Self-* distributed query region covering in wireless sensor networks

Preethi Linga
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/1630

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
SELF-DISTRIBUTED QUERY REGION COVERING IN WIRELESS SENSOR NETWORKS

by

Preethi Linga

Bachelor of Science
Andhra University, India
1999

A thesis submitted in partial fulfillment of the requirements for the

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

Graduate College
University of Nevada, Las Vegas
May 2004

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The Thesis prepared by

PREETHI LINGA

Entitled

SELF-* DISTRIBUTED QUERY REGION COVERING IN WIRELESS SENSOR NETWORKS

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

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Co-Chair

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative
ABSTRACT

Self-* Distributed Query Region Covering in Wireless Sensor Networks

by

Preethi Linga

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

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

Late Mark Weiser envisioned the next (i.e., the post-PC) era of computing, called *ubiquitous computing*. In ubiquitous world, we expect to see thousands of invisible computing devices used per person, maybe, even in a household. Recent technological advances in microsensors, wireless networking and communications, and embedded processing made it possible. We can now build ad-hoc networks composed of a large number of low-cost, low-power, and small sensor nodes. The sensor networks are expected to be very large. In many applications, they will be densely deployed. These networks are energy constrained. The topology also is expected to change very frequently. Considering all the constraints discussed above, the sensor network must be *self-configuring* and *self-reconfiguring* or *self-healing*.

In sensor networks, *queries* are sent from some devices (could be a PDA, a laptop, or any computer) to sense some data/events over some time period and a geographical region, called *query region*. The query region is usually a subset of the total region covered by all the sensors in the network. Since the sensors are usually densely deployed, there are usually
a lot more sensors than required to process a given query. One possible way to minimize usage of energy is not to keep all sensor nodes fully active all the time. Some of them can be put to passive mode some times while others are active in sensing the data in the environment. The above scenario is modeled as an optimization problem in sensor networks, called the connected sensor cover problem. Given a query over a sensor network, select a minimum or nearly minimum set of sensors, called connected sensor cover, such that the selected sensors cover the query region, and form a connected network among themselves. In its general form, this problem is known to be NP-hard.

In this thesis research, we design the first, fully distributed, strictly localized, scalable, and self-* solution to the connected sensor cover problem. The Self-* concept has been used to include many fault-tolerant properties like self-configuring, self-reconfiguring/self-healing, etc. We will present a self-stabilizing solution and show that this solution is both self-configuring and self-healing. In a self-stabilizing system, every computation, starting from an arbitrary state, eventually reaches a state which satisfies the specification. Nodes achieve the global objective by using only local computations. Local algorithm based sensor networks are more robust, and scale better. The proposed solution is space optimal in terms of number of states used per node. Although the solution is not optimal in terms of the number of nodes in the cover set, but the initial simulation results show that our solution was able to eliminate a high degree of redundant nodes. So, the results look good. Another feature of the proposed algorithm is that the faults are contained only within the neighborhood of the faulty nodes.
TABLE OF CONTENTS

ABSTRACT ................................................................. iii

ACKNOWLEDGMENTS .................................................. viii

CHAPTER 1 INTRODUCTION ........................................... 1
  Contributions ......................................................... 3
  Outline of the Thesis ............................................... 4

CHAPTER 2 SELF-* SYSTEMS ....................................... 5
  Overview ............................................................... 5
  Ubiquitous/Pervasive Computing ................................. 10
  Self-stabilizing Systems ............................................ 13

CHAPTER 3 WIRELESS NETWORKS ............................... 16
  Mobile Wireless Networks ......................................... 16
    Infrastructured/Cellular Wireless Networks ................. 17
    Infrastructureless/Ad Hoc Wireless Networks ............. 18
  Wireless Sensor Networks ........................................ 19
    Overview ............................................................ 20
    Architecture and Applications .................................. 22
    Power Awareness .................................................... 26
    Data Dissemination ............................................... 27
    Data Fusion ........................................................... 32
    Routing Protocols ................................................ 34
    Localization Systems ............................................. 40
    Time Synchronization ............................................. 41
    Other Problems ..................................................... 42

CHAPTER 4 CONNECTED COVER .................................. 44
  Motivation ............................................................ 45
  Related Work ......................................................... 46
  Preliminaries ........................................................ 51
    Model ................................................................. 51
    Self-stabilizing Program ........................................... 54
    Problem Description .............................................. 55
  Connected Sensor Cover Algorithm .............................. 59
    Approach to the Solution ........................................ 61
    Normal Behavior .................................................. 63
    Faults and Recovery .............................................. 70
  Correctness of the Solution ....................................... 72

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ACKNOWLEDGMENTS

I am grateful to my thesis advisor, Dr. Ajoy K. Datta for chairing my committee and
advising this work. I am indebted to him for his whole-hearted support, encouragement,
enthusiasm, and inspiration throughout my graduate study, and most importantly, his faith
and belief in me. I owe a lot to the co-chair of the examination committee, Dr. Maria
Gradinariu at IRISA/Universite Rennes 1, France, for her numerous contributions in this
research — pointing out to us a published much weaker solution to an important problem,
helping and guiding me throughout the work, and her patience even when I tried to rush.

Special thanks to Philippe Raipan-Parvédy at IRISA/Universite Rennes 1, France for
simulating the algorithm and sending us the results. Philippe’s simulation gave us a lot of
confidence. I am also grateful to Dr. John Minor, Dr. Wolfgang Bein, Dr. Venkatesan
Muthukumar, and Dr. Emmanuelle Anceaume for their participation in my committee.

This thesis is dedicated to my parents for their love, faith and support. My special
gratitude goes to my sister Swathi, and my brother-in-law Satish for their support and
encouragement without whom I would not have made so far. I am very fortunate to have a
wonderful set of friends, all of whom made my study at UNLV enjoyable and memorable.
CHAPTER 1

INTRODUCTION

After spending the first era of computing with mainframes, we are now in the era of personal computing. The next wave, the third era of computing was visioned by Late Mark Weiser. In 1988 at the Computer Science Lab at Xerox PARC, he articulated the next age of computing, called ubiquitous computing [145], also called calm technology [149]. Later, it was given another name, called pervasive computing [120]. In ubiquitous world, we expect to see thousands of invisible computing devices used per person, maybe, even in a household.

How can we make this ubiquitous environment a reality? Recent technological advances in microsensors, wireless networking and communications, and embedded processing made it possible. We can now build ad-hoc networks composed of a large number of low-cost, low-power, and small sensor nodes. These ad-hoc wireless sensor networks [48] have applications everywhere — military, business, commercial, health, and home. Although sensor networks have some similarities with wireless ad-hoc networks (like MANETs) and mobile cellular networks, there are enough differences which demand seeking new sets of protocols. Today, networked sensors can be built using commercial components off the shelf. A research team at the University of California at Berkeley is attempting to design a system within a size of a few cubic millimeters [113]. Top silicon companies, like Intel Corporation [82] are also in the business of manufacturing such devices. DARPA has initiated a research, called

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Network Embedded Systems Technology (NEST) [108].

The sensor networks are expected to be very large. In many applications, they will be densely deployed. These networks are energy constrained. Not only the sensors have limited battery power, it is extremely difficult if not impossible to replace the battery. They may be deployed in inaccessible terrains or disaster areas. So, it is very important to design energy efficient sensor networks to enable untethered and unattended operation for an extended period of time. The topology may change very frequently due to various reasons, like position, reliability, available energy, malfunctioning, etc. Thus, designing wireless sensor networks is challenging.

Deploying pre-configured network of a huge number of sensors is impractical. Expecting to be able to manually maintain that size of a network is absurd. Considering all these constraints, the sensor network must be self-configuring and self-maintaining or self-healing. The term Self-* has been used to include all the properties like self-organizing, self-configuring, self-healing, etc. In this thesis, we will present a self-stabilizing solution to an important energy saving problem in sensor networks. Then we will show that this solution can also be considered as a self-* solution. In a self-stabilizing system, every computation, upon starting from an arbitrary state, eventually reaches a state from where the computation satisfies the specification. The paradigm of self-stabilization, introduced by Dijkstra in 1974 [39], is considered to be the most unified strategy to design fault-tolerant systems. Although it was intended to handle transient faults, research has shown that all types of faults can be dealt with in a stabilizing manner.

In sensor networks, queries are sent to sense some data/events over a query region. Considering the limited energy available, designing an energy saving query response protocol
for sensor networks is very important. A new problem, called connected sensor cover was introduced in [63] to model the query response system. The problem can be informally defined as follows: Given a query over a sensor network, select a minimum or nearly minimum set of sensors, called connected sensor cover, such that the selected sensors cover the query region, and form a connected network among themselves. In its general form, this problem is known to be NP-hard [63, 93].

1.1 Contributions

This thesis research involves two main topics: design of self-* systems and design of energy-efficient protocols for sensor networks. Many terms like self-organizing, self-maintaining, and self-healing, etc. have been introduced in the literature in the last few years. The first contribution of this research is writing a survey of the self-* systems which includes the above systems and self-stabilizing systems. We discuss ubiquitous/pervasive computing, IBM’s autonomic computing, and recovery-oriented computing and self-repairing computers. We also briefly summarize the main results of the self-stabilizing systems.

The second contribution of this thesis is an extensive study of various aspects of wireless sensor networks. We report the current results of many important problems (like data dissemination, data aggregation, localization systems, time synchronization) in this area. We also make an attempt to connect the self-* systems and wireless sensor networks.

The third and main contribution of this research is to design a self-* power-efficient solution to the connected sensor cover problem. This is the first localized distributed solution to the connected sensor cover problem. The proposed solution is also the first self-* protocol of the problem. By localized, it means that sensor nodes communicate only with their neighbors. Localized solutions in large networks are desirable due to their high relia-
bility and scalability. We implemented the self-* properties by using the self-stabilization paradigm. Our solution can handle different types of faults including node and link (wireless communication) failures, change of power level, and memory corruption.

