1-1-2007

Self-* distributed query region covering in sensor networks

Ai Yamazaki
University of Nevada, Las Vegas

SELF-* DISTRIBUTED QUERY REGION COVERING IN SENSOR NETWORKS

by

Ai Yamazaki

Bachelor of Science
University of Nevada, Las Vegas
2001

Bachelor of Arts
University of Nevada, Las Vegas
2005

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
August 2007

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval

The Graduate College
University of Nevada, Las Vegas

AUGUST 13, 2007

The Thesis prepared by

AI YAMAZAKI

Entitled

SELF -* DISTRIBUTED QUERY REGION COVERING IN SENSOR NETWORKS

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

1017-53

experimentally. The simulation results show that our solutions provide better performance in terms of coverage than pre-existing self-stabilizing algorithms.

ABSTRACT

Self-* Distributed Query Region Covering in Sensor Networks

by

Ai Yamazaki

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

Wireless distributed sensor networks are used to monitor a multitude of environments for both civil and military applications. Sensors may be deployed to unreachable or inhospitable areas. Thus, they cannot be replaced easily. However, due to various factors, sensors' internal memory, or the sensors themselves, can become corrupted. Hence, there is a need for more robust sensor networks. Sensors are most commonly densely deployed, but keeping all sensors continually active is not energy efficient. Our aim is to select the minimum number of sensors which can entirely cover a particular monitored area, while remaining strongly connected. This concept is called a *Minimum Connected Cover* of a query region in a sensor network. In this research, we have designed two *fully distributed*, robust, *self-*$*$ solutions to the minimum connected cover of query regions that can cope with both transient faults and sensor crashes. We considered the most general case in which every sensor has a different sensing and communication radius. We have also designed extended versions of the algorithms that use multi-hop information to obtain better results utilizing small *atomicity* (i.e., each sensor reads only one of its neighbors' variables at a time, instead of reading all neighbors' variables). With this, we have proven*self-*$*$ ($self-configuration$, $self-stabilization$, and $self-healing$) properties of our solutions, both analytically and

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I would like to express sincere appreciation to my thesis advisor, Dr. Ajoy K. Datta for his guidance, insight, and support for this research. His trust and confidence in my abilities have truly encouraged me throughout my graduate study. I would like to offer a special thanks to Dr. Maria Gradinariu at IRISA/Universite Rennes 1, France, for giving us direction for research. Her advice and contribution was essential in completing this work. Thanks go to Rajesh Patel for simulating the algorithms, whose comments and suggestions made our work greatly refined. I am also grateful to Dr. John Minor, Dr. Yoohwan Kim, and Dr. Venkatesan Muthukumar, for their participation in my committee.

My special gratitude goes to my family. I would like to dedicate this thesis to them for their understanding, motivation, and patience. I am thankful to all faculty members and friends who made my stay at the University of Nevada, Las Vegas a memorable and valuable experience.

CHAPTER 1

INTRODUCTION

Recent technological advances in wireless networking, memory, and embedded microprocessors have made it possible to produce tiny, low-cost, and low-powered sensor nodes. A large number of those tiny sensors compose wireless sensor networks which have revolutionized our world. Wireless distributed sensor networks are used to monitor a multitude of environments for both civil and military applications, such as traffic monitoring, wildlife habitat monitoring, home security, and battlefield awareness. Unlike current Internet information gathering services, wireless sensor networks provide users more localized and application-specific data in a timely manner.

One of the most well-known sensor nodes is the MICA Mote which was initially developed at the University of California at Berkeley. Today, it is commercially available to the public by Crossbow Technology, Inc. [3] and has been widely used by researchers. DARPA has initiated a research project called Network Embedded Systems Technology (NEST) [4]. Funded by DARPA, the Smart Dust project was also developed at UC Berkeley attempting to design a sensor system which could be integrated into a package only a few cubic millimeters in size [50]. Top Silicon Valley companies, like Intel Corporation are also in the business of manufacturing such devices [5].

Since these network sensors have limited battery power, the life span of sensor networks is usually expected to be very short. Thus, they are energy constrained. Also, due to

various factors, sensors' internal memory, or the sensors themselves, can become corrupted. Unfortunately, as sensors may be deployed in unreachable or inhospitable areas, they cannot be replaced easily. Although they are most commonly densely deployed, keeping all sensors continually active is not energy efficient due to the above reason.

In addition to energy consumption, the topology of a network may change frequently due to malfunctions or environmental situations. Thus, it is impractical to pre-configure a network and deploy each sensor in a certain deterministic position for a large number of sensors.

Due to all these constraints, there is a need for more fault-tolerant and energy-efficient sensor networks which must be *self-configuring* and *self-maintaining* or *self-healing*. The term *Self-*$*$ has been used to describe all these properties like self-organizing, self-configuring, self-healing, etc. In this thesis, we will present a *self-stabilizing* solution to this very challenging energy saving problem in sensor networks. Then we will show that this solution can also be considered as a self-$*$ solution.

A sensor network's topology can be described using a graph $G(V, E)$, where vertices are sensor nodes and edges are communication links between sensors. Every sensor has a certain *communication range* and can communicate with only those sensors which are located within its communication range. If two sensors are located within each other's communication range, then there is a bi-directional communication link and they are called *neighbors*. Similarly, every sensor has a certain range it can sense or gather data which is called a sensor's *sensing range*. A group of sensors is said to cover a specific region when the union of the sensing disks of these sensors completely cover this region.

In sensor networks, queries may be sent to "sense" data or events over a particular region, called a *query region*. Our aim is to select the minimum number of sensors which

can entirely cover a particular query region, while remaining strongly connected. This concept is called *Minimum Connected Cover* of a query region in a sensor network. In its general form, this problem is known to be NP-hard [38, 45].

1.1 Contributions

The main goal of our research is to design an energy-efficient query response sensor network protocol. Two main topics are considered in this research: the design of wireless sensor networks and the design of self-$*$ systems. The first contribution of this thesis is the discussion of the wireless sensor networks. We discuss the current trend and solutions to many significant problems in this area. The second contribution is the study of the self-$*$ systems which includes the properties of self-organizing, self-maintaining, and self-healing amongst others.

The most important contribution of our research is to connect the self-$*$ systems and wireless sensor networks to design a self-$*$ energy-efficient solution to the minimum connected sensor cover problem. Considering the most general case in which every sensor has a different sensing and communication radius, we have designed two *fully distributed*, robust, $self-*$ solutions to the minimum connected cover of query regions that can cope with both transient faults and sensor crashes. We have also designed extended versions of the algorithms which use multi-hop information to obtain better results utilizing small *atomicity* (i.e., each sensor reads only one of its neighbors' variables at a time, instead of reading all neighbors' variables at once).

1.2 Outline of the Thesis

In Chapter 2, we discuss mobile wireless networks such as cellular networks and wireless ad hoc networks. This chapter also includes the overview of wireless sensor networks. We then delve into self-$*$ systems in Chapter 3. Descriptions of many types of fault-tolerant systems are presented in the context of self-$*$. The motivation of this research, previous solutions in related areas, the model, and the program used in our solutions are described in Chapter 4. The problem of minimum connected sensor cover is formally defined in this chapter. Our self-stabilizing solutions for this problem is presented in Chapters 5, 6, and 7. We include the proof of correctness for each solution in each chapter. In Chapter 5, the single-hop UID based algorithm is introduced. A modified version of that algorithm is given in Chapter 6. In Chapter 7, we present multi-hop algorithms. Simulation results for all algorithms presented in this paper are included in Chapter 8. Finally, we summarize our research, and present further concepts for future research in Chapter 9.

CHAPTER 2

WIRELESS NETWORKS

In this chapter, we will give brief descriptions of wireless networks, which include cellular wireless networks and ad hoc wireless networks to understand the difference between sensor networks and other types of wireless networks. The overview of sensor networks will also be presented. Wireless networks have been playing an important role in the development of communication technology in the last century and it is still growing rapidly. It is an information transmission system that uses electromagnetic waves such as radio waves instead of physical wires. Examples of wireless networks are WLAN (wireless local area networks), GSM (Global System for Mobile communications), and D-AMPS (Digital Advanced Mobile Phone Service).

2.1 Mobile Wireless Networks

In recent years, our society has become more information oriented and the demand of information accessibility has been growing rapidly. The advantage of using a wireless network is its convenience. Via WLAN, users can access the internet anywhere outside their work place such as remote offices or even coffee shops. Also, its expandability removes the need of physically rewiring the existing network when a machine is added or removed. With these advantages, mobile wireless networks have been experiencing a tremendous growth in popularity amongst people who want information and connectivity anytime and anywhere. This growth has led to many technological advances in this field, and as a result, small, low

cost, and powerful mobile computing devices such as Personal Digital Assistants (PDAs), smart mobile phones, and laptop computers have been developed. Although mobility is one of the key factors of these computing devices' popularity, it makes maintaining communication among the various types of mobile devices critical and challenging. Recent advances in wireless communication technologies have enabled wireless mobile devices to communicate with each other in various ways. Mobile Wireless Networks can be classified into two branches; infrastructured (cellular) and infrastructureless (ad hoc) wireless networks [35]. Both aim to provide reliable communications and computing environment where users are not tethered to their information source.

2.1.1 Infrastructured/Cellular Wireless Networks

In an infrastructure/cellular wireless network, access points (or base stations) are required which enable the mobile devices to connect to each other. Those access points are distributed along a wired backbone and usually connected to a fixed network infrastructure or to the Internet, and act as routers or gateways to forward packets to other devices. Cellular networks are divided into cells and each cell is associated with a base station and covered by this base station. Within its coverage, a base station can communicate directly with mobile hosts by sending and receiving signals. The communication between one mobile host to another is established via a base station and point-to-point connections are usually not established among mobile hosts. A mobile host is able to move from one cell to another. However, to do so, it must cease communication with the old base station and begins communication with the new base station, which is called a handoff. The handoff should not disconnect the existing communication and should not be detectable by a user [51].

Examples of this kind of wireless network are Global System for Mobile Communications (GSM), Universal Mobile Telecommunication System (UMTS), Wireless Local Loop (WLL),

and Wireless Local Area Network (WLAN).

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

2.1.2 Infrastructureless/Ad Hoc Wireless Networks

There may be a need for efficient and dynamic communication of independent mobile users when no fixed wired infrastructure is available. A few examples are emergency/rescue operations, disaster recovery, and military networks. In such situations, organized communication networks can not be relied upon. Thus, establishing reliable networks quickly among a collection of mobile hosts without any centralized administration is required. As such, the development of mobile devices and their networks have been receiving more and more interest.

A network which does not rely on any wired backbone, base stations, or a central controller is called an ad hoc network. In this type of network, communication between mobile hosts is peer-to-peer, so each host has direct communication with another. Hosts also act as relay nodes to forward data packets. Such a network is often called a Mobile Ad Hoc Network (MANET) [2, 33].

The set of applications for a MANET is diverse, ranging from large scale, mobile, and highly dynamic networks, to small and static networks that are constrained by power sources. Examples of applications arenas are military battlefield, civilian environments, and emergency operations.

There are several characteristics of MANET which differ significantly from other types of networks: Since hosts are mobile in MANET, the network topology may change dynamically. However, because there is no centralized controller, each host must be able to detect

this topology change and re-configure the network every time the change occurs in a fully distributed manner. Also, multiple hop routing algorithms may be needed as every host may not be within the communication range of every other host. Hence, the intermediate nodes must serve as routers for other nodes in the network so that data packets can be forwarded to their destinations. In addition to these, MANET has a fluctuating link capacity. Factors such as link quality, fading, noise, and interference are key issues. Security and interception problems are of a concern as well, especially in military applications. Therefore, designing the protocol for MANET is very crucial and those issues must be carefully examined before widespread commercial deployment.

2.2 Wireless Sensor Networks

Sensor networks are one of the main topics of research for this thesis. We will present an overview of the sensor networks in this next section. Some issues and concepts associated with sensor networks are also included.

2.2.1 Overview

Wireless sensor networks have been recognized as one of the most important technologies for the future. It allows us to instrument, observe, and respond to phenomena in the surrounding environment. A number of sensors spread across a geographic area compose sensor networks. Recent technological advancement has enabled the production of small, low-cost, low-powered, distributed sensing devices. These devices are called sensor nodes. They are very different from traditional desktop and server systems [41]. Each sensor node has wireless communication capability and some level of computational ability for signal processing and networking of data, but has a limited energy source. Sensor nodes are usually static. However, some nodes can be mobile. It seems that sensor networks have similarities

with wireless ad hoc networks such as MANET and mobile cellular networks. However the followings characteristics of sensor networks [17] suggest that many recommended protocols for the above two platforms may not be well suited for sensor networks.

1. Battery Power/Energy: Sensor nodes have a limited energy supply. Usually, once deployed, batteries cannot be replaced or recharged.

2. Communication: The communication (transmission) range of a sensor node is limited. Also, the quality of a wireless connection between sensor nodes is limited due to various reasons such as latency and bandwidth.

3. Memory: Sensor nodes have limited memory, hence limited computational power.

4. Location sensing: Sensor nodes may or may not be supported by satellite location determination system such as GPS.

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

6. Size: The number of nodes in a sensor network can be several orders of magnitude higher than the nodes in other ad hoc networks. The number may be in the thousands or millions.

7. Deployment: Sensor nodes are usually densely deployed for the purpose of fault tolerances. Typically, they are deployed randomly. However, in some applications, deterministic deployment is also available.

8. Unattended operations: Depending on the application, sensor nodes are unattended for long periods of time. In most cases, physical maintenance may not be feasible.

9. Topology Changes and Failures: The topology of sensor networks may change dynamically, due to change of position, reachability (e.g., jamming, noise, obstacles, etc.), available energy, malfunctioning, etc. Also sensor nodes can fail easily due to the low cost in manufacturing or environmental threats such as destruction by animals or vehicles. Therefore sensor networks should be self-healing, as well as self-organizing (Chapter 3).

Sensors are used as both data generators and routers. Networked sensor nodes can aggregate data to provide a rich, multi-dimensional view of the environment. *Source* sensors detect the event or gather data. Sources are usually located where the environmental activities of interest are expected to take place. *Sink* nodes are basically monitoring terminals such as mobile PDAs or laptops. They are connected to other networks such as the Internet and provide remote access to data from the sensor network.

Architecture. Other recent advancements in technology have made common hardware platforms, sensors, and radios widely available. Such low-cost, off-the-shelf devices enabled the great development in the field of sensor networks.

Sensor nodes are typically composed of on-board sensors, a processor, a small amount of memory, a wireless modem, and a limited energy source. Overall prototypes of currently available sensor nodes are very similar, but their size and shape come in great varieties, from WINS NG 2.0, whose size is more than 5000 cubic centimeters to Smart Dust which is a device a cubic millimeter in size. Their diversity is natural due to different types of applications. One of the most widely used nodes is MICA Mote [3]. MICA motes were originally introduced by US Berkeley research group. It features an Atmega 128L processor, 4 KB of RAM, a 916 MHz transceiver, TinyOS operation system, and runs on 2 AA batteries. TinyOS is a small micro-threaded OS and it can provide the system

software support to operate and manage such small smart devices [40]. Five requirements for networked sensor systems were given in [41]. They are (a) small physical size and low power consumption, (b) concurrency-intensive operation, (c) limited physical parallelism and controller hierarchy, (d) diversity in design and usage, and (e) robust operation.

One of the most well-known projects in sensor networks is the aforementioned *Smart Dust* project at Berkeley [50]. The researchers aim was the designing of networked sensors with limits on size and power resources, called smart dust with required sensing, communication, and computing hardware, along with a power supply, within the size of a few cubic millimeters.

In [5] the current research on heterogeneous sensor networks at Intel Corporation is presented. In this research, a group at Intel is exploring the deployment of heterogeneous sensor networks in theme parks. These networks could be used for monitoring water quality, providing Internet access to park visitors, or for the overall improvement of park management.

DARPA founded a program called *Network Embedded Systems Technology (NEST).* [4] describes many projects under NEST. A fundamental question for the NEST program is how should deeply embedded, diffuse sensor networks be programmed? The goal of the NEST program is to achieve "fine-grain" fusion of physical and information processes.

Applications. Today, there are many different types of sensors such as seismic, infrared, acoustic, visual, and radar amongst others. Hence there are a wide variety of conditions that can be monitored by sensor nodes that include temperature, humidity, pressure, noise, and vehicular movement. Also, sensor nodes can be used for continuous sensing or event detection. Consequently, application fields of sensor networks are limitless. The followings are a few examples:

- Military applications: Sensor networks can be used to detect biological and chemical attacks and create warning systems. Also they can be used to monitor an ally's condition and status.

- Environmental applications: One interesting example of this area was presented in [47]. Sensors were deployed on Great Duck Island in Maine for habitat monitoring. Forest fire detection and flood detection systems are also good examples in this category.

- Health applications: Doctors can monitor the current condition of patients by using sensors which may detect heart rate or blood pressure.

- Commercial applications: There are numerous applications in this field. Inventory management, intruder detection, and vehicle tracking use sensor networks to attain a so called smart environment.

Many requirements for the above mentioned application areas may be unique and not suitable for traditional ad hoc networks. For instance, in military applications, there is a heightened chance that nodes will be destroyed by an enemy. Because sensor nodes are cheap and disposable, they can be deployed densely to tolerate a node's fault. Therefore, in the future, wireless senor networks will be an integral part of our lives.

Sensor network services. According to [39], several services must be provided by sensor networks in addition to low-level networking. Such services are unique to sensor networks. The following are some examples described in [39].

