\Rightarrow \overline{AB} > 1$.

Hence we arrive at a contradiction.

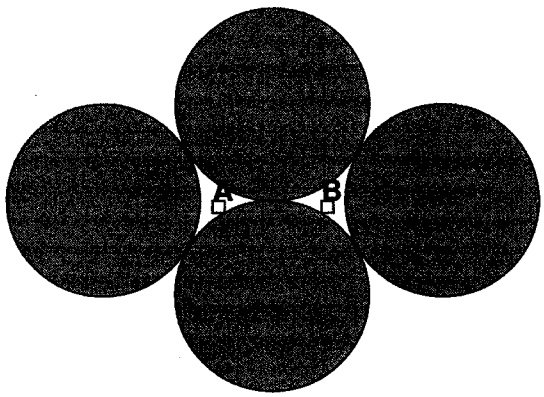


Figure 1. Case 1 (Two uncovered gap regions between 4 *chosen* sensors).

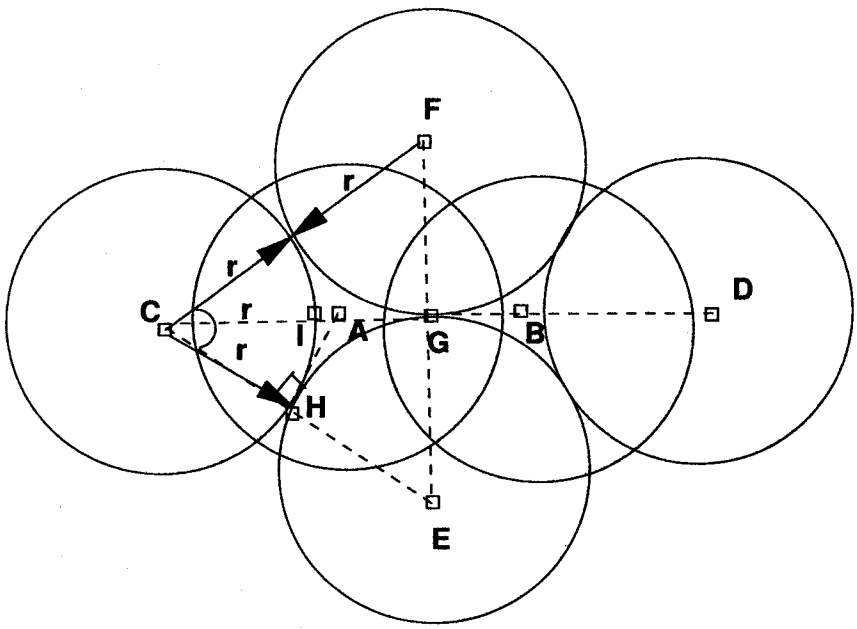


Figure 2. Case 1 ($\overline{AB} > R_C$).

Case 2:

The sensors in the Maximal Independent Set chosen by the MISNode(i) predicate formed an initial pattern of coverage in which there is one uncovered region between the sensing disks of four of these sensors. An optimal MIS satisfying this case is shown in Figure 6.3.

$\Rightarrow$ If we let Node A be the sensor in the uncovered region between the four sensors and Node B be a sensor in an uncovered region outside of the four sensors, then by similar reasoning as Case 1, the distance between Node A and Node B is less than or equal to R_C.

$\Rightarrow$ If we let $R_C = 1$, in Figure 3 $\overline{AB} < 1$

$\Rightarrow$ Since $\overline{AB} = \overline{CD}$, $\overline{AB} = 2R_C = 2$

$\Rightarrow \overline{AB} > 1$

Hence we arrive at a contradiction.

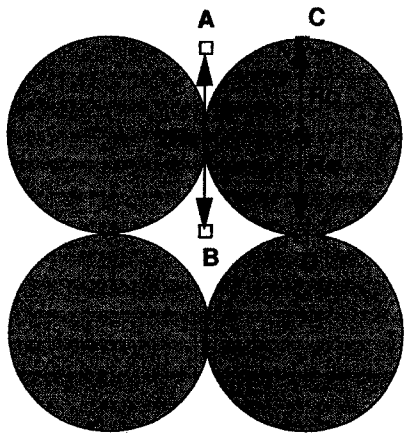


Figure 3. Case 2 (An "optimal" MIS in which Sensor A and Sensor B are not neighbors).

Case 3:

A suboptimal MIS in which there is one uncovered region between the four nodes is shown in Figure 6.4.

Since the graph is densely populated, there will be more than one *unchosen* node in each uncovered region.

$\Rightarrow$ There may be many *undecided* sensors that are not neighbors of *chosen* sensors, unless they are the "least UID" neighbors of *chosen* sensors, or that have part of their transmission disks not covered by *chosen* sensors.

$\Rightarrow$ Predicate $FillNode(i)$ and $\mathcal{A}_3$ will mark these nodes as *chosen* and $Redundant_1(i)$ will not unmark these nodes.

$\Rightarrow$ Since the sensing disk of each of these sensors spans a distance of $2R_C$, and yet each sensor will remain *chosen* if it is located at a distance of greater than one R_C from another *chosen* node, each of these uncoverd regions will eventually be covered by *chosen* nodes and will remain covered by these *chosen* nodes.

Hence we arrive at a contradiction. □

Lemma 0.0.2 (Connectivity). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm 1 $\mathcal{MCSC}$ forms a connected graph.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 1 do not form a connected subgraph.

$\Rightarrow$ There exists a sensor in the final $\mathcal{MCSC}$, lets name it Sensor A, that is marked *chosen* and is not adjacent to another *chosen* sensor.

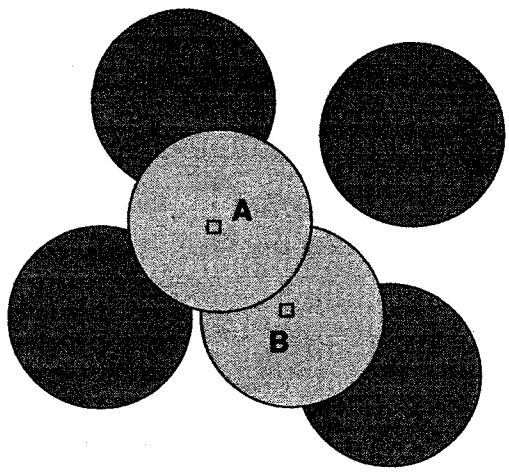


Figure 4. Case 3 (A "suboptimal" MIS in which Sensor A and Sensor B are not neighbors).

$\Rightarrow$ Sensor A is marked *chosen* and is not within the transmission disk of another *chosen* sensor.

$\Rightarrow$ Sensor A is marked *chosen* and does not have a *chosen* neighbor.

$\Rightarrow$ $BridgeNode(i)$ and $FillNode(i)$ did not mark an *unchosen* sensor that is also the "least UID" neighbor of Sensor A, let's name it Sensor B, as *chosen*, or $Redundant_1(i)$ unmarked this sensor.

Case 1:

There is no *unchosen* node within the transmission disk of Sensor A that is the "least UID" neighbor of Node A.

$\Rightarrow$ Since all sensors are initially *unchosen*, and, if changed to *undecided*, can only change to *chosen* by executing $\mathcal{A}_3$ or *unchosen* by executing $\mathcal{A}_1$, and since Sensor A has no *chosen* neighbors, all neighbors of Sensor A will be *unchosen*.

$\Rightarrow$ One of these *unchosen* neighbors of Sensor A will also have the least UID of all the neighbors of Sensor A.

Hence we arrive at a contradiction.

Case 2:

$Redundant_1(i)$ unmarked Sensor B.

$\Rightarrow$ Since Sensor B is *unchosen* and is also the sensor having the least UID of all the neighbors of Sensor A, after changing to *undecided* by executing $\mathcal{A}_2$, Sensor B will also evaluate $\neg LeastUIDNgbr(i,j)$ as false.

$\Rightarrow$ Sensor B will also evaluate $Redundant_1(i)$ as false.

$\Rightarrow$ Sensor B will not be unmarked by $Redundant_1(i)$.

Hence we arrive at a contradiction. □

Theorem 0.0.1 ($\mathcal{L}_{MCSC}$ satisfies specification). *Any system configuration satisfying the legitimacy predicate $\mathcal{L}_{MCSC}$ (per Definition 0.0.1) satisfies the specification of the connected sensor cover problem (as given by Specification 0.0.1).*

Proof. The coverage and connectivity properties have been proven in Lemmas 0.0.1 and 0.0.2, respectively. The definition of $\mathcal{L}_{MCSC}$ implies that in a legitimate configuration, there exist no redundant *chosen* sensor, meaning that all redundant sensors have been identified and are marked *unchosen*. Therefore, the connected cover set $\mathcal{MCSC}_Q$ computed at this point is the smallest possible by Algorithm 1 $\mathcal{MCSC}$. □

Property 0.0.1. *The system defined by the legitimacy predicate $\mathcal{L}_{MCSC}$ is silent.*

Proof. In any configuration satisfying $\mathcal{L}_{MCSC}$, all actions of Algorithm 1 $\mathcal{MCSC}$ are disabled. □

Lemma 0.0.3 (Closure). *The legitimacy predicate $\mathcal{L}_{MCSC}$ is closed.*

Proof. Property 0.0.1 asserts the closure of $\mathcal{L}_{MCSC}$. □

5.6.2 Proof of Convergence

The goal of this section is to prove that starting from any arbitrary configuration of the system of sensors, Algorithm 1 $\mathcal{MCSC}$ guarantees that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

Proof. We formulate this proof by contradiction. Suppose that starting from any arbitrary configuration of the system of sensors, Algorithm 1 $\mathcal{MCSC}$ does not guarantee that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

$\Rightarrow$ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

$\Rightarrow$ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration in which all nonredundant sensors are marked *chosen* and all redundant sensors are marked *unchosen*.

Case 1:

There exists a configuration in which a (nonredundant) sensor that may evaluate $MISNode(i)$ or $FillNode(i)$ as true, does not do so and does not execute $\mathcal{A}_3$.

$\Rightarrow$ A sensor whose sensing disk intersects with the query region and whose sensing disk does not intersect with a *chosen* sensor, or an *undecided* Sensor A that is not the neighbor of any other *undecided* Sensor B whose UID is greater than that of Sensor A, or that is the "least UID" neighbor of Sensor B, is not marked *chosen*.

$\Rightarrow$ Since any query region sensor that is initially *unchosen*, is nonredundant, and whose sensing disk does not intersect with a *chosen* sensor will evaluate $MISNode(i)$

as true, this node will evaluate the guard of $\mathcal{A}_3$ as true.

$\Rightarrow$ This (nonredundant) node will execute $\mathcal{A}_3$ and will change to *chosen*.

Hence we arrive at a contradiction.

Alternatively, since a query region sensor, let's name it Sensor B, that is initially *unchosen* and that is the "least" UID neighbor of a *chosen* (or *undecided*) sensor, and that has no other *undecided* neighbors, will evaluate $BridgeNode(i)$ as true.

$\Rightarrow$ Sensor B will execute $\mathcal{A}_2$ and change to *undecided*.

Or if Sensor B is initially *undecided*, it will then evaluate $FillNode(i)$ as true and will evaluate the guard of $\mathcal{A}_3$ as true.

$\Rightarrow$ This (nonredundant) sensor will execute $\mathcal{A}_3$ and will change to *chosen*.

Hence we arrive at a contradiction.

Case 2: The nonredundant query region sensor is initially marked *chosen*, but executes $Redundant(i)$ and is unmarked.

$\Rightarrow$ Since any nonredundant sensor is one that may be located in an uncovered region and one whose sensing disk is needed to cover the query region, if this sensor is *chosen* and yet is not the neighbor of another *chosen* sensor having a greater UID than its own, then the sensor will evaluate $Redundant_2(i)$ as false and will not become unmarked.

Hence we arrive at a contradiction.

Case 3:

If a redundant sensor is marked as *chosen*, $Redundant_1(i)$ or $Redundant_2(i)$ will not unmark this sensor.

$\Rightarrow$ Since any redundant sensor is one which is not needed to ensure coverage

nor connectivity and which is *undecided* and is the "lesser", but not "least UID", neighbor of an *undecided* sensor, or that is *chosen* and is the "lesser", but not "least UID", neighbor of another *chosen* sensor, such a sensor will evaluate $Redundant_1(i)$ or $Redundant_2(i)$ as true and will subsequently execute $\mathcal{A}_1$.

$\Rightarrow$ Any such redundant sensor will become unmarked by rule $\mathcal{A}_1$.

Hence we arrive at a contradiction. □

5.6.3 Proof of Self-∗

5.6.3.1 Self-configuring

From the proofs of closure and convergence, it was shown that starting from any initial configuration, Algorithm 1 $\mathcal{MCSC}$ forms a network topology in which all members of the minimum connected sensor cover are connected, and are thus able to communicate with each other, either directly or indirectly. It was also shown that starting from any arbitrary state, the given query region will eventually be completely covered. By executing the rules of Algorithm 1 $\mathcal{MCSC}$, network sensors will self-configure to establish a topology that enables communication and sensing coverage under stringent energy constraints. Hence Algorithm 1 $\mathcal{MCSC}$ is self-configuring.

5.6.3.2 Self-healing

Proof. We formulate this proof by contradiction. Suppose Algorithm 1 $\mathcal{MCSC}$ is not self-healing.

$\Rightarrow$ If a nonredundant node fails, a redundant node joins the network, or if there is an arbitrary corruption of the state variables of nodes, including the $Status_i$ variable, then part of the query region may become uncovered, or may be covered by a redundant node.

Case 1:

If a nonredundant node fails, then part of the query region becomes uncovered.

$\Rightarrow$ Since the graph is densely populated, there is a portion of the graph in which an *unchosen* sensor that is in this uncovered region, and that is the "least UID" neighbor of all *undecided* nodes within its transmission disk, does not execute $\mathcal{A}_2$ and $\mathcal{A}_3$ to become *chosen*.

But since an *unchosen* node in this uncovered region (that is the "least UID" neighbor of all *undecided* nodes within its transmission disk) will evaluate $BridgeNode(i)$ as true, and $FillNode(i)$ as true, this node will execute $\mathcal{A}_2$ and $\mathcal{A}_3$ to become *chosen*.

Hence we arrive at a contradiction.

Case 2:

If a part of the query region is covered by a redundant node, then since any node that is *chosen* or *undecided* and that is not the "least UID" neighbor of another *undecided* or *chosen* node, but that has a "lesser" UID than this node, will not evaluate $BridgeNode(i)$ nor $FillNode(i)$ as true, this node will not execute $\mathcal{A}_2$ and change to *undecided*, nor will it execute $\mathcal{A}_3$ and change to *chosen*.

$\Rightarrow$ This redundant node will not cover part of the query region.

Hence we arrive at a contradiction.

Case 3:

If there is an arbitrary corruption of the state variables of nodes, including the $Status_i$ variable, then part of the query region may become uncovered, or may be covered by a redundant node.