1.2 Outline of the Thesis

We start with a survey of self-* systems in Chapter 2. This chapter includes descriptions of many types of fault-tolerant systems, all under the common framework of self-*. We briefly describe the basic ideas of mobile wireless networks including cellular networks and mobile ad-hoc networks (in Section 3.1 in Chapter 3). In Section 3.2 in Chapter 3, we present a detail survey of wireless sensor networks. Chapter 4 includes the main contribution of this thesis. In that chapter, we introduce the connected sensor cover problem, and present a self-stabilizing solution to the problem, followed by its proof of correctness. Discussion about the complexity of the algorithm, simulation results, and other properties are also included in the same chapter. In Section 4.8, we will give some ideas to extend this research. Finally, in Chapter 5, we present some concluding remarks and a few future directions.
CHAPTER 2

SELF-* SYSTEMS

A number of definitions have been proposed in the literature to capture the meaning of distributed systems. A distributed system [134] can be defined as an interconnected collection of autonomous computers, processes, or processors (also, called nodes). Tanenbaum [133] added one extra condition to the above definition — the existence of the collection of nodes must be transparent to users of the system. 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 [110]. Many authors (e.g., Tanenbaum in [133]) made an attempt to distinguish between distributed systems and computer networks. However, the difference between the two systems in modern computing is very subtle.

One of the main topics of research in this thesis is self-* systems. In the following, we start with an overview of self-* systems (Section 2.1). We describe many terms currently being used in the broad area of fault-tolerant computing.

2.1 Overview

Software systems are being used for almost all business-critical applications. Thus, the availability of these systems is extremely important. The system must be able to adjust
to different inputs, adapt to all possible changes of the environment, and handle different faults. The different concepts or terms encapsulated in *self-* have been introduced to characterize different ways of detecting, adjusting, and recovering from the above situations. Unfortunately, these terms have not been formalized in a coherent manner. In the following, we will informally describe these concepts with examples from the current literature. In Section 2.3, we give an overview of the concept of self-stabilization which has been an active area of research for more than twenty-five years.

In [53], *self-* has been used “to capture many recent buzzwords in a single meta-buzzword”. According to [53], a *self-* system should be self-configuring, self-organizing, self-tuning, self-healing, and self-managing. The following is from [130]: “Such research 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.” Human organization (specifically, corporate structure) was used as an analogy to explain the concept of *self-* in [130]. The authors explored the analogy by using several examples from the corporate world. Some of the examples are “worker/supervisor hierarchies”, “avoidance of micro-management”, “complaint-based tuning”, “risk analysis”, and “observe, diagnose, and repair loop”. The above characteristics make the human organizations (like corporate offices) self-organizing and resilient to failures. The insights gained from the human organizations combined with concepts from AI and storage systems were used to design *self-* storage systems [53]. The goals of the *self-* storage systems proposed in [53] were reduction of human administration, and increase of levels of reliability, availability, and performance.
A system is considered to be *self-configuring* if starting from an arbitrary starting state and an arbitrary input, the system will eventually satisfy the specification or start behaving properly. A similar concept, called *self-organizing* was formally defined in [5]. The concept was then applied to study peer-to-peer systems. A *self-healing* system automatically recovers from different perturbations and dynamic changes. A self-healing system can also be characterized as a *self-maintaining* system.

The Dynamic Reconfiguration subsystem (DRSS) [64] is a very well-known general architecture supporting the development of self-configuring and self-maintaining systems. A prototype of DRSS has been implemented using Microsoft’s .NET platform. DRSS was used to achieve fault-tolerance in Aladdin Home Networking Project [141]. The Aladdin project is an implementation of a highly reliable system infrastructure for connecting various in-home networks. The self-organization was formally defined based on the locality principle in [5]. Locality principle means that a node maintains only a limited amount of information with respect to a bounded set of nearby nodes. The authors used this new definition in peer-to-peer networks to reorganize links and spontaneously group nodes using some special criteria.

A self-configuring and self-healing algorithm for multi-hop wireless networks was presented in [158]. The algorithm self-configures nodes in a 2D plane into a cellular hexagonal structure satisfying certain properties. The algorithm also self-heals under various perturbations in dynamic networks.

Distributed self-configuring algorithms have been proposed in the field of robotics [36, 132, 139]. The problem in general in this field is for a system of multiple mobile robots to communicate with each other to form a geometric pattern. The algorithms are self-
configuring in the sense that they can start from any shape, but eventually after some finite moves will converge to a specified shape. The final patterns considered include circles, points on a polygon, straight lines, etc. Some algorithms considered robots with memory, i.e., they can remember all the steps taken in the past. Some algorithms considered oblivious robots. In [36], anonymous and oblivious robots were considered. Refer to [25] for a good survey on cooperative mobile robotics.

The number of computer devices is increasing very fast — at a compound rate of 38% [56]. Many of these computers are also interconnected. As more systems are becoming connected to a more diverse set of systems and environments, planned maintenance of computers are becoming more of an impossible task to manage. The phenomenal growth of computer systems and the invention of Internet has also made managing and maintaining computers too complex. This had made a big impact on the cost of maintaining computers. The cost of employing network administrators to keep the computers up and running has been rising. The following is from [56]: “According to a study published in March 2002 by researchers at the University of California at Berkeley, the labor costs outstrip equipment by factors of three to 18, depending on the type of system. And one third to one half the total budget is spent preventing or recovering from crashes. And no wonder: a system failure at a brokerage or credit-card authorization center can run up millions of dollars per hour in lost business.” Hardware and software faults, and human errors are unavoidable. So, to meet these challenges, recovery based approach to achieving fault-tolerant systems is the way to follow. Systems should be self-managing in the sense that all tasks in all phases in the life cycle of the system are automatic. In the following, we will discuss two suggested approaches to solve the system maintenance and management problem: autonomic
computing and recovery-oriented computing.

IBM's solution to the above unmanageable task is the introduction of the new model of computing, called autonomic computing [79]. 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." An autonomic system is defined by eight characteristics as follows. (1) It needs to "know itself". (2) It must configure and reconfigure itself under varying (and in the future, even unpredictable) conditions. (3) It never settles for the status in quo — it always looks for ways to optimize its workings. (4) It must perform something akin to healing — it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction. (5) It must be an expert in self-protection. (6) It must know its environment and the context surrounding its activity, and act accordingly. (7) It cannot exist in a hermetic environment. (8) It will anticipate the optimized resources needed while keeping its complexity hidden. A special issue of IBM Systems Journal has been dedicated to autonomic computing [80]. We quote the following from the cover page of the special issue "The development of autonomic computing will make systems capable of self-configuring, self-healing, self-optimizing, and self-protecting, analogous to the abilities of living organisms with autonomic nervous systems".

A new approach to building highly reliable systems, called recovery-oriented computing was introduced in [51, 109]. They called such systems self-repairing computers. The group at Berkeley and Stanford are working on applying this concept to designing highly-dependable Internet services. The basic approach used in their work is based on failure recovery. Some important characteristics of recovery-oriented computing have been iden-
tified in [109]. They are “isolation and redundancy”, “system-wide support for undo”, “integrated diagnosis support”, “online verification of recovery mechanisms”, “design for high modularity, measurability, and restartability”, and “dependability/availability benchmarking”.

2.2 Ubiquitous/Pervasive Computing

The term ubiquitous computing was coined by late Mark Weiser at Xerox Palo Alto Research Center (PARC) to describe a vision of future technology that would always be available, often monitoring or anticipating the user’s needs, even when the user was not explicitly aware of the technology [145]. The following quotation is from his paper [145]: “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.” This new era of computing is based on two key challenges: invisible computing and calm technology [149]. The main slogan for this computing is “invisible everywhere” approach [146]. Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user. By invisible, it means that the tool does not intrude on our consciousness; we focus on the task, not the tool. The writing or literacy technology may be the oldest technology in this category of invisible computing. When we see something written, we just use it, but do not feel it. The electricity is another example. Electric motors or some other devices are everywhere. Eyeglasses are a good tool — we look at the world, not the eyeglasses. The canes used by blind people are good examples too. They feel the street, not the canes. The goal of the calm technology [149] is to send information in a calm manner. However, information technology is more often the enemy of calm. The examples are pagers, cellphones, TV,
radio, etc. The calm technology engages, and moves back and forth between two things: peripheral (or sensory) processing and center of processing. A good example is driving a car. When we drive a car, we look at the road, we hear the noise in the car from fellow passengers, radio, etc., but we do not hear the noise of the engine. In this case, the center of processing is looking at the road, and the periphery is the noise of the engine. If something goes wrong in the engine, the two processing types will swap between them — the engine will be the main focus of attention.

Unlike virtual reality, ubiquitous computing endeavors to integrate information displays into the everyday physical world. Unlike PDA's, ubiquitous computing envisions a world of fully connected devices, with cheap wireless networks everywhere. It postulates that we need not carry anything with us since information will be accessible everywhere. Unlike the intimate agent computer that responds to one's voice and is a personal friend and assistant, ubiquitous computing envisions computation primarily in the background where it may not even be noticed [147].

The emergence of network enabled devices and the promise of ubiquitous network connectivity have made the development of pervasive computing environments an attractive research goal [117]. Ubiquitous computing is now also called pervasive computing [120]. Pervasive computing is about making our lives simpler through digital environments that are sensitive, adaptive, and responsive to human needs. In a way, pervasive computing subsumes mobile computing as well as distributed systems. Mobile computing goal of “anytime anywhere” connectivity is extended to “all the time everywhere” by integrating pervasive-ness support technologies such as interoperability, scalability, smartness, and invisibility.

New hardware systems design for ubiquitous computing has been oriented towards ex-
Experimental platforms for systems and applications of invisibility. The initial incarnation of ubiquitous computing was in the form of “tabs”, “pads”, and “boards” built at Xerox PARC, 1988-1994. In hardware, we have mobile devices, sensors, and even smart appliances. Sensors can use the Piconet module to enable wireless connectivity [48]. Active badges [143], the ParcTab [1], the ORL location system [144], the PinPoint Positioning Systems (LPS) [150], and the Factoid [49] are all examples of this class of networked sensors (discussed in detail in Section 3.2). Supporting software technologies include digital signal processing and object-oriented programming.