- *Localization.* In many sensor network protocols, nodes are required to know their own positions. However, not all sensor nodes are equipped with a GPS system. Therefore,

they must have the ability to compute geographic location automatically. Measuring the distance to neighboring nodes can be done by measuring the receiving signal strength (RF-based ranging) or measuring the difference in arrival times of simultaneously transmitted radio (acoustic ranging).

- *Time synchronization.* Often, sensors are required to collaborate to detect an event. For this purpose, it is essential for sensor networks to have the ability of time synchronization. Typically, this can be implemented by using time-stamped messages. Time synchronization can also be used to ensure collision-free communication between sensors.

- *Remote programming.* Usually, sensor networks are application specific and their tasks are pre-programmed before deployment. It is highly desirable if the tasks are re-programmable when environments or targets have changed.

- *Security.* Although the importance of security issues has not been recognized as much as other areas of sensor networks have been, recently this issue has gained extensive attention. As sensor networks became more and more popular and used in many applications, security can become a serious problem, especially in military applications. Designing a mechanism, such as message encryption or authentication, is quite a challenging problem since sensor networks have limited transmission bandwidth and limited computational power.

CHAPTER 3

SELF-∗ SYSTEMS

One of the main topics of research in this thesis is *self-∗ systems.* In the following, we start with a brief description of distributed systems which is a computational model commonly used in the design of self-stabilizing algorithms. Then we will give an overview of *self-∗ systems* in Section 3.1. We will describe many terms currently being used in the broad area of fault-tolerant computing. Also, an overview of the concept of self-stabilization which is currently a very active area of research will be given in Section 3.3.

Distributed Systems. A number of definitions have been proposed in the literature to capture the meaning of *distributed systems.* A *distributed system* is a communication network, multiprocessor computers, and can be a single multitasking computer [28]. Also, the existence of the collection of these nodes must be transparent to the system user. Although the processors in distributed systems are autonomous in nature, they may need to communicate with each other to *coordinate* their actions and achieve a reasonable level of *cooperation* [49]. A program composed of executable statements are run by each computer. Each execution of a statement changes the computer's local memory content, hence the computer's state. Consequently, a distributed system is modeled as a set of n state machines that communicate with each other. There are mainly two models for communications between machines; message passing and shared memory. In the message passing model, machines communicate with each other by sending and receiving messages. While in the

shared memory model, communication is carried out by writing in and reading from the shared memory. This model will be described in detail in Section 4.3.

3.1 Overview

Software systems are used everywhere. Thusly commercially available software systems must be able to adjust to different inputs and handle different faults so that they can be used in many different environments. The different concepts or terms encapsulated in *self-∗* have been introduced to characterize different ways of detecting, adjusting, and recovering from such changes. Because these terms have not been formally defined, we will informally describe them with examples from other sources of literature.

A self-∗ system should be self-configuring, self-organizing, self-contained, self-healing, and self-managing [34]. According to [54], research in self-∗ systems is "a direct response to the shift from needing bigger, faster, stronger computer systems to the need for less human-intensive management of the systems currently available. System complexity has reached the point where administration generally costs more than hardware and software infrastructure." The goals of the self-∗ systems are reduction of human administration and maintenance, and an increase of reliability, availability, and performance.

A system is considered to be *self-configuring* if starting from an arbitrary state and an arbitrary input, the system will eventually satisfy the specification of an application or start behaving properly in finite steps. Therefore, a self-configuring system is the system which can configure and reconfigure itself under varying conditions or faults. A similar concept of self-organizing was defined in [8]. In this paper, this concept was applied to study peer-to-peer systems based on the locality principle. Example applications can be seen in the field of robotics [25]. The problem considered in these papers is for a system of multiple mobile

robots to be able to communicate with each other and form a certain geometric pattern. Since each robot can start from any arbitrary position, but eventually converges to a final shape, proposed solutions are considered to be self-configuring.

A *self-contained* system is a system in which only local neighbors are affected by any faults or topology change. Thus, if a fault occurs, nodes which are located more than several hops away should not be aware of it.

A *self-healing* system automatically recovers from different perturbations and dynamic changes. In [9], a self-healing network (SHN) for supporting scalable and fault-tolerant runtime environments was presented. It was designed to support message transmission via multiple nodes while protecting against failures. Finally, within a *self-maintaining* system, all tasks in all phases in the life cycle of the system are automatic so that it can reduce the system administrator's tasks. As the number of computer devices continue to increase exponentially, planned maintenance of computers are becoming more and more of an impossible task to manage. As well, the cost of employing network administrators to keep these computers up and running has been rising. In [13], the authors defined this concept from the system administrators perspective as a system which maintenance will only be required at fixed intervals and the required tasks will be clearly defined at maintenance time. *Autonomic computing* is IBM's solution to the above management problem [1]. On October 15th, 2001, Paul Horn, Senior Vice President of IBM Research suggested a solution: "Build computer systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies."

Another approach which was introduced in [31, 48] was *recovery-oriented computing*, with such systems being called *self-repairing computers*. This concept can be applied to designing highly dependable Internet services.

3.2 Ubiquitous/Pervasive Computing

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" [59]. These are the words of the late Mark Weiser who was the chief technologist at Xerox PARC and considered the father of *ubiquitous computing.* He described this new era as that of most computers vanishing into the background and being "nearly invisible" from users, but would always be available, which was called *invisible computing*, and one of the key concepts of his vision. This invisible tool is one that does not intrude on our consciousness so that we can focus on the task. An example of this concept is eyeglasses. We look at the world, not the eyeglasses. Computers should be the same. They would be available and prevalent throughout the physical environment without users actually having awareness of them. Another key concept was presented in [60], known as *calm technology.* The goal of "calm" technology is to send information in a calm manner. Technology such as cellphones and TVs are often the antithesis of this concept. However, calm technology allows the user to choose what information is needed and what information is peripheral (or sensory) to reduce information overload, while still allowing the user to move easily from the center of information to periphery and back. This can be performed by giving more detail to the periphery. In [60], an example of this calm technology is shown by comparing a video conference and a phone conference. The video conference can give participants visual knowledge of details such as facial expression or body posture, so that participants are more confident about what information is important, hence a more "calm" environment than that of a phone conference.

Ubiquitous computing is about making our lives simpler through digital environments that are sensitive, adaptive, and responsive to human needs. It is now a framework for

new and exciting research in the field of computer science, which includes mobile devices, sensors, and many smart appliances.

3.3 Self-stabilizing Systems

In 1973, Dijkstra introduced the term *self-stabilization* in the world of computer science [27, 26], which was a concept of fault-tolerance. Unfortunately, only a handful of people had become aware of its importance until Lamport endorsed this as "Dijkstra's most brilliant work" and "a milestone in work on fault-tolerance" in his invited talk at the ACM Symposium on Principles of Distributed Computing in 1983. Today, it is one of the most active areas of research in the field of computer science.

A system is considered self-stabilizing if starting from any arbitrary state (possibly a fault state) it is guaranteed to converge to a legitimate state which satisfies its problem specification in a finite number of steps. Once it converges to a legitimate state, it must stay in that legitimate state thereafter unless a fault occurs. With respect to behavior, it can also be defined as a system starting from an arbitrary state, reaching a state in finite time from which it starts behaving correctly according to its specification. This self-stabilization enables systems to recover from a transient fault automatically.

According to [10, 11], self-stabilization can be defined in terms of two properties; *closure* and *convergence*. Closure means that if a system is in a correct (or legitimate) state, it is guaranteed to stay in a correct state, if no fault occurs. On the other hand, convergence means that starting from any arbitrary state, it is guaranteed that the system will eventually reach a correct state in finite steps. In order for a system to be self-stabilizing, it must satisfy both of these properties.

In the area of network protocols, self-stabilization has been extensively studied. Pro-

tocols like routing, high-speed networks, sensor networks, and connection management are just a part of many applications of self-stabilization. Also, there exist many self-stabilizing distributed solutions for graph theory problems, for example, spanning tree constructions, maximal matching, search structures, and graph coloring. Many self-stabilizing solutions for numerous classical distributed algorithms were proposed as well. Those include mutual exclusion, token circulation, leader election, distributed reset, termination detection, and propagation of information with feedback [28].

In the study of self-stabilization, several aspects of models have been considered, such as the following:

- Interprocess Communication: shared registers or message passing.
- Fairness: weakly fair, strongly fair, or unfair.
- Atomicity: composite or read/write atomicity.
- Types of Daemon: central or distributed.

All together proving stabilization programs are quite challenging. Two techniques have been commonly used in research literature: convergence stair [37] and variant function [43] methods. Furthermore, many general methods of designing self-stabilizing programs have been proposed which include diffusing computation [12], silent stabilization [29], local stabilizer [7],local checking and local correction [14, 56], counter flushing [57], self-containment [36], snap-stabilization [20], super-stabilization [30], and transient fault detector [15].

Self-stabilization is a significant concept in the study of sensor networks. Due to the dynamic nature of sensor network topology (Section 2.2), the protocols for setting up and organizing sensor networks are often required to be self-stabilizing.

CHAPTER 4

MINIMUM CONNECTED SENSOR COVER PROBLEM

Throughout our research, we extensively studied two main fields; wireless sensor networks and self-∗ system, which were discussed in earlier chapters. Using the knowledge we acquired by studying those topics, we designed robust, distributed, and self-∗ protocols to solve the minimum connected sensor cover problem that is a significant issue in sensor networks. The outline of this chapter is as follows: The motivation of this research is first stated. Then, we discuss how other issues and topics described in earlier chapters are related to the minimum connected sensor cover problem. In section 4.2, we describe some various results in related areas. The model and the program including its notation used in our algorithms are described in section 4.3. Also, a formal definition of self-stabilization is given. Finally, we state both an informal explanation and formal specification of the problem to be solved in this section.

We report the main results of our research in the next four chapters. In Chapter 5, an algorithm to solve the minimum connected sensor cover (Algorithm $\mathcal{SHID}$) is presented. We discuss the modified versions of this algorithm in Chapters 6 and 7. A detailed informal description, formal algorithms, and proof of the algorithms are included in each chapter. We also discuss simulation results of all algorithms in Chapter 8.

4.1 Motivation

Wireless sensor networks are usually composed of a large number of tiny sensor nodes with finite energy sources. In many applications, sensors are deployed in an area where once deployed it is infeasible to replace or recharge a sensor's battery or energy supply. Therefore sensor networks are usually expected to have a very short life. Also, in a typical sensor network, the topology of the network is dynamic due to nodes' movement, failure, energy consumption, or other varying reasons. Hence, designing a robust, energy efficient sensor network which will allow uninterrupted operation and can adopt rapid topology change is critical. That is, sensor networks should be designed as self-$*$ systems.

In sensor networks, queries are sent to nodes from a few external devices to gather data. The information to be gathered by a sensor network may concern only a particular sub-set of the monitored area, called a query region. Since the sensors are usually densely deployed, considering the energy constrained nature of sensor networks, all sensors inside the query region should not be actively participating in sensing the data. To minimize the network energy consumption and prolong the network life time, some sensors should be placed in a passive mode. However, the active sensors must be able to cover the whole query region and maintain network connectivity. Thus, the minimum connected sensor cover problem had arisen.

4.2 Related Work

The problem of computing a minimum connected cover of a query region was first introduced in [38]. Two self-organizing solutions were also presented in [38]. However, both solutions follow a greedy strategy and none of the solutions are localized. The first solution is centralized — a fixed leader chooses the nodes to be part of the cover. In the

second solution, a particular sensor node (which is not always the same node) behaves as the coordinator or leader. This special node collects global information in order to select the nodes to be included in the final cover set.

The issues of coverage and connectivity, and the relationship between them, were analyzed in [58]. Since different applications or the degree of fault-tolerance may require different degrees of coverage, a Coverage Configuration Protocol (CCP) was presented in [58]. The CCP protocol was designed to maximize the number of sleeping nodes, while maintaining two conditions: (1) Every point in the query region is covered by at least K sensors (K-Coverage), and (2) all nodes are connected via K disjoint paths (K-connectivity).

In [58, 67], it is stated that if the communication range is at least twice the sensing range, then complete coverage implies connectivity. A more general form of this theorem was proven in [58]. That is, K-coverage implies K-connectivity if the communication range is at lease twice the sensing range. When the above condition does not hold, CCP cannot guarantee network connectivity. So, CCP was integrated with SPAN [19] to provide both coverage and connectivity.

SPAN is a connectivity maintenance protocol in which a node volunteers to be a coordinator when it finds that two of its neighbors cannot communicate with each other directly or indirectly.To reduce the number of redundant coordinators, after a certain delay, only a single node announces its decision to be a coordinator. This protocol, however, cannot configure a network to a specific degree of connectivity. It can only preserve the network's original connectivity.

A similar approach was discussed in ASCENT [18]. In this paper, the goal was to maintain a certain data delivery ratio. ASCENT nodes locally measure their connectivity using a number of active neighbors and message losses to decide if they should be active or

passive. This protocol can automatically configure the network connectivity. However, it does not guarantee complete coverage of the query region.

A number of optimal conditions for coverage were established in [67] which show that minimizing the number of working nodes is equivalent to minimizing the overlap of sensing areas of all the nodes. The authors defined the optimal positions of the nodes that have minimal overlap of sensing areas. In this optimal position, any three nodes should form an equilateral triangle with side length 3 $\sqrt{r}$, where r is the sensing radius. Optimal Geographical Density Control (OGDC) algorithm for coverage was proposed based on those optimal conditions. However, the result of this is valid only when complete coverage implies connectivity (as discussed above).

A scheduling protocol for coverage was proposed in [55]. The authors here assume a circular sensing area and allow a node to turn off only if its sensing area is completely covered by its neighbors. Then, nodes use a random delay to announce their decision to turn off. Unfortunately, the issue of connectivity was not addressed in [55].

The GAF protocol [63] uses GPS to reduce redundant nodes when maintaining routing paths in ad-hoc networks. A randomized probing-based density control algorithm was used to maintain coverage despite node failures in the PEAS protocol [65]. In this algorithm, a node can enter into a working state when there is no other working node within a certain distance c. As nodes do not have location information, this can be checked by broadcasting a message with a communication range c and receiving a reply. A communication range (also called the probing range) can be changed to provide different degrees of coverage. Although these solutions are efficient in fault free environments, they are neither fault-tolerant nor self-stabilizing. Since this algorithm must be re-executed in order to repair an overlay, in this scheme, every member of the network must be notified of any corruption and of the

need to re-execute the algorithm.

Recent solutions to the connected cover problem address fault-tolerance issues by reinforcing the degree of coverage and connectivity. In [68], the problem of k-coverage was addressed; the protocol ensures that any sensor is covered by k other sensors. This work is further extended in [69] to the k-coverage and k-connectivity problem. The proposed solution involves the computation of a Voronoi diagram for independent sensor nodes. However, the implementation of local Voronoi diagrams is not addressed, nor are transient faults.

In [23, 24], decentralized, self-stabilizing, and fault-tolerant algorithms for the minimum connected covering of a query region in sensor networks were proposed. The first solution in [24] uses a greedy strategy. It requires the knowledge of the distance to the center of the query region. That is, the region is covered in successive waves from outside to inside. The coverage stops once the wave reaches the center of the monitored area. The second solution proposed in the same paper uses a pruning strategy. Redundant nodes are removed from the final cover if their removal does not disconnect their respective neighborhoods, and if their sensing regions are completely covered by their chosen neighbors. In [23] as well, another pruning-based algorithm was proposed. Nodes are considered redundant if their sensing regions are covered by other chosen nodes, and if their chosen neighbors are connected through a connection path. This solution assumes that each node's sensing radius and communication radius are equal, and all sensors have equal sensing (therefore equal communication) radii. Algorithms proposed in this thesis are designed based on this algorithm, but with a different assumption.

The pruning-based algorithm used in [23] and our algorithms are similar to pruning used in the computation of connected dominating sets [16, 21, 42, 44, 46, 61, 62]. 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 is a dominating set which is also a connected sub-graph. A node is a dominator of another node if the second node is in the transmission range of the first node. Two pruning dominating set rules were proposed in [62] and extended in [61]. Rule 1 unmarks a host u if all of its neighbors are covered by another marked host, and if its UID is less than another marked host's UID. Rule 2 unmarks a host u if its neighbors are covered by two other directly connected marked hosts, and if its UID is less than both of these hosts. However, these rules do not ensure if a host u itself is covered before unmarked. In [21], Rule k was proposed. In this rule, a host is unmarked if it is covered by k other hosts and if its UID is the least of all marked neighbors' UID's. A localized algorithm for connected dominating set proposed in [46] improved the result of [21]. However, it requires additional message exchanges since each node decides whether it should be dominant by using the information received from its neighbors. Furthermore the synchronization among nodes is needed, hence it is difficult to implement.

4.3 Preliminaries

4.3.1 Model

Sensor Network. Sensor Networks usually consist of a large number of sensor nodes which are also referred to as simply *sensors* or *node*. In this paper, we use the terms *node* and *sensor* interchangeably. Since deploying large numbers of sensors in certain positions is usually infeasible, most of the time they are randomly deployed in a geographic region. In research fields, the sensor network is typically modeled as a *directed* communication graph $G(V, E)$, where V is the set of vertices (or sensors) and E is the set of directed edges (or communication links) between sensors. Thus there is a bi-directional link between sensor i

and sensor j if and only if $(i, j) \in E$ and $(j, i) \in E$.

The *communication region*, also called the *transmission region*, of sensor i is 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* (or *communication radius*) 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 and sensor j is called a neighbor of sensor i. An edge $(i, j) \in E$ in the graph G indicates that j is a *neighbor* of i. 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. That is, there is a bi-directional link between sensor i and sensor j and there exist two edges $(i, j) \in E$ and $(j, i) \in E$ in the graph G.