$\Rightarrow$ If the $Status_i$ variable for a node is initially *undecided* or *chosen*, then part of the query region may become uncovered, or may be covered by a redundant node.

Since $FillNode(i)$ evaluates to true if a node is *undecided*, and is not the neighbor

of any *undecided* sensor having a greater UID than its own, or if it is the "least UID" neighbor of any *undecided* sensor, irregardless of whether it was initially *undecided*, and since a *chosen* node will cover part of the query region, such an arbitrary corruption will still allow a node to execute $\mathcal{A}_3$ and cover the query region.

Hence we arrive at a contradiction.

Alternatively, since $Redundant_1(i)$ will unmark a sensor even if it is initally *undecided* and is the neighbor of another *undecided* sensor having a greater UID than its own, but is not the "least UID" neighbor of this sensor, then part of the query region will not be covered by a redundant node.

Hence we arrive at a contradiction.

Alternatively, since $Redundant_2(i)$ will unmark a *chosen* sensor, irregardless of whether it was initially *chosen*, that is the "lesser UID" neighbor of another *chosen* sensor, but that does not have the least UID out of all the neighbors of this sensor, then part of the query region will not be covered by a redundant node.

Hence we arrive at a contradiction. □

5.6.3.3 Self-∗

Using the concept of self-stabilization, the self-configuring and self-healing features of our solution have been implemented. Since the paradigm of self-stabilization subsumes all other self-∗ properties, our solution is truly fault-tolerant in terms of the self-∗ feature.

CHAPTER 6

SECOND $\mathcal{MCSC}$ ALGORITHM

6.1 Description of Second $\mathcal{MCSC}$ Algorithm and Data Structures Used

The description of the second $\mathcal{MCSC}$ algorithm is very similar to the first $\mathcal{MCSC}$ algorithm and can be referred to in Section 5.1. In addition to this, the assumptions and data structures used for the second $\mathcal{MCSC}$ algorithm are the same as those for the first $\mathcal{MCSC}$ algorithm and can be also be referred to in Section 5.1.

6.2 Predicates Used in Second $\mathcal{MCSC}$ Algorithm

The predicate $QryRgnIntrsctn(i)$ evaluates to true if the sensing disk of Sensor i intersects with **some** portion of the query region. $NgbrOfChsn(i)$ evaluates to true if Sensor i is a neighbor of any sensor whose status is *chosen*. $ENgbrOfChsn(i,j)$ evaluates to true if Sensor i is a neighbor of any sensor, excluding Sensor j, whose status is *chosen*. Predicate $IsLeastUIDNgbr(i,x)$ evaluates to true if Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID. $LessNotLeastNgbrOfChsn(i)$ evaluates to true if Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID. The predicate $GrtrOrLeastNgbrOfChsn(i)$ evaluates to true if Sensor i is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor.

SensorCover(i), evaluates to true if the status of Sensor i is *unchosen*, the sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the"least UID" neighbor. Predicate $MCSCNode(i)$ evaluates to true if Sensor i is an *undecided* sensor and is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor, or there is a sensor within the transmission disk of Sensor i that is not the neighbor of a *chosen* sensor.

$Redundant_1(i)$ evaluates to true if Sensor i is an *undecided* sensor and is the "lesser" neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor. Finally, $Redundant_2(i)$ evaluates to true if the status of Sensor i is *chosen*, Sensor i has a smaller UID than another *chosen* Sensor j that is within its transmission disk, but Sensor i does not have the smallest UID out of all the neighbors of Sensor j, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor that is not Sensor i.

6.3 Normal Execution of Second $\mathcal{MCSC}$ Algorithm

In this algorithm, every sensor sends its closed neighbor set (including the value of $Status_i$ of the sensors in this set), to all of its neighbors. The steps of the algorithm are as follows:

1. The algorithm marks all *unchosen* sensors whose sensing regions intersect with some portion of the query region (R_Q), and that are not the neighbors of *chosen* sensors whose UID's are greater than their own, or for which these sensors are not the"least UID" neighbors, as *undecided*.

2. $MCSCNode(i)$ checks if Sensor i is *undecided*, and if a neighbor of Sensor i (i.e., a sensor within Sensor i's transmission disk) is not "dominated" by a

chosen sensor (i.e., is not within the transmission disk of a *chosen* sensor), or if Sensor i is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor. In this case, the sensing disk of Sensor i is needed in the final cover set, and hence Sensor i changes its status to *chosen*.

3. $Redundant_1(i)$ is used to unmark any *undecided* sensor that is the "lesser" neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and whose entire transmission disk is covered by *chosen* sensors. In this case, the status of the *undecided* sensor is changed to *unchosen*.

4. $Redundant_2(i)$ removes redundant sensors from the final cover set as follows. If all of the neighbors of Sensor i are within the transmission disk of some *chosen* sensor, and Sensor i is the "lesser" neighbor of a *chosen* sensor, but is not the node with the smallest UID out of all the neighbors of this *chosen* sensor, then Sensor i and all of its neighbors are "dominated" by a *chosen* sensor. In this case, Sensor i should not be in the final MCDS, and thus changes its status to *unchosen*.

5. Finally, action $\mathcal{A}_1$ ensures that any redundant sensor or any sensor whose sensing disk does not intersect with the query region, has its status changed to *unchosen*.

6. All *chosen* sensors are in the final MCDS.

6.4 Faults and Recovery of Second $\mathcal{MCSC}$ Algorithm

In this section, we focus on the fault handling features of the proposed algorithm (Algorithm $MCSC$). There are three variables used in the solution: S_i, UID_i, and $Status_i$ for a Sensor i. So, we need to show that our solution can cope with all possible corruptions associated with these three variables. In the following, we will make an

attempt to list all or most of the important types of faults, and show how they are dealt with in Algorithm *MCSC*. **(1) Wrong initialization of the $Status_i$ variable.** As discussed in the previous subsection, all sensors, if properly initialized, start as *unchosen*. *(a) Sensor i is initialized to* ***undecided*** . Assume that Sensor i is initialized to *undecided*. If i is not a redundant node, then i remains *undecided*, and subsequently changes to *chosen*. (see Actions $\mathcal{A}_2$ and $\mathcal{A}_3$). That is, no correction is necessary. If i is redundant, then it will satisfy the predicate $Redundant_1(i)$ and will change to *unchosen*. *(b) Sensor i is initialized to* ***chosen*** . If the sensing disk of Sensor i does not intersect with the query region, then, by executing $\mathcal{A}_1$, Sensor i will change to *unchosen*. So, no correction is necessary. If Sensor i is redundant, then then it will satisfy the predicate $Redundant_2(i)$, and will change to *unchosen*. If it is nonredundant then Sensor i is necessary, either to ensure coverage or connectivity, and should not be unmarked. **(2) Wrong initialization of the UID_i variable.** *(a) Sensor i is initialized to a UID that is used to identify another Sensor.* If Sensor i is redundant, then any other Sensor within the transmission disk of Sensor i, that has a larger UID than Sensor i and for which Sensor i is not the "least UID" neighbor, will cause Sensor i to evaluate $Redundant(i)$ as true and to become unmarked, if all of Sensor $i's$ neighbors are covered by *chosen* nodes. If it is nonredundant, then Sensor i is needed in the final cover set, and should not be unmarked. **(3) Weakening or Failure of sensors, both in terms of communication and sensing ability.** The weakening or failure of sensors will affect the sensing and communication range of the sensors. In other words, the constant set R_S or R_C will change. Change of R_S or R_C may change the values of $Redundant(i)$, $SensorCover(i)$, and $MCSCNode(i)$. All these changes will be reflected in the change of values of the guards of the corresponding actions. So, eventually, the status of the affected nodes will change due to the execution of these actions. However, these changes will not affect the execution of these actions by the neighbors of the affected nodes. Therefore, any changes in the $Status_i$ variable of

the affected nodes will be handled as mentioned earlier.

Algorithm 2 Connected Sensor Cover Algorithm (Algorithm 2 $\mathcal{MCSC}$) for Sensor i.

Constants:

R_Q:: Query region;

N_i:: Set of sensors within the communication range of Sensor i;

Shared Variables:

S_i:: Sensing region of Sensor i;

UID_i:: Unique user identification number of Sensor i;

$Status_i \in \{unchosen, undecided, chosen\}$:: Status of Sensor i;

Predicates:

$QryRgnIntrsctn(i) \equiv S_i \cap R_Q \neq \emptyset$;
$\equiv$ sensing disk of Sensor i intersects with some portion of query region;

$NgbrOfChsn(i) \equiv (\exists j : i \in N_j \wedge Status_j = chosen)$;
$\equiv$ Sensor i is a neighbor of a chosen sensor;

$ENgbrOfChsn(i,j) \equiv (\exists k : i \in N_k \wedge Status_k = chosen \wedge k \neq j)$;
$\equiv$ Sensor i is a neighbor of a chosen sensor that is not Sensor j;

$IsLeastUIDNgbr(i,x) \equiv i \in N_x \wedge (\forall j \in N_x : j \neq i \wedge UID_i < UID_j)$;
$\equiv$ Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID;

$LessNotLeastNgbrOfChsn(i) \equiv (\exists j : i \in N_j : Status_j = chosen \wedge UID_i < UID_j \wedge \neg IsLeastUIDNgbr(i,j))$;
$\equiv$ Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID;

$GrtrOrLeastNgbrOfChsn(i) \equiv (\forall j : i \in N_j : Status_j \neq chosen \vee UID_i > UID_j \vee IsLeastUIDNgbr(i,j))$;
$\equiv$ Sensor i is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor;

$SensorCover(i) \equiv Status_i = unchosen \wedge QryRgnIntrsctn(i) \wedge GrtrOrLeastNgbrOfChsn(i)$;
$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor;

Algorithm 2 Connected Sensor Cover Algorithm (Algorithm 2 $\mathcal{MCSC}$) for Sensor i (Continued)

$MCSCNode(i) \equiv Status_i = undecided \wedge (GrtrOrLeastNgbrOfChsn(i) \vee (\exists j \in N_i : \neg NgbrOfChsn(j)))$;

$\equiv$ Sensor i is an *undecided* sensor and is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which Sensor i is not the "least UID" neighbor, or there is a sensor within the transmission disk of Sensor i that is not the neighbor of a *chosen* sensor;

$Redundant_1(i) \equiv Status_i = undecided \wedge LessNotLeastNgbrOfChsn(i) \wedge (\forall j \in N_i : NgbrOfChsn(j))$;

$\equiv$ Sensor i is an *undecided* sensor and is the "lesser" neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor;

$Redundant_2(i) \equiv Status_i = chosen \wedge LessNotLeastNgbrOfChsn(i) \wedge (\forall j \in N_i : ENgbrOfChsn(j,i))$;

$\equiv$ status of Sensor i is *chosen*, Sensor i has a smaller UID than another *chosen* Sensor j that is within its transmission disk, but Sensor i does not have the smallest UID out of all the neighbors of Sensor j, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor that is not Sensor i.

$Redundant(i) \equiv Redundant_1(i) \vee Redundant_2(i)$;

Actions:

$\mathcal{A}_1 :: \neg QryRgnIntrsctn(i) \vee Redundant(i)$

$\longrightarrow Status_i = unchosen$;

$\mathcal{A}_2 :: SensorCover(i)$

$\longrightarrow Status_i = undecided$;

$\mathcal{A}_3 :: MCSCNode(i)$

$\longrightarrow Status_i = chosen$;

6.6 Correctness of Second $\mathcal{MCSC}$ Algorithm

Definition 0.0.2. *The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$) if the following conditions are true with respect to a query region:*

i) All non-redundant sensors are marked chosen.

ii) All redundant sensors are marked unchosen.

6.6.1 Proof of Closure

Lemma 0.0.4 (Coverage). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm 2 $\mathcal{MCSC}$ completely covers the query region R_Q.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 2 do not completely cover the query region.

$\Rightarrow$ There is some portion of the query region that is not covered by a *chosen* node.

Since $\mathcal{A}_2$ states that a sensor will change to *undecided* if it is *unchosen*, its sensing disk intersects with some portion of the query region, and if it is not the neighbor of a *chosen* sensor whose UID is greater than its own or for which it is not the "least UID" neighbor, and since the graph is densely populated and all sensors are initially *unchosen*, there will always exist a set of *undecided* nodes, whose sensing disks intersect with the query region, and that will be located at a distance greater than the communication radius, but may also be located less than twice the communication radius from another *chosen* node and from another *undecided* node.

Since an *undecided* node's sensing disk spans a distance of $2R_C$, the union of the sensing disks of all *chosen* nodes and all such *undecided* nodes located at a distance greater than R_C but less than $2R_C$ from any *chosen* or *undecided* node, will completely cover the query region.

Since any *undecided* node will either change to *chosen* by $MCSCNode(i)$ or *unchosen* by $Redundant_1(i)$, and since all such *undecided* nodes are located at a distance greater than R_C from any *chosen* node, each such *undecided* node will evaluate $GrtrOrLeastNgbrOfChsn(i)$ as true and $LessNotLeastNgbrOfChsn(i)$ as false and will change to *chosen* by Rule $\mathcal{A}_3$.

The union of the sensing disks of all nodes that were initially *chosen* and all such *undecided* nodes that changed to *chosen* by executing $\mathcal{A}_3$, completely cover the query region.

Since $Redundant_2(i)$ will only evaluate to true if a node evaluates $LessNotLeastNgbrOfChsn(i)$ as true, and all of its neighbors are covered by a *chosen* node, the $Redundant_2(i)$ predicate will only unmark any of these *chosen* nodes if its entire transmission disk is completely covered by some other *chosen* node.

The sensing disks of all *chosen* sensors in the final $\mathcal{MCSC}$ completely cover the query region.

Hence we arrive at a contradiction. □

Lemma 0.0.5 (Connectivity). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm 2 $\mathcal{MCSC}$ forms a connected graph.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the

sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 2 do not form a connected subgraph.

$\Rightarrow$ There exists a sensor in the final $\mathcal{MCSC}$, lets name it Sensor A, that is marked *chosen* and is not adjacent to another *chosen* sensor.

$\Rightarrow$ Sensor A is marked *chosen* and is not within the tranmission disk of another *chosen* sensor.

$\Rightarrow$ Sensor A is marked *chosen* and does not have a *chosen* neighbor.

Case1 :

$SensorCover(i)$ and $MCSCNode(i)$ did not mark an *unchosen* sensor that is also a neighbor with a greater UID or the "least UID" neighbor of Sensor A, let's name it Sensor B, as *chosen*, or $Redundant_2(i)$ unmarked this sensor.

Since all sensors can have a status of *unchosen*, *undecided*, or *chosen*, and Sensor A has no *chosen* neighbors, all of Sensor A's neighbors must be either *unchosen* or *undecided*.

$\Rightarrow$ Since Sensor A has no *chosen* neighbors, and since all *undecided* neighbors of Sensor A that evaluate $MCSCNode(i)$ as true will change to *chosen*, all *undecided* neighbors of Sensor A must have evaluated $MCSCNode(i)$ as false.