Ubiquitous computing offers a framework for new and exciting research across the spectrum of computer science. The challenge is to create a new kind of relationship of people to computers, one in which the computer would have to take the lead in becoming vastly better at getting out of the way so people could just go about their lives. The most ubiquitous current informational technology embodied in artifacts is the use of written symbols, primarily words, but including also pictographs, clocks, and other sorts of symbolic communication [148].

The ubiquitous computing will bring information technology beyond the big problems like corporate finance and school homework, to the little annoyances like “Where are the car keys?”, “Can I get a parking place?”, and “Is that shirt I saw last week at Macy’s still on the rack [149]?”. This new trend of computing seems likely to provide a framework for interesting and productive work for many more years or decades, but we have much to learn about the details.
2.3 Self-stabilizing Systems

The concept of self-stabilization has been known for about thirty years as a paradigm of designing fault-tolerant systems. This concept was introduced to computer science by Dijkstra [40, 39] and later strongly endorsed by Lamport [94]. The idea of self-stabilization was used in other areas (such as numerical analysis, control theory, systems science, etc.) long before Dijkstra coined the word “self-stabilization”. Many definitions of self-stabilization exist in the literature, and unfortunately, the stabilizing research community did not converge on a single definition. One widely accepted definition is as follows: A self-stabilizing system, regardless of its initial state, converges in finite time to a set of states that satisfy its specification. It can also be defined with respect to behavior instead of state as follows: A self-stabilizing system, starting from an arbitrary state, reaches a state in finite time such that it starts behaving according to its specification. In this section, we will give an informal overview of some aspects of stabilization. Our goal is not to give a comprehensive summary of the whole area. Readers can refer to [70] for an almost current on-line bibliography of stabilizing literature, [41] for the only book on this topic, [50, 58, 122] for surveys of the area, and [7, 10] for an introduction to the concept of self-stabilization.

Although Dijkstra's work [39] did not mention of any application of self-stabilization to fault-tolerance, there has been a lot of research on this topic. The dissertations [6, 89] are excellent sources on this topic. The self-stabilization was defined in terms of closure and convergence in [6, 8]. Closure refers to the property which requires that during all system executions, the system stays within some set of legal or desirable set of states unless a fault occurs. Convergence requires the system to reach a legal state from any arbitrary (possibly illegal) state in finite steps. A system is self-stabilizing if it satisfies both closure and
convergence properties. In [6, 8], a comprehensive study of different types of faults (such as crash, stuck-at, fail-stop, omission, timing, performance and Byzantine) and how they are accommodated in their definition of stabilization (in terms of closure and convergence) was included. The first formal definition of fault-tolerance was given in [8]. The results like [6, 8] and some others (refer [70]) in subsequent years establish the fact that self-stabilization is the most unified strategy of achieving fault-tolerance in distributed systems. Previous attempts were specific to some technologies, architectures, and applications. In [89], it was shown that a fault-tolerant program is a composition of a fault-intolerant program and a set of fault-tolerance components. A method for designing multi-tolerant programs (ones that tolerate multiple types of faults) was also presented in [89]. It was shown in [83] that a sequence of crashes can drive a protocol into arbitrary global states.

Self-stabilization has been extensively used in the area of network protocols. Numerous papers have been written on protocols like routing (including cut-through, wormhole), alternating bit, sliding window, session control, congestion control, connection management, high-speed networks, sensor networks, and max-flow computation. Refer to [41, 70] for the pointers. Many of these protocols also consider message losses and duplications, and node/link failures.

There exist many self-stabilizing distributed solutions for graph theory problems. Examples are different types of spanning trees, finding center and median, maximal matching, search structures, and graph coloring. Stabilization has been applied to solving many classical distributed algorithms. Examples include mutual exclusion, token circulation, leader election, synchronization and clocks, distributed reset, distributed diffusing computation, termination detection, and propagation of information with feedback. Problems have been
considered in different topologies (e.g., ring and tree) as well.

Numerous models have been considered in the literature. There exist several dimensions of the model, such as execution model (shared registers and message passing), fairness (weakly fair, strongly fair, and unfair), granularity of the atomic step (composite versus read/write atomicity), and types of daemons (central and distributed). Stabilization time complexity and space complexity have been two factors of the stabilizing solutions. Several optimal solutions have been proposed.

Proving stabilization program is quite challenging. Two techniques have been commonly used in the literature: convergence stair [60] and variant function [87] methods. Proof techniques for randomized algorithms, refer to [18, 44, 69, 75]. Many general methods of designing self-stabilizing programs have been proposed. We mention some of them here without any description: diffusing computation [9], silent stabilization [42], local stabilizer [3], local checking and local correction [12, 136], distributed program checking [14], counter flushing [137], window washing [32], self-containment [55], snap-stabilization [33], super-stabilization [43], power supply [2], and transient fault detector [17].
CHAPTER 3

WIRELESS NETWORKS

We will present various concepts and issues related with wireless sensor networks in this chapter. To understand the distinction between sensor networks and other types of wireless networks, in Section 3.1, we give a very brief overview of wireless networks.

3.1 Mobile Wireless Networks

In recent years, mobile computing has enjoyed a tremendous growth in popularity due to various technological advances and rapid development of small, inexpensive, and powerful mobile computing devices ranging from Personal Digital Assistants (PDAs), mobile phones, handhelds, and wearable computers to laptops. Due to the added facility of mobility in these computing devices, maintaining communication among the various types of mobile devices is critical and challenging. An overview of important issues of distributed computing in a mobile environment was given in [62]. With the development in wireless communication technologies in decades, wireless mobile units can communicate with each other in various ways. Mobile Wireless Networks can be classified into infrastructured (cellular) and infrastructureless (ad hoc) wireless networks [54]. Both aim to provide ubiquitous communications and computing environment where users are untethered from their information source.
3.1.1 Infrastructured/Cellular Wireless Networks

In an infrastructure/cellular wireless network, the mobile devices connect to each other through an access point, or use a more sophisticated intermediary such as a base station (gateway or router). Cellular networks divide the geographical area they serve into smaller regions, called cells. Each cell has a base station, also referred to as the mobile service station (MSS). Several mobile hosts (MH) may be present in a cell. The MHs can move from one cell to another [115]. All mobile nodes are one-hop away from a base station. Mobile nodes directly communicate with access points or base stations, and usually do not establish point-to-point connections with other mobile nodes. Access points are usually connected to the rest of the network or the Internet. Each access point has a coverage area which it is able to send signals to and receive signals from other mobile nodes. Nodes within the area of an access point are able to communicate directly with that access point. But, as the mobile node moves from the coverage area of one access point into that of another, a handoff occurs, where the node ceases communication with the old access point and begins communicating with the new access point. The handoff should be completely seamless so that the user is not aware of the fact that a transition in coverage areas has occurred. Typical examples of this kind of wireless networks 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. WAP products are typical examples of the commercial application for this type of wireless networks.
3.1.2 Infrastructureless/Ad Hoc Wireless Networks

Mobile users may need to communicate in situations where no fixed wired infrastructure is available. For example, a group of researchers en-route to a conference may meet at the airport and require to connect to the wide area network, students may need to interact during a lecture, or firemen need to connect to an ambulance en-route to an emergency scene. In such situations, a collection of mobile hosts with wireless network interfaces may form a temporary network without the aid of any established infrastructure or centralized administration. Such networks received considerable attention in recent years in both commercial and military applications, due to their attractive properties of building a network on the fly and not requiring any pre-planned infrastructure such as a base station or a central controller. Such an interconnection between mobile computers is called an ad-hoc network, in conformance with current usage within the IEEE 802.11 subcommittee. Communication in ad hoc networks is peer-to-peer as the mobile nodes communicate directly with one another. A good introduction to MANET and a nice collection of chapters on routing protocols in MANET can be found in [111].

In short, a MANET is a collection of wireless mobile nodes that can dynamically form a network to exchange information without any aid from a pre-existing fixed network infrastructure. This is a very important part of communication technology that supports truly pervasive/ubiquitous computing, because in many contexts, information exchange among mobile units cannot rely on any fixed network infrastructure but on the rapid configuration of wireless connections on the fly [131].

Mobile ad-hoc networks differ significantly from existing networks as follows:

1. All nodes can move.
2. The topology of interconnections may be quite dynamic. It may change rapidly and unpredictably over time.

3. Since every computer may not be within the communication range of every other computer, multiple hops may be needed. Hence, the nodes must serve as routers for other nodes in the network so that data packets can be forwarded to their destinations.

The set of applications for MANETs is diverse, ranging from large scale, mobile, and highly dynamic networks, to small and static networks that are constrained by power sources. Typical applications include commercial sector, military battlefield, civilian environments, emergency operations, and personal area network (PAN). Regardless of the attractive applications, the features of MANET introduce several challenges that must be studied before a wide commercial deployment can be expected that include routing, security, reliability, quality of service (QoS), internetworking, and power Consumption.

3.2 Wireless Sensor Networks

In this section, we will give an overview of the characteristics of sensor nodes and sensor networks, along with the issues/problems associated with the sensor networks. Our goal here is not to write a comprehensive survey of related work. We selected some key issues, which we describe briefly with some pointers to the current literature. In Chapter 2, we described various self-* properties. Sensor networks are expected to be self-configuring and self-organizing.
3.2.1 Overview

As the post-PC era emerges, several new niches of computer system designs are taking shape with characteristics that are quite different from traditional desktop and server regimes [73]. Recent technological advances such as development of MEMS microsensors [105], wireless networking and communications, electronics, and embedded processing made it possible to manufacture low-cost, low-power, small (in size) sensor node based ad-hoc networks.