A *sensing region* of sensor i is the area in which sensor i can detect a given physical phenomenon at a desired confidence level. Although the sensing regions can be any convex shape, we chose a circular sensing region as the basis for our algorithms. A *sensing range* or *sensing radius* of sensor i indicates the maximum distance between sensor i and any point p in the sensing region of sensor i. A point p in a field is said to be *covered* (or monitored) by a sensor i if the Euclidean distance between p and i is less than the sensing range of sensor i.

A *communication path* from i to j is a direct path of sensors. Where i_x is a neighbor of i_{x+1} for $1 \leq x \leq m - 1$, the sequence of sensors can be expressed as $i = i_1, i_2, \ldots, i_m = j$. The *communication distance* from sensor i to sensor j is the number of sensors, or length, of the shortest communication path from i to j.

Program. As a communication model, we assumed that the program of every processor consists of a set of *shared variables* as well as *local variables* and a finite set of actions. This model is called the *local shared memory model* of communication used by Dijkstra [26]. Processors (or nodes) communicate with each other by the use of shared variables. Each processor can read and write its own shared variables. However, each processor can only read the shared variables owned by a neighboring processor. Local variables are local to each processor and can be read and updated only by the owner of the local variables.

In the program of p, the guard (or predicate) of an action is a boolean expression involving the variables of p and its neighbors. One or more variables of p is updated by a statement of an action of p. Each action can be expressed with the following structure: $< label >:: < guard > \longrightarrow < statement >$. Only if its guard evaluates to true, can an action be executed. By assuming that the actions are atomically executed, we used a model known as *composite atomicity* [28]. In other words, when the evaluation of a guard and the corresponding statement of an action are executed, they are performed in one atomic step. In distributed systems, multiple processors are able to execute an action concurrently as long as there is no influence of one action upon another. As an example, within a shared memory model, a processor can not write to variable v while another processor is reading from it.

The values of a node's variables define the *state* of that node and the states of all nodes determine the *state* of a system. In this thesis, the state of a node is referred to as (*local*) *state* and the state of a system is referred to as (*global*) *configuration.*

Assume distributed protocol $\mathcal{P}$ is a collection of binary transition relations denoted by $\mapsto$, on $\mathcal{C}$, where $\mathcal{C}$ is the set of all possible configurations of the system. Then, a *computation* of a protocol $\mathcal{P}$ is defined as a *maximal* sequence of configurations $e = \gamma_0, \gamma_1, ..., \gamma_i, \gamma_{i+1}, ...,$

where $i \geq 0, \gamma_i \mapsto \gamma_{i+1}$ in a single *computation step*, if γ_{i+1} exists, or γ_i is a terminal configuration.

If the sequence is either infinite, or it is finite and no action of $\mathcal{P}$ is enabled in the final configuration, then the sequence is considered as *maximal*. We assumed that all computations considered in this paper are maximal.

The notation $\mathcal{E}$ is used to define the set of all possible computations of a protocol $\mathcal{P}$ in system S.

If a node u has an action A such that the guard of A is true in γ, then u is said to be *enabled* or have a *privilege* in $\gamma \in \mathcal{C}$. Similarly, an action A is said to be enabled at u if the guard of A is true at u in $\gamma \in \mathcal{C}$.

If a node u has been enabled in γ_i and not enabled in γ_{i+1} without executing any action between these two configurations, then it is said that u has executed a *disable action* in $\gamma_i \mapsto \gamma_{i+1}$. This is due to at least one neighbor of a node u that has changed its state between γ_i and γ_{i+1}, and this change effectively has made the guard of all actions of u false.

We assumed the *asynchronous* model for the timing model, in which processors execute their programs at different speeds. In this model, a *scheduler*, also know as *daemon*, determines which processors execute the next step. In this paper, we considered a *distributed daemon*. In each computational step, if there exists at least one node that is enabled, the distributed daemon selects a non-empty subset of enabled nodes to execute an action. We also assumed a *weakly fair* daemon, which only ensures that in an infinite execution, each processor takes an infinite number of steps. It also means that if a node p is continuously enabled, then p will be eventually chosen by the daemon to execute an action.

4.3.2 Self-stabilizing Program

Fault Model. This research pertains to various types of faults as follows:

- The program (or code) of the algorithm cannot be corrupted, but the state or configuration of the system may be arbitrarily corrupted.
- The faults caused by node crash or malfunction can fail-stop nodes.
- The nodes may recover from the fault or join the network at any time.
- The network topology may change due to faults.
- Arbitrary faults may occur in any finite number, in any order, at any frequency, and at any time.

Self-stabilization [28]. 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* if and only if the following two conditions hold:

Closure: Every computation of $\mathcal{A}$ starting from a configuration satisfying $\mathcal{L}_{\mathcal{A}}$ preserves $\mathcal{L}_{\mathcal{A}}$.

Convergence: Every computation of $\mathcal{A}$ starting from an arbitrary configuration contains a configuration that satisfies $\mathcal{L}_{\mathcal{A}}$.

Informally, closure property means that once the system is in a legitimate configuration, then it stays in a legitimate configuration until a fault occurs. Convergence property guarantees that from any arbitrary configuration the system will converge to the legitimate configuration in a finite number of steps.

4.3.3 Problem Specification

In this section, we formally define the problem of Connected Cover of a Query Region in sensor networks.

Definition 4.1 (Connected Sensor Cover) *Consider a sensor network G with a set of n sensors $S = (I_1, I_2, \ldots, I_n)$, where each sensor I_i is assigned a sensing radius S_i. Given a query region R_Q in the sensor network, a set of sensors $SC_Q = I_{i_1}, I_{i_2}, \ldots, I_{i_m}$ is called a* connected sensor cover *for the query region R_Q if the following two conditions hold:* **(a) Coverage:** $R_Q \subseteq (S_{i_1} \cup S_{i_2} \cup \ldots S_{i_m})$. **(b) Connectivity:** *The communication graph induced by SC_Q is strongly connected such that any two sensors in this set can communicate with each other directly or indirectly.*

A set of sensors that satisfies only condition (a) above is called a *sensor cover* for R_Q in the sensor network.

Definition 4.2 (Minimum Connected Sensor Coverage Problem) *Given a query region over a sensor network, the minimum connected sensor coverage problem is to find the set of the smallest number of sensors which satisfies the two conditions of the connected sensor cover.*

Additionally, we require the algorithm (for solving the above problem) to be self-organizing, self-stabilizing, and self-healing [28, 66]. That is, regardless of the initial state (wrong initialization of the local variables, memory or program counter corruptions) nodes self-configure (self-organize) using only local information in order to make the system self-stabilize to a *legitimate state*. The legitimate state is defined with respect to a minimal connected cover formed out of the nodes that can communicate with each other either directly or indirectly. Upon stabilization, each sensor in the query region will know if it should act as an active or a passive node for the application. If a sensor is in the final minimal connected sensor cover set, it will stay active and participate in sensing/gathering information in response to a query. If a sensor becomes passive, then it will not participate

in the sensing and communication role in the application, but instead, enter into the power saving mode. However, in the power saving mode, it will still do some local checking to detect faults or network topology changes. Under such perturbations, the minimal connected cover should be able to self-heal without any external intervention and the impact should be confined within a tightly bound region around the disturbed area. In our proposed algorithms, a *chosen* sensor is an active sensor and an *unchosen* sensor is a passive sensor.

CHAPTER 5

SINGLE-HOP UID-BASED ALGORITHMS

5.1 Description and Data Structures

In this section, we will present the single-hop, UID based self stabilizing solution to the minimum connected cover problem. In this paper, we refer to this algorithm as $\mathcal{SHID}$.

First, we will state our assumptions and then we will explain the data structures used in the algorithm.

Throughout our research, we consider highly dense sensor networks. That is, there should always be enough sensors to cover the query region at any time, even if some sensors fail. This is stated in the following assumptions we made.

Assumption 5.1

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

(ii) There exist numerous redundant sensors which are either boundary or interior sensors with respect to the query region.

Another important assumption that we must make is regarding each sensor's sensing and communication radius. One of the main goals of our research is to design an algorithm which can cope with variable sensing and communication radii of sensors. As mentioned in the related work, Section 4.2, most of the previous research has been done by assuming

fixed sensing and communication radii and all sensors have equal sensing and communication radii. In [58, 67], the authors assume that the sensing radius is twice the size of the communication radius. In [23], it is assumed that the sensing radius equals the communication radius for the sensors. However, due to the nature of sensor networks, some sensors may consume more energy than others. So, as their energy levels decrease, their sensing and communication radii may decrease at varying rates. Also, in a self-stabilizing system, which is described in section 3.3, the initial configuration can be arbitrary. From the reasons above, it is impractical to assume that all sensors have fixed sensing and communication radii. Therefore we made the following assumption:

Assumption 5.2

Each sensor has variable sensing and communication radii.

This assumption means that the communication radius may not be equal to the sensing radius of the sensors, and the communication radii and the sensing radii of all sensors may not be equal, either. As well, they can be changed over time due to various reasons.

Data Structures. The data structure $Info$ has fields UID, *Status*, *Position*, *Rc*, *Rs*, S, and $MinUID$. UID represents the unique identifier (UID) of a sensor, which is a positive integer. *Status* represents the status of a sensor. The status of a sensor may be *unchosen*, *undecided*, or *chosen*. A node with the status *chosen* is part of the connected cover. *Position* represents a geometric location or coordinate of a sensor. *Rc* and *Rs* represent a communication radius and a sensing radius of a sensor, respectively, and S represents a sensing region of a sensor. Finally, $MinUID$ represents the minimum UID amongst all of a sensor's neighbors' UIDs. All sensors that have the minimum UID within a particular

chosen sensor's neighborhood are needed to ensure connectivity.

Sensor i has three shared variables: $Self\_Info_i$, N_i and N_Info_i. $Self\_Info_i$ is a data type of $Info$ which contains Sensor i's own information. N_i is a set of sensors within the communication range of Sensor i. Since we assume that each sensor has a different communication radius, we include only those sensors which have bi-directional communication with Sensor i to be within the neighbor set N_i. From this point onward, communication neighbors are known by the term *neighbors* and refers to only those sensors which have bi-directional communication. We use the term *sensing neighbors* to represent the sensors that are located within each others' sensing disks. *Sensing neighbors* may or may not be communication neighbors. N_Info_i is a set of δ $Info$ structures containing $Self\_Info_j$ of all sensors j in N_i.

The local variables of Sensor i are $2N\_Info_i$, which is a set of δ^2 $Info$ structures containing all 2-hop neighbors j's $Self\_Info_i$, and NN_i, which is a set of N_j for Sensor i's all neighbors j. That is, NN_i is a set of sensors located, at most, two hops away from Sensor i. Although this algorithm uses 2-hop information to compute the redundant cover, messages are exchanged only within a single hop.

Macro. We introduce the macro $Read(j)$ to gather sensor i's neighbors' information using small atomicity. That is, Sensor i reads only one of its neighbor's shared variables in one atomic step, instead of reading all neighbors' shared variables.

When there is a timeout, a *timeout* action is enabled and the macro $Read(j)$ reads one of Sensor i's neighbors j's variables. Sensor i reads Sensor j's $Self\_Info_j$, and includes it in its N_Info_i. If there is a duplicate, (i.e., Sensor i reads Sensor j for a second time), then $Self\_Info_j$ overwrites the old data. Finally, Sensor i needs to gather information from sensors located 2-hops away by reading its neighbor j's neighbor information N_Info_j.

Thus, NN_i and $2N\_Info_i$, will be updated. We assume that there is a function $F(j, NN_i)$, $j \in Ni$, that returns N_j. Then the $Next(N_i)$ function updates the pointer to the next neighbor.

5.2 Predicates

The predicate $QryRgnIntrsctn(i)$ checks if the sensing disk of Sensor i intersects with any portion of the query region. If it does, then this predicate is evaluated as true. The predicate $SnsngNgbr(i, j)$ returns true if Sensor i and Sensor j are located within each others' sensing disks. This predicate is needed to ensure the coverage condition since each sensor has variable sensing and communication radii, so the communication neighbors may not cover each others sensing disks. The predicate $CvrSnsngByChsn(i)$ is true when the sensing disk of Sensor i is covered by a subset of *chosen* sensors that are located within two communication hops from Sensor i. Similarly, $NeighborsConnectivity(i)$ is true when all pairs of Sensor i's *chosen* neighbors are connected by *chosen* sensors located within two communication from Sensor i. The predicate $LstUIDNgbr(i, j)$ evaluates to true if Sensor i has the least UID amongst the neighbors of Sensor j. $GrtrLstOrNotNgbrOfChsn(i)$ is true if Sensor i doesn't have any *chosen* communication neighbor nor any *chosen* sensing neighbor, otherwise Sensor i has the greater UID than its *chosen* neighbors, or Sensor i has the least UID amongst the neighbors of Sensor j.

The predicate $SensorCover(i)$ evaluates to true if Sensor i's status is *unchosen*, the sensing disk of Sensor i intersects with any portion of query region, and the predicate $GrtrLstOrNotNgbrOfChsn(i)$ is true. $MCSCNode(i)$ checks if the predicate $GrtrLstOrNotNgbrOfC$ is true or a part of the sensing disk of Sensor i is uncovered. In this case, if Sensor i's status is *undecided*, then this predicate evaluates to true. When Sensor i's status is either

undecided or *chosen*, and $GrtrLstOrNotNgbrOfChsn(i)$ is false, $CvrSnsngByChsn(i)$ is true, and $NeighborsConnectivity(i)$ is true, then the predicate $Redundant(i)$ evaluates to true.

5.3 Normal Execution

The steps of the algorithm are as follows:

1. The algorithm marks an *unchosen* sensor whose sensing region intersects with any portion of the query region (RQ) as *undecided*, if one of the following is true: 1) The sensor does not have a *chosen* neighbor which is also a sensing neighbor. 2) Its UID is greater than a UID of a *chosen* neighbor which is also a sensing neighbor. 3) Among the neighbors of a *chosen* sensor, it has the minimum UID.

2. $MCSCNode(i)$ checks if Sensor i's status is *undecided*, and if one of the following is true: 1) The sensor does not have any *chosen* neighbors, which are also sensing neighbor. 2) Its UID is greater than a *chosen* neighbor's UID. 3) It is the minimum UID neighbor of a *chosen* sensor. 4) A part of the sensing disk of Sensor i is not covered by a *chosen* sensor. In this case, the sensing disk of Sensor i is needed in the final cover set, so Sensor i changes its status to *chosen*.

3. $Redundant(i)$ removes any *undecided* or *chosen* sensor that has a smaller UID than a *chosen* neighbor's UID and is not a minimum UID neighbor of a *chosen* sensor, if its entire sensing disk is covered by *chosen* sensors and all *chosen* neighbors of this sensor are connected through a second path. In this case, the status of such a sensor is changed to *unchosen* (rule $\mathcal{A}_1$).

4. Rule $\mathcal{A}_1$ also ensures that 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 query region connected cover.

5.4 Fault and Recovery

In this section, we explain how the proposed algorithms $\mathcal{SHID}$handles the fault. There are seven variables in the $Info$ structure used in the solutions for a Sensor i: UID, *Status*, *Position*, R_C, R_S, S, and $MinUID$. Since the algorithm assumes variable R_C and R_S (as well as S) in Assumption 5.2, we do not consider the weakening or changing of a sensor's sensing and communication range as a fault. For the purpose ofthis algorithm, UID cannot be corrupted. However all other variables can be corrupted. So, we need to show that our solutions can deal with all possible corruptions associated with these variables. In the following, we will show how they are handled in the Algorithm $\mathcal{SHID}$.

1. Wrong initialization of the *Status* variable.

   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) 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*. Thus, no correction is necessary. If Sensor i is redundant, then it will satisfy the predicate *Redundant*(i), and will change to *unchosen*. If it is non-redundant,

then Sensor i is necessary, either to ensure coverage or connectivity, and should not be unmarked.

2. Wrong initialization of the *Position* variable.

   The variable *Position* is computed by some other protocols or possibly by an equipped GPS. If this variable is not initialized correctly, Algorithm $\mathcal{SHID}$may produce the wrong result. However, we assume the existence of the self-stabilizing protocols to compute this value and it will eventually stabilize and produce the correct *Position* value. After that, the algorithm will also stabilize and compute the correct result in finite steps.

3. Wrong initialization of the $MinUID$ variable.

   Each sensor reads one of the neighbors and updates $MinUID$ variable upon every timeout. So, when a sensor finishes reading all neighbors for the first time, $MinUID$ will contain the correct value. After that, the algorithm will stabilize and compute the correct result in finite steps.

5.6 Correctness

In this section, we will show the correctness of Algorithm $\mathcal{SHID}$. We will prove that the algorithm produces the solution which satisfies the specification of the connected sensor cover problem. First, we will give a definition of a legitimacy predicate with respect to the specification of the $\mathcal{MCSC}$ problem. Then, in the following sections, we will prove that the final set produced by the algorithm when the system is in a legitimate state satisfies the coverage and connectivity properties as defined in Section 4.3.3. Also, we will prove that the system reaches a legitimate state in finite steps, regardless of the initial configuration or

Algorithm 5.5.1 Query Region Connected Sensor Cover Algorithm for Sensor i ($\mathcal{SHID}$).