$\Rightarrow$ All *undecided* neighbors of Sensor A must have evaluated $GrtrOrLeastNgbrOfChsn(i)$ as false, and all neighbors of these *undecided* sensors must have evaluated $NgbrOfChsn(j)$ as true.

$\Rightarrow$ Since $LessNotLeastNgbrOfChsn(i)$ is the negative of $GrtrOrLeastNgbrOfChsn(i)$, and all neighbors of these *undecided* sensors evaluated $NgbrOfChsn(j)$ as true, all *undecided* neighbors of Sensor A must have

changed to *unchosen* after evaluating $Redundant_1(i)$ as true and executing $\mathcal{A}_1$.

$\Rightarrow$ All neighbors of Sensor A are *unchosen*.

$\Rightarrow$ The "least UID" neighbor of Sensor A must be *unchosen*.

Hence we arrive at a contradiction.

Case 2:

Sensors A and B are *chosen* neighbors, but Sensor A or Sensor B was unmarked by $Redundant_2(i)$.

As shown in Case 1, since the "least UID" neighbor of Sensor A must be an *unchosen* sensor, let's name it Sensor B, and since Sensor B will change to *chosen* after executing $\mathcal{A}_2$ and $\mathcal{A}_3$, then Sensor B cannot evaluate $LessNotLeastNgbrOfChsn(i)$ as true.

$\Rightarrow$ Sensor B cannot evaluate $Redundant_2(i)$ as true.

$\Rightarrow$ Sensor B cannot be unmarked by $Redundant_2(i)$.

Hence we arrive at a contradiction.

Alternatively, since Sensor A has a greater UID than Sensor B, Sensor A cannot evaluate $LessNotLeastNgbrOfChsn(i)$ as true.

$\Rightarrow$ Sensor A cannot be unmarked by $Redundant_2(i)$.

Hence we arrive at a contradiction. □

Theorem 0.0.2 ($\mathcal{L}_{MCSC}$ satisfies specification). *Any system configuration satisfying the legitimacy predicate $\mathcal{L}_{MCSC}$ (per Definition 0.0.2) satisfies the specification of the connected sensor cover problem (as given by Specification 0.0.1).*

Proof. The coverage and connectivity properties have been proven in Lemmas 0.0.3 and 0.0.5, respectively. The definition of $\mathcal{L}_{MCSC}$ implies that in a legitimate config-

uration, there exist no redundant *chosen* sensor, meaning that all redundant sensors have been identified and are marked *unchosen*. Therefore, the connected cover set $\mathcal{MCSC}_Q$ computed at this point is the smallest possible by Algorithm 2 $\mathcal{MCSC}$. □

Property 0.0.2. *The system defined by the legitimacy predicate $\mathcal{L}_{MCSC}$ is silent.*

Proof. In any configuration satisfying $\mathcal{L}_{MCSC}$, all actions of Algorithm 2 $\mathcal{MCSC}$ are disabled. □

Lemma 0.0.6 (Closure). *The legitimacy predicate $\mathcal{L}_{MCSC}$ is closed.*

Proof. Property 0.0.2 asserts the closure of $\mathcal{L}_{MCSC}$. □

6.6.2 Proof of Convergence

The goal of this section is to prove that starting from any arbitrary configuration of the system of sensors, Algorithm 2 $\mathcal{MCSC}$ guarantees that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

Proof. We formulate this proof by contradiction. Suppose that starting from any arbitrary configuration of the system of sensors, Algorithm 2 $\mathcal{MCSC}$ does not guarantee that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

⇒ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

⇒ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration in which all nonredundant sensors are marked *chosen* and all redundant sensors are marked *unchosen*.

Case 1:

$\Rightarrow$ There exists a configuration in which a (nonredundant) sensor whose status is *unchosen*, whose sensing disk intersects with some portion of the query region, and that may evaluate $GrtrOrLeastNgbrOfChsn(i)$ and $MCSCNode(i)$ as true, does not do so and does not execute $\mathcal{A}_3$.

$\Rightarrow$ A query region sensor which is *unchosen*, not the neighbor of a *chosen* whose UID is greater than its own or for which it is not the "least UID" neighbor, and that has part of its transmission disk not covered by another *chosen* sensor, is not marked *chosen*.

$\Rightarrow$ Since any query region sensor that is initially *unchosen*, and is nonredundant because it is not the "lesser" neighbor of a *chosen* sensor nor the "least UID" neighbor of this *chosen* sensor, and which has a sensor within its transmission disk that is not the neighbor of a *chosen* sensor, will evaluate $QryRgnIntrsctn(i)$ and $GrtrOrLeastNgbrOfChsn(i)$ and $MCSCNode(i)$ as true, this node will evaluate the guard of $\mathcal{A}_2$ and $\mathcal{A}_3$ as true.

$\Rightarrow$ This (nonredundant) sensor will execute $\mathcal{A}_2$, followed by $\mathcal{A}_3$, and will change to *chosen*.

Hence we arrive at a contradiction.

Case 2:

The nonredundant query region sensor is initially marked *chosen*, but executes $Redundant(i)$ and is unmarked.

$\Rightarrow$ Since this sensor executed $Redundant(i)$, it is the neighbor of a *chosen* sensor having a greater UID than itself, but is not the "least UID" neighbor of this *chosen* sensor, and all sensors within its transmission disk are neighbors of a *chosen* sensor.

$\Rightarrow$ This sensor's entire transmission (and sensing) disk is covered by the sensing disks of other *chosen* sensors.

$\Rightarrow$ This sensor is redundant.

Hence we arrive at a contradiction.

Case 3:

If a redundant sensor is marked as *chosen* or *undecided*, $Redundant_1(i)$ or $Redundant_2(i)$ will not unmark this sensor.

$\Rightarrow$ Since a redundant sensor is one whose entire sensing disk is covered by the sensing disks of other *chosen* sensors, and whose removal will not leave part of the query region uncovered, such a redundant sensor having a smaller UID than its *chosen* neighbor, but that is not the "least UID" neighbor of this *chosen* sensor, will evaluate $LessNotLeastNgbrOfChsn(i)$ as true, and will have all of its neighbors evaluate $NgbrOfChsn(j)$ and $ENgbrOfChsn(j, i)$ as true.

$\Rightarrow$ Such a (redundant) sensor will evaluate $Redundant_1(i)$ and $Redundant_2(i)$ as true.

$\Rightarrow$ Such a (redundant) sensor will execute $\mathcal{A}_1$ and will be unmarked.

Hence we arrive at a contradiction.

□

6.6.3 Proof of Self-$*$

6.6.3.1 Self-configuring

From the proofs of closure and convergence, it was shown that starting from any initial configuration, Algorithm 2 $\mathcal{MCSC}$ forms a network topology in which all members of the minimum connected sensor cover are connected, and are thus able to communicate with each other, either directly or indirectly. It was also shown that starting from any arbitrary state, the given query region will eventually be completely

covered. By executing the rules of Algorithm 2 $\mathcal{MCSC}$, network sensors will self-configure to establish a topology that enables communication and sensing coverage under stringent energy constraints. Hence Algorithm 2 $\mathcal{MCSC}$ is self-configuring.

6.6.3.2 Self-healing

Proof. We formulate this proof by contradiction. Suppose Algorithm 2 $\mathcal{MCSC}$ is not self-healing.

$\Rightarrow$ If a nonredundant node fails, a redundant node joins the network, or if there is an arbitrary corruption of the state variables of nodes, including the $Status_i$ variable, then part of the query region may become uncovered, or may be covered by a redundant node.

Case 1:

If non-redundant node fails, then part of the query region becomes uncovered.

$\Rightarrow$ Since the graph is densely populated, there is a portion of the graph in which an *unchosen* sensor that is in this uncovered region, does not execute $\mathcal{A}_2$ and $\mathcal{A}_3$ to become *chosen*. But since this *unchosen* sensor is not covered by a *chosen* sensor, and since all *unchosen* sensors will not be the neighbors of any *chosen* sensor, and since this node will also have part of its transmission disk not covered by a *chosen* sensor, it will evaluate the guard of $\mathcal{A}_2$ as true and $MCSCNode(i)$ as true.

$\Rightarrow$ This node will execute $\mathcal{A}_2$, followed by $\mathcal{A}_3$, and will become *chosen*.

Hence we arrive at a contradiction.

Case 2:

If part of the query region is covered by a redundant node, then since any node that is the "lesser", but not "least UID", neighbor of a *chosen* node, and whose entire transmission disk is covered by *chosen* nodes, is redundant and will not evaluate

$GrtrOrLeastNgbrOfChsn(i)$ as true, this node will not execute $\mathcal{A}_2$ and change to *undecided*, nor will it execute $\mathcal{A}_3$.

$\Rightarrow$ This node cannot change to *chosen* to cover the query region.

Hence we arrive at a contradiction.

Case 3:

If there is an arbitrary corruption of one of the state variables of nodes, including the $Status_i$ variable, then part of the query region may become uncovered, or may be covered by a redundant node.

$\Rightarrow$ If the $Status_i$ variable for a node is initially *undecided* or *chosen*, then part of the query region may become uncovered, or may be covered by a redundant node.

Since $MCSCNode(i)$ evaluates to true if an *undecided* sensor is not the neighbor of a *chosen* sensor having a greater UID than its own or for which it is not the "least UID" neighbor, and if it has part of its transmission disk uncovered, regardless of whether it was initially undecided, and since a *chosen* node will cover part of the query region, such an arbitrary corruption will still allow a node to execute $\mathcal{A}_3$ and cover the query region.

Hence we arrive at a contradiction.

Alternatively, since $Redundant_1(i)$ will unmark a sensor even if it is initially *undecided* and is the "lesser" neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and it has all parts of its transmission disk covered by a *chosen* sensor, then part of the query region will not be covered by a redundant node.

Hence we arrive at a contradiction.

Alternatively, since $Redundant_2(i)$ will unmark a *chosen* sensor that is a "lesser", but not "least UID", neighbor of another *chosen* sensor, and whose transmission disk is completely covered by other *chosen* sensors, regardless of whether it was initially *chosen*, part of the query region will not be covered by a redundant node.

Hence we arrive at a contradiction. □

6.6.3.3 Self-∗

Using the concept of self-stabilization, the self-configuring and self-healing features of our solution have been implemented. Since the paradigm of self-stabilization subsumes all other self-∗ properties, our solution is truly fault-tolerant in terms of the self-∗ feature.

CHAPTER 7

THIRD $\mathcal{MCSC}$ ALGORITHM

7.1 Description of Third $\mathcal{MCSC}$ Algorithm and Data Structures Used

The description of, and assumptions for, the third $\mathcal{MCSC}$ algorithm is very similar to the first $\mathcal{MCSC}$ algorithm and can be referred to in Section 5.1. In addition to this, the data structures used for the third $\mathcal{MCSC}$ algorithm are similar to those used for the first $\mathcal{MCSC}$ algorithm (Section 5.1), except that the status of a sensor may be *unchosen*, *undecided*, *removed*, or *chosen*.

7.2 Predicates Used in Third $\mathcal{MCSC}$ Algorithm

The predicate $Cycle(x, y)$ determines if there exists a cycle such that Sensors x, i, and y are vertices in the cycle, and all other vertices in this cycle are *chosen* sensors. Its steps are as follows:

1. A vertex i sends a $FindCycle(i, x)$ message and a $FindCycle(i, y)$ message to x and y, respectively. As a $FindCycle(i, x)$ or $FindCycle(i, y)$ message travels, the path is recorded and piggybacked onto the $FindCycle(i, x)$ or $FindCycle(i, y)$ message. Each node traversed in this path is recorded.

2. Sensor x and Sensor y then send these search messages to all neighbors having a status of *chosen*.

3. If a node receives a $FindCycle(i, x)$ or a $FindCycle(i, y)$ message, it then, in turn, forwards this message to all of its *chosen* neighbors (floods the network).

4. If any node receives **both** $FindCycle(i, x)$ and $FindCycle(i, y)$ messages, then there is a cycle, and node i can then be removed.

   (a) This node then sends a $FoundCycle(x, y)$ message to vertex i along the shorter path that is recorded in either $FindCycle(i, x)$ or $FindCycle(i, y)$.

5. The $Cycle(x, y)$ predicate returns true if the $FoundCycle(x, y)$ message has been received by vertex i, within $2D$ rounds, in which D is the diameter of the network and *round* refers to a computation $e \in \mathcal{E}$ in which every continuously enabled processor has taken one atomic step (as defined in Section **??**) .

Predicate $Adjacent(x, y)$ evaluates to true if x and y are neighbors. The predicate,
$IsLeastUIDNgbr(i, x)$, evaluates to true if Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID. $HasChsnNgbr(x)$ evaluates to true if Sensor x has a neighbor that has a status of *chosen*. Predicate $ENgbrOfChsn(i, j)$ evaluates to true if Sensor i is a neighbor of a *chosen* sensor that is not Sensor j.

The predicate $NonAdjacentNgbrs(i)$ evaluates to true if Sensor i has two neighbors that are not adjacent (are not neighbors of each other). $QryRgnIntrsctn(i)$ evaluates to true if the sensing disk of Sensor i intersects with **some** portion of the query region. $NonRemovable(i)$ evaluates to true if Sensor i has two neighbors for which the $Cycle(x, y)$ predicate does not evaluate to true.

The predicate, $LesserNgbrOfChsn(i)$, evaluates to true if Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own. $LessNotLeastNgbrOfChsn(i)$ evaluates to true if Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID. Finally, $Connector(i)$ evaluates to true if Sensor i is an *unchosen* sensor and there exists a neighbor of Sensor i that is *chosen* or *removed* and that does not have any *chosen* neighbors, and Sensor i is the neighbor of this *chosen* or *removed* sensor

that has the smallest UID.

The predicate *Redundant*(i) unmarks Sensor i if it is a *chosen* sensor and is the neighbor of a *chosen* sensor having a greater UID than its own, but is not the neighbor of this sensor having the smallest UID, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor that is not Sensor i.

7.3 Normal Execution of Third $\mathcal{MCSC}$ Algorithm

We will explain the normal execution of the protocol; i.e., assuming that the system starts from a good initial configuration (all sensors are initially *unchosen*) and that no faults occur during the execution of the protocol. The steps of the algorithm are as follows:

1. The algorithm marks **all** sensors whose sensing region intersects with some portion of the query region (R_Q), that have two nonadjacent neighbors, and that are not the neighbors of *chosen* sensors having greater UID's than their own, as *undecided*.

2. The algorithm then attempts to place an *undecided* Sensor i in the final $\mathcal{MCSC}$, by checking if it is nonremovable. A vertex is nonremovable if its removal results in a disconnected graph. This is determined as follows:

   (a) If any two neighbors (x, y) of the *undecided* vertex i do not have a cycle that has, as a path in this cycle, vertices $(..., x, i, y, ...)$, then this vertex cannot be removed. In other words, there must be a cycle between every pair of neighbors of *undecided* vertex i, in which all sensors in this cycle are *chosen* sensors (except Sensor x and Sensor y), before it is removable. This is determined by the *Cycle*(x, y) predicate, which was elaborated upon before.