A sensor network usually is composed of a scalable large number of nodes with highly constrained power sources. The nodes are typically static in nature. However, some or all nodes could be mobile. In many situations, the sensor nodes are deployed randomly in inaccessible terrains or disaster areas. The sensor networks should be self-organizing and self-healing (or fault-tolerant) (see Chapter 2). The routing algorithms should be energy-efficient. Considering all the above constraints, sensor networks seem to have some similarities with wireless ad-hoc networks (like MANETs) (Section 3.1.2) and mobile cellular networks (Section 3.1.1). However, many protocols suggested for the above two platforms may not be well suited for sensor networks for the following reasons [4, 154]:

**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 thousands or millions.

**Density:** Sensor nodes are expected to be densely deployed. The density can range from few sensor nodes to few hundred sensor nodes in a region, which can be less than 10 m in diameter. They can be very close or directly inside the area to be observed. They can be deployed inside a large machinery, at the bottom of an ocean, in a biologically or chemically hazardous area, in a battlefield, etc.
Topology Changes and Failures: The topology of sensor networks may change more frequently. Sensor nodes can be spreaded in as a mass or placed individually. They can be deployed in various ways, e.g., dropping from aircrafts, delivering in an artillery shell, rocket, or missile, and placing one by one using a robot or a human. The topology may change due to change of position, reachability (e.g., jamming, noise, obstacles, etc.), available energy, malfunctioning, etc. Sensor nodes can fail easily due to the low cost in manufacturing or environmental threats. They could also be destroyed by animals or vehicles.

Communication Method: Broadcast communication is typically used in sensor network, whereas most other ad-hoc networks use point-to-point communication.

Constraints: Constraints like power, computational ability, and memory are more stringent in sensor networks. Sensor networks use batteries which may last only a few days, whereas nodes in other wireless ad-hoc networks are usually powerful computers.

Bandwidth: The required bandwidth of sensor networks is low, of the order of 1 – 100 kb/s.

Despite all the above constraints, we desire a robust, long-lived sensor network out of such fallible, short-life sensor nodes [154]. Wireless sensor networks improve sensing accuracy by providing distributed processing of a vast information collected by the source nodes [68]. The sensor nodes are used to collect useful information such as acoustic, light, and seismic data/environment. The sensors are used as both data generators and routers. Networked sensors can aggregate such data to provide a rich, multi-dimensional view of the environment. Networking also improves the fault-tolerance. The sensor network can afford
to be more focused on some critical events detected by the source sensors. Sources are usually located where the environmental activities of interest are expected to take place. This kind of networks can also improve the remote access to sensor data by providing sink nodes which are connected to other networks such as the Internet. The sinks are basically monitoring terminals. They may be mobile PDAs, laptops, or static access points.

3.2.2 Architecture and Applications

The sensor networks are becoming very common platforms for various applications such as health, home, commercial, and military applications. Some specific applications include health (monitoring patients), environmental monitoring (e.g., traffic, habitat, security), industrial sensing and diagnostics (e.g., factories, appliances, managing inventory, monitoring product quality, monitoring disaster areas), infrastructure maintenance (e.g., power grids, water distribution, waste disposal), and battlefield awareness (e.g., military command, control, communications, computing, intelligence, surveillance, reconnaissance, and multitarget tracking). See [4] for a good survey and [124, 159] for the current trends in sensor networks. A very interesting article on the future of sensor networks was written by Pister [112]. His views there are along the characteristics of ubiquitous/pervasive computing Section 2.2.

WINS [114] is one of the early major projects on designing networks using embedded sensors. The WINS project was initiated at the University of California, Los Angeles by Pottie and Kasier [114] in 1993. The network, called wireless integrated network sensors (WINS) was designed to provide a distributed network and Internet access to sensors, controls, and processors deeply embedded in equipment, facilities, and the environment. This project used the advances in microsensor technology, low-power signal processing, low-power computation, low-cost wireless networking to produce a compact embedded sys-
tern at a cost much lower than conventional wireline sensor and actuator systems. WINS is a self-organizing and self-configuring system. The architecture of a WINS node (taken from [129]) is shown in Figure 3.2.1. The sensor needs to be continuously active. Once an event is detected, then the node identifies the event, determines if further processing is needed, and communicates with other WINS nodes if needed.

![Figure 3.2.1: The architecture of a WINS node.](image)

Today, networked sensors can be constructed using commercial components on the scale of a square inch in size and a fraction of a watt in power [73]. A novel system architecture of networked sensors was given in [73]. The authors of [73] designed a tiny microthreaded OS, called TinyOS [135] to provide the system software support to manage and operate this class of tiny smart devices. Five requirements for networked sensor systems were given in [73]. 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 Smart Dust project at
Berkeley [85, 113]. This project investigated the technological opportunities and challenges of designing networked sensors with limits on size and power resources. Their main goal is to design a device (called smart dust) with required sensing, communication, and computing hardware, along with a power supply, within a size of a few cubic millimeters. This processor is an ATMEL [11] 4 MHz, 8 bit micro-controller with 8 K bytes of program memory and 512 bytes of data memory. It also includes a radio with a single channel RF transceiver operating at 916 MHz and capable of transmitting at 10 Kbps using on-off-keying encoding [73, 151]. A consortium including U. C. Berkeley and others devised a second generation mote, called MICA [34, 76]. This new mote takes less power and is smaller in size.

Piconet [19] is another prototype embedded network. It was developed at the Olivetti and Oracle Research Laboratory, Cambridge. The Active Badge Location System [142] has studied the utility of networked sensors. A set of algorithms for establishing and maintaining (i.e., self-organization) wireless sensor networks was described in [129].

The article [82] describes the current research on heterogeneous sensor networks at Intel Corporation. An 802.11 mesh network comprised of high-end node, such as Intel XScale based nodes, is overlaid on a sensor network formed using Intel motes. The experimental results show that heterogeneous networks enhance overall performance. 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, for providing Internet access to park visitors, or for overall improvement of park management.

ZebraNet is a very recent and important project being conducted at The Princeton University [157]. Funded by a research grant from the National Science Foundation through their Information Technology Research (ITR) initiative, ZebraNet is a project to explore
wireless protocols and position-aware computation from a power-efficient perspective. Essentially ZebraNet is a power-aware wireless ad hoc sensor network, but with more serious bandwidth and computational needs than most prior sensor network research problems [97]. The ZebraNet Project is a good example for habitat applications of mobile sensor networks as this project is aimed for wildlife tracking. This system includes custom tracking collars (nodes) carried by animals under study across a large wild area [84]. The collars operate as peer-to-peer network to deliver logged data back to researchers. The collars include global positioning system (GPS), Flash memory, wireless transceivers, and a small CPU. The collars will also be fitted with a solar panel to recharge the battery during the day [123].

*Network Embedded systems Technology (NEST)* is a program funded by DARPA. There are many projects under NEST and their descriptions are available at [108]. 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 enable “fine-grain” fusion of physical and information processes. The aim is to develop sensor and information system technology and systems with application to battle space awareness, targeting, command and control, and the supporting infrastructure.

*Micro-Adaptive Multi-domain Power aware Sensors (µAMPS)* is a project being conducted at MIT. The goal of this project is to develop a framework for implementing adaptive energy-aware distributed microsensors. The µAMPS project focuses on innovative energy-optimized solutions at all levels of the system hierarchy, from the physical layer and communication protocols up to the application layer and efficient DSP design for microsensor nodes [102].
3.2.3 Power Awareness

Minimizing energy consumption is an important challenge in mobile networking. The source of energy for a node is most often an attached battery cell. Since the size of a cell is limited, the amount of available energy is also limited. Therefore, sensor network architectures and applications, as well as deployment strategies, must be developed with low energy consumption as one of the important requirements [128]. Every message sent and every computation performed drains the battery. Energy optimization, in the case of sensor networks, is much more complex, since it involves not only reducing the energy consumption of a single sensor node but also maximizing the lifetime of an entire network. The network lifetime can be maximized only by incorporating energy awareness into every stage of wireless sensor network design and operation, thus empowering the system with the ability to make dynamic tradeoffs among energy consumption, system performance, and operational fidelity [118].

The power consumption of each node in an ad-hoc wireless system can be divided according to functionality into: (1) the power utilized for the transmission of a message; (2) the power utilized for the reception of a message; and (3) the power utilized while the system is idle. This suggests two complementary levels at which power consumption can be optimized: (1) minimizing power consumption during the idle time and (2) minimizing power consumption during communication [96].

A recent and very important body of work concerns optimizing power consumption during idle time rather than during the time of communicating messages [28, 153]. The basic idea is that nodes do not need to be listening and consuming power when they are not involved in sending, forwarding, or receiving data. The PAMAS protocol applies this result at the MAC level [126]. Span [28] is a good example of such type of algorithms. It is
a distributed, randomized algorithm where nodes make local decisions on whether to sleep or to join a forwarding backbone as a coordinator.

Other way of optimizing power consumption is by reducing the overhead involved in routing protocols. Routing protocols for wireless ad hoc networks should be able to perform local collaboration to reduce bandwidth requirements [67]. Sensor networks contain too much data for an end-user to process. Therefore, automated methods of combining or aggregating the data into a small set of meaningful information is required. Localized distributed routing protocols are better when compared to globalized algorithms as there is no need to propagate any topology changes (that are very frequent in ad hoc networks). Low-Energy Adaptive Clustering Hierarchy (LEACH) is a clustering-based protocol that minimizes the energy dissipation in sensor networks. Some power aware routing protocols (including LEACH) for sensor networks are discussed in more detail in Section 3.2.6.

3.2.4 Data Dissemination

Data dissemination and data collection are very important tasks in sensor networks. The protocols for the above need to be energy-efficient because energy is considered to be the most crucial resource in sensor networks. It is extremely difficult, if not impossible to recharge batteries of thousands of devices in remote or hostile environments [88]. Studies [101] show that the energy consumption is dominated by the cost of transmitting and receiving messages. The importance of saving communication cost for sensor nodes is also supported by data from popular prototypes of sensor network devices, such as MICA2 [34]. As mentioned before, sensor nodes are subject to failures at any time. However, the data forwarding and monitoring should be available over long period of time [154].

Govindan, et. al. [48, 61] suggested creating ad-hoc smart environments based on sen-
sensor networks. The goal is to achieve overall accuracy and reduced costs by using a large number of inexpensive, short-range sensors rather than a few expensive and complex long-range sensors. The authors [61] predicted that by the next century, low-level and low-power wireless communication protocols will be developed to coordinate miniature sensors to design the ad-hoc smart environments. These smart environments can be effectively used to collect and disseminate information in various situations. Ad-hoc smart environments must provide exception-free, unattended operation (using autonomous nodes). Nodes must be completely self-configuring and robust to changes in condition. The environments must automatically adapt to changes in environment and requirements. These networks must be data-centric (meaning the applications will focus on the data generated by sensors), application specific, and resource constrained. The sensor networks should use localized algorithms, meaning the nodes communicate only with sensors within some neighborhood. Nodes achieve a global objective by using only local computations. Local algorithm based sensor networks are more robust, and scale better.