Constants:
R_Q:: Query region;

Structure:
$Info\{$
UID :: Unique user identification number
$Status \in \{unchosen, undecided, chosen\}$ :: Status of a sensor
$Position$ :: Geometric location or coordinate of a sensor
R_C :: Communication radius of a sensor
R_S :: Sensing radius of a sensor
S :: Sensing region of a sensor
$MinUID$ :: minimum UID amongst all of a sensor's neighbors' UIDs
$\}$

Shared Variables:
$Info\ Self\_Info_i$ :: One structure that contains information for $Sensor_i$
$Set\ N\_Info_i$ :: Set of δ structures that contain all neighbors' information
$Set\ N_i :: \{j \in V | Dist(i,j) \leq R_{C_i} \wedge Dist(i,j) \leq R_{C_j}\}$

Local Variables:
$Set\ 2N\_Info_i$ :: Set of $\delta_i + \sum \delta_{j \in N_i}$ structures that contain all 2-hop neighbors' information
$Set\ NN_i$:: Set of N_j, $\forall j \in N_i$

Macro:
$Read(j)$
$N\_Info_i = N\_Info_i \bigcup Self\_Info_j$
$NN_i = NN_i \bigcup N_j$
$2N\_Info_i = 2N\_Info_i \bigcup N\_Info_j$
$j = Next(N_i)$

Predicates:
$QryRgnIntrsctn(i) \equiv Self\_Info_i.S \cap R_Q \neq \emptyset$;
$\equiv$ sensing disk of Sensor i intersects with some portion of query region;

$Dist(i,j) \equiv$ Returns the Euclidean distance between Sensor i and Sensor j;

$SnsngNgbr(i,j) \equiv (\forall j$: Dist(i,j) $\leq min(Self\_Info_i.R_S, N\_Info_i.Self\_Info_j.R_S))$;
$\equiv$ Sensor i and Sensor j are located within each others' sensing disks;

$CvrSnsngByChsn(i) \equiv (\exists A : \forall j,k \in A, j \in N_i \wedge k \in N_j$
$\wedge N\_Info_i.Self\_Info_j.Status = 2N\_Info_i.N\_Info_j.Self\_Info_k.Status = chosen$
$\wedge\ Self\_Info_i.S \subset \bigcup_{j,k \in A} N\_Info_i.Self\_Info_j.S, 2N\_Info_i.N\_Info_j.Self\_Info_k.S)$;
$\equiv$ Sensing region of Sensor i is covered by a subset of *chosen* sensors that are located no farther than two communication hops from Sensor i;

$NeighborsConnectivity(i) \equiv (\forall j,t \in N_i, N\_Info_i.Self\_Info_j.Status = N\_Info_i.Self\_Info_t.Status =$
$chosen, \exists k \neq i, 2N\_Info_i.N\_Info_j.Self\_Info_k.Status = chosen \wedge j,t \in N_k)$;
$\equiv$ All *chosen* pairs of neighbors of Sensor i are connected by a *chosen* node;

$LstUIDNgbr(i,j) \equiv i \in N_j \wedge (Self\_Info_i.UID = N\_Info_i.Self\_Info_j.MinUID)$;
$\equiv$ Sensor i is a neighbor of Sensor j, and is also the neighbor of Sensor j having the least UID;

$GrtrLstOrNotNgbrOfChsn(i) \equiv (\forall j : i \in N_j, N\_Info_i.Self\_Info_j.Status \neq chosen \vee$
$\neg SnsngNgbr(i,j) \vee Self\_Info_i.UID > N\_Info_i.Self\_Info_j.UID \vee$
$LstUIDNgbr(i,j))$;
$\equiv$ Sensor i is not the communication neighbor, nor the sensing neighbor of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor;

$SensorCover(i) \equiv Self\_Info_i.Status = unchosen \wedge QryRgnIntrsctn(i) \wedge$
$GrtrLstOrNotNgbrOfChsn(i)$;
$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the communication neighbor, nor the sensing neighbor of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor;

$MCSCNode(i) \equiv Self\_Info_i.Status = undecided \wedge (GrtrLstOrNotNgbrOfChsn(i) \vee$
$\neg CvrSnsngByChsn(i))$;
$\equiv$ Sensor i is an *undecided* sensor and is not the communication neighbor, nor the sensing neighbor of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor, or a part of the sensing disk of Sensor i is not covered by a *chosen* sensor;

$Redundant(i) \equiv (Self\_Info_i.Status = undecided \vee Self\_Info_i.Status = chosen) \wedge$
$\neg GrtrLstOrNotNgbrOfChsn(i) \wedge CvrSnsngByChsn(i) \wedge NeighborsConnectivity(i)$;
$\equiv$ Sensor i is an *undecided* or a *chosen* sensor and is the "lesser" communication and sensing neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and the entire sensing disk of Sensor i is covered by *chosen* sensors, and *chosen* neighbors of Sensor i are connected through a second path;

Actions:

$\mathcal{A}_1 :: \neg QryRgnIntrsctn(i) \vee Redundant(i)$
$\longrightarrow Self\_Info_i.Status = unchosen$;

$\mathcal{A}_2 :: SensorCover(i)$
$\longrightarrow Self\_Info_i.Status = undecided$;

$\mathcal{A}_3 :: MCSCNode(i)$
$\longrightarrow Self\_Info_i.Status = chosen$;

$\mathcal{A}_4 :: Timeout \wedge j \in N_i$
$\longrightarrow Read(j)$;

type of faults occurring in the system. Finally, these results will be used to prove Algorithm $\mathcal{SHID}$satisfies self-* properties.

Definition 5.1 *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 5.1 (Coverage) *In any legitimate configuration, the chosen set computed by Algorithm 5.5.1 completely covers the query region R_Q.*

Proof. We prove this lemma by contradiction. Suppose the query region is not completely covered by the sensing disks of the sensors in the final set chosen by Algorithm 5.5.1.

By the action $\mathcal{A}_2$, a sensor will change to *undecided* if it is *unchosen*, if its sensing disk intersects with some portion of the query region, and if it is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose UID is greater than its own, unless, amongst all the neighbors of this *chosen* sensor it has the minimum UID. Since the graph is densely populated and all sensors are initially *unchosen*, there will always exist a set of *unchosen* sensor nodes, whose sensing disks intersect with the query region and that is located at an uncovered area. Since any *unchosen* sensor which is located at the uncovered area does not have any *chosen* sensing neighbors, it will evaluate $SensorCover(i)$ to true by the first condition. Therefore by Rule $\mathcal{A}_2$, it changes its status from *unchosen* to *undecided.* Any *undecided* node will either change to *chosen* by $MCSCNode(i)$ or *unchosen* by $Redundant(i)$. Since all such *undecided* nodes are located outside of any *chosen* sensor's sensing disk, each of them will evaluate $GrtrLstOrNotNgbrOfChsn(i)$ as true and evaluate $Redundant(i)$ as false because the nodes' entire sensing disk is not

covered by *chosen* sensors. Thus, a node changes its status from *undecided* to *chosen* by Rule $\mathcal{A}_3$. Therefore, any uncovered area in the query region will be eventually covered by all such sensors that change their status to *chosen* by executing Rule $\mathcal{A}_3$. $Redundant(i)$ will be evaluated to true only if a node evaluates $GrtrLstOrNotNgbrOfChsn(i)$ as false, and if its entire sensing disk is covered by other *chosen* sensors and the removal of it does not disconnect its *chosen* neighbors. Thusly, once the query region is completely covered, $Redundant(i)$ will not unmark the sensor which removal will result in the creation of an uncovered area of the query region. The sensing disks of all *chosen* sensors in the final set completely cover the query region. Therefore we arrive at a contradiction. □

Lemma 5.2 (Connectivity) *In any legitimate configuration, the chosen set computed by Algorithm 5.5.1 forms a connected graph.*

Proof. We also prove this lemma by contradiction. Suppose the sensing disks of the sensors in the final chosen set computed by Algorithm 5.5.1 do not form a connected subgraph. Hence, there exists a sensor j in the final chosen set that is marked *chosen* and is not adjacent to another *chosen* sensor. More precisely, Sensor j is marked *chosen* and does not have any communication *chosen* neighbor.

Assume $SensorCover(i)$ and $MCSCNode(i)$ did not mark an *unchosen* Sensor i that is the minimum UID neighbor of Sensor j, as *chosen*, or $Redundant(i)$ unmarked this sensor. Since Sensor i is the minimum UID neighbor of Sensor j, Sensor j saves the UID of Sensor i in $Self\_Info_j.MinUID$. Also, since all sensors can have a status of *unchosen*, *undecided*, or *chosen*, and Sensor j has no *chosen* neighbors, Sensor i's status must be either *unchosen* or *undecided*. If Sensor i's status is *unchosen*, Sensor i evaluates $LstUIDNgbr(i, j)$ to true and thus evaluates $GrtrLstOrNotNgbrOfChsn(i)$ to true. Therefore, it changes its status

from *unchosen* to *undecided* after executing Rule $\mathcal{A}_2$, and then to *chosen* after executing Rule $\mathcal{A}_3$. If Sensor i's status is *undecided*, for the same reason as above, it changes its status from *undecided* to *chosen* by executing Rule $\mathcal{A}_3$. Also, Sensor i cannot evaluate $Redundant(i)$ as true since it has evaluated $GrtrLstOrNotNgbrOfChsn(i)$ to true.

Once again we arrive at a contradiction. □

Theorem 5.1 ($\mathcal{L}_{MCSC}$ satisfies specification) *Any system configuration satisfying the legitimacy predicate $\mathcal{L}_{MCSC}$ (per Definition 5.1) satisfies the specification of the minimal connected sensor cover problem (as given by Specification 4.2).*

Proof. The coverage and connectivity properties have been proven in Lemmas 5.1 and 5.2, respectively. By Definition 5.1, there exists no redundant *chosen* sensor in a legitimate configuration. That is, all redundant sensors have been identified and marked *unchosen.* Therefore, the connected cover set $\mathcal{MCSC}$ computed at this point is the smallest possible by Algorithm $\mathcal{SHID}$. □

Property 5.1 *The system defined by the legitimacy predicate $\mathcal{L}_{MCSC}$ is silent.*

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

Lemma 5.3 (Closure) *The legitimacy predicate $\mathcal{L}_{MCSC}$ is closed.*

Proof. Property 5.1 asserts the closure of $\mathcal{L}_{MCSC}$. □

5.6.2 Proof of Convergence

In this section, we aim to prove that starting from any arbitrary configuration of the system, or occurrence of any type of faults in the system, Algorithm $\mathcal{SHID}$guarantees that

in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

Lemma 5.4 (Convergence) *Starting from an arbitrary configuration, Algorithm $\mathcal{SHID}$ reaches a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.*

Proof. We will again prove this lemma by contradiction. Suppose that starting from any arbitrary configuration of a system of sensors, Algorithm 5.5.1 does not guarantee that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$. Hence, there exists a configuration in which, after any finite number of steps, the system will never reach a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$. That is, there exists a configuration in which, after any finite number of steps, the system will never reach a configuration in which all non redundant sensors are marked *chosen* and all redundant sensors are marked *unchosen*. This is described in the following cases.

Case 1: There exists a configuration in which a non redundant *unchosen* sensor which may evaluate $GrtrLstOrNotNgbrOfChsn(i)$ as true, but does not do so and does not change its status to *chosen*. That is, a query region sensor which is *unchosen*, and does not have any sensing neighbor whose UID is greater than its own or it is the minimum UID neighbor of a *chosen* sensor, is not marked as *chosen*, even if part of its sensing disk is not covered by a subset of *chosen* sensors. Since any query region sensor that is initially *unchosen*, and is non-redundant because it does not have any *chosen* sensing neighbor whose UID is greater than its own or it is the minimum UID neighbor of a *chosen* sensor, will evaluate $QryRgnIntrsctn(i)$, $GrtrLstOrNotNgbrOfChsn(i)$, and then $SensorCover(i)$ as true, the query region sensor changes its status to *undecided*. Also, since this sensor's sensing disk is not completely covered by *chosen* sensors, it will evaluate $MCSCNode(i)$

as true and change its status to *chosen* by executing Rule A3.

Hence we arrive at a contradiction.

Case 2: The non-redundant query region sensor is initially marked *chosen*, but executes *Redundant*(i) and is unmarked. Since this sensor executed *Redundant*(i), its entire sensing disk must be covered by a set of *chosen* sensors, it must have a *chosen* neighbor which has a greater UID than its own, and it must not be the minimum UID neighbor of a *chosen* sensor. Therefore this sensor is a redundant sensor.

Hence we arrive at a contradiction.

Case 3: A redundant sensor is marked as *chosen* or *undecided*, but *Redundant*(i) will not unmark this sensor. A redundant sensor is the one whose entire sensing disk is covered by the sensing disks of other *chosen* sensors, and it must have a *chosen* neighbor which has a greater UID than its own, and it must not be the minimum UID neighbor of a *chosen* sensor. Subsequently, it will evaluate $GrtrLstOrNotNgbrOfChsn(i)$ as false and $CvrSnsngByChsn(i)$ as true. Thus, by executing Rule $\mathcal{A}_1$, any redundant sensors will be unmarked.

Hence we arrive at a contradiction. □

5.6.3 Proof of Self-∗

5.6.3.1 Self-configuring

Due to the nature of Sensor networks, in most case, sensors are deployed randomly and densely distributed. This along with dynamic environmental changes make manual configuration of such systems extremely difficult. Therefore in sensor networks, the property of self-configuring to establish a topology that provide communication and sensing coverage gains further needs. From the proofs of closure (Lemma 5.3) and convergence (Lemma 5.4), it was shown that starting from any initial configuration, Algorithm 5.5.1 forms a

network topology in which all sensors in the set $\mathcal{MCSC}$ are connected, and are thus able to communicate with each other, either directly or indirectly. Also we have shown that starting from any arbitrary state, the given query region will eventually be completely covered. By executing the rules of Algorithm 5.5.1, network sensors will self-configure to establish a required topology under stringent energy constraints. Ergo, Algorithm 5.5.1 is self-configuring.

5.6.3.2 Self-healing

Self-healing (also called self-reconfiguration) is another important concept which makes the wireless sensor networks more robust systems. Our proposed solution is self-healing under various perturbations, such as node joins, failures, and state corruption. We will prove this by contradiction. Suppose Algorithm 5.5.1 is not self-healing. Thus, if a non-redundant node fails, a redundant node joins the network, or if there is an arbitrary corruption of the state variables of nodes, $Status_i$, then part of the query region may become uncovered, or may be covered by a redundant node. These perturbations will be demonstrated in the following cases.

Case 1:

If a non-redundant 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*. But since this *unchosen* sensor is not covered by a *chosen* sensor (i.e. it does not have any *chosen* sensing neighbors), it will evaluate $\mathcal{A}_2$ as true and $\mathcal{A}_3$ as true.

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 a non-redundant node fails, then part of the query region is covered by a redundant node. Since any node which has a UID less than its *chosen* sensing neighbor, but not a minimum UID neighbor of a *chosen* node, and whose entire transmission disk is covered by *chosen* nodes, is redundant and will not evaluate $GrtrLstOrNotNgbrOfChsn(i)$ as true. This node will not execute $\mathcal{A}_2$ and change to *undecided*, nor will it execute $\mathcal{A}_3$.

Thus, 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 state variables $Status_i$, then part of the query region may become uncovered, or may be covered by a redundant node. 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 sensing neighbor of a *chosen* sensor which has a greater UID than its own, or if it is the minimum UID neighbor of a *chosen* sensor, and if it has part of its sensing disk uncovered, then such an arbitrary corruption will still allow an *undecided* non-redundant node to execute $\mathcal{A}_3$ and change its status to *chosen*. Therefore, this sensor will cover the query region.

Hence we arrive at a contradiction.

Alternatively, $Redundant(i)$ will unmark a sensor even if it is initially *undecided* or *chosen*, if it has a smaller UID than a *chosen* sensing neighbor, it is not the minimum UID neighbor of a *chosen* sensor, and its entire sensing disk is covered by another *chosen* sensor. Therefore, a redundant node will not be included in the set $\mathcal{MCSC}$ to cover the query region.

Hence we arrive at a contradiction.

5.6.3.3 Self-∗

Our solution has been implemented with the self-configuring and self-healing features using the concept of self-stabilization. Since the paradigm of self-stabilization includes all other self-∗ properties, our solution is truly fault-tolerant in terms of the self-∗ feature.

CHAPTER 6

SINGLE-HOP RS-BASED ALGORITHMS

6.1 Description and Data Structures

In the previous algorithm, the UID is used to solve the contention of sensors and remove the redundant sensors. That is, between two neighboring sensors, the one with the greater UID remains in the final cover set, and among the neighbors of a *chosen* sensor, the one with the minimum UID also remains to maintain the connectivity. In this section, we will show a modification of Algorithm $\mathcal{SHID}$, which uses the sensing region instead of the UID for the above purpose, and we refer to this algorithm as $\mathcal{SHRS}$(Algorithm 6.2.1). The idea behind this is that since every sensor has a different sensing radius, keeping the sensors which have a larger coverage region results in a smaller number of sensors in the final set. Note that although the algorithm uses the sensing region instead of the UID, in the case of two or more sensors having the same sensing region, the UID is still needed as a deciding factor.

The following are necessary changes.

- The data structure $Info$ includes the variable $MaxR_SUID$ instead of $MinUID$. It represents the UID of the sensor with the maximum R_S amongst all of a sensor's neighbors. If several sensors have the same sensing radius, then the one with the greatest UID will be selected.

- The predicate $LstUIDNgbr(i, j)$ changes to $GrtstRsNgbr(i, j)$. To maintain connec-

tivity in Algorithm $\mathcal{SHRS}$, the predicate $GrtstRsNgbr(i, j)$ selects the sensor which has the greatest R_S among the neighbors of a chosen sensor.