3. If a vertex is removable (or not nonremovable), and its status is *undecided*, then its status becomes *removed*.

4. If a vertex is *chosen* or *removed* and if it does not have any *chosen* neighbors, then its *unchosen* neighbor, that is also the neighbor having the smallest UID of all its neighbors, is marked as *chosen*.

5. An *undecided* vertex that is nonremovable is marked *chosen*.

6. If a *chosen* Sensor i is the neighbor of another *chosen* sensor having a greater UID than its own, but is not the neighbor of this sensor having the smallest UID, and if all sensors within Sensor $i's$ transmission disk are neighbors of some *chosen* sensor that is not Sensor i, then Sensor i is unmarked.

7. All *chosen* vertices are in the final $\mathcal{MCSC}$.

7.4 Faults and Recovery of Third $\mathcal{MCSC}$ Algorithm

In this section, we focus on the fault handling features of the proposed algorithm (Algorithm $MCSC$). There are three variables used in the solution: S_i, UID_i, and $Status_i$ for a Sensor i. So, we need to show that our solution can cope with all possible corruptions associated with these three variables. In the following, we will make an attempt to list all or most of the important types of faults, and show how they are dealt with in Algorithm $MCSC$. **(1) Wrong initialization of the $Status_i$ variable.** As discussed in the previous subsection, all sensors, if properly initialized, start as *unchosen*. *(a) Sensor i is initialized to undecided.* Assume that Sensor i is initialized to *undecided*. If i is not a redundant node, then i remains *undecided*, and subsequently changes to *chosen*. (see Actions $\mathcal{A}_2$ and $\mathcal{A}_3$). That is, no correction is necessary. If i is redundant, then it will satisfy the predicate $Redundant(i)$ after executing $\mathcal{A}_3$, or will execute $\mathcal{A}_4$, and will either change to *unchosen* or *removed*. *(b) Sensor i is initialized to removed.* Assume that Sensor i is initialized as a *removed* sensor. If the sensing disk of Sensor i does not intersect with the query region, then, by executing $\mathcal{A}_1$, Sensor i will change to *unchosen*. So, no correction is necessary. If

Sensor i's sensing disk does intersect with the query region, then if it does not have a *chosen* neighbor, after evaluating $Connector(i)$ as true, its *unchosen* neighbor, having the least UID, will be marked as *chosen* by $\mathcal{A}_3$. Therefore, since Sensor i's neighbor was marked to ensure connectivity, Sensor i is not redundant, and should not be unmarked. *(c) Sensor i is initialized to chosen.* If the sensing disk of Sensor i does not intersect with the query region, then, by executing $\mathcal{A}_1$, Sensor i will change to *unchosen*. So, no correction is necessary. If Sensor i is redundant, then then it will satisfy the predicate $Redundant(i)$, and will change to *unchosen*. If it is nonredundant then Sensor i is necessary, either to ensure coverage or connectivity, and should not be unmarked. **(2) Wrong initialization of the UID_i variable.** *(a) Sensor i is initialized to a UID that is used to identify another Sensor.* If Sensor i is redundant, then any other Sensor within the transmission disk of Sensor i, that has a larger UID than Sensor i, will cause Sensor i to evaluate $Redundant(i)$ as true and to become unmarked. If it is nonredundant, then Sensor i is needed in the final cover set, and should not be unmarked. **(3) Weakening or Failure of sensors, both in terms of communication and sensing ability.** The weakening or failure of sensors will affect the sensing and communication range of the sensors. In other words, the constant set R_S or R_C will change. Change of R_S or R_C may change the values of $Redundant(i)$ and $Connector(i)$. All these changes will be reflected in the change of values of the guards of the corresponding actions. So, eventually, the status of the affected nodes will change due to the execution of these actions. All changes of the $Status_i$ variable have already been discussed in earlier cases above.

Algorithm 3 Connected Sensor Cover Algorithm (Algorithm 3 $\mathcal{MCSC}$) for Sensor i.

Constants:

R_Q:: Query region;

N_i:: Set of sensors within the communication range of Sensor i;

Shared Variables:

S_i:: Sensing region of Sensor i;

UID_i:: Unique user identification number of Sensor i;

$Status_i \in \{unchosen, undecided, removed, chosen\}$:: Status of Sensor i;

Predicates:

$Cycle(x, y) \equiv \exists cycle : \{\ldots, x, i, y, \ldots\}$ are vertices in the cycle, and all other vertices in the cycle are *chosen* sensors;
$\equiv$ there exists a cycle such that Sensors x, i, and y are vertices in the cycle, and all other vertices in this cycle are *chosen* sensors;

$Adjacent(x, y) \equiv x \in N_y \wedge y \in N_x$;
$\equiv$ Sensor x is a neighbor of Sensor y, and Sensor y is a neighbor of Sensor x;

$IsLeastUIDNgbr(i, x) \equiv i \in N_x \wedge (\forall j \in N_x : j \neq i \wedge UID_i < UID_j)$;
$\equiv$ Sensor i is a neighbor of Sensor x, and is also the neighbor of Sensor x having the least UID;

$HasChsnNgbr(x) \equiv \exists i \in N_x : Status_i = chosen$;
$\equiv$ Sensor x has a *chosen* neighbor;

$ENgbrOfChsn(i, j) \equiv (\exists k : i \in N_k \wedge Status_k = chosen \wedge k \neq j)$;
$\equiv$ Sensor i is a neighbor of a *chosen* sensor that is not Sensor j;

$NonAdjacentNgbrs(i) \equiv \exists x \in N_i \wedge \exists y \in N_i : \neg Adjacent(x, y)$;
$\equiv$ Sensor i has two neighbors that are not neighbors of each other;

$QryRgnIntrsctn(i) \equiv S_i \cap R_Q \neq \emptyset$;
$\equiv$ sensing disk of Sensor i intersects with some portion of query region;

$NonRemovable(i) \equiv \exists x \in N_i \wedge \exists y \in N_i : \neg Cycle(x, y)$;
$\equiv$ Sensor i has two neighbors between which there is no cycle that includes *chosen* sensors in this cycle.;

$LesserNgbrOfChsn(i) \equiv (\exists j : i \in N_j \wedge Status_j = chosen \wedge UID_i < UID_j)$;
$\equiv$ Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own;

$LessNotLeastNgbrOfChsn(i) \equiv (\exists j : i \in N_j \wedge Status_j = chosen \wedge UID_i < UID_j \wedge \neg IsLeastUIDNgbr(i, j))$;
$\equiv$ Sensor i is a neighbor of a *chosen* sensor whose UID is greater than its own, but Sensor i is not the neighbor of this sensor that has the smallest UID;

Algorithm 3 Connected Sensor Cover Algorithm (Algorithm 3 $\mathcal{MCSC}$) for Sensor i (Continued)

$Connector(i) \equiv Status_i = unchosen \wedge (\exists j \in N_i : (Status_j = chosen \vee Status_j = removed) \wedge \neg HasChsnNgbr(j) \wedge IsLeastUIDNgbr(i,j))$;

$\equiv$ Sensor i is an *unchosen* sensor and there exists a neighbor of Sensor i that is *chosen* or *removed* and that does not have any *chosen* neighbors, and Sensor i is the neighbor of this *chosen* sensor having the smallest UID;

$Redundant(i) \equiv Status_i = chosen \wedge LessNotLeastNgbrOfChsn(i) \wedge (\forall j \in N_i : ENgbrOfChsn(j,i))$;

$\equiv$ Sensor i is a *chosen* sensor and is the neighbor of a *chosen* sensor having a greater UID than its own, but is not the neighbor of this sensor having the smallest UID, and all sensors within the transmission disk of Sensor i are neighbors of a *chosen* sensor that is not Sensor i;

Actions:

$\mathcal{A}_1 :: \neg QryRgnIntrsctn(i) \vee Redundant(i)$

$\longrightarrow Status_i = unchosen$;

$\mathcal{A}_2 :: QryRgnIntrsctn(i) \wedge NonAdjacentNgbrs(i) \wedge \neg LesserNgbrOfChsn(i)$

$\longrightarrow Status_i = undecided$;

$\mathcal{A}_3 :: (NonRemovable(i) \wedge Status_i = undecided) \vee Connector(i)$

$\longrightarrow Status_i = chosen$;

$\mathcal{A}_4 :: \neg NonRemovable(i) \wedge Status_i = undecided$

$\longrightarrow Status_i = removed$;

7.6 Correctness of Third $\mathcal{MCSC}$ Algorithm

Definition 0.0.3. *The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$) if the following conditions are true with respect to a query region:*

i) All non-redundant sensors are marked chosen.

ii) All redundant sensors are marked unchosen.

7.6.1 Proof of Closure

Lemma 0.0.7 (Coverage). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm $\mathcal{MCSC}$ completely covers the query region R_Q.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 3 do not completely cover the query region.

$\Rightarrow$ There is a portion of the query region that does not lie within the sensing disk of a *chosen* sensor.

$\Rightarrow$ Since the graph is densely populated and the communication radius is equal to the sensing radius, there exists a sensor within this uncovered portion of the query region, let's call it Node A, that does not lie within the transmission disk of a *chosen* sensor.

$\Rightarrow$ Since every sensor will have two nodes located at opposite ends of its sensing disk that are non-adjacent neighbors, and since Node A is not located within the transmission disk of a *chosen* sensor, and since Node $A's$ sensing disk intersects with a portion of the query region, Node A will evaluate $QryRgnIntrsctn(i)$,

$NonAdjacentNgbrs(i)$, and $\neg LesserNgbrOfChsn(i)$ as true and will change its status to *undecided.*

$\Rightarrow$ Since Node A is not located within the sensing disk of a *chosen* sensor, Node A will not be located within the transmission disk of a *chosen* sensor.

$\Rightarrow$ Part, if not all, of Node $A's$ transmission disk will not lie within the transmission disk of a *chosen* sensor.

Case 1: There exists an x and a y which are neighbors of Node A, for which $\neg Cycle(x, y)$ will evaluate to true.

$\Rightarrow$ Node A will evaluate $NonRemovable(i)$ to true.

$\Rightarrow$ Node A will execute $\mathcal{A}_3$ and will change to *chosen.*

$\Rightarrow$ Since Node A is *chosen* and is also located within its own transmission disk, Node A does lie within the transmission disk of a *chosen* sensor.

Hence we arrive at a contradiction.

Case 2: If all neighbors of Node A evaluate $Cycle(x, y)$ to true, but if only Node A is not a neighbor of a *chosen* node, then Node A will execute $\mathcal{A}_4$ and will change to *removed.*

$\Rightarrow$ Since Node A does not have a *chosen* neighbor, and since all *undecided* nodes must change to either *chosen* after executing $\mathcal{A}_3$ or *removed* after executing $\mathcal{A}_4$, and since any neighbor of Node A will find that Node A is not covered by a *chosen* node and will evaluate $\neg NonRemovable(i)$ as false, then all neighbors of Node A must be *unchosen.*

$\Rightarrow$ Any of these neighbors of Node A may evaluate $Connector(i)$ as true, execute $\mathcal{A}_3$, and change to *chosen.*

$\Rightarrow$ Node A does lie within the transmission disk of a *chosen* sensor.

Hence we arrive at a contradiction.

Case 3: Sensor A does lie within the transmission disk of another *chosen* sensor, let's call it Sensor B, but Sensor A or Sensor B was unmarked by *Redundant*(i).

$\Rightarrow$ When Sensor A or Sensor B is unmarked, the portion of the query region covered by Sensor A or Sensor B will be uncovered.

Since *Redundant*(i) will evaluate to true only if Sensor i and all of Sensor $i's$ neighbors are neighbors of a *chosen* node, Sensor $A's$ entire transmission disk must be covered by a *chosen* node before it is unmarked by *Redundant*(i).

$\Rightarrow$ If Sensor A or Sensor B is unmarked, the portion of the query region covered by Sensor A or Sensor B must be covered by other *chosen* sensor(s).

Hence we arrive at a contradiction. □

Lemma 0.0.8 (Connectivity). *In any legitimate configuration, the connected set cover $\mathcal{MCSC}_Q$ computed by Algorithm $\mathcal{MCSC}$ forms a connected graph.*

Proof. We prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final $\mathcal{MCSC}$ chosen by Algorithm 3 do not form a connected subgraph.

$\Rightarrow$ There exists a sensor in the final $\mathcal{MCSC}$, lets name it Sensor A, that is marked *chosen* and is not adjacent to another *chosen* sensor.

$\Rightarrow$ Sensor A is marked *chosen* and is not within the transmission disk of another *chosen* sensor.

$\Rightarrow$ Sensor A is marked *chosen* and does not have a *chosen* neighbor.

$\Rightarrow$ Rule $\mathcal{A}_3$ did not mark an *unchosen* neighbor of Sensor A as *chosen*, or $Redundant(i)$ unmarked this node.

Case 1: Since Sensor A does not have a *chosen* neighbor and Sensor $A's$ status is *chosen*, either there are no *unchosen* sensors that are neighbors of Sensor A, or the "least UID" neighbor of Sensor A is not an *unchosen* sensor.

$\Rightarrow$ Since all sensors are initially *unchosen*, and the sensing disk of Sensor A intersects with some portion of the query region, there is no *unchosen* sensor within the query region that is a neighbor of Sensor A, and that evaluated $\neg LesserNgbrOfChsn(i)$ as false.

$\Rightarrow$ All sensors that are neighbors of Sensor A evaluated $\neg LesserNgbrOfChsn(i)$ as true.

$\Rightarrow$ There is no neighbor of Sensor A that has a smaller UID than Sensor A.

$\Rightarrow$ Sensor A has the smallest UID of all its neighbors.

$\Rightarrow$ Only *undecided* nodes are neighbors of Sensor A, or the least UID neighbor of Sensor A is an *undecided* node.

$\Rightarrow$ If we name such an *undecided* neighbor of Sensor A as Sensor B, then Sensor B will either change to *chosen* by $\mathcal{A}_3$, or *removed* by $\mathcal{A}_4$.

$\Rightarrow$ If Sensor B had changed to *chosen* by $\mathcal{A}_3$, then Sensor A would have a *chosen* neighbor.

$\Rightarrow$ Sensor B must have changed to *removed* by rule $\mathcal{A}_4$ after evaluating $\mathcal{A}_4$ as true.

$\Rightarrow$ Sensor B evaluated $\neg NonRemovable(i)$ as true.

$\Rightarrow (\forall x \in N_B \wedge \forall y \in N_B) : Cycle(x, y)$

$\Rightarrow$ All parts of Sensor $B's$ transmission disk are covered by a *chosen* node.

$\Rightarrow$ Since Sensor B is a neighbor of Sensor A, Sensor A is covered by, and is a neighbor of, a *chosen* node.

$\Rightarrow$ Sensor A does have a *chosen* neighbor.

Hence we arrive at a contradiction.

Case 2: Sensor A does have a *chosen* neighbor, but this *chosen* neighbor, let's name it Sensor B, was unmarked by $Redundant(i)$.