A localized and data-centric data dissemination algorithm for sensor networks, called directed diffusion was presented in [48, 66, 81]. All nodes are application aware. This scheme saves energy by selecting empirically good paths and by caching and processing data in-network (e.g., data aggregation). Each node names data that it generates with one or more attributes. Other nodes (called sinks) may express interests, based on these attributes. For each active task, the sink periodically broadcasts an interest message to each of its neighbors. Other (intermediate) nodes propagate interests. Interests establish a reverse data path for data that matches the interest. This path has an associated gradient. The gradients direct the diffusion of data. As it propagates, intermediate nodes can cache or
locally transform (e.g., aggregate) data. Caching and aggregation can increase the efficiency, robustness, and scalability of coordination. When an interest arrives at a data producer, that source begins producing data. The first data message sent from the source is marked as exploratory and is sent to all neighbors that have matching gradients. When exploratory data reaches the sink, the sink reinforces its preferred neighbor, establishing a reinforced gradient towards the sink. The reinforced neighbor reinforces its neighbor in turn, all the way back to the data source or sources, resulting in a chain of reinforced gradients from all sources to all sinks. Subsequent data messages are not marked exploratory, and are sent only on reinforced gradients rather than to all neighbors. A particular feature of this algorithm is that a node in the network may make a local decision (based possibly on perceived traffic characteristics like the observed delay difference between events received along different paths) to draw data from one or more neighbors in preference to other neighbors. Such techniques are called localized algorithms.

The directed diffusion protocol has been termed as two-phase pull protocol in [66]. Two more diffusion protocols, called one-phase pull and push along with the proper applications and suitability of these three algorithms were described in [66]. Declarative Routing Protocol (DRP) [31] is another recent data dissemination protocol. Directed Diffusion [81] and DRP are similar in that they both take the data-centric naming approach to enable in-network data aggregation.

The diffusion algorithms presented in [81] assumed periodic, low-rate, flooding of events that enabled local re-routing around failed nodes. Such flooding wastes a lot of energy. An alternative scheme was proposed in [52] where multipath routing is used to increase resilience to node failure. Multipath routing has been studied in both wired and wireless networks.
(like MANETs). The two main purposes of using multipaths are load balancing and reliable data delivery. Both are useful for sensor networks as well. But, the main contribution of the work in [52] is quickly finding alternate paths between source and sink. In addition to the primary path, they maintain a small number of alternative paths that can be used in case the primary path fails. Two types of alternate paths have been considered: disjoint multipaths and braided multipaths. Disjoint paths are node-disjoint with the primary path, while the braided paths need not necessarily be completely node-disjoint with the primary path. Disjoint path strategy is more resilient but uses more energy than braided path strategy.

A family of adaptive dissemination protocols, called SPIN (Sensor Protocols for Information via Negotiation) for wireless sensor networks was proposed in [68]. SPIN uses metadata negotiation and resource-adaptation to overcome several deficiencies in traditional dissemination approaches like flooding and gossiping [65]. SPIN focuses on the efficient dissemination of individual sensor observations to all the sensors in a network assuming that all sensors could be sink nodes. This increases the fault-tolerance of the system. Also, a critical piece of information can be disseminated to all the nodes. The above can be implemented using classic flooding. However, there are three deficiencies of simple flooding. They are implosion (a node always sends data to its neighbors, regardless of whether or not the neighbor has already received the data from another source), overlap (some nodes often cover overlapping geographic areas, and nodes often gather overlapping pieces of sensor data), and resource blindness (nodes do not modify their activities based on the amount of energy available to them at a given time). SPIN overcomes the above problems using two key ideas: negotiation and resource-adaptation. To deal with implosion and overlap,
nodes in SPIN negotiate with each other before transmitting data. This avoids sending unnecessary data. Nodes describe data by using meta-data in the negotiation process. Thus, exchanging sensor data may be expensive, but exchanging data (meta-data) about data need not be. Nodes poll their resources (e.g., energy) before transmitting data. This allows sensors lacking in energy to cut back on certain activities.

Another group of data dissemination protocols have been proposed, which considered multiple sources and multiple mobile sinks. Sink mobility makes a dissemination protocol more challenging. A Scalable Energy-efficient Asynchronous Dissemination protocol (SEAD) has been proposed in [88]. SEAD is a distributed self-organizing protocol. The experimental results show that SEAD is more energy-efficient than [81, 155]. In this work, the sinks are considered to be mobile. The communication consists of three main tasks: building the dissemination tree (called d-tree), disseminating data, and maintaining linkage to mobile sinks. The main focus of this work is to minimize communication cost in terms of energy. When mobile sinks join the tree, SEAD does not use flooding to find an entry to the tree. Flooding is used in [81, 155]. Flooding uses a lot of energy and incurs unnecessary collisions. In SEAD, the mobile sink selects one of its neighboring sensor nodes to send a join query to the source of the tree. The selected sensor node is called the sink's access node. The access node is used to represent the moving sink when the optimal d-tree is built. Static access nodes amortize the overhead in the presence of mobility. Access nodes keep track of the current position of the corresponding mobile nodes. The tree delivers data to the fixed access node. In turn, the access node delivers the data to the sink without exporting the sink's location information to the rest of the tree. The tree is updated only when the access node changes (as opposed to every time some node moves). Source data is
replicated at selected nodes between the source and sinks. The replica temporarily stores the latest data incoming from the source and asynchronously disseminates it to others along the tree. The replica placement strategy locally readjusts the tree in the neighborhood of the gate replica to further reduce communication energy. The constructed tree is managed to accommodate mobile sinks or defective regions such as a group of congested or failed nodes. TD3DD, a Two-Tier Data Dissemination protocol [155] is another protocol which considers mobile sinks.

3.2.5 Data Fusion

Motivation behind sensor fusion research was discussed in [22]. Although sensors are used in many real world applications, they are still not that accurate. Fusing sensor data is a method to overcome this drawback. Combining readings from multiple sensors may increase the degree of accuracy. Multiple sensors can also help in making a decision as a group (see group decision-making systems below). We can remove inconsistencies, and get a clearer and better interpretation of readings input from individual sensors. Other advantages are the increase of reliability and reduction of cost. The flow of data in general in sensor process consists of the following stages: the raw data (detection) is put into some sort of computable format (preprocessing). The computable data from many sources is combined (fusion) and evaluated by the system (interpretation).

*Multi-sensor fusion* [22, 98] refers to combining the readings from several sensors into a uniform data structure. This concept can be applied to any systems involving signal processing. These applications include aeronautics (e.g., air traffic control system, navigation system, and location finding system used in military aircrafts), manufacturing (e.g., robot sensors), remote sensing (e.g., weather forecasting, space shuttle, and pollution con-
trol), hazardous environments (using autonomous and semi-autonomous devices), traffic control (air, shipping, railways, and highway traffic, and safety features in automobiles), and medical applications.

Multiple sensors are used in group decision-making systems [138]. Group decision-making problems appear in many large-scale systems, including many real-world situations. Application areas include financial institutions, air-traffic control, oil exploration, medical diagnosis, military command and control, electric power networks, weather prediction, and industrial organizations. For example, a military commander may use data from radar and multiple sensors along with intelligence information to decide to attack or retreat. Many distributed detection network topologies have been designed to implement various organizational structures for group decision-making systems. Refer to [138] for details.

Data fusion or in-network aggregation techniques for sensor networks have been reported in [91, 100]. In future sensor networks, we expect to see a fast processor on a single small sensor with a good size memory and a radio transceiver. The main challenges involved in data fusion are its time-sensitive nature and the need for synchronization of the data from multiple streams. An additional requirement of a fusion application is the scarcity of power in the individual nodes. In this regard (power), data fusion and power-aware routing techniques [27, 127] have some similar objectives. The benefit of power awareness in sensor networks is discussed in Section 3.2.3.

The rapid improvement of microsensor technology has influenced the emergence of a key area in signal processing, called Collaborative Signal and Information Processing (CSIP) [92] in microsensor networks. CSIP research into microsensor networks has focused on developing new methods and algorithms for representing, storing, and processing spatially...
distributed, multimodal information. Refer to [92] for a collection of articles on this topic.

Data can be collected from many independent sensors, and then combined or fused into one reliable reading. Sensor data fusion technique and Byzantine Agreement problem [95] have been combined to design a hybrid algorithm to design a reliable data fusion technique [21]. Byzantine failures model any arbitrary type of processor malfunction. Byzantine generals problem [95] is a distributed consensus problem [99]. Any solution to this problem must satisfy the following redundancy: the number of processors used must be more than three times the number of faulty processors. In sensor networks, data fusion technique must use redundant sensors to rely on the input readings.

Two types of fusion techniques, called value fusion and decision fusion were introduced in [30]. The purpose of these techniques is to design fault-tolerant fusion technique for collaborative signal processing systems. In value fusion, all sensors exchange their measured values and then each sensor makes its own individual and independent decision by fusing the collected values. In decision fusion, each sensor may make its independent decision using its own measured values and then sensors may exchange their decisions among each other to arrive at a consensus by fusing all decisions. In this work, both fault-free and faulty sensors were considered in the experiments. The fault model considered was Byzantine type [95]. The results showed that value fusion is superior to decision fusion when the sensor network is highly reliable and fault-free. The performances of these two fusion techniques reverse in a faulty environment.

3.2.6 Routing Protocols

A clustering-based energy-efficient routing protocol, called Low-Energy Adaptive Clustering Hierarchy (LEACH) for sensor networks was proposed in [67]. LEACH was shown to be
more energy-efficient compared to some other existing routing protocols. In this work, it was assumed that the data being sensed by the nodes in the network will be transmitted to a control center or base station where the end-user can access the data. The base station is fixed and located far from the sensors. All nodes in the network are homogeneous and energy-constrained. The above two assumptions imply that the communication between the sensor nodes and the base station is expensive in terms of energy consumption, and there are no special high-energy nodes through which communication can proceed. The key features of LEACH are: (a) Localized coordination and control for cluster set-up and operation. (b) Randomized rotation of the cluster “base stations” or “cluster-heads” and the corresponding clusters. (c) Local compression to reduce global communication.