- The predicate $GrtrLstOrNotNgbrOfChsn(i)$ changes to $GrtrGrtstOrNotNgbrOfChsn(i)$. Sensor i evaluates this predicate as true if one of the following is true: 1) Sensor i does not have a *chosen* neighbor which is also a sensing neighbor. 2) Sensor i's R_S is greater than a *chosen* sensing neighbor's R_S. 3) Sensor i has the greatest R_S among the neighbors of a *chosen* sensor.

Algorithm $\mathcal{SHRS}$shows only the modified parts while the rest remain the same.

6.3 Correctness

Even if a sensing radius is used instead of an UID for the redundant deciding factor, the proofs for the correctness of Algorithm 6.2.1 are very similar to the proofs for Algorithm 5.5.1 and can be referred to in Section 5.6.

Algorithm 6.2.1 Query Region Connected Sensor Cover Algorithm for Sensor i ($\mathcal{SHRS}$).

Changed Structure:

$Info\{$

UID :: Unique user identification number

$Status \in \{unchosen, undecided, chosen\}$ :: Status of a sensor

$Position$ :: Geometric location or coordinate of a sensor

R_C :: Communication radius of a sensor

R_S :: Sensing radius of a sensor

S :: Sensing region of a sensor

$MaxR_SUID$ :: UID of sensor with maximum R_S amongst all of a sensor's neighbors; in case of a tie, sensor with greatest UID is selected

$\}$

Changed Predicates:

$GrtstRsNgbr(i,j) \equiv i \in N_j \wedge (Self\_Info_i.R = N\_Info_i.Self\_Info_j.MaxR_SUID)$;

$\equiv$ Sensor i is a neighbor of Sensor j, and is also the neighbor of Sensor j having the greatest sensing radius;

$GrtrGrtstOrNotNgbrOfChsn(i) \equiv (\forall j : i \in N_j, N\_Info_i.Self\_Info_j.Status \neq chosen \vee \neg SnsngNgbr(i,j) \vee Self\_Info_i.R_S > N\_Info_i.Self\_Info_j.R_S \vee GrtstRsNgbr(i,j))$;

$\equiv$ Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor;

$SensorCover(i) \equiv Self\_Info_i.Status = unchosen \wedge QryRgnIntrsctn(i) \wedge GrtrGrtstOrNotNgbrOfChsn(i)$;

$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor;

$MCSCNode(i) \equiv Self\_Info_i.Status = undecided \wedge (GrtrGrtstOrNotNgbrOfChsn(i) \vee \neg CvrSnsngByChsn(i))$;

$\equiv$ Sensor i is an *undecided* sensor and is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor, or a part of the sensing disk of Sensor i is not covered by a *chosen* sensor;

$Redundant(i) \equiv (Self\_Info_i.Status = undecided \vee Self\_Info_i.Status = chosen) \wedge \neg GrtrGrtstOrNotNgbrOfChsn(i) \wedge CvrSnsngByChsn(i) \wedge NeighborsConnectivity(i)$;

$\equiv$ Sensor i is an *undecided* or a *chosen* sensor and is the communication and sensing neighbor of a *chosen* sensor that has a greater R_S than its own, but is not the "$MaxR''_S$ neighbor of this sensor, the entire sensing disk of Sensor i is covered by *chosen* sensors, and *chosen* neighbors of Sensor i are connected through a second path;

CHAPTER 7

MULTI-HOP ALGORITHMS

7.1 Description and Data Structures

In Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$, a sensor uses only 2-hop information to check if its entire sensing region is covered by a subset of *chosen* sensors and if every pair of its *chosen* neighbors has an alternate communication path. If the sensing radius is greater than the communication radius, then 2-hop information is not enough to verify this coverage condition. Since we did not assume any limitation for the sensing radius, it is possible that the sensing region of Sensor i is covered by the sensors that are located several hops away from Sensor i. Similarly, it is possible that *chosen* neighbors are connected to each other via paths of more than two hops. Thus, to obtain a better approximation, multi-hop information is required. Therefore, in Algorithms $\mathcal{MHID}$(algorithm 7.3.1) and $\mathcal{MHRS}$(algorithm 7.3.2), our goal is to further reduce the number of nodes in the final set by increasing the available information in exchange for the cost of extra communication. We would like to collect upto a maximum of H-hop count information, where H is a constant. Algorithm $\mathcal{MHID}$is a modification of Algorithm $\mathcal{SHID}$, and similarly, Algorithm $\mathcal{MHRS}$is a modification of Algorithm $\mathcal{SHRS}$. Algorithm $\mathcal{MHRS}$shows only the parts which are different from Algorithm $\mathcal{MHID}$.

Data Structures. The information required to compute the coverage condition is the UID, S, and coordinates of all *chosen* sensors within H-hops of Sensor i. Also, the hop

count should be recorded. To collect this information, flooding is too expensive, and there are many redundant messages. So, we assume that there is a self-stabilizing BFS tree construction running in the background. Each *chosen* sensor maintains its own BFS tree, with a height of H rooted to itself. The information of all *chosen* sensors within H-hops from Sensor i will be passed along using these BFS trees which Sensor i receives only from its parent sensor. Therefore, each sensor has to maintain a set of parent pointers P_i. The number of parent pointers per node is less than or equal to the number of *chosen* sensors within H-hops.

To gather information from *chosen* sensors located more than 2-hops away, the following change has been made in the data structure and the read action of Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$. The data structure *Root*, which contains the root node's UID, S, *Position*, and Hop, was added. Also, there are extra shared variables $Root\_Info_i$ and C_i. $Root\_Info_i$ is a set of *Root* structures, and C_i is a set of *chosen* sensors within H-hop distance. A set of parent pointers is kept as a local variable.

Macro. There are two separate read macros dependent upon whether or not Sensor i's neighbor j is i's parent. If j is not i's parent, then the read action is the same as that of Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$. If j is the parent of i, when timeout occurs, i reads the root information, as well as j's information and j's neighbors' (i's 2-hop neighbors') information. After Sensor i reads $Root\_Info_j$ from j, i increments Hop count in $Root\_Info_i.Root_R$. If it is greater than $H-1$, then this data is discarded, and R is removed from C_i.

7.2 Predicates

This multi-hop information is applied to the predicates $CvrSnsngByChsn(i)$ and $NeighborsConnectivity(i)$. $CvrSnsngByChsn(i)$ is evaluated as true if Sensor i's sensing

disk is covered by a subset of *chosen* sensors that are located no farther than H communication hops from Sensor i. For the predicate $NeighborsConnectivity(i)$ to check the connectivity of Sensor i's chosen neighbors, the new predicate $Cycle(x, y)$ is added, which is evaluated as true 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. Hence, the predicate $NeighborsConnectivity(i)$ is true when all pairs of *chosen* neighbors of Sensor i are connected by a path of *chosen* sensors located within H-hops from Sensor i.

7.4 Correctness

The purpose of multi-hop algorithms is to obtain a better approximation of MCSC by using the information of sensors which are located several hops away. Such multi-hop information is used in the predicates $CvrSnsngByChsn(i)$ and $NeighborsConnectivity(i)$, which are checked in $Redundant(i)$ to remove the redundant sensors. A sensor is redundant and unmarked by Rule $\mathcal{A}_1$ if the following criteria are met; if a sensor's status is either *chosen* or *undecided*; it has a sensing neighbor which has a greater UID than its own; it is not the minimum UID neighbor of a *chosen* sensor; and both $CvrSnsngByChsn(i)$ and $NeighborsConnectivity(i)$ are evaluated as true. In multi-hop algorithms, $CvrSnsngByChsn(i)$ is evaluated as true if a sensor's entire sensing disk is covered by a subset of *chosen* sensors which are located within H-hops. Similarly, $NeighborsConnectivity(i)$ is evaluated as true if every pair of *chosen* neighbors of a sensor is able to communicate with each other using an alternative path made by *chosen* sensors within H-hops. Therefore, to prove the correctness of multi-hop algorithms (Algorithm 7.3.1 and 7.3.2), we need to prove multi-hop information gathering by Algorithms $\mathcal{MHID}$and $\mathcal{MHRS}$. The rest of the proofs are the same as the proofs of $\mathcal{SHID}$, and can be referenced in Section 5.6.

Algorithm 7.3.1 Query Region Connected Cover Algorithm for Sensor i ($\mathcal{MHID}$).

Constants:
R_Q:: Query region;
H:: Hop count

Structure:
$Info$\{
UID :: Unique user identification number
$Status \in \{unchosen, undecided, chosen\}$ :: Status of a sensor
$Position$ :: Geometric location or coordinate of a sensor
R_C :: Communication radius of a sensor
R_S :: Sensing radius of a sensor
S :: Sensing region of a sensor
$MinUID$ :: minimum UID amongst all of a sensor's neighbors' UIDs
\}
$Root$\{
UID :: Unique user identification number
$Position$ :: Geometric location or coordinate of a sensor
R_S :: Sensing radius of a sensor
Hop :: Hop count
\}

Shared Variables:
$Info\ Self\_Info_i$ :: One structure that contains information for $Sensor_i$
$Set\ Root\_Info_i$ :: Set of structure Root
$Set\ N\_Info_i$ :: Set of δ structures that contain all neighbors' information
$Set\ N_i :: \{j \in V | Dist(i,j) \leq R_{C_i} \wedge Dist(i,j) \leq R_{C_j}\}$
$Set\ C_i$ :: Set of *chosen* sensors within n Hops

Local Variables:
$Set\ 2N\_Info_i$ :: Set of $\delta_i + \sum \delta_{j \in N_i}$ structures that contain all 2-hop neighbors' information
$Set\ NN_i$:: Set of N_j, $\forall j \in N_i$
$Set\ P_i$:: Set of parent pointers

Macro:
$Read(j)$
$N\_Info_i = N\_Info_i \bigcup Self\_Info_j$
$NN_i = NN_i \bigcup N_j$
$2N\_Info_i = 2N\_Info_i \bigcup N\_Info_j$
$j = Next(N_i)$

$Read\_Parent(j)$
$N\_Info_i = N\_Info_i \bigcup Self\_Info_j$
$NN_i = NN_i \bigcup N_j$
$2N\_Info_i = 2N\_Info_i \bigcup N\_Info_j$
$C_i = C_i \bigcup C_j$
$Root\_Info_i = Root\_Info_i \bigcup Root\_Info_j$
$AdjustRoot\_Info_i and\ C_i$ :: Increment $Root\_Info_i.Root_R.Hop$ by 1. If it becomes > H - 1, then discard the data and remove R from C_i.
$j = Next(N_i)$

Predicates:

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

$Dist(i,j) \equiv$ Returns the Euclidean distance between Sensor i and Sensor j;

$SnsngNgbr(i,j) \equiv (\forall j$: Dist(i,j) $\leq min(Self\_Info_i.R_S, N\_Info_i.Self\_Info_j.R_S))$;
$\equiv$ Sensor i and Sensor j are located within each others' sensing disks;

$Cycle(x,y) \equiv \exists cycle : \{\ldots, x, i, y, \ldots\}$ are vertices in the cycle and all other vertices in the cycle are elements of C_i (*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;

$CvrSnsngByChsn(i) \equiv (\exists A : \forall j \in A \wedge j \in C_i \wedge Self\_Info_i.S \subseteq (\bigcup Root\_Info_i.Root_r.S)$;
$\equiv$ Sensing region of Sensor i is covered by a subset of *chosen* sensors that are located no farther than H communication hops from Sensor i;

$NeighborsConnectivity(i) \equiv (\forall j, t \in N_i$,Cycle(j, t));
$\equiv$ All *chosen* pairs of neighbors of Sensor i are connected by a path of *chosen* nodes;

$LstUIDNgbr(i,j) \equiv i \in N_j \wedge (Self\_Info_i.UID = N\_Info_i.Self\_Info_j.MinUID)$;
$\equiv$ Sensor i is a neighbor of Sensor j, and is also the neighbor of Sensor j having the least UID;

$GrtrLstOrNotNgbrOfChsn(i) \equiv (\forall j : i \in N_j, N\_Info_i.Self\_Info_j.Status \neq chosen \vee$
$\neg SnsngNgbr(i,j) \vee Self\_Info_i.UID >$
$N\_Info_i.Self\_Info_j.UID \vee LstUIDNgbr(i,j))$;
$\equiv$ Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor;

$SensorCover(i) \equiv Self\_Info_i.Status = unchosen \wedge QryRgnIntrsctn(i) \wedge$
$GrtrLstOrNotNgbrOfChsn(i)$;
$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor;

$MCSCNode(i) \equiv Self\_Info_i.Status = undecided \wedge (GrtrLstOrNotNgbrOfChsn(i) \vee$
$\neg CvrSnsngByChsn(i))$;
$\equiv$ Sensor i is an *undecided* sensor and is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose UID is greater than its own unless it is the "least UID" neighbor of this *chosen* sensor, or a part of the sensing disk of Sensor i is not covered by a *chosen* sensor;

$Redundant(i) \equiv (Self\_Info_i.Status = undecided \vee Self\_Info_i.Status = chosen) \wedge$
$\neg GrtrLstOrNotNgbrOfChsn(i) \wedge CvrSnsngByChsn(i) \wedge NeighborsConnectivity(i)$;
$\equiv$ Sensor i is an *undecided* or a *chosen* sensor and is the "lesser" communication and sensing neighbor of a *chosen* sensor, but is not the neighbor of this sensor that has the smallest UID, and the entire sensing disk of Sensor i is covered by *chosen* sensors, and *chosen* neighbors of Sensor i are connected through a second path;

Actions:

$\mathcal{A}_1 :: \neg QryRgnIntrsctn(i) \vee Redundant(i)$
$\longrightarrow Self\_Info_i.Status = unchosen;$

$\mathcal{A}_2 :: SensorCover(i)$
$\longrightarrow Self\_Info_i.Status = undecided;$

$\mathcal{A}_3 :: MCSCNode(i)$
$\longrightarrow Self\_Info_i.Status = chosen;$

$\mathcal{A}_4 :: Timeout \wedge j \in N_i \wedge \neg(j \in P_i)$
$\longrightarrow Read(j);$

$\mathcal{A}_5 :: Timeout \wedge j \in N_i \wedge j \in P_i$
$\longrightarrow Read\_Parent(j);$

7.4.1 Proof of Multi-hop Information Gathering

Lemma 7.1 *The coverage and connectivity related information of every chosen sensor will eventually reach every sensor within H-hops.*

Proof. We prove this lemma by contradiction. Suppose there exists a sensor which never receives the information of *chosen* sensors located within H-hops from it. Although we assume that every sensor might have a different communication radius, we limit the neighbor set so that neighbors always have a bi-directional communication link. Thus, if the distance from Sensor A to Sensor B is n-hops, then the distance from Sensor B to Sensor A is also n-hops.

We assume that there is a self-stabilizing BFS tree construction running in the background, and each *chosen* sensor maintains its own BFS tree, with a height of H, rooted to itself. Since the BFS tree spans all sensors within H-hops, there is always a path from a *chosen* root sensor to every sensor within H-hops from the root. Thus, all information sent out from the root along the BFS tree eventually reaches all sensors within H-hops. That is, every sensor is able to receive the information of all *chosen* sensors located within H-hops from itself.

Hence we arrive at a contradiction. □

Algorithm 7.3.2 Query Region Connected Cover Algorithm for Sensor i ($\mathcal{MHRS}$).

Changed Structure:

$Info\{$
- $Status \in \{unchosen, undecided, chosen\}$ :: Status of a sensor
- $Position$ :: Geometric location or coordinate of a sensor
- R_C :: Communication radius of a sensor
- R_S :: Sensing radius of a sensor
- S :: Sensing region of a sensor
- $MaxR_SUID$ :: UID of a sensor with the maximum R_S amongst all of a sensor's neighbors; in case of a tie, the sensor with greatest UID is selected

$\}$

$Root\{$
- $Position$ :: Geometric location or coordinate of a sensor
- R_S :: Sensing radius of a sensor
- Hop :: Hop count

$\}$

Changed Predicates:

$GrtstRsNgbr(i,j) \equiv i \in N_j \wedge (Self\_Info_i.UID = N\_Info_i.Self\_Info_j.MaxR_SUID)$;
$\equiv$ Sensor i is a neighbor of Sensor j, and is also the neighbor of Sensor j having the greatest sensing radius;

$GrtrGrtstOrNotNgbrOfChsn(i) \equiv (\forall j : i \in N_j, N\_Info_i.Self\_Info_j.Status \neq chosen \vee \neg SnsngNgbr(i,j) \vee Self\_Info_i.R_S > N\_Info_i.Self\_Info_j.R_S \vee GrtstRsNgbr(i,j))$;
$\equiv$ Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor;

$SensorCover(i) \equiv Self\_Info_i.Status = unchosen \wedge QryRgnIntrsctn(i) \wedge GrtrGrtstOrNotNgbrOfChsn(i)$;
$\equiv$ status of Sensor i is *unchosen*, sensing disk of Sensor i intersects with some portion of query region, and Sensor i is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor;

$MCSCNode(i) \equiv Self\_Info_i.Status = undecided \wedge (GrtrGrtstOrNotNgbrOfChsn(i) \vee \neg CvrSnsngByChsn(i))$;
$\equiv$ Sensor i is an *undecided* sensor and is not the communication neighbor, nor the sensing neighbor, of a *chosen* sensor whose sensing radius is greater than its own unless it is the "$MaxR''_S$ neighbor of this *chosen* sensor, or a part of the sensing disk of Sensor i is not covered by a *chosen* sensor;

$Redundant(i) \equiv (Self\_Info_i.Status = undecided \vee Self\_Info_i.Status = chosen) \wedge \neg GrtrGrtstOrNotNgbrOfChsn(i) \wedge CvrSnsngByChsn(i) \wedge NeighborsConnectivity(i)$;
$\equiv$ Sensor i is an *undecided* or a *chosen* sensor and is the communication and sensing neighbor of a *chosen* sensor that has a greater R_S than its own, but is not the "$MaxR''_S$ neighbor of this sensor, and the entire sensing disk of Sensor i is covered by *chosen* sensors, and *chosen* neighbors of Sensor i are connected through a second path;

CHAPTER 8

SIMULATION AND RESULTS

8.1 Discussion of Result

Algorithms $\mathcal{SHID}$(Algorithm 5.5.1), $\mathcal{SHRS}$(Algorithm 6.2.1), $\mathcal{MHID}$(Algorithm 7.3.1), and $\mathcal{MHRS}$(Algorithm 7.3.2) compute a minimum connected sensor cover for a query region. Moreover, all algorithms are fault-tolerant in terms of the self-$*$ feature.