Since Sensor A no longer has a *chosen* neighbor, and Sensor $B's$ status is *unchosen*, and Sensor B is a neighbor of Sensor A, either Sensor B, or any other *unchosen* neighbor of Sensor A can evaluate $Connector(i)$ as true, execute $\mathcal{A}_3$, and change to *chosen*.

$\Rightarrow$ Before an *unchosen* node evaluates $Connector(i)$ as true and executes $\mathcal{A}_3$, it must have evaluated $IsLeastUIDNgbr(i, j)$ as true.

$\Rightarrow$ This node, once it executes $\mathcal{A}_3$, will also evaluate $LessNotLeastNgbrOfChsn(i)$ as false.

$\Rightarrow$ Sensor B cannot be unmarked by $Redundant(i)$.

Hence we arrive at a contradiction. □

Theorem 0.0.3 ($\mathcal{L}_{MCSC}$ satisfies specification). *Any system configuration satisfying the legitimacy predicate $\mathcal{L}_{MCSC}$ (per Definition 0.0.3) satisfies the specification of the connected sensor cover problem (as given by Specification 0.0.1).*

Proof. The coverage and connectivity properties have been proven in Lemmas 0.0.7 and 0.0.8, respectively. The definition of $\mathcal{L}_{MCSC}$ implies that in a legitimate configuration, there exist no redundant *chosen* sensor, meaning that all redundant sensors

have been identified and are marked *unchosen*. Therefore, the connected cover set $\mathcal{MCSC}_Q$ computed at this point is the smallest possible by Algorithm $\mathcal{MCSC}$. □

Property 0.0.3. *The system defined by the legitimacy predicate $\mathcal{L}_{MCSC}$ is silent.*

Proof. In any configuration satisfying $\mathcal{L}_{MCSC}$, all actions of Algorithm $\mathcal{MCSC}$ are disabled. □

Lemma 0.0.9 (Closure). *The legitimacy predicate $\mathcal{L}_{MCSC}$ is closed.*

Proof. Property 0.0.3 asserts the closure of $\mathcal{L}_{MCSC}$. □

7.6.2 Proof of Convergence

The goal of this section is to prove that starting from any arbitrary configuration of the system of sensors, Algorithm $\mathcal{MCSC}$ guarantees that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

Proof. We formulate this proof by contradiction. Suppose that starting from any arbitrary configuration of the system of sensors, Algorithm $\mathcal{MCSC}$ does not guarantee that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

⇒ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

⇒ There exists a configuration in which, after any finite number of steps, the system will never reach a configuration in which all nonredundant sensors are marked *chosen* and all redundant sensors are marked *unchosen*.

Case 1: There exists a configuration in which a (nonredundant) query region sensor that is not the "lesser" neighbor of a *chosen* sensor, that has two non-adjacent neighbors, that has two neighbors between which there is no cycle that includes *chosen* sensors in this cycle, and that may evaluate $NonAdjacentNgbrs(i)$, $\neg LesserNgbrOfChsn(i)$, and $NonRemovable(i)$ as true, does not do so and does not execute $\mathcal{A}_3$.

$\Rightarrow$ A sensor having two nonadjacent neighbors which is not the neighbor of a *chosen* sensor having a greater UID than its own, and that has two neighbors between which there is no cycle including *chosen* sensors in this cycle, is not marked *chosen*.

$\Rightarrow$ Since any query region sensor that is initially *unchosen*, and is nonredundant because it is not the "lesser" neighbor of a *chosen* sensor, has two nonadjacent neighbors, and that has two neighbors between which there is no cycle including chosen sensors in this cycle, will evaluate $NonAdjacentNgbrs(i)$, $\neg LesserNgbrOfChsn(i)$, and $NonRemovable(i)$ as true, this node will evaluate the guard of $\mathcal{A}_2$, and then $\mathcal{A}_3$ as true.

$\Rightarrow$ This (nonredundant) sensor will execute $\mathcal{A}_2$, followed by $\mathcal{A}_3$, and will change to *chosen*.

Hence we arrive at a contradiction.

Case 2: The nonredundant query region sensor is initially marked *chosen*, but executes $Redundant(i)$ and is unmarked.

$\Rightarrow$ Since this sensor executed $Redundant(i)$, it is the neighbor of a *chosen* sensor having a greater UID than itself, and all sensors within its transmission disk are neighbors of a *chosen* sensor.

$\Rightarrow$ This sensor's entire transmission (and sensing) disk is covered by the sensing disks of other *chosen* sensors.

$\Rightarrow$ This sensor is redundant.

Hence we arrive at a contradiction.

Case 3: If a redundant sensor is marked as *chosen*, $Redundant(i)$ will not unmark this sensor.

Since any redundant sensor is one whose entire sensing disk is covered by the sensing disks of other *chosen* sensors, and whose removal will not leave part of the query region uncovered, such a sensor will evaluate $LesserNgbrOfChsn(i)$ as true, and will have all of its neighbors evaluate $ENgbrOfChsn(j, i)$ as true.

$\Rightarrow$ Such a (redundant) sensor will evaluate $Redundant(i)$ as true.

$\Rightarrow$ Such a (redundant) sensor will execute $\mathcal{A}_1$ and will be unmarked.

Hence we arrive at a contradiction. □

7.6.3 Proof of Self-$*$

7.6.3.1 Self-configuring

From the proofs of closure and convergence, it was shown that starting from any initial configuration, Algorithm $\mathcal{MCSC}$ forms a network topology in which all members of the minimum connected sensor cover are connected, and are thus able to communicate with each other, either directly or indirectly. It was also shown that starting from any arbitrary state, the given query region will eventually be completely covered. By executing the rules of Algorithm $\mathcal{MCSC}$, network sensors will self-configure to establish a topology that enables communication and sensing coverage under stringent energy constraints. Hence Algorithm $\mathcal{MCSC}$ is self-configuring.

7.6.3.2 Self-healing

Proof. We formulate this proof by contradiction. Suppose Algorithm $\mathcal{MCSC}$ is not self-healing.

$\Rightarrow$ If a nonredundant node fails, a redundant node joins the network, or if there is an arbitrary corruption of the $Status_i$ variable of nodes, then part of the query region may become uncovered, or may be covered by a redundant node.

Case 1: If a nonredundant node fails, then part of the query region becomes uncovered.

Since the graph is densely populated, there is a portion of the graph in which an *unchosen* sensor that is in this uncovered region, does not execute $\mathcal{A}_2$ and $\mathcal{A}_3$ to become *chosen*.

$\Rightarrow$ However, since this *unchosen* sensor has two nonadjacent neighbors, is not the "lesser" neighbor of a *chosen* sensor, and has two neighbors between which there is no cycle that includes chosen sensors in this cycle, it will evaluate the guard of $\mathcal{A}_2$ as true and $NonRemovable(i)$ as true.

$\Rightarrow$ This node will execute $\mathcal{A}_2$, followed by $\mathcal{A}_3$, and will become *chosen*.

Hence we arrive at a contradiction.

Case 2: A part of the query region is covered by a redundant node.

Since any node that is the "lesser" neighbor of a *chosen* node, and whose entire transmission disk is covered by *chosen* nodes, is redundant and will not evaluate $\neg LesserNgbrOfChsn(i)$ as true, this node will not execute $\mathcal{A}_2$ and change to *undecided*, nor will it execute $\mathcal{A}_3$.

$\Rightarrow$ This node cannot change to *chosen* to cover the query region.

Hence we arrive at a contradiction.

Case 3: If there is an arbitrary corruption of the $Status_i$ variable of nodes, then part of the query region may become uncovered, or may be covered by a redundant node.

$\Rightarrow$ If the $Status_i$ variable for a node is initially *undecided*, *chosen*, or *removed*, then part of the query region may become uncovered, or may be covered by a redundant node.

Since $LesserNgbrOfChsn(i)$ evaluates to false if a node, regardless of its initial status, is not the "lesser" neighbor of a *chosen* node, and $NonRemovable(i)$ will evaluate to true if an *undecided* node has two neighbors for which $\neg Cycle(x, y)$ evaluates to true, and since a *chosen* node will cover part of the query region, such an arbitrary corruption will still allow a node to execute $\mathcal{A}_2$ and $\mathcal{A}_3$ and cover the query region.

Hence we arrive at a contradiction.

Alternatively, since $Redundant(i)$ will unmark a sensor if it is *chosen*, is the "lesser" neighbor, but not the neighbor having the smallest UID, of a *chosen* node, and if all parts of its transmission disk are covered by *chosen* nodes, if the $Status_i$ variable of a redundant node is initially *chosen*, is initially *undecided*, or changes from *removed* to *undecided*, and then this node changes to *chosen* by executing $\mathcal{A}_3$, it will become unmarked.

$\Rightarrow$ Part of the query region will not be covered by a redundant node.

Hence we arrive at a contradiction. □

7.6.3.3 Self-*

Using the concept of self-stabilization, the self-configuring and self-healing features of our solution have been implemented. Since the paradigm of self-stabilization subsumes all other self-* properties, our solution is truly fault-tolerant in terms of the self-* feature.

CHAPTER 8

SIMULATION AND RESULTS

8.1 Discussion of Results

Algorithms 1, 2, and 3 compute a minimum connected sensor cover for the query region. Moreover, all three algorithms are fault-tolerant in terms of the self-$*$ feature.

In our simulations, for the first set of experiments, we assumed that nodes are chosen and randomly deployed on a grid of size 500 $\times$ 500 (300,000 nodes). Similar to [26, 44, 55] we consider the sensing region associated with a sensor modeled as a circular region around itself. We considered a homogeneous network of 300,000 nodes (i.e. all sensors had the same sensing region — circular of radius 6). We then used varying sizes for a query region, and measured the number of sensors in the final minimum connected cover set, the number of query region sensors (dominated) per MCSC sensor, and the stabilization time for Algorithms 1, 2, 3, and Rule k [15]. The query region used in each simulation varied from 60 $\times$ 60 graph units to 120 $\times$ 120 graph units, in intervals of 10 graph units. The results of this simulation are summarized in Table 1 and Figures 5(a) - (c) in the next section.

The simulations summarized in Table 2, Table 3, and Figures 5(d)-(i) were performed with a query region of size 90 $\times$ 90 graph units. The total number of sensors deployed, and the size of the radius of communication of the sensors were varied in Tables 2 and 3, respectively.

Table 1. Number of MCSC Sensors, Query Region Sensors per MCSC Sensor, and Stabilization Times for Algorithms 1, 2, 3, and Rule k at Various Query Region Sizes.

		Size of Query Region (n x n units)						
		60	**70**	**80**	**90**	**100**	**110**	**120**
Alg. 1	**Number of MCSC Sensors**	155	203	246	286	342	387	475
Alg. 1	**Qry Rgn Sensors / MCSC Sensor**	45.6	43.6	46.5	48.9	47.2	50.1	47.1
Alg. 1	**Stabilization Time (min.)**	38.1	73.0	135.2	192.4	259.5	376.9	498.7
Alg. 2	**Number of MCSC Sensors**	166	207	257	309	375	432	502
Alg. 2	**Qry Rgn Sensors / MCSC Sensor**	40.5	42.6	42.5	44.4	43.2	44.4	44.4
Alg. 2	**Stabilization Time (min.)**	10.2	14.7	19.7	27.9	36.2	47.1	62.5
Alg. 3	**Number of MCSC Sensors**	187	238	287	364	519	661	708
Alg. 3	**Qry Rgn Sensors / MCSC Sensor**	37.8	38.2	39.0	37.4	31.5	29.3	31.7
Alg. 3	**Stabilization Time (min.)**	10.7	17.5	27.2	42.3	64.3	108.3	176.0
Rule k	**Number of MCSC Sensors**	191	244	297	343	410	513	595
Rule k	**Qry Rgn Sensors / MCSC Sensor**	37.5	37.0	37.9	39.8	40.0	38.2	38.3
Rule k	**Stabilization Time (min.)**	4.6	6.0	8.3	11.4	16.0	22.4	29.6

As shown in Table 1, at all query region sizes, Algorithm 1 produced the least nodes in the final cover set, Algorithm 2 produced a greater number of nodes in the final cover set than Algorithm 1 but fewer nodes in the final cover set than Algorithm 3 and Rule k. Algorithm 3 produced a greater number of nodes in the final cover set than Algorithm 1 and Algorithm 2 at all query region sizes tested. However, it produced a final cover set that was smaller than Rule k's at query region sizes that were less than 90 $\times$ 90 square graph units and larger than Rule k's at query region sizes greater than this. Rule k produced the greatest number of nodes in the final cover set at query region sizes that were less than 90 $\times$ 90 square graph units, but produced fewer nodes in the final cover set than Algorithm 3 at query region sizes that were greater than 90 $\times$ 90 square graph units. This was due to the fact that Algorithm 1 has the strongest redundancy predicate, since it only requires that a Sensor i be the neighbor of a chosen sensor and also have a smaller UID than this *chosen* sensor but not the least UID out of all the neighbors of this *chosen* sensor, before it is unmarked. Algorithms 2 and 3 have a redundancy predicate that is weaker than that of Algorithm 1 but stronger than that of Rule k, since it requires that a Sensor i be the neighbor of a *chosen* sensor, and also have a smaller UID than this *chosen* sensor but not the least UID out of all the neighbors of this *chosen* sensor, and that all sensors within the transmission disk of Sensor i are also neighbors of a *chosen* sensor, before Sensor i is unmarked. Also, since Algorithm 3 uses the $Connector(i)$ predicate to ensure connectivity and uses the $LessNotLeastNgbrOfChsn(i)$ predicate as part of its redundancy predicate, in any particular covered area of the query region, only the node with the greatest and the least UID will be marked as *chosen*. In addition to this, Rule k has the weakest redundancy predicate, since it requires that all sensors within the transmission disk of Sensor i be covered by marked sensors and that Sensor i also has the least UID out of all the nodes that cover its transmission disk, before it is unmarked. Also, as shown in Figure 5(b), each sensor in the final cover set chosen by Algorithms 1

and 2 "dominated" a greater number of nodes than Rule k. Thus Algorithms 1 and 2 "dominated" a greater number of nodes than Rule k. Algorithm 3 "dominated" a greater number of nodes than Rule k at query region sizes less than 90 *times* 90 square graph units, but fewer number of nodes than Rule k at query region sizes greater than this. Thus Algorithms 1 and 2 did outperform Rule k in the sense that they allowed more nodes to be in an "inactive" state at all the query region sizes tested in our simulation, and Algorithm 3 outperformed Rule k at query region sizes less than 90×90 square units. However, as shown in Figure 5(c), Algorithm 1 had the highest stabilization time of all the algorithms. This increased stabilization time is attributed to the fact that Algorithm 1 has the strongest redundancy predicate, and therefore will incur the greatest time cost when unmarking redundant *chosen* nodes and again producing a sensor cover consisting of nonredundant nodes after restabilization. Furthermore, the stabilization time of Algorithm 3 is greater than Algorithm 2 and Rule k. This is due to the fact that Algorithm 3 has a redundancy predicate that is not weaker than that of both algorithms, and yet sends FindCycle(i, x) and FindCycle(i, y) messages that must travel throughout the network.