In the simulation study conducted in the research reported in [67], two types of protocols were considered: a direct communication protocol, where each sensor sends its data directly to the base station, and a minimum-energy (or power-aware) routing protocol (e.g., [27, 127]). Two types of energy spent by the protocols were considered: to transmit and to receive a message. In the direct communication protocol, if the base station is far away from the nodes, each node will expend a large amount of transmission power. However, the experimental studies showed that if either the base station is close to the nodes, or the energy required to receive data is large, the direct communication method expends less energy (sum of transmitting and receiving) than the minimum-energy routing protocol. In the minimum-energy protocol, the nodes closest to the base station will be used to route a large number data messages to the base station. Thus, these nodes will die out quickly, causing the energy required to get the remaining data to the base station to increase and more nodes to die. This will create a cascading effect that will shorten system lifetime. In
addition, as nodes close to the base station die, that area of the environment is no longer being monitored. For similar reasons, in direct communication protocol, the nodes farthest from the base station will die out first.

In a clustering-based approach, nodes are organized into clusters that communicate with a local base station, and these local base stations transmit the data to the global base station, where it is accessed by the end-user. This greatly reduces the distance nodes need to transmit their data, as typically the local base station is close to all the nodes in the cluster. Although this seems to be an energy-efficient solution, local base stations may die out quickly as they are being heavily used. This problem is solved in LEACH. LEACH is a self-organizing, adaptive clustering protocol that uses randomization to distribute the energy load evenly among the sensors in the network. Sensors elect themselves to be local cluster-heads at any given time with a certain probability. In order to keep the cluster-heads alive for a longer period of time, the cluster-heads are randomly rotated. Another advantage of this is that the nodes die in a random fashion which avoids any particular section of the environment not being sensed.

GRAdient Broadcast (GRAB) [154] targets at robust data delivery in an extremely large sensor network made of highly unreliable nodes. The algorithm is based on four ideas: cost field setup, credit-based adjustable mesh forwarding, density control, and cost field refreshment.

Cost field setup: Every node computes its minimum cost to reach the sink, i.e., effectively sets up the minimum cost (shortest) path to the sink. The cost computation is initiated by the sink node by broadcasting an advertising message only once. All intermediate nodes also broadcast/forward the cost message only once. So, flooding is avoided to compute the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
cost. The minimum cost path is used by any source to deliver the data to the sink.

**credit-based adjustable mesh forwarding:** The report message carrying the data (to be disseminated) includes two fields: minimum required cost from the source to the sink and the cost consumed so far for traversing from the source to the current (intermediate) node. An intermediate node forwards the message only if the consumed cost so far plus the minimum cost from this node to the sink is equal to the source's cost. Thus, the minimum cost paths are maintained without maintaining the next-hop node information.

Sending the report message through a single path is prone to node failures and interferences. A unique credit-based mesh forwarding method is used to enhance robustness and achieve flexible tradeoff between robustness and energy by controlling the credit carried by the messages. The credit is the total amount of energy budget given for this message less the minimum cost of the source. That is, a source having some extra energy to spare for sending a report message can afford to travel more paths during the delivery process. The intermediate nodes use the same scheme to forward the message. These multiple forwarding paths interleave and form a forwarding mesh. The width of the mesh decides the degree of robustness of the data (report message) dissemination, and is dependent on the amount of credit available.

**density control:** Nodes are monitored to make sure that enough nodes are awake to cover the geographical region by keeping other nodes turned off. The sleeping nodes wake up later and replace nodes with very low energy or failed nodes.

**cost field refreshment:** Topology changes due to various reasons will affect the minimum costs. A hybrid approach of both event driven and timeout is used to refresh the costs. The sink monitors certain properties of received packets. If any major change is noticed,
it rebroadcasts a cost advertise message to rebuild the cost field. In addition, if no report message is received for certain time, the sink also refreshes the cost field when the timeout occurs.

Rumor Routing protocol [20] takes a new approach to designing routing protocol. The main idea is to efficiently distributing queries to nodes that have observed events in the network. An event is an abstraction, identifying anything from a set of sensor readings to the node's processing capabilities. The goal was to design a protocol which is tunable, and allows for tradeoffs between setup overhead and delivery reliability. The directed diffusion method [81] floods queries. In GRAB [154], the sink node initiates cost field computation by starting a restricted form of broadcasting. Thus, there is an overhead of establishing the cost field. However, the report (data) messages are delivered following the minimum cost paths cheaply and reliably. The rumor routing [20] uses a set of long-lived agents to establish paths directed towards the events they encounter. Whenever an agent crosses a path leading to an event it has not seen yet, it adapts its behavior and creates path that leads to both events. The agents also tries to optimize the paths. The geographical routing techniques such as GEAR [156] and [86] rely on localized nodes, and provide savings over a complete network flood by limiting the flooding in a geographical region. The method in [119] allows access to named data by hashing the name to a geographic region in the network.

Recently, results of an extensive study of issues related to routing in sensor networks have been reported in [152]. The authors estimated link connectivity statistics dynamically using an efficient yet adaptive link estimator, explored storing the link status and routing information in a neighborhood table with constant space regardless of cell density, and
studied routing on dense sensor networks with simple, low-power radios and limited storage.

A wide range of routing alternatives in terms of hop distribution, path reliability, and stability of the routes on large networks were studied. This study gave a lot of insights into the interactions across the system layers involved in the routing problem for sensor networks.

In [78], a spatiotemporal multicast protocol, called mobicast [77] was used to study distribution of messages to nodes in a delivery zone that evolves over time in some predictable manner. Mobicast provides reliable and just-in-time message delivery to mobile delivery zones on top of a random network topology. Many sensor networks (e.g., habitat monitoring and intruder tracking) need to track down physical entities that move in the environment. In such applications, only sensors close to an interesting physical entity should participate in the aggregation of data associated with that entity as other sensors far away waste precious energy without improving sensing fidelity. An active sensor group moving at the same velocity as the mobile entity is maintained. A protocol for activating and deactivating sensors is deployed. Only a small number of sensors should be active to provide continuous coverage, while most sensors sleep and periodically wake up to poll active sensors and enter the active mode if necessary. Another module of the system presented in [78] is a communication mechanism that enables sensors to actively push information about a known entity to other sensors or actuators before the entity reaches their vicinity. Mobicast allows applications to specify their spatiotemporal constraints by requesting a mobile delivery zone. Thus, mobicast provides a powerful communication abstraction for supporting local coordination and data aggregation in sensor networks.
3.2.7 Localization Systems

Due to the proliferation of sensor devices and the goal of creating ubiquitous environment [145], the computing elements (sensors) will be ubiquitous and pervasive [38, 15]. Realization or implementation of pervasive environment will require some context-aware, location-dependent applications which adapt their behavior and user interface to the current location in space [117]. The problem of localization is to determine where a given node is physically located in a network. This is a challenging yet extremely crucial problem for many embedded sensor applications like environmental monitoring of water and soil [23]. Localization helps reduce power consumed in multihop wireless networks. In context-aware applications, localization allows proper selection of appropriate devices, and support useful coordination among devices. The location information can also be effectively used in routing protocols such as geographic routing, information dissemination protocols, and sensor query processing systems. The Global Positioning System (GPS) [74] solves the localization problem in outdoor environments for PC-class nodes. However, for large embedded sensor networks due to their many resource, cost, and size constraints, GPS is not suitable. Localization has been a very active area of research in the sensor networks area [23, 24, 57, 116, 117, 121]. All these algorithms are suitable for wireless and distributed embedded sensor networks. There are various characteristics of these algorithms. Some of them self-configure, i.e., autonomously measure and adapt its properties to environmental conditions (rather than rely on design-time pre-configuration or manual reconfiguration) in order to achieve ad-hoc deployment and robust, unattended operation in any environment [24]. Algorithms [116] using only estimate of its distance to a few nearby (neighbor) nodes are preferred. Anchor-free algorithms [116] are also desirable. Anchor-based algo-
rithms [23, 121] assume that a non-negligible fraction of nodes are anchor nodes that already know their location.

### 3.2.8 Time Synchronization

Time synchronization is a critical task in sensor networks for various purposes like sensor data fusion, coordinated actuation, and power-efficient duty cycling. Consider sensor networks for some environmental application [47]. Mobile computing devices equipped with sensors, clocks, and short range radio are deployed in the environment to measure various things. The devices record the time when they detect or no longer detect the phenomenon and communicate this information to other devices as they pass by. In order to determine the direction of the phenomenon, temporal ordering of these events originating from different devices (and thus different clocks) has to be determined. To estimate the speed of the phenomenon, time differences between events originating from different devices have to be calculated. Time synchronization is also useful for estimating proximity of distances between smart things by taking into account the points in time when a certain phenomenon in the environment (e.g., sound, light, air pressure) is sensed by different smart things. It is expected that the sensors be tightly synchronized on the order of 1 \( \mu \text{sec} \). Sensor networks are used in many applications where very accurate time is needed. The Network Time Protocol (NTP) [106] has been used to maintain the Internet clocks. However, much improved time synchronization protocols have been proposed for the sensor networks. See [45, 46, 47, 71] for some of these protocols.

Collision-free communication is a desirable property for sensor networks. First, the collided messages cannot be used. Second, collisions waste energy. A stabilizing collision-free diffusion protocol was presented in [90]. In addition, the diffusion algorithm was used
to obtain time-division multiplexing. The proposed algorithm maps a general network to a grid structure. Thus, the collision free algorithm is based on a grid where nodes are assumed to know their location in the grid. So, this algorithm is not suitable for dynamic networks. This drawback is removed in [72]. In [72], a self-stabilizing distributed dynamic TDMA slot assignment algorithm is presented. Nodes in this algorithm do not use any local information making the algorithm suitable for dynamic networks.