In our simulations, we assumed that nodes are randomly deployed on a grid of size 500 $\times$ 500 (350,000 nodes). Similar to [38, 53, 67] we considered the sensing region associated with a sensor to be a circular region centered around the sensor itself. We considered a network of 350,000 nodes in which sensors had both sensing radii and transmission radii that varied in size from 0 to 8 units. However, in some of the cases tested in our simulations, we restricted the sizes of sensors' sensing and transmission radii to be within a certain range. For Algorithms $\mathcal{MHID}$and $\mathcal{MHRS}$, we went the query region boundaries for the hop count. Specifically, a hop count of 8 was used.

The query region used in our simulations was 15 x 15 square graph units. We measured the number of sensors in the final minimum connected cover set, the number of query region sensors covered per MCSC sensor, the average number of sensors within a sensor's sensing disk, and the stabilization times for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, $\mathcal{MHRS}$, Algorithm $MCSC$ [24], and Rule k [21]. We also computed an approximation ratio for each algorithm. This approximation ratio was used as a measure of optimality for each

algorithm and was computed as the ratio between the number of query region sensors covered per MCSC sensor and the average number of sensors within a sensor's sensing disk. The smaller this ratio is, the closer the final cover set chosen by a particular algorithm is to an optimal minimum connected cover set.

We used varying relative sizes of R_C and R_S for our simulations. Cases tested include:

- $R_C \geq R_S$ (all sensors had the same size of radii of communication).
- $R_C \geq R_S$ (all sensors had the same size of sensing radii).
- $R_C \geq R_S$ (sensors had different sizes of R_C and R_S).
- $R_C > R_S$ (all sensors had the same size of radii of communication).
- $R_C > R_S$ (all sensors had the same size of sensing radii).
- $R_C = R_S$ (all sensors had the same size of R_C and R_S).
- $R_C < R_S$ (all sensors had the same size of radii of communication).
- $R_C < R_S$ (all sensors had the same size of sensing radii).
- $R_C < R_S$ (sensors had different sizes of R_C and R_S).
- All sensors had equal sizes of radii of communication but unequal sizes of sensing radii.
- All sensors had equal sizes of sensing radii but unequal sizes of radii of communication.
- All sensors had unequal sizes of radii of communication and unequal sizes of sensing radii.

Figure 8.5 shows simulation screenshots for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, and $\mathcal{MHRS}$with variable sensing and communication radii. From the simulation screenshots we can conclude that all these algorithms can produce the final cover set that completely cover the query region since there are no black spots inside the query region.

Tables 8.1 and 8.2 show the number of $MCSC$ sensors, query region sensors per $MCSC$ sensor, average number of sensors per sensing disk, and stabilization times for these cases. These results, excluding stabilization times, are also shown in Figures 8.1, 8.2, and 8.3. Tables 8.3 and 8.4, and Figure 8.4, show the approximation ratios for the algorithms tested for each of these cases.

As shown in Table 8.1 and Figure 8.1, in all cases tested except all R_S equal, Algorithm $\mathcal{MHID}$produced a cover set that contained fewer, or in some cases nearly the same, number of nodes as that of Algorithm $\mathcal{SHID}$. This implies that for a UID based algorithm, multi-hop coverage and connectivity does significantly reduce the number of nodes in the final cover set for nearly all ranges of size of sensing and transmission radii. Also, in more than half the cases tested, specifically in all cases tested except $R_C > R_S$ (all R_S equal), $R_C = R_S$ (all R_C and R_S equal), $R_C < R_S$ (all R_C equal), $R_C < R_S$ (all R_S equal), and $R_C < R_S$ (R_C and R_S unequal), Algorithm $\mathcal{MHRS}$produced a cover set that contained fewer, or nearly the same, number of nodes than Algorithm $\mathcal{SHRS}$. This also implies that for an algorithm based upon the size of R_S, multi-hop coverage and connectivity does reduce the number of nodes in the final cover set for most of the ranges of size of sensing and transmission radii.

Tables 8.3, 8.4, and Figure 8.4 show that for a UID based algorithm, multi-hop coverage does significantly improve the algorithm's approximation ratio. Also, in most cases tested, for a R_S based algorithm, multi-hop coverage significantly improves the algorithm's approx-

imation ratio. This implies that an algorithm that uses multi-hop coverage and connectivity produces a final cover set that is closer to an optimal minimum connected cover set.

Algorithm $\mathcal{SHRS}$that uses two-hop coverage and connectivity had a better approximation ratio than the UID based algorithm ($\mathcal{SHID}$) that uses two-hop coverage and connectivity, when the size of R_C was less than R_S. This is due to the fact that when $R_C < R_S$, the SnsngNgbr(i,j) predicate will evaluate to true when Sensor j is a neighbor of Sensor i. Thus, ¬SnsngNgbr(i,j) will evaluate to false in the predicate GrtrGrtstOrNotNgbrOfChsn(i), and the sensor with the greatest R_S, within Sensor i's neighborhood, will evaluate this predicate to true and be marked as chosen by $\mathcal{A}_2$. This allows the query region to be covered with fewer nodes. However, in nearly all cases, if R_C is greater than or equal to R_S, Algorithm $\mathcal{SHRS}$has a worse approximation ratio than Algorithm $\mathcal{SHID}$. This is due to the fact that a sensor with a sensing radius that is smaller than R_C may be able to evaluate ¬SnsngNgbr(i,j) to true in GrtrGrtstOrNotNgbrOfChsn(i), even though it may have a small R_S. Thus, it can evaluate this predicate as true and change to *chosen*, even though there may be more suitable sensors (those with larger R_S) outside Sensor i's neighborhood.

As an improvement, as shown in Table 8.3 and Figure 8.4, an R_S based algorithm using multi-hop coverage (Algorithm $\mathcal{MHRS}$) produced a better cover set than all of our other algorithms when $R_C > R_S$ (all R_C equal) and produced one of the lowest approximation ratios obtained by our algorithms (2.4). This is due to CvrSnsngByChsn(i) and NeighborsConnectivity(i) predicates having a greater chance of being evaluated to true as sensors further than 2-hops from Sensor i are considered. Subsequently, as a greater number of sensors are marked as *chosen*, a sensor that may not have been the most suitable to be marked may evaluate GrtrGrtstOrNotNgbrOfChsn(i) as false, evaluate Redundant(i) as true, and

unmark itself. This leads to fewer nodes in the final cover set.

As shown in Table 8.3 and Figure 8.4, the approximation ratio for the multi-hop, R_S based algorithm ($R_C > R_S$ and all R_C equal) is equal to that of Algorithm $MCSC$ ($R_C = R_S$ and all R_C and R_S equal). However, Algorithm $\mathcal{MHRS}$can also produce a cover set that is connected and that completely covers R_Q at <u>all</u> ranges of size of sensing and transmission radii.

Also, our Algorithm$\mathcal{MHRS}$produces better approximation ratios than Rule k for most cases, when $R_C \geq R_S$ or $R_C > R_S$. This improvement may be attributed to a greater number of nodes that were unmarked by Algorithm $\mathcal{MHRS}$'s redundancy predicate. Since the CvrSnsngByChsn(i) predicate in this algorithm considers nodes that can be located further than two hops from Sensor i, there is a greater chance that a node evaluates this predicate to true and becomes unmarked by Redundant(i) in Algorithm $\mathcal{MHRS}$, than a node evaluating the redundancy predicate of Rule k to true. As a result, a greater number of nodes will be unmarked by Redundant(i) in Algorithm $\mathcal{MHRS}$. Algorithm $\mathcal{MHID}$also produces better approximation ratios than Rule k in most cases when $R_C \geq R_S$ or $R_C > R_S$. In addition to this, when $R_C > R_S$ and all R_S are equal, Algorithm $\mathcal{SHID}$and $\mathcal{SHRS}$produce better approximation ratios than Rule k. In contrast to Rule k, our algorithms can also produce a cover set that is connected and that completely covers R_Q at <u>all</u> ranges of size of sensing and transmission radii.

As shown in Table 8.1, for nearly all cases, the stabilization times for Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$are less than or equal to that of Rule k. This shows that these algorithms outperform Rule k in terms of stabilization time. Although Algorithms $\mathcal{MHID}$and $\mathcal{MHRS}$have a greater stabilization time, in most cases when $R_C \geq R_S$ or $R_C > R_S$, these algorithms outperform Rule k in terms of producing a more optimal cover set. Also,

in most cases, Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$outperformed Algorithm $MCSC$ in terms of stabilization time.

Finally, in more than half the cases when $R_C \geq R_S$ or $R_C > R_S$, the approximation ratios for Algorithm $\mathcal{MHRS}$were similar to that of Algorithm $MCSC$. These results lead us to believe that the ability to produce a better approximation to an optimal cover set, combined with the ability to completely cover and produce a connected cover set for *all* ranges of sizes of the radii of communication and sensing radii, justify the increase in stabilization time and message complexity required for multi-hop coverage and connectivity.

8.2 Tables

		Relative Size of R_C or R_S					
		$R_C \geq R_S$ all R_C =	$R_C \geq R_S$ all R_S =	$R_C \geq R_S$ $R_C \wedge R_S \neq$	$R_C > R_S$ all R_C =	$R_C > R_S$ all R_S =	$R_C = R_S$ all R_C, R_S =
Alg. $\mathcal{SHID}$	**Number of MCSC Sensors**	158	35	140	163	40	30
Alg. $\mathcal{SHID}$	**Qry Rgn Sensors / MCSC Sensor**	4.2	18.3	4.5	4.1	16.7	21.0
Alg. $\mathcal{SHID}$	**Avg # of Sensors / Sensing Disk**	21.9	58.0	22.2	15.2	41.5	88.9
Alg. $\mathcal{SHID}$	**Stabilization Time (min.)**	2.7	1.2	1.8	2.7	1.2	1.3
Alg. $\mathcal{SHRS}$	**Number of MCSC Sensors**	144	40	147	164	35	25
Alg. $\mathcal{SHRS}$	**Qry Rgn Sensors / MCSC Sensor**	4.4	15.6	4.7	4.0	19.3	24.4
Alg. $\mathcal{SHRS}$	**Avg # of Sensors / Sensing Disk**	25.6	64.7	26.9	17.0	45.6	88.3
Alg. $\mathcal{SHRS}$	**Stabilization Time (min.)**	2.2	1.2	3.0	3.3	1.2	1.2
Alg. $\mathcal{MHID}$	**Number of MCSC Sensors**	75	25	139	126	41	27
Alg. $\mathcal{MHID}$	**Qry Rgn Sensors / MCSC Sensor**	9.6	26.6	4.8	5.2	15.3	24.4
Alg. $\mathcal{MHID}$	**Avg # of Sensors / Sensing Disk**	27.4	63.2	23.8	15.0	36.8	97.1
Alg. $\mathcal{MHID}$	**Stabilization Time (min.)**	619.5	4.2	94.5	76.5	3.6	2.1
Alg. $\mathcal{MHRS}$	**Number of MCSC Sensors**	137	38	71	123	42	34
Alg. $\mathcal{MHRS}$	**Qry Rgn Sensors / MCSC Sensor**	4.9	16.4	9.0	5.3	15.3	19.4
Alg. $\mathcal{MHRS}$	**Avg # of Sensors / Sensing Disk**	22.2	55.0	22.0	12.6	38.0	98.0
Alg. $\mathcal{MHRS}$	**Stabilization Time (min.)**	124.0	8.9	11.2	41.5	7.1	1.7
Alg. $MCSC$	**Number of MCSC Sensors**						16
Alg. $MCSC$	**Qry Rgn Sensors / MCSC Sensor**						42.9
Alg. $MCSC$	**Avg # of Sensors / Sensing Disk**						104.9
Alg. $MCSC$	**Stabilization Time (min.)**						1.4
Rule k	**Number of MCSC Sensors**						20
Rule k	**Qry Rgn Sensors / MCSC Sensor**						33.7
Rule k	**Avg # of Sensors / Sensing Disk**						101.1
Rule k	**Stabilization Time (min.)**						2.7

Table 8.1: Number of MCSC Sensors, Query Region Sensors per MCSC Sensor, Average Number of Sensors per Sensing Disk, and Stabilization Times for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, $\mathcal{MHRS}$, Algorithm $MCSC$, and Rule k at Various Sizes of R_C and R_S.

		Relative Size of R_C or R_S					
		$R_C < R_S$ **all** $R_C =$	$R_C < R_S$ **all** $R_S =$	$R_C < R_S$ $R_C \wedge R_S \neq$	**all** $R_C =$	**all** $R_S =$	$R_C \wedge R_S \neq$
Alg. $\mathcal{SHID}$	**Number of MCSC Sensors**	62	280	332	144	179	227
Alg. $\mathcal{SHID}$	**Qry Rgn Sensors / MCSC Sensor**	11.4	2.4	2.0	4.4	3.5	2.7
Alg. $\mathcal{SHID}$	**Avg # of Sensors / Sensing Disk**	106.6	97.3	100.9	44.7	37.3	42.2
Alg. $\mathcal{SHID}$	**Stabilization Time (min.)**	1.3	1.3	1.3	1.3	1.2	1.3
Alg. $\mathcal{SHRS}$	**Number of MCSC Sensors**	58	275	310	144	194	235
Alg. $\mathcal{SHRS}$	**Qry Rgn Sensors / MCSC Sensor**	10.9	2.3	2.1	4.2	3.2	2.8
Alg. $\mathcal{SHRS}$	**Avg # of Sensors / Sensing Disk**	68.3	92.8	104.4	41.7	39.4	45.7
Alg. $\mathcal{SHRS}$	**Stabilization Time (min.)**	1.2	1.2	1.2	1.3	1.2	1.2
Alg. $\mathcal{MHID}$	**Number of MCSC Sensors**	62	277	338	92	204	227
Alg. $\mathcal{MHID}$	**Qry Rgn Sensors / MCSC Sensor**	10.5	2.4	1.9	6.8	3.1	2.8
Alg. $\mathcal{MHID}$	**Avg # of Sensors / Sensing Disk**	95.0	96.6	89.4	41.1	34.7	41.2
Alg. $\mathcal{MHID}$	**Stabilization Time (min.)**	4.1	11.2	7.3	14.8	83.3	5.6
Alg. $\mathcal{MHRS}$	**Number of MCSC Sensors**	73	314	361	145	178	237
Alg. $\mathcal{MHRS}$	**Qry Rgn Sensors / MCSC Sensor**	8.5	2.2	1.9	4.5	3.6	2.5
Alg. $\mathcal{MHRS}$	**Avg # of Sensors / Sensing Disk**	88.7	104.9	100.6	42.5	37.4	39.3
Alg. $\mathcal{MHRS}$	**Stabilization Time (min.)**	8.5	2.7	72.9	14.7	4.5	5.5

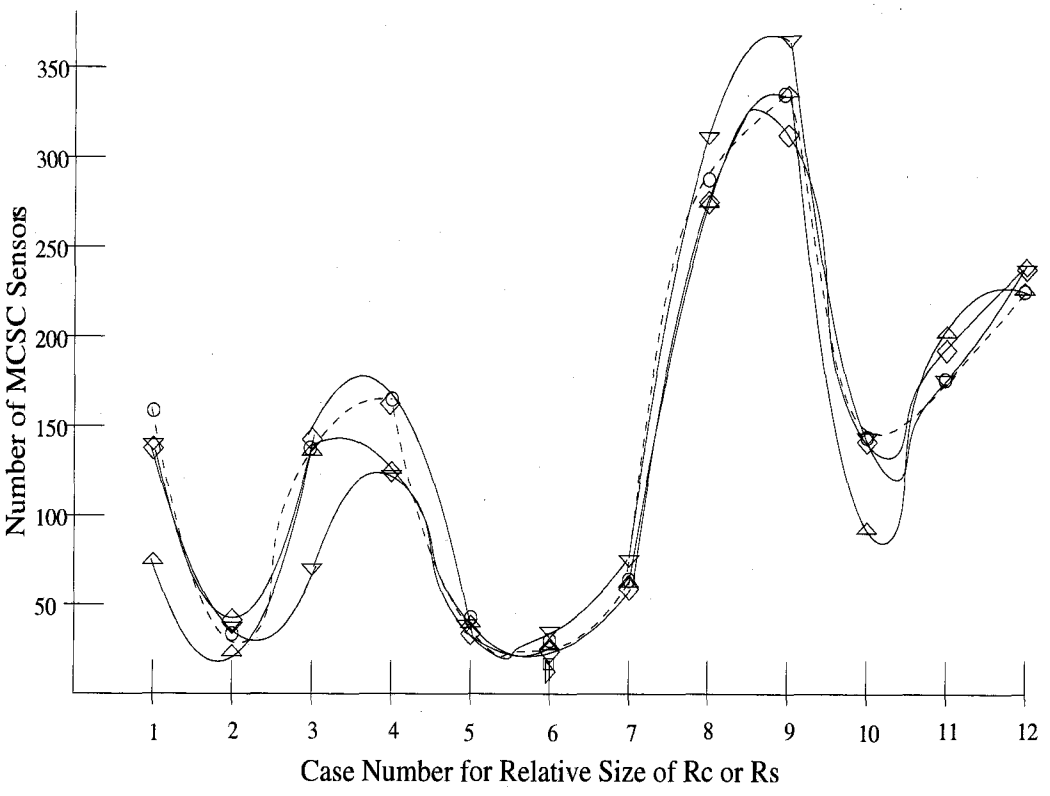
Table 8.2: Number of MCSC Sensors, Query Region Sensors per MCSC Sensor, Average Number of Sensors per Sensing Disk, and Stabilization Times for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, $\mathcal{MHRS}$, Algorithm $MCSC$, and Rule k at Various Sizes of R_C and R_S.