Table 2. Number of MCSC Sensors, Query Region Sensors per MCSC Sensor, and Stabilization Times for Algorithms 1, 2, 3, and Rule k at Various Sensor Densities.

		Number of Sensors (x 100,000)				
		1.5	**2.0**	**2.5**	**3.0**	**3.5**
Alg. 1	**Number of MCSC Sensors**	294	273	283	286	296
Alg. 1	**Qry Rgn Sensors / MCSC Sensor**	22.8	33.8	40.1	48.9	53.4
Alg. 1	**Stabilization Time (min.)**	13.3	43.8	103.8	192.4	285.3
Alg. 2	**Number of MCSC Sensors**	316	313	318	309	317
Alg. 2	**Qry Rgn Sensors / MCSC Sensor**	22.1	28.9	35.9	44.4	49.7
Alg. 2	**Stabilization Time (min.)**	9.9	14.6	20.8	27.9	35.7
Alg. 3	**Number of MCSC Sensors**	341	347	349	364	470
Alg. 3	**Qry Rgn Sensors / MCSC Sensor**	20.1	26.6	32.9	37.4	34.2
Alg. 3	**Stabilization Time (min.)**	8.0	16.5	26.6	42.3	64.4
Rule k	**Number of MCSC Sensors**	332	344	342	343	360
Rule k	**Qry Rgn Sensors / MCSC Sensor**	20.8	26.4	34.1	39.8	44.7
Rule k	**Stabilization Time (min.)**	2.2	4.1	7.6	11.4	18.2

Furthermore, Table 2 shows that the size of the final cover sets produced by Algorithms 1 and 2 is smaller than that produced by Algorithm 3 and Rule k. Therefore, both Algorithms 1 and 2 outperformed Rule k in terms of the size of the final cover set at all sensor densities tested in our simulation. The final cover sets produced by Algorithm 3 and Rule k were very similar in terms of size, when the total number of sensors in the simulation was less than 300,000 nodes. Therefore, both algorithms produced nearly the same number of nodes in the final cover set, when the total number of nodes deployed was less than 300,000 nodes.

The number of $MCSC$ sensors for both Algorithms 1 and 2 did not monotonically increase when the node density was increased, while that of Rule k did increase sharply when the node density was greater than 300,000 nodes per 500×500 graph units. This may be attributed to the fact that at higher node densities, there may have been a greater number of nodes that covered any particular marked sensor's transmission disk, and thus a less likelihood that a marked sensor had the least UID of all the sensors that covered its transmission disk. Therefore, fewer nodes would have been unmarked at higher node densities by Rule k.

Table 3. Number of MCSC Sensors, Query Region Sensors per MCSC Sensor, and Stabilization Times for Algorithms 1, 2, 3, and Rule k at Varying Sizes of R_C

		Size of Radius of Communication				
		6	**7**	**8**	**9**	**10**
Alg. 1	**Number of MCSC Sensors**	286	223	177	143	113
Alg. 1	**Qry Rgn Sensors / MCSC Sensor**	48.9	61.3	77.0	97.3	120.9
Alg. 1	**Stabilization Time (min.)**	192.4	245.1	317.8	442.0	505.2
Alg. 2	**Number of MCSC Sensors**	309	245	183	151	120
Alg. 2	**Qry Rgn Sensors / MCSC Sensor**	44.4	54.9	74.1	89.4	114.4
Alg. 2	**Stabilization Time (min.)**	27.9	28.4	31.7	34.3	40.8
Alg. 3	**Number of MCSC Sensors**	364	352	275	237	221
Alg. 3	**Qry Rgn Sensors / MCSC Sensor**	37.4	39.2	49.2	58.0	61.9
Alg. 3	**Stabilization Time (min.)**	42.3	51.5	56.0	73.3	81.1
Rule k	**Number of MCSC Sensors**	343	278	222	177	145
Rule k	**Qry Rgn Sensors / MCSC Sensor**	39.8	49.4	61.8	77.2	95.8
Rule k	**Stabilization Time (min.)**	11.4	15.4	18.8	22.9	29.5

Table 3 and Figure 5(g) show that Algorithms 1, 2, 3, and Rule k produced smaller final cover sets as the radius of communication of the sensors was increased. However, Algorithms 1 and 2 produced smaller cover sets than Rule k at all sizes of the radius of communication that were tested. Also, as shown in Figure 5(h), each sensor in the final cover set chosen by Algorithms 1 and 2 "dominated" a greater number of nodes than Rule k, at all sizes of the radius of communication that were tested. This indicates that Algorithms 1 and 2 outperformed Rule k, in terms of the size of the final cover set and the number of query region sensors covered by each node in the final cover set, at all sizes of the radius of communication that were tested. Also, both Algorithms 3 and Rule k produced a cover set that was very similar in size, when the size of the radius of communication of the sensors was 6 and the size of the query region was 90×90 graph units.

As the size of the radius of communication was increased, each sensor chosen by Algorithms 1, 2, and 3 also "dominated" a greater number of query region sensors. This seems intuitive since the size of the radius of communication is equal to the size of the radius of the sensing disk of sensors in Algorithms 1, 2, and 3. Therefore, as the radius of communication was increased in size, there were a greater number of nodes within the transmission disk, and thus within the sensing disk, of chosen sensors in the simulation. Thus, in Algorithms 1, 2, and 3, there was a smaller probability of nodes being chosen by $\mathcal{A}_2$ and $\mathcal{A}_3$. Also, since there was an increased likelihood that a node was the neighbor of another *chosen* sensor that had a greater UID than its own but was not the "least UID" neighbor of this *chosen* sensor, a greater number of *chosen* sensors may have been unmarked by the redundancy predicates of both algorithms.

The stabilization times of both Algorithm 2 and Rule k were very similar at all sizes of the radius of communication that were tested. Also, despite the fact that Algorithm 1 had a higher stabilization time than Algorithm 2, 3, and Rule k, Algorithm 1 still produced fewer nodes in the final cover set. While Rule k does stabilize faster

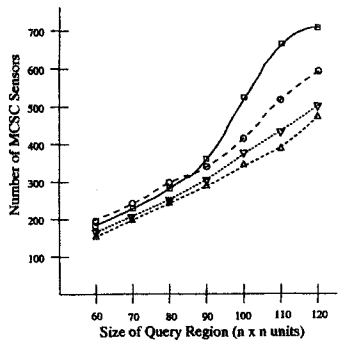
than Algorithms 1, 2, and 3, the slower stabilization times seem justified due to the fact that the latter three algorithms do not compromise connectivity, nor coverage.

The time complexity of Algorithms 1 and 2 is $O(\Delta^2)$, where Δ is the maximum degree of a node in the network. The time complexity of Algorithm 3 is $O(D)$, in which D is the diameter of the network. The stabilization times of all three algorithms measured during simulation, however, may increase due to the time cost associated with unmarking redundant *chosen* nodes and again producing a sensor cover consisting of nonredundant nodes after restabilization.

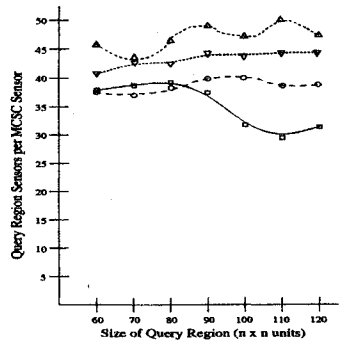
The screenshots in Figures 6, 7, and 8 show the final cover sets that are produced by Algorithms 1, 2, and 3, respectively, when the radius of communication is 8. In all screenshots, each sensor is depicted as a black spot, and areas that are occupied by sensors are shown as black areas. Also, the query region is outlined by a red square, and the sensing disk of each *chosen* sensor is depicted as a light blue circle with a white border. Any uncovered regions within the query region will be shown as black areas within the red rectangle.

In addition to this, Algorithms 1, 2, and 3 are fault-tolerant in terms of the self-$*$ feature. This implies that Algorithms 1, 2, and 3 are also self-contained, meaning that the number and location of nodes affected by a faulty node, are minimally contained within the neighborhood of the faulty sensor. It also implies that the system self-heals after restabilization, without any external intervention. This is shown in the screenshots in Figures 9, 10, 11 and in Figures 12 and 13. The screenshots in Figures 9, 10, and 11 are those of Algorithms 1, 2, and 3, respectively, when there are two faulty nodes that are neighbors of each other. The screenshots in Figures 12 and 13 are those of Algorithms 1 and 2, respectively, when there are two faulty nodes that are not neighbors of each other. In these screenshots, the sensing disks of faulty nodes are pink and those of nodes that were faulty and changed their status after restabilization are green. In this simulation, the sensing disks of nodes that were not faulty and yet changed their status after restabilization should have changed from

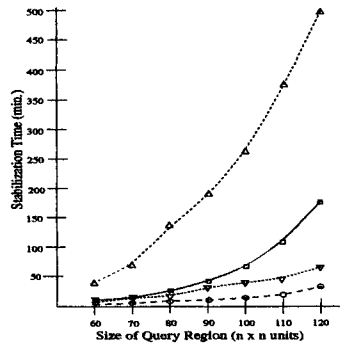
light blue to yellow. As the simulation shows, in all three algorithms, when a node's status is corrupted by an arbitrary fault, the system is self-contained and self-heals after restabilization, without any external intervention.



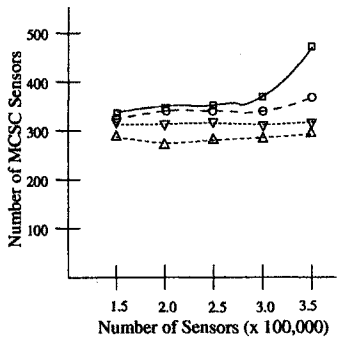
(a) Number of MCSC Sensors for Various Query Region Sizes



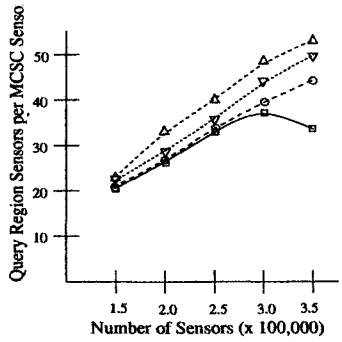
(b) Query Region Sensors per MCSC Sensor for Various Query Region Sizes



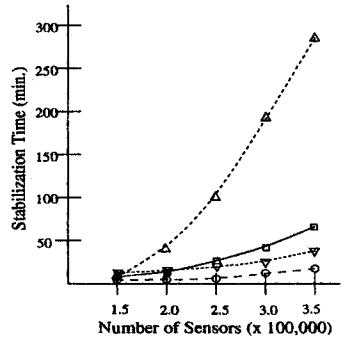
(c) Stabilization Time (minutes) for Various Query Region Sizes



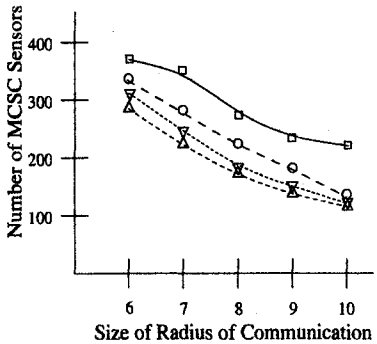
(d) Number of MCSC Sensors for Various Sensor Densities



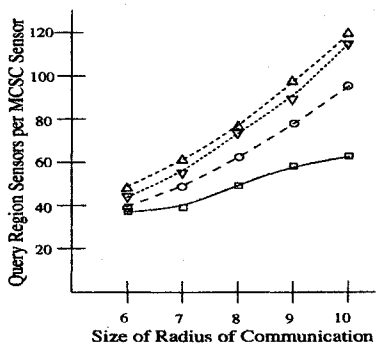
(e) Query Region Sensors per MCSC Sensor for Various Sensor Densities



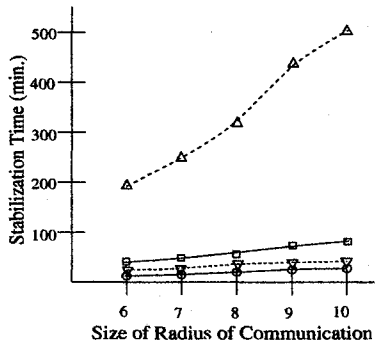
(f) Stabilization Time (minutes) for Various Sensor Densities



(g) Number of MCSC Sensors for Various Sizes of Radius of Communication



(h) Query Region Sensors per MCSC Sensor for Various Sizes of Radius of Communication



(i) Stabilization Time (minutes) for Various Sizes of Radius of Communication

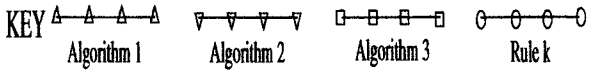


Figure 5. Graphs of Experimental Results.