3.2.9 Other Problems

Media access control (MAC) and transmission control protocols are extremely important for sensor networks. Although protocols for these problems exist in traditional networks, but different wireless technologies, application characteristics, and resource constraints demand new investigation of these protocols for the sensor networks. Such protocols were presented in [151].

The exposure problem was introduced in [104]. It is a measure of how well an object, moving on an arbitrary path, can be observed by the sensor network over a period of time. The minimal exposure path provides valuable information about the worst case exposure-based coverage in sensor networks. The Exposure problem is related with the coverage problem (discussed in detail in Section 4.2 in Chapter 4). Deployment of sensors depends on the applications. It can be predetermined when the environment is sufficiently known and under control, in which case the sensors can be strategically hand placed. The deployment can be random (e.g., by air-dropping) if the environment is unknown or hostile. Deployment strategies for target detection over a region of interest were studied in [29]. In this application, sensors use group-decision making to achieve a global decision [22, 138] (see Section 3.2.5 for details). Since the local observations made by the sensors depend on
their position, the performance of the detection algorithm is a function of the deployment. The path exposure [104] was used to evaluate the performance of the deployment strategies. The main goal of this research was to determine the number of sensors to be deployed to carry out target detection in a region of interest. The tradeoffs lie among the network performance, the cost of the sensors deployed, and the cost of deploying the sensors.

A scalable and fault-tolerant distributed clustering algorithm for sensor networks, called LOCI was presented in [107]. Clustering is a well-known approach to achieve efficient, scalable control in large networks. Every cluster will have a leader node. The proposed clustering method is uniform in the sense that all nodes within a radius $R$ from a leader node will belong to the cluster of that leader and no node further than $mR$ (where $m$ is a constant $\geq 2$) from the leader will be part of the cluster of the particular leader. The article [107] also includes a fault-tolerant solution to a distributed tracking service [13] using LOCI. A self-stabilizing solution to the pursuer-evader problem in sensor networks was given in [37]. Two solutions were presented — one is more energy-efficient and the other faster in tracking evader.
CHAPTER 4

CONNECTED COVER

We report the main results of this thesis research in this chapter. As discussed in earlier chapters, the two main fields we studied are self-* systems and wireless sensor networks. After carrying out an extensive study of those topics, we designed a local, distributed, and self-* protocol as a solution to a very important problem in sensor networks, called connected sensor cover problem. The roadmap of this chapter is as follows: In the next section, we state the motivation of this research. We informally state how some other problems/topics studied in previous chapters are related to the problem solved in this chapter. Then in Section 4.2, we describe some results in related areas. In Section 4.3, we first state the model used in writing the algorithm. The program (including its notations) used is reported next. We also give a formal definition of self-stabilization in this section. Finally, in this section, we give both an informal explanation and formal statement of the problem solved in this chapter. In Section 4.4, the connected sensor cover algorithm (Algorithm MCSC) is presented. In that section, we include a detail informal description and a formal algorithm. The proof of the algorithm is given in Section 4.5. We also discuss about simulation and other properties of the algorithm in Sections 4.6 and 4.7, respectively. We end this chapter (Section 4.8) with some ideas which we would like to explore to extend this research beyond this thesis.
4.1 Motivation

As discussed in Chapters 1 and 3, the sensor networks are usually composed of a very large number of low-cost sensor nodes with highly constrained energy sources. In many applications, they will be densely deployed. To make the situation worse, unlike MANET's, it is usually impossible to replace the battery of sensor nodes, meaning they have a very short life. These networks can be deployed everywhere — home, industrial plants, environmentally disaster and inaccessible areas. The topology of sensor networks is expected to be very dynamic due to various reasons. Despite all the above constraints and obstacles, we still must be able to design robust sensor networks which will allow uninterrupted operation even in an unattended environment for a long period of time. Considering the size and applications as discussed above, it is obvious that the sensor networks should be designed as self-* systems (Chapter 2).

In sensor networks, queries are sent from some devices (could be a satellite, PDA, a laptop, or any computer) to sense some data/events over some time period and a geographical region, called query region. A query could be like "Every I ms for the next Y seconds, tell me how many vehicles of type T are moving in direction D in region R". The query region is usually a subset of the total region covered by all the sensors in the network. Considering the limited energy available, one of the most important goals in any protocols on sensor networks is to save energy. Since the sensors are usually densely deployed, there are usually a lot more sensors than required to process a given query. One possible way to minimize usage of energy is not to keep all sensor nodes fully active all the time. Some of them can be put to passive mode some times while others are active in sensing the data in the environment. However, for the sensor networks to be effective, the active nodes must be
able to cover the whole query region and maintain the network connectivity at all times.

In [63], a new optimization problem in sensor networks, called connected sensor cover, was introduced. The objective of the problem is to select a minimum or nearly minimum set of sensors, called connected sensor cover, such that the selected sensors cover the query region and can (directly or indirectly) communicate with each other. The research in this thesis is motivated by the work of [63].

Data dissemination, data aggregation, and routing protocols are presented in Sections 3.2.4 and 3.2.6 in Chapter 3. These protocols are used to efficiently broadcast (or route) and forward messages from sinks, aggregate messages from the sources to the sinks. In the recent years, a number of protocols were developed to design robust broadcasting or forwarding data messages. As mentioned earlier, in sensor networks, queries are sent from devices external to the sensor network. Therefore, the query needs to be broadcast to the sensor nodes or routed to a particular sensor node (maybe, the node closest to the device) which would initiate the connected sensor cover algorithm. Similarly, after the connected sensor cover is computed, the result need to be reported back to the device which originated the query.

4.2 Related Work

The problem addressed in this thesis, the connected sensor cover, was introduced in [63]. Two self-organizing solutions were presented in [63]. None of the solutions is localized. The first solution is centralized. The second solution was claimed to be distributed. However, it is not a distributed solution in true sense. In every stage of the distributed solution, a particular sensor node behaves as the coordinator or leader. This special node collects all the global information about the possible new sensor nodes to be added, then decides
which ones to choose. We now describe the main idea about the solutions in [63]. A greedy approach is taken to select the best possible set of sensors. We need to define a few terms to describe the algorithms. A subelement is defined as a minimal region that is formed by an intersection of a number of sensing regions. A subelement is termed as a valid subelement if its region intersects with the query region. In every stage of the algorithm, a set of sensors lying on the best candidate path is added to the existing set of connected sensors. A candidate path consists of a path of a sensors that form a communication path connecting a candidate sensor with some sensor already included in the connected sensor cover set. A sensor is a candidate to be chosen if its sensing region intersects with the sensing region of some sensor in the current connected set of sensors. The best candidate path is the one which covers the maximum number of uncovered valid subelements per sensor. So, the goal of the algorithm is to select the path of connected sensors such that the maximum number of uncovered subelements belong to this path.

The issues of coverage and connectivity, and the relations between them were analyzed in a unified framework in [140]. Different applications may require different degrees of sensing coverage — a location being covered by only one node vs. more than one nodes. The degree of coverage may also be decided by the degree of fault-tolerance to be maintained by the application. Changes in application modes or environmental conditions may demand a change of coverage requirement. A surveillance sensor network may initially maintain a low degree of coverage required for distributed detection. After an intruder is detected, however, the region in the vicinity of the intruder may reconfigure itself to achieve a higher degree of coverage required for distributed tracking. In [140], a Coverage Configuration Protocol (CCP) that can dynamically configure networks to provide different feasible degrees of
coverage requested by applications was presented. In this work, CCP was integrated with a
connectivity maintenance protocol (SPAN [28]) to provide both coverage and connectivity
guarantees. The integrated coverage and connectivity problem solved in [140] is as follows:
Given a coverage (or query) region $A$ and a sensor coverage degree $K_s$-covered (i.e., every
location inside $A$ is covered by at least $K_s$ nodes), find the maximum number of nodes
that are scheduled to sleep under the constraints that the remaining nodes must guarantee
two conditions (a) $A$ is at least $K_s$-covered and (ii) all active nodes are connected. The
first interesting result obtained in [140] is the sufficient condition for a 1-covered sensor
network to be connected. Satisfying $R_c \geq 2R_s$ (where $R_c$ and $R_s$ are the communication
and sensing ranges, respectively) for a 1-covered network implies the connectivity. That
is, if the communication range is at least twice the sensing range, all we have to do is
to configure the sensors such that they are covered. That would imply the connectivity.
Another important result on the relation between the degrees of coverage and connectivity
established in [140] is the following: If $R_c \geq 2R_s$, then $K_s$-coverage of a convex region
implies a $K_s$-connectivity of the communication graph. (The connectivity of a graph is
the minimum number of nodes that must be removed in order to partition the graph into
more than one connected components.) The protocol CCP implements the $K_s$ coverage
and $K_s$ connectivity in a network which satisfies the condition $R_c \geq R_s$. The main task of
the protocol is for every sensor node to decide if it is eligible to be active at a particular
time. A sensor is ineligible to become active if all the intersecting points inside its sensing
region are at least $K_s$-covered. Sensor nodes exchange beacons (HELLO messages) among
their neighbors to compute all the intersecting points inside their sensing circles. When the
condition $R_c \geq 2R_s$ is not satisfied by the network, CCP alone does not solve the integrated
problem. In that situation, the problem is solved by composing CCP with SPAN [28]. The summary of the results obtained from the various experiments conducted in the research in [140] is the following: CCP can provide one-coverage while keeping a significantly smaller number of active nodes than some existing protocols. CCP can effectively enforce different coverage degrees specified by the application.

A self-configuring strategy, called ASCENT, to form a sensor network that provides communication and sensing coverage under stringent energy constraints was presented in [26]. The goal is to keep enough nodes to cover the region without too many packet losses and maintain the connectivity. If too few nodes are deployed, packet loss will increase. If too many nodes are deployed, the collisions will increase and energy will be wasted unnecessarily. ASCENT works in phases. A node starts in a listening-only phase, called neighbor discovery phase, where each node estimates the number of neighbors actively transmitting messages computed locally. Then the nodes enter into the join decision phase, where they decide whether to join the network. The decision is based on the estimate of the number of active neighbors and the data message loss. If a node decides to join for a long time, it enters into an active phase and starts sending routing control and data messages. If a node decided not to join the network, it enters into the adaptive phase, where it turns itself off for a period of time, or reduces its transmission range.