		Relative Size of R_C or R_S					
		$R_C \geq R_S$ **all** $R_C =$	$R_C \geq R_S$ **all** $R_S =$	$R_C \geq R_S$ $R_C \wedge R_S \neq$	$R_C > R_S$ **all** $R_C =$	$R_C > R_S$ **all** $R_S =$	$R_C = R_S$ **all** R_C, R_S $=$
Alg. $\mathcal{SHID}$	**Approximation Ratio**	5.2	3.2	4.9	3.7	2.5	4.2
Alg. $\mathcal{SHRS}$	**Approximation Ratio**	5.8	4.1	5.7	4.3	2.4	3.6
Alg. $\mathcal{MHID}$	**Approximation Ratio**	2.9	2.4	4.9	2.9	2.4	4.0
Alg. $\mathcal{MHRS}$	**Approximation Ratio**	4.6	3.4	2.4	2.4	2.5	5.1
Alg. $MCSC$	**Approximation Ratio**						2.4
Rule k	**Approximation Ratio**						3.0

Table 8.3: Approximation Ratios for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, $\mathcal{MHRS}$, Algorithm $MCSC$, and Rule k at Various Sizes of R_C and R_S.

		Relative Size of R_C or R_S					
		$R_C < R_S$ **all** $R_C =$	$R_C < R_S$ **all** $R_S =$	$R_C < R_S$ $R_C \wedge R_S \neq$	**all** $R_C =$	**all** $R_S =$	$R_C \wedge R_S \neq$
Alg. $\mathcal{SHID}$	**Approximation Ratio**	9.4	41.4	49.9	10.1	10.6	15.6
Alg. $\mathcal{SHRS}$	**Approximation Ratio**	6.3	40.3	49.7	9.9	12.3	16.3
Alg. $\mathcal{MHID}$	**Approximation Ratio**	9.0	40.9	47.6	6.1	11.3	14.9
Alg. $\mathcal{MHRS}$	**Approximation Ratio**	10.4	47.7	54.1	9.5	10.4	15.5

Table 8.4: Approximation Ratios for Algorithms $\mathcal{SHID}$, $\mathcal{SHRS}$, $\mathcal{MHID}$, and $\mathcal{MHRS}$, at Various Sizes of R_C and R_S.



Case Number Key

1. Rc >= Rs (all Rc equal)
2. Rc >= Rs (all Rs equal)
3. Rc >= Rs (Rc, Rs unequal)
4. Rc > Rs (all Rc equal)
5. Rc > Rs (all Rs equal)
6. Rc = Rs (all Rc, Rs equal)
7. Rc < Rs (all Rc equal)
8. Rc < Rs (all Rs equal)
9. Rc < Rs (Rc, Rs unequal)
10. all Rc equal
11. all Rs equal
12. Rc and Rs unequal

Algorithm Key

Alg. SHID
Alg. MHID
Alg. MCSC
Alg. SHRS
Alg. MHRS
Rule k

Figure 8.1: Number of MCSC Sensors for Various Relative Sizes of R_C or R_S

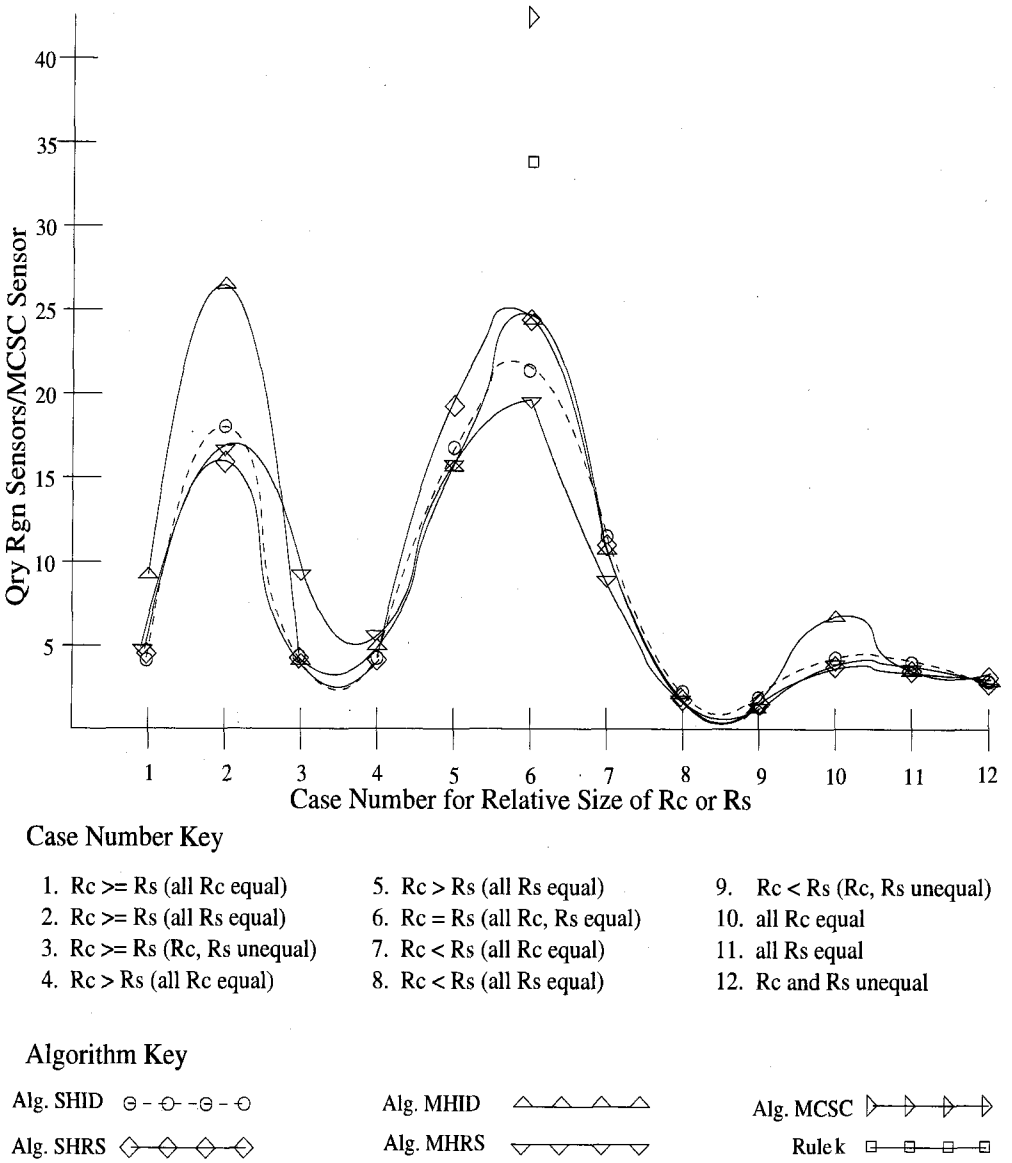
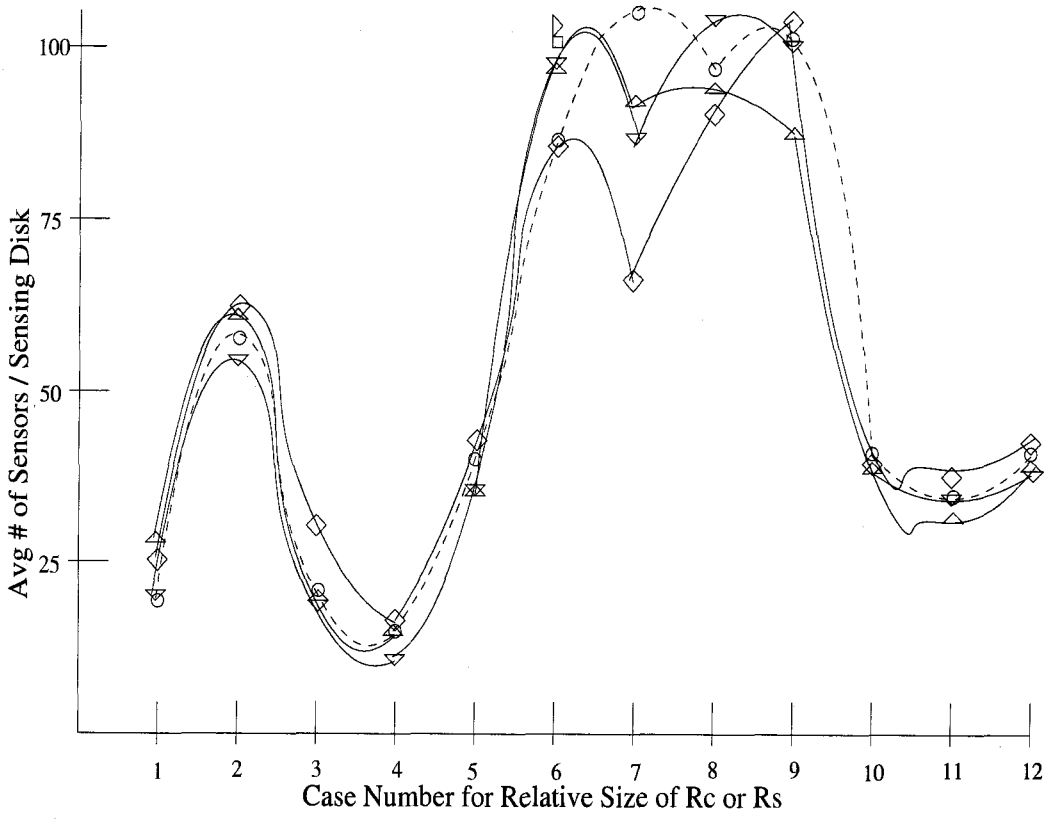


Figure 8.2: Query Region Sensors per MCSC Sensor for Various Relative Sizes of R_C or R_S



Case Number Key

1. Rc >= Rs (all Rc equal)
2. Rc >= Rs (all Rs equal)
3. Rc >= Rs (Rc, Rs unequal)
4. Rc > Rs (all Rc equal)
5. Rc > Rs (all Rs equal)
6. Rc = Rs (all Rc, Rs equal)
7. Rc < Rs (all Rc equal)
8. Rc < Rs (all Rs equal)
9. Rc < Rs (Rc, Rs unequal)
10. all Rc equal
11. all Rs equal
12. Rc and Rs unequal

Algorithm Key

Alg. SHID
Alg. SHRS
Alg. MHID
Alg. MHRS
Alg. MCSC
Rule k

Figure 8.3: Average Number of Sensors per Sensing Disk for Various Relative Sizes of R_C or R_S

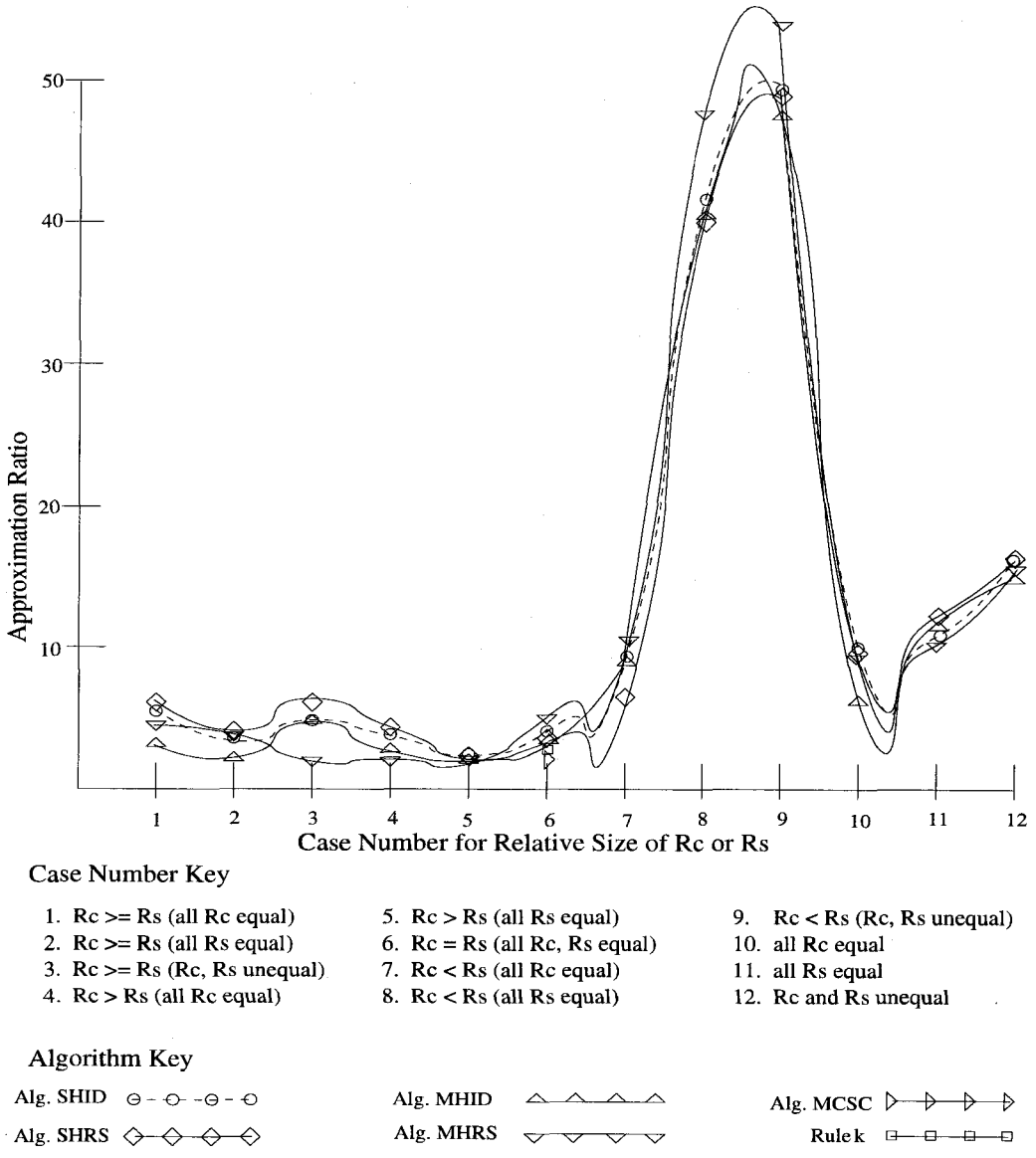
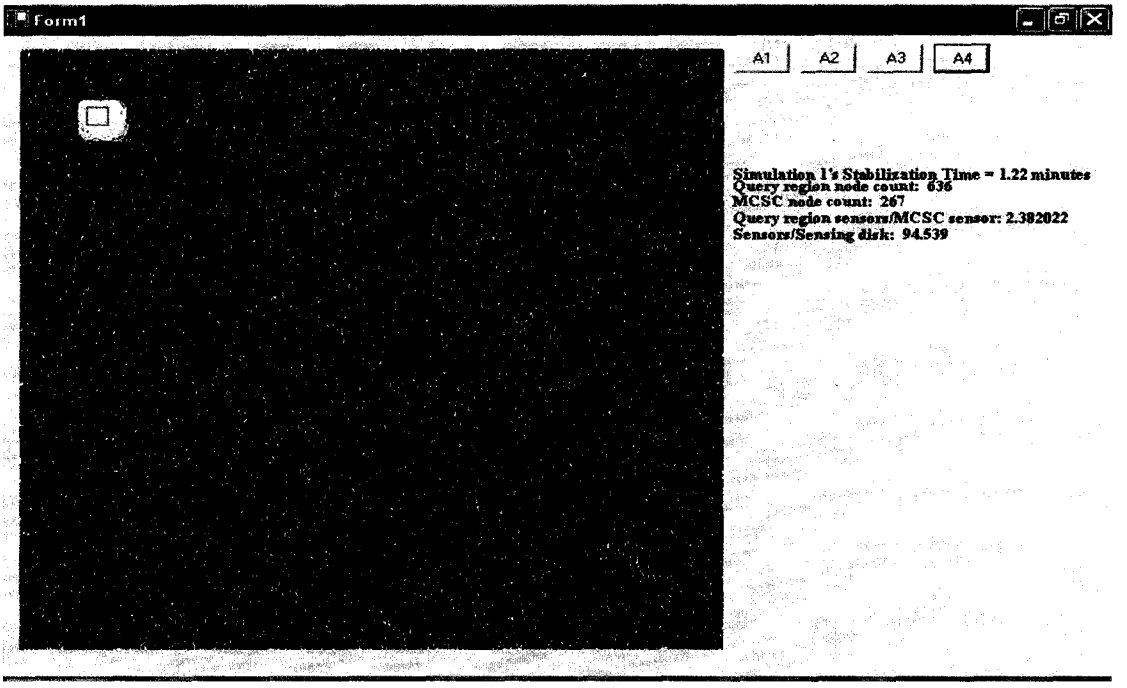
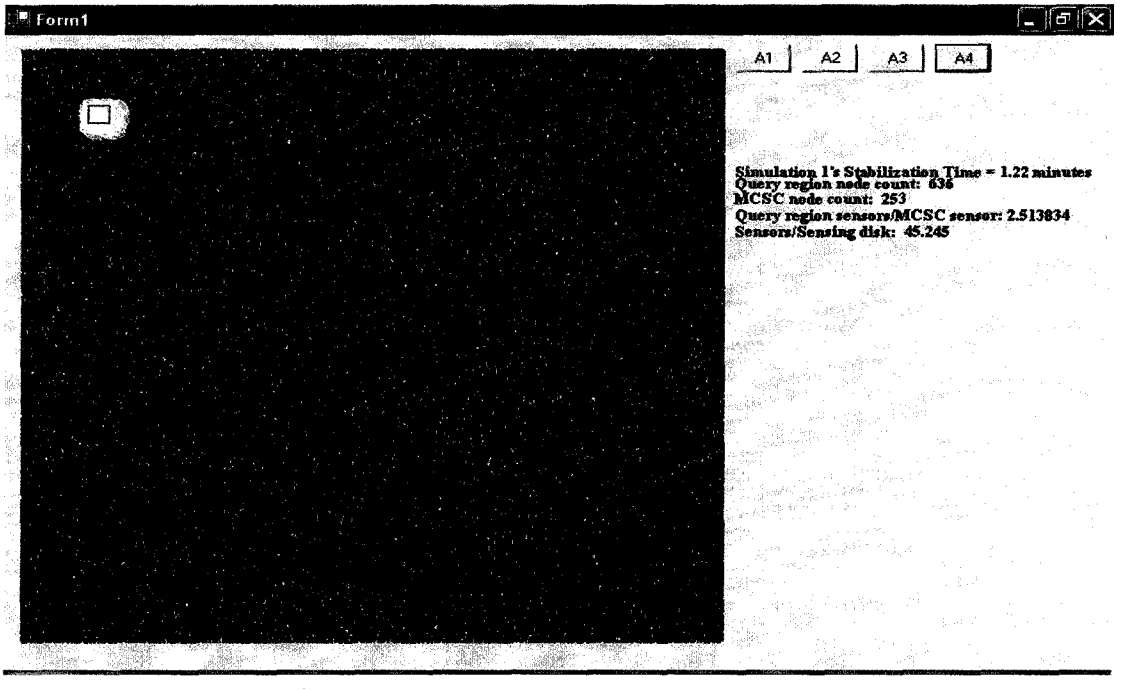