Figure 6. Screenshot of Simulation of Algorithm 1 When the Radius of Communication is 8

Figure 7. Screenshot of Simulation of Algorithm 2 When the Radius of Communication is 8

Figure 8. Screenshot of Simulation of Algorithm 3 When the Radius of Communication is 8

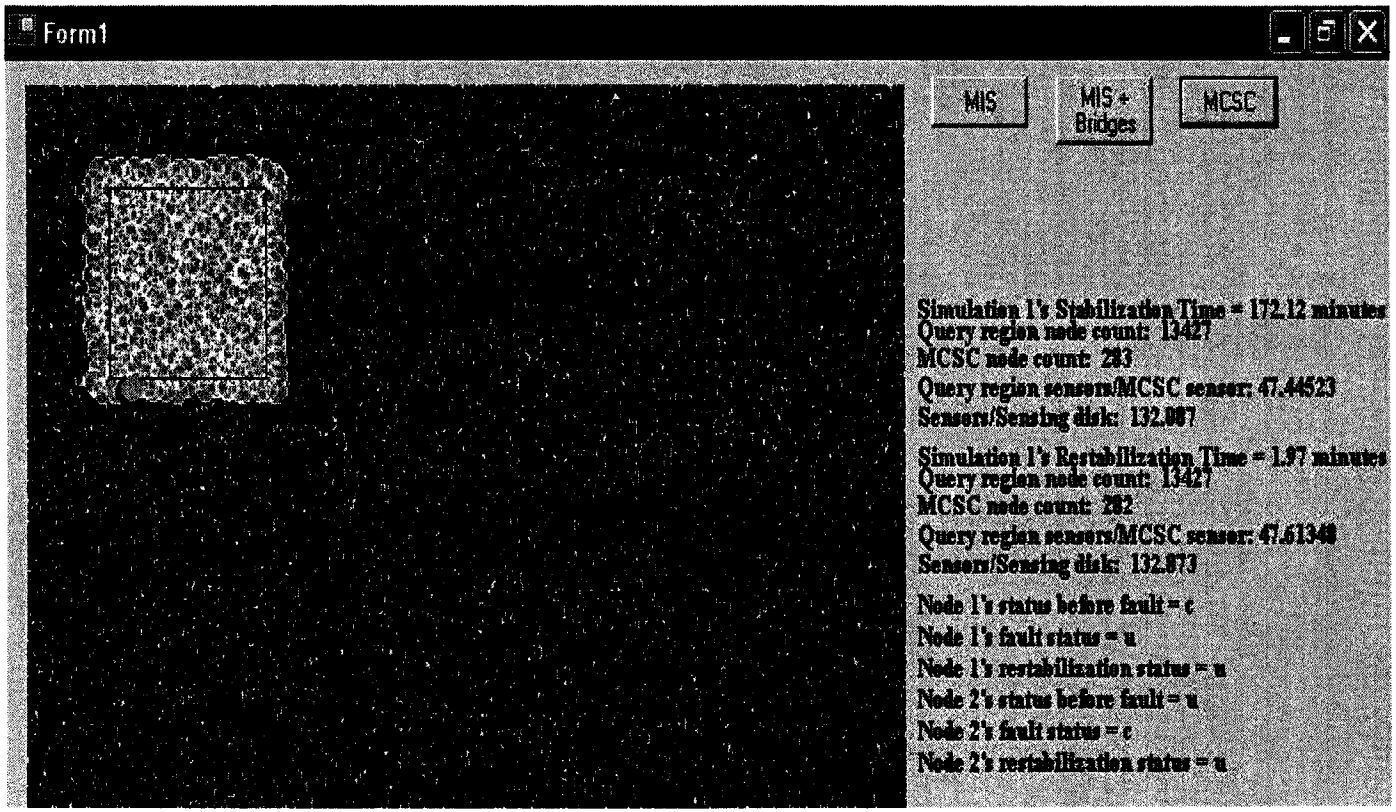


Figure 9. Screenshot of Simulation of Self-Containment of Algorithm 1 With 2 Neighboring Faults

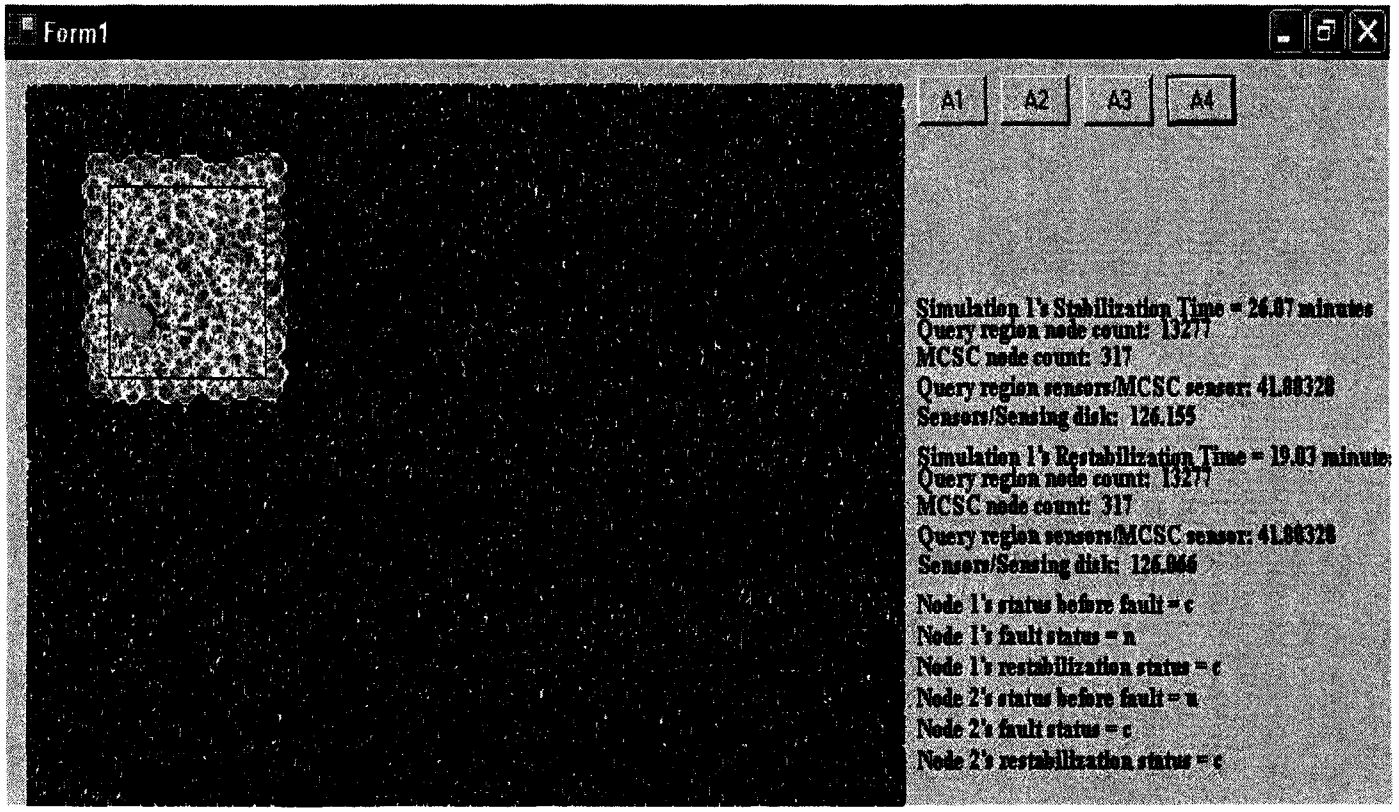


Figure 10. Screenshot of Simulation of Self-Containment of Algorithm 2 With 2 Neighboring Faults

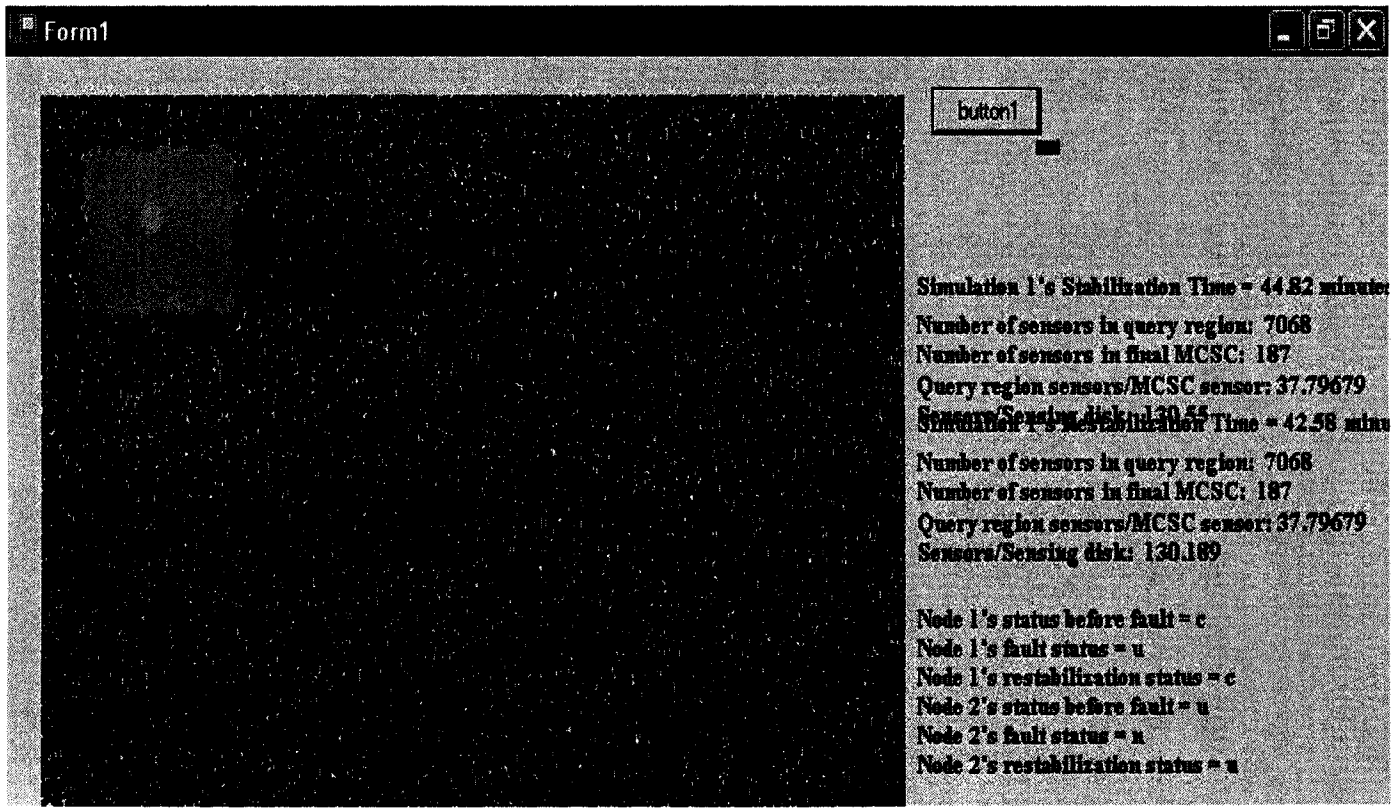


Figure 11. Screenshot of Simulation of Self-Containment of Algorithm 3 With 2 Neighboring Faults

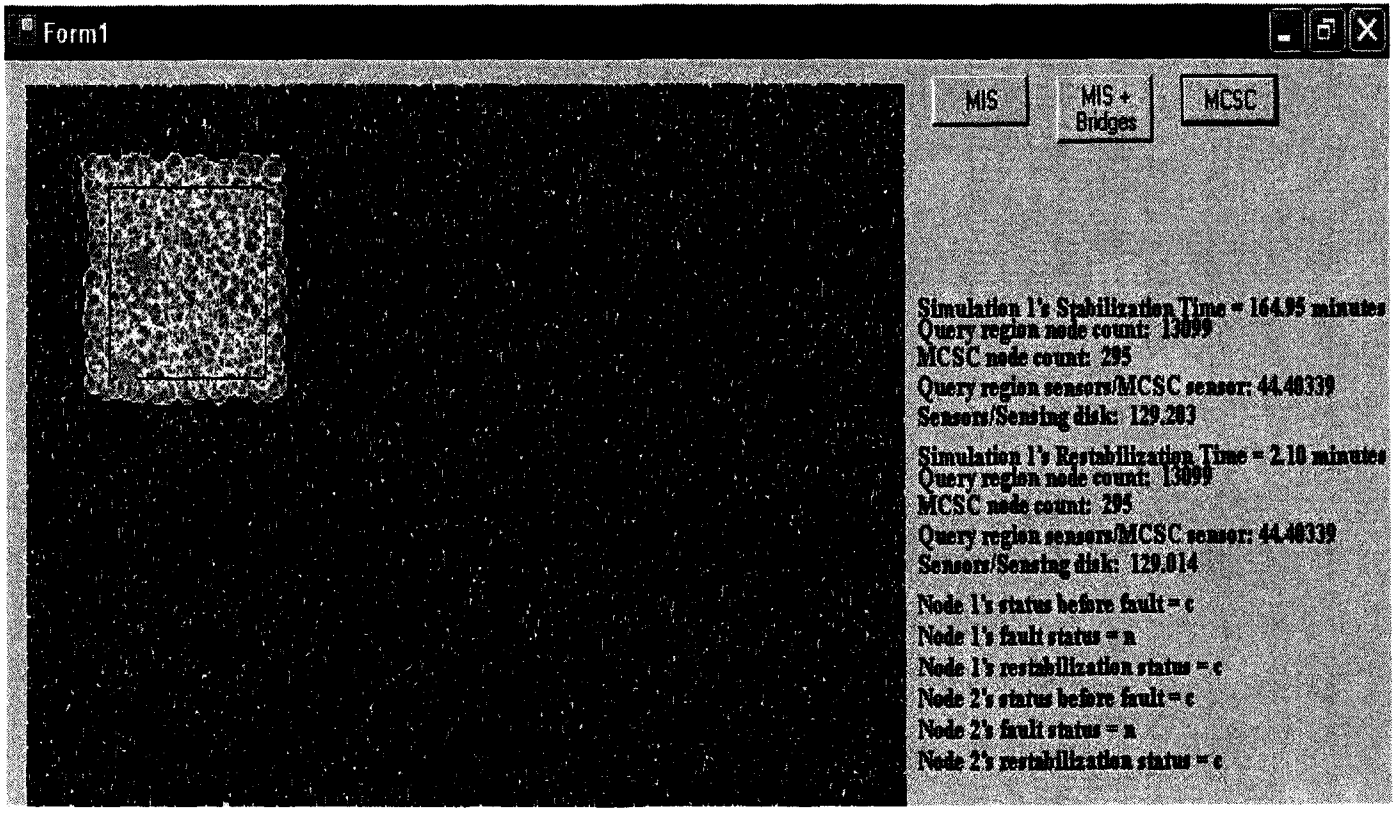


Figure 12. Screenshot of Algorithm 1's Self-Containment Simulation With 2 Non-Neighboring Faults

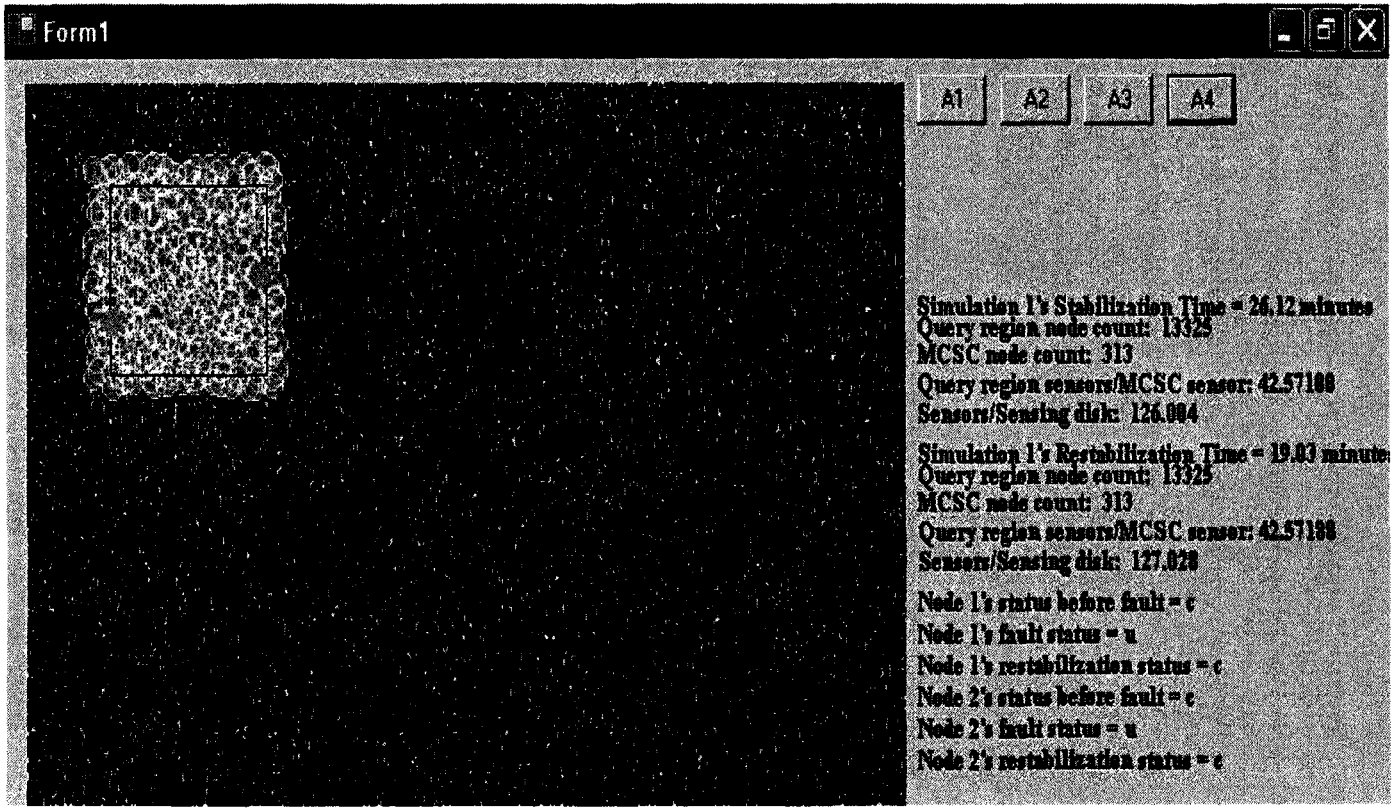


Figure 13. Screenshot of Algorithm 2's Self-Containment Simulation With 2 Non-Neighboring Faults

CHAPTER 9

CONCLUSION AND FUTURE RESEARCH

The main motivation of our research was to design a totally distributed self-$*$ query response system in sensor networks. We presented three local, distributed, scalable, self-$*$ solutions to the minimum connected sensor cover problem and showed how these solutions are self-organizing and self-healing as well. The algorithms are also self-$*$ contained, meaning that after a fault occurs in the system, after restabilization, only nodes within the locality of the faulty nodes change status. Throughout the design process, we followed a power-aware approach. Although our goal was to design a minimal size sensor cover, we used power-awareness as a strong guide in our design, and accepted a slight degree of suboptimality.

The minimum connected cover set produced by Algorithms 1, 2, and 3 are minimal in the sense that they do not include another cover set. Algorithms 1 and 2 outperformed Rule k in terms of producing a smaller final cover set at all query region sizes that were tested in our simulation. Algorithm 3 outperformed Rule k in terms of producing a smaller final cover set at query region sizes that were less than 90×90 square graph units. Also, at all sensor densities and all sizes of the radius of communication that were tested, both Algorithms 1 and 2 outperformed Rule k in terms of producing a smaller final cover set. The final cover sets produced by Algorithm 3 and Rule k were very similar in terms of size, when the total number of sensors deployed was less than 300,000 nodes, and the size of the query region was 90×90 square graph units. Despite the fact that the stabilization time of Algorithm 1 is greater than that of Algorithm 2, 3, and Rule k, the final cover set produced by Algorithm 1 is smaller than that produced by Algorithm 2, 3, and Rule

k. Also, Algorithm 1, Algorithm 2, and Algorithm 3 are truly fault-tolerant and are self-contained, meaning that after a fault occurs in the system, after restabilization, only nodes within the locality of the faulty nodes change status.

This research showed that the concept of self-stabilization subsumes many other self-$*$ properties. The connected sensor cover problem is a global task since nodes cannot locally compute the final response to the query. However, we still required our algorithms to be *local* in the sense that no node in the proposed algorithms collect global information, and no node behaves as a special node in any stage of the execution of the algorithms. In our solution, every node can decide if it should be *unchosen*, *undecided*, *chosen*, or *removed* (in the case of Algorithm 3), during the computation of the response to a query, based upon local information. In summary, we achieved a global objective by using local algorithms.

Sensing coverage characterizes the monitoring quality provided by a sensor network in a designated region. Different applications may require different degrees of sensing coverage. In this regard, we can extend our solution in a couple of ways. Firstly, we may write a parametric solution where the input query will include the degree of coverage expected. The redundancy predicate will be relaxed to allow the corresponding higher degree of coverage. Secondly, we can simply assume a particular degree (> 1) of coverage in our algorithm. Similar to the implementation of a higher degree of coverage to achieve better robustness, we may also require a higher degree of connectivity for the same purpose (i.e., to increase the level of fault-tolerance). We can extend the neighborhood connectivity checking to k-node ($k > 1$) disjointness in the communication graph. Unfortunately, higher degree of coverage/connectivity would require more communication cost, i.e., consuming more power. We can conduct a study on the trade off between connected cover size optimality vs. robustness and energy efficiency.

Also, our work can be extended by finding an algorithm to form a minimum connected "clusterhead" set, such that every node in the graph G(V, E) is either in

the "clusterhead" set, adjacent to a node in the "clusterhead" set, or adjacent to a neighbor of a node in the "clusterhead" set. Nodes in this "clusterhead" set will then be responsible for aggregating, routing, or transmitting data that has been collected from the query region.

Our work may also be extended to include sensors with sensing or transmission radii that are different in size. That is, we may increase or decrease the sensing radii of sensors used in our research, and study the effect of this change upon the size and degree of coverage of the final cover set that is obtained.

BIBLIOGRAPHY

[1] IBM03 http://www.research.ibm.com/autonomic/.

[2] ATM Atmel, inc. at90s4434/ls4434/s8535/ls8535. Preliminary (Complete) Datasheet.

[3] IETF05 Ietf working group: Mobile ad-hoc networks (manet). http://www.ietf.org/html.charters/manet-charter.html.

[4] NES03 Nest project at berkeley. http://webs.cs.berkeley.edu/nest-index.html.

[5] AN03 Wireless ad hoc networks. http://w3.antd.nist.gov/wahn_ssn.shtml.

[6] Cro03 Crossbow technology mica2 wireless measurement system datasheet, 2003. http://www.xbow.com/Products/Wireless_Sensor_Networks.htm.

[7] AGS93 NT Adams, R Gold, BN Schilit, MM Tso, and R Want. An infrared network for mobile computers. In *USENIX Symposium on Mobile and Location-Independent Computing*, pages 41–51, Aug 1993.

[8] AD97 Y Afek and S Dolev. Local stabilizer. In *Israel Symposium on Theory of Computing Systems*, pages 74–84, 1997.

[9] Aro92 A Arora. *A foundation of fault-tolerant computing.* Ph.D. dissertation, The University of Texas at Austin, Dec 1992.

[10] AG94 A Arora and MG Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[11] APV91 B Awerbuch, B Patt-Shamir, and G Varghese. Self-stabilization by local checking and correction. In *FOCS91 Proceedings of the Thirtyfirst Annual IEEE Symposium on Foundations of Computer Science*, pages 268–277, 1991.

[12] CA04 J Carle and D Simplot-Ryl. Energy-efficient area monitoring for sensor networks. *IEEE Press*, pages 40–46, Feb 2004.

[13] CJB01 B Chen, K Jamieson, H Balakrishnan, and R Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *MobiCom02 Proceedings of the Seventh Annual Inernational Conference on Mobile Computing and Networking*, pages 85–96, Jul 2001.

[14] CDP03 A Cournier, AK Datta, F Petit, and V Villain. Enabling snap-stabilization. In *IEEE Twentythird International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 12–19, May 2003. Providence, Rhode Island.

[15] DW03 F Dai and J Wu. Distributed dominant pruning in ad hoc networks. *Proceedings of ICC'03*, 2003.

[16] Dij74 EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17(11):643–644, Nov 1974.

[17] Dij73 EW Dijkstra. Ewd386 the solution to a cyclic relaxation problem. In *Selected Writings on Computing: A Personal Perspective*, pages 34–35. Springer-Verlag, 1982. EWD386's original date is 1973.

[18] Dol00 S Dolev. *Self-Stabilization.* MIT Press, 2000.

[19] DGS96 S Dolev, MG Gouda, and M Schneider. Memory requirements for silent stabilization. In *PODC96 Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 27–34, 1996.

[20] D04 J Dowling, R Cunningham, E Curran, and V Cahill. Component and system-wide self-* properties in decentralised distributed systems. *SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems*, Jun 2004.

[21] EGH02 D Estrin, R Govindan, J Heidemann, and S Kumar. Next century challenges: Scalable coordination in sensor networks. *Mobile Computing and Networking*, pages 263–270, 1999.

[22] FP03 Armando Fox and David Patterson. Self-repairing computers. *Scientific American*, Jun 2003.

[23] F00 M Frodigh, P Johansson, and P Larsson. Wireless ad hoc networking: the art of networking without a network. *Ericsson Review*, (4):248–263, 2000.

[24] GGH96 S Ghosh, A Gupta, T Herman, and SV Pemmaraju. Fault-containing self-stabilizing algorithms. In *PODC96 Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 45–54, May 1996.

[25] GM91 MG Gouda and N Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458, 1991.

[26] GDG03 H Gupta, SR Das, and Q Gu. Connected sensor cover: Self-organization of sensor networks for efficient query execution. In *MobiHoc03 Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 189–200, 2003.

[27] HKB03 WR Heinzelman, J Kulik, and H Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. MOBICOM*, pages 174–185, 1999.

[28] HT03 T Herman and S Tixeuil. A distributed tdma slot assignment algorithm for wireless sensor networks. Technical Report 1370, LRI, Université Paris-Sud XI, Sep 2003.

[29] C01 J Hill, R Szewczyk, and A Woo. Tinyos: Operating system for sensor networks. http://www.eecs.berkeley.edu/IPRO/Summary/01abstracts/szewczyk.1.html.

[30] HSW00 J Hill, R Szewczyk, A Woo, S Hollar, D Culler, and K Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000. ASPLOS-IX.

[31] ISS05 F Ingelrest, D Simplot-Ryl, and I Stojmenovic. Smaller connected dominating sets in ad hoc and sensor networks based on coverage by two-hop neighbors. Technical report, Institut National De Recherche En Informatique Et En Automatique, Apr 2005.

[32] I00 C Intanagonwiwat, R Govindan, and D Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *ACM Mobicom 2000*, Apr 2000.

[33] KKP99 JM Kahn, RH Katz, and KSJ Pister. Next century challenges: Mobile networking for smart dust. In *International Conference on Mobile Computing and Networking(MOBICOM)*, pages 271–278, Nov 1999.

[34] Kes88 JLW Kessels. An exercise in proving self-stabilization with a variant function. *Information Processing Letters*, 29:39–42, 1988.

[35] KU04 F Kuhn, T Moscibroda, and R Wattenhofer. Initializing newly deployed ad hoc and sensor networks. *MobiCom '04*, 2004.

[36] KAR00 VSA Kumar, S Arya, and H Ramesh. Hardness of set cover with intersection 1. In *ICALP00 Proceedings of the Twentyseventh International Colloquium on Automata, Languages and Programming*, pages 624–635, 2000.

[37] L01 A Lim. Distributed services for information dissemination in self-organizing sensor networks. *Journal of Franklin Institute*, 338:707–727, 2001.

[38] LI04 H Liu, Y Pan, and J Cao. An improved distributed algorithm for connected dominating sets in wireless ad hoc networks. *Proceedings of the ISPA '04*, Dec 2004.

[39] M02 A Mainwaring, J Polastre, R Szewczyk, D Culler, and J Anderson. Wireless sensor networks for habitat monitoring. *WSNA '02*, Sep 2002.

[40] Mil94 DL Mills, Z Yang, and TA Marsland. Internet time synchronization: The network time protocol. *Global States and Time in Distributed Systems, IEEE Computer Society Press*, 1994.

[41] Pat03 David Patterson. Recovery-oriented computing — overview. http://roc.cs.berkeley.edu/.

[42] Pis03a KSJ Pister. Smart dust. http://robotics.eecs.berkeley.edu/ pister/ SmartDust.

[43] R99 EM Royer and C Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, Apr 1999.

[44] SSS03 S Shakkottai, R Srikant, and N Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In *INFOCOM03 Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1073–1083, Apr 2003.

[45] Sun01 J Sun. Mobile ad hoc networking: An essential technology for pervasive computing. In *Proceedings International Conferences on Info-Tech & Info-Net*, pages 316–321, 2001.

[46] Tel00 G Tel. *Introduction to distributed algorithms.* Cambridge University Press, second edition, 2000.

[47] Var93 G Varghese. *Self-stabilization by local checking and correction.* Ph.D. dissertation, MIT, 1993.

[48] Var94 G Varghese. Self-stabilization by counter flushing. In *PODC94 Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 244–253, 1994.

[49] WXZ03 X Wang, G Xing, Y Zhang, C Lu, R Pless, and C Gill. Integrated coverage and connectivity configuration in wireless sensor networks. In *ACM SenSys03 Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 28–39, Nov 2003.

[50] WB96 M Weiser and JS Brown. The coming age of calm technology. Technical report, Xerox PARC, Oct 1996.

[51] WC01 A Woo and DE Culler. A transmission control scheme for media access in sensor networks. In *Proc. ACM/IEEE Mobicom, Mobile Computing and Networking*, pages 221–235, 2001.

[52] WU02 J Wu. Extended dominating-set-based routing in ad hoc wireless networks with unidirectional links. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):866–881, Sep 2002.

[53] WU99 J Wu and H Li. On calculating connected dominating sets for efficient routing in ad hoc wireless networks. *Proceedings of DialM'99*, pages 7–14, 1999.

[54] YG03 Y Yao and J Gehrke. Query processing for sensor networks. *Proceedings of the 2003 CIDR Conference*, 2003.

[55] ZH03 H Zhang and JC Hou. Maintaining sensing coverage and connectivity in large sensor networks. Technical Report UIUCDCS-R-2003-2351, University of Illinois at Urbana Champaign, Jun 2003.

VITA

Graduate College
University of Nevada, Las Vegas

Rajesh Patel

Home Address:
2001 Shelbyville Street
Henderson, NV 89052

Degrees:
Bachelor of Science, Biological Sciences
University of Southern California

Bachelor of Science, Nuclear Medicine
University of Nevada, Las Vegas

Special Honors and Awards:
Graduate Assistantship/Teaching Assistantship (School of Computer Science), Cum Laude, The Chancellor's List 2004-2005, Howard Hughes Research Fellowship, UNLV Physics Award, Dean's Honor List, The National Dean's List, Who's Who Among Students in American Universities and Colleges

Publications:
Distributed Self-* Minimum Connected Covering of a Query Region in Sensor Networks, *ISPAN 2005*

Thesis Title: Distributed Self-* Minimum Connected Sensor Cover Algorithms

Thesis Examination Committee:
Chairperson, Dr. Ajoy K. Datta, Ph. D.
Co-Chairperson, Dr. Maria Gradinariu, Ph. D.
Committee Member, Dr. John Minor, Ph. D.
Committee Member, Dr. Yoohwan Kim, Ph. D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph. D.