Unreliable sensor networks were studied in [125]. A sensor grid network of unit area was considered. The authors derived the necessary and sufficient conditions for the network to remain covered and connected in terms of the probability of a node to be active (i.e., not failed) and transmission radius of the nodes. It was shown, a large network can maintain connectivity and coverage even with highly unreliable and small transmission powered sensor
nodes. The diameter of the random grid network was also computed. This result (the diameter) can be very useful to design efficient routing, dissemination, and aggregation protocols.

Computational geometry techniques were used to design coverage algorithms in [103]. The proposed algorithms use geolocation information of the sensor nodes. A pre-processing algorithm is used to know the location of the sensors. This scheme uses GPS. Both deterministic and stochastic coverage cases were considered in this research. Two types of coverage have been studied in this work: worst and best case coverage. In the worst case coverage, attempts are made to quantify the quality of service by finding areas of lower observability from sensor nodes and detecting breach regions. In the best case coverage, finding areas of high observability from sensors and identifying the best support and guidance regions are of primary concern.

The problem of placement of sensor nodes has been studied in [128]. The sensor nodes cannot be placed deterministically for two main reasons. First, they are often deployed in remote or inhospitable areas. Moreover, the sensor nodes available currently do not support dynamic self-adjustment of positions. Secondly, the large number of sensor nodes makes deterministic placement impractical due to the increased cost and latency. Typically, in remote or inaccessible areas, the sensor nodes are dropped in bulk from an aircraft [85]. This kind of ad-hoc sensor networks must be designed to satisfy the following three conditions and constraints [26]: ad-hoc deployment (the sensors may not be deployed in regular fashion; uniform deployment may not correspond to uniform connectivity due to obstacles, etc.), energy constraints (the nodes will be untethered for power as well as communications), and unattended operation under dynamics (the size of the network will not support manual
configuration, and the environment dynamics will preclude design-time pre-configuration).

A heuristic was reported in [128] to organize the sensor nodes into mutually exclusive sets, where the members of each set cover the monitored area. So, it is necessary to keep only the members of any one of these sets active and the rest inactive at any time. After some time (decided by the protocol), the active set is deactivated and a set from the previously active sets is made active. This process is repeated until the sensors go out of power.

4.3 Preliminaries

4.3.1 Model

Sensor Network. In this research, we consider sensor networks [63, 140] consisting of a large number of sensors (also referred as sensor nodes and nodes in this thesis) randomly distributed in a geographical region. We model the sensor network as a directed communication graph $G(V, E)$, where each node in $V$ represents a sensor, and each edge $(i, j) \in E$, called communication edge, indicates that $j$ is a neighbor of $i$ (see Definition 4.3 below).

Definition 4.1 (Sensing Region and Sensing Range) For a sensor $i$, there is a region, called sensing region, which signifies the area in which the sensor $i$ can sense a given physical phenomenon maintaining a desired confidence level. The sensing range of a sensor $i$ indicates the maximum distance between $i$ and any point $p$ in the sensing region of $i$.

Definition 4.2 (Covered) A point $p$ is covered (or monitored) by a sensor node $i$ if the Euclidean distance between $p$ and $i$ is less than the sensing range of $i$.

Assumption 4.1 (Circular Sensing Region) The sensing regions are of any convex shape.

For the sake of simplicity, especially, for showing examples, the sensing regions are assumed
to be circular. However, our solution works for any convex shape.

Definition 4.3 (Communication Region, Communication Range, and Neighbors)
The communication region of a sensor $i$ (also called the transmission region) defines the
area in which $i$ can communicate directly (i.e., in single hop) with other sensor nodes. The
maximum distance between node $i$ and any other node $j$, where $j$ is in the communication
region of $i$, is called the communication range of $i$. Node $i$ can communicate with node $j$
(i.e., $i$ can send a message to $j$) if the Euclidean distance between them is less than the
communication range of $i$. Then $i$ is called a neighbor of $j$, and this relation is represented
by a directed edge $(i, j)$. The set of neighbors of $i$ is represented by $N_i$. Two nodes $i$ and $j$
can communicate directly with each other only if $i \in N_j \land j \in N_i$, i.e., they are neighbors of
each other. If $i$ and $j$ are neighbors of each other, then there are two edges between them:
$(i, j)$ and $(j, i)$.

Definition 4.4 (Communication Path and Communication Distance) A directed path
(sequence) of sensors $i = i_1, i_2, \ldots, i_m = j$, where $i_x$ is a neighbor of $i_{x+1}$ for $1 \leq x \leq m - 1$,
is called a communication path from $i$ to $j$. The length of the shortest (communication) path
(which is the number of sensors on the shortest path) from $i$ to $j$ is called the communication
distance from $i$ to $j$.

Program. We consider the local shared memory model of communication as used by Di-
jkstra [39]. The program of every processor consists of a set of shared variables (henceforth,
referred to as variables) and a finite set of actions. A can only write to its own variables,
and read its own variables and variables owned by the neighboring nodes.

Each action is of the following form: $<\text{label}> :: <\text{guard}> \rightarrow <\text{statement}>$. The
guard of an action in the program of $p$ is a boolean expression involving the variables of $p$ and its neighbors. The statement of an action of $p$ updates one or more variables of $p$. An action can be executed only if its guard evaluates to true. We assume that the actions are atomically executed, meaning, the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step. This model is known as *composite atomicity* [41].

The state of a node is defined by the values of its variables. The state of a system is the product of the states of all nodes. We will refer to the state of a node and system as a *(local)* state and *(global)* configuration, respectively. Let a distributed protocol $P$ be a collection of binary transition relations denoted by $\rightarrow$, on $C$, the set of all possible configurations of the system. A *computation* of a protocol $P$ is a maximal sequence of configurations $e = \gamma_0, \gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots$, such that for $i \geq 0, \gamma_i \rightarrow \gamma_{i+1}$ (a single *computation step*) if $\gamma_{i+1}$ exists, or $\gamma_i$ is a terminal configuration. The *Maximality* means that the sequence is either infinite, or it is finite and no action of $P$ is enabled in the final configuration. All computations considered in this paper are assumed to be maximal. The set of all possible computations of $P$ in system $S$ is denoted as $E$. A node $p$ is said to be *enabled* in $\gamma$ ($\gamma \in C$) if there exists an action $A$ such that the guard of $A$ is true in $\gamma$. We consider that any node $p$ executed a *disable action* in the computation step $\gamma_i \rightarrow \gamma_{i+1}$ if $p$ was enabled in $\gamma_i$ and not enabled in $\gamma_{i+1}$, but did not execute any action between these two configurations. (The disable action represents the following situation: At least one neighbor of $p$ changed its state between $\gamma_i$ and $\gamma_{i+1}$, and this change effectively made the guard of all actions of $p$ false.) Similarly, an action $A$ is said to be enabled (in $\gamma$) at $p$ if the guard of $A$ is true at $p$ (in $\gamma$).
We assume a *weakly fair and distributed daemon*. The *weak fairness* means that if a node \( p \) is continuously enabled, then \( p \) will be eventually chosen by the daemon to execute an action. The *distributed* daemon implies that during a computation step, if one or more nodes are enabled, then the daemon chooses at least one (possibly more) of these enabled nodes to execute an action.

### 4.3.2 Self-stabilizing Program

**Fault Model.** [5, 107] This research deals with various types of faults.

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

- The nodes may crash. That is, the faults can fail-stop nodes.

- The nodes may recover or join the network.

- The topology (both actual and logical topologies) may change due to faults.

- Faults may occur in any finite number, in any order, at any frequency, and at any time.

**Self-stabilization.** In Section 4.3.1, we defined \( C \) as the set of all possible configurations of the system. The relation \( c \vdash Pred \) means that an element \( c \in C \) satisfies the predicate \( Pred \) defined on the set \( C \). A predicate is *non-empty* if there exists at least one element that satisfies the predicate.

We define a special predicate true as follows: *for any* \( c \in C \), \( c \vdash true \).

Let \( \mathcal{A} \) be an algorithm, and \( \mathcal{R} \) and \( \mathcal{S} \) be predicates defined on the configurations of \( \mathcal{A} \).
Closure: $\mathcal{R}$ is closed in $\mathcal{A}$ if every computation of $\mathcal{A}$ starting from a configuration satisfying $\mathcal{R}$ preserves $\mathcal{R}$.

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

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

Self-stabilization: Let $\mathcal{L}_A$ be a non-empty legitimacy predicate of an algorithm $\mathcal{A}$ with respect to a specification predicate $\text{Spec}$ such that every configuration satisfying $\mathcal{L}_A$ satisfies $\text{Spec}$. Algorithm $\mathcal{A}$ is self-stabilizing with respect to $\text{Spec}$ iff true converges to $\text{Spec}$ in $\mathcal{A}$.

4.3.3 Problem Description

As discussed in Section 4.1, the main motivation of this research is to save energy used in a large sensor network where the nodes cannot be deployed in a predetermined or manual manner. Our research is focused on designing a query-response system. A query in sensor networks asks for some data/measurements/events sensed/observed over some period of time at some frequency over a geographical region. Upon receiving a query, the sensors will sense or measure the data, collaborate among themselves to disseminate or fuse the collective data to the sink of the query. Although a query can be initiated in the whole geographical region, but typically, a query refers to a subset of the region, called the query region denoted by $R_Q$ in this report. So, the sensors only inside the query region should be involved in generating the response to the query.
We assume a very dense network of sensors, i.e., with lot of redundant sensors in the whole geographic region. That is normally done if the sensors are randomly deployed for an application. Considering the redundancy and our goal of designing a power-aware query-response system, all sensors inside the query region should not be actively participating in the protocol to answer the query. Our approach (similar to the one in [63]) to save the energy is for the sensors inside $R_Q$ to self-organize to form a logical network sufficient enough to cover (see Definitions 4.1 and 4.2) the query region. However, in order for the sensors in the logical network (i.e., the region