Figure 8.4: Approximation Ratios of all Algorithms for Various Relative Sizes of R_C or R_S

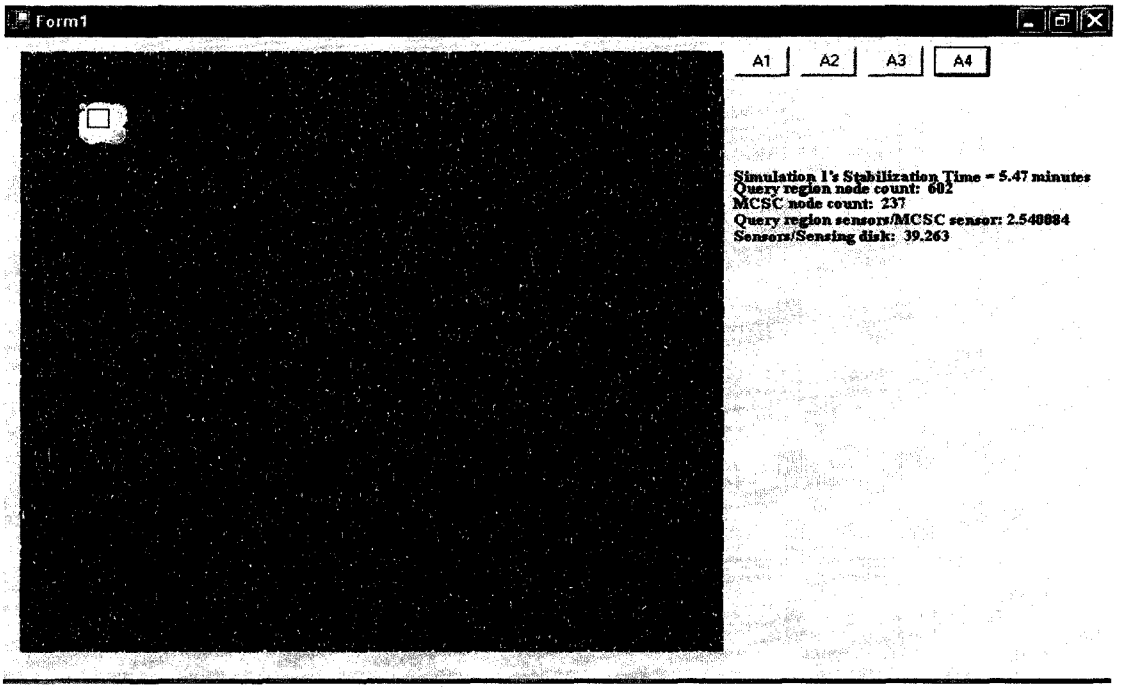
8.4 Screenshots



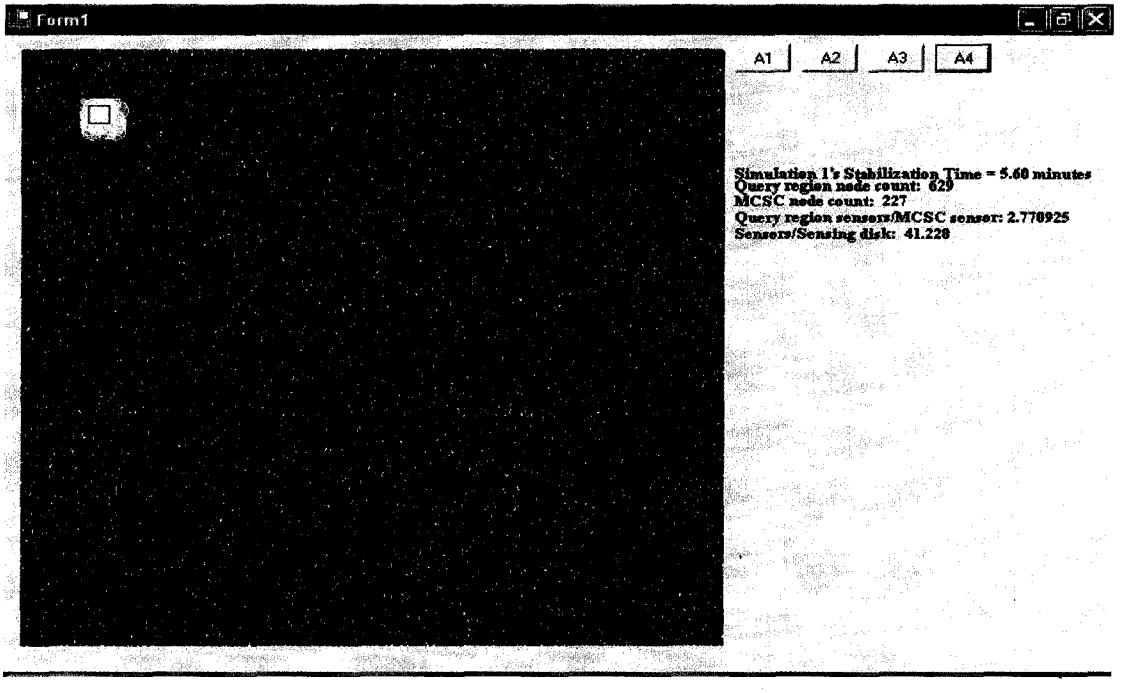
(a) Screenshot of Simulation of SHID with various sensing/communication radii



(b) Screenshot of Simulation of SHRS with various sensing/communication radii



(c) Screenshot of Simulation of MHID with various sensing/communication radii



(d) Screenshot of Simulation of MHRS with various sensing/communication radii

Figure 8.5: Screenshots of Simulations of Algorithms SHID, SHRS, MHID and MHRS with various sensing and communication radii

CHAPTER 9

CONCLUSION AND FUTURE RESEARCH

We started this research by studying wireless networks and self-$*$ systems. Both are very active research areas to make ubiquitous computing a reality. Our main focus was wireless sensor networks which already have numerous applications in vast and varying fields.

From its nature, sensor networks have various constraints which are well-known. The most significant issue within the scope of sensor networks is energy. Designing energy efficient, as well as reliable and scalable, sensor network protocols is highly desirable, yet very challenging. Considering the size, frequency of topology changes, and energy level changes, it is essential for sensor networks to have the properties of self-organizing, self-configuring, and self-healing.

The main goal of our research was to design a completely distributed self-$*$ query region covering sensor network. Assuming the most general cases of nodes' power limitations, we presented two local, and two multi-hop, distributed, scalable, and self-$*$ solutions to the minimum query region connected cover problem which can cope with variable sensing and transmission ranges. We demonstrated how these solutions are self-organizing and self-healing as well. They are also self-contained because once faults occur, they can be corrected within their neighborhood. Our solutions use 2-hop locality in Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$, and H-hop locality in Algorithms $\mathcal{MHID}$and $\mathcal{MHRS}$. In Algorithms $\mathcal{SHID}$and $\mathcal{SHRS}$, nodes acquire 2-hop information by reading neighbors' shared variables. Although

multi-hop communication is used in Algorithm $\mathcal{MHID}$and $\mathcal{MHRS}$, it is used only to check the coverage and connectivity conditions of chosen nodes, and the entire network topology is still unknown to each node. In that sense, we can still consider these algorithms as local algorithms. In addition, our solutions can easily be transformed to conform to power-aware systems by modifying an UID-based to a R_S-based algorithm. Similarly, instead of eliminating nodes based on the value of their identifiers or their sensing radius, nodes can be eliminated based on the state of their batteries, that is, selecting only nodes with strong energetic capabilities in the final cover. However, it is possible that competing nodes may have equal energy resources, since they may execute the same pattern of actions. Only in this case, the identifiers would be used to break the symmetry.

We formally proved the self-* properties of our solutions. Moreover, we have conducted extended simulations using the measures of stabilization time, the number of nodes in the final cover, the number of query region sensors covered per MCSC sensor, and the average number of sensors within a sensor's sensing disk. Our experiments demonstrated that under certain conditions, our proposed algorithms perform better than the self-stabilizing algorithms proposed in [24, 23]. Furthermore, they also produce a cover set that is connected and completely covers query regions at all ranges of size of sensing and transmission radii.

The proposed algorithms are also extended studies of the locality, in that multi-hop information exchange can be used to produce a better approximation to an optimal cover set. Although we proved that multi-hop communications greatly improve the result of the minimum connected cover problem, sensors consume the most energy during intercommunication between sensors. Therefore, it is crucial to find an optimal hop count so that a network life time is prolonged by compromising the optimality of the minimum connected cover set. The trade-off between the energy consumption by communication vs. the energy

saving by reducing the number of active nodes would be an interesting topic for future research.

The area of connected coverage in sensor networks still raises a broad class of challenges. Since the coverage characterizes the monitoring quality provided by a sensor network, different applications may require different degrees of sensing coverage. Similarly, for the purpose of fault-tolerance to network failure, such as packet loss, higher degree of connectivity may be required. In general, this is known as the k-connected k-coverage problem, and has been studied in fault-free environments in [68, 69]. However, an interesting open issue would be to address this problem in fault-prone environments, and in its generalized form: self-stabilizing k-coverage k-connectivity of query regions. Mobile sensor networks is another possible future topic. Assuming that sensors have some mobility, finding self-$*$ solutions for the minimum connected cover problem would be an interesting area of research. However, mobility would probably consume more energy, so it would be quite challenging to design energy-efficient protocols in such networks.

BIBLIOGRAPHY

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

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

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

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

[5] Hetergeneous Sensor Networks. Intel Corporation. http://www.intel.com/research/exploratory/heterogeneous.htm.

[6] 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.

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

[8] E Anceaume, M Gradinariu, and M Roy. Self-organizing systems case study: peer-to-peer networks. In *DISC03 Distributed Computing 17th International Symposium, Short papers proceedings*, pages 1–8, 2003.

[9] T Angskun, G Fagg, G Bosilca, J Pjesivac-Grbovic, and J Dongarra. Self-Healing Network for Scalable Fault Tolerant Runtime Environments. In *DAPSYS 2006, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems*, pages 21-23, September 2006.

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

[11] A Arora and MG Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

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

[13] S Asami, N Talagala, and D Patterson. Designing a self-maintaining storage system. In *Proceedings of the Sixteenth IEEE Symposium on Mass Storage Systems*, pages 222–233, March 1999.

[14] 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.

[15] J Beauquier, S Delaët, S Dolev, and S Tixeuil. Transient fault detectors. In *(DISC'98) Proceedings of the Twelfth International Symposium on DIStributed Computing*, number 1499, pages 62–74. Springer-Verlag, 1998.

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

[17] D Carman, P Kruus and B Matt. Constraints and Approaches for Distributed Sensor Network Security. *NAI Labs Technical Report*, September 1, 2000.

[18] A Cerpa and D Estrin. Ascent: Adaptive self-configuring sensor networks topologies. In *INFOCOM02 Proceedings of the Conference on Computer Communications*, Jun 2002.

[19] 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.

[20] 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.

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

[22] F Dai and J Wu. On Constructing k-Connected k-Dominating Set in Wireless Networks. *IPDPS'05 19th IEEE International Parallel and Distributed Processing Symposium.* page 81a, 2005

[23] AK Datta, M Gradinariu, and R Patel. Distributed self-$*$ minimum connected covering of a query region in sensor networks. In *ISPAN '05*, pages 448–453, 2005.

[24] AK Datta, P Linga, M Gradinariu, and P Raipin-Parvedy. Self-* distributed query region covering in sensor networks. In *SRDS05 24th IEEE Symposium on Reliable Distributed Systems*, pages 50–59, Oct 2005.

[25] X Défago and A Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *POMC02 Workshop on Principles of Mobile Computing*, 2002.

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

[27] 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.

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

[29] 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.

[30] S Dolev and T Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 1997.

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

[32] Roy Friedman, Maria Gradinariu, and Gwendal Simon. Locating cache proxies in manets. In *MobiHoc*, pages 175–186, 2004.

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

[34] GR Ganger, JD Strunk, and AJ Klosterman. Self-* storage: Brick-based storage with automated administration. Technical Report CMU-CS-03-178, Carnegie Mellon University, Aug 2003.

[35] VK Garg and JE Wilkes. *Wireless and personal communications systems.* Prentice Hall, 1996.

[36] 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.

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

[38] 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.

[39] J Heidemann and R Govindan. An Overview of Embedded Sensor Networks. *Technical Report ISI-TR-2004-594, USC/Information Sciences Institute*, November, 2004. http://www.isi.edu/ johnh/PAPERS/Heidemann04a.html.

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

[41] 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.

[42] 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, April 2005.

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

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

[45] 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.

[46] 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.

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

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

[49] D Peleg. *Distributed Computing A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

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

[51] R Prakash, N Shivaratri, and M Singhal. Distributed dynamic fault tolerant channel allocation for mobile computing. *IEEE Transactions on Vehicular Technology*, 48(6), Nov 1999.

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

[53] 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.

[54] JD Strunk and GR Ganger. A human organization analogy for self-* systems. Technical report, FCRC Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems In conjunction with Federated Computing Research Conference, Jun 2003.

[55] D Tian and ND Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA02 Proceedings of the First Workshop on Sensor Networks and Applications*, pages 32–41, Sep 2002.

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

[57] 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.

[58] 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.

[59] M Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, Sep 1991.

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

[61] 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.

[62] 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.

[63] Y Xu, J Heidemann, and D Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom02 Proceedings of the Seventh Annual Inernational Conference on Mobile Computing and Networking*, pages 70–84, 2001.

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

[65] F Ye, G Zhong, J Cheng, S Lu, and L Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *ICDCS03 Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 1–10, 2003..

[66] H Zhang and A Arora. GS3: Scalable self-configuring and self-healing in wireless networks. In *PODC02 Proceedings of the Twentyfirst Annual ACM Symposium on Principles of Distributed Computing*, 2002.

[67] 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.

[68] Zongheng Zhou, Samir R. Das, and Himanshu Gupta. Connected k-coverage problem in sensor networks. In *ICCCN*, pages 373–378, 2004.

[69] Zongheng Zhou, Samir R. Das, and Himanshu Gupta. Fault tolerant connected sensor cover with variable sensing and transmission ranges. In *SECON*, 2005.

VITA

Graduate College
University of Nevada, Las Vegas

Ai Yamazaki

Local Address:
4248 Spencer st. apt.230
Las Vegas, NV 89119

Home Address:
296 Yubure Yamakita-machi Ashigarakami-gun
Kanagawa, Japan 258-0123

Degrees:
Bachelor of Science, Hotel Administration, 2001
University of Nevada, Las Vegas

Bachelor of Arts, Computer Science, 2005
University of Nevada, Las Vegas

Thesis Title: Self-* Distributed Query Region Covering in Sensor Networks

Thesis Examination Comittee:
Chairperson, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. John T. Minor, Ph.D.
Committee Member, Dr. Yoohwan Kim, Ph.D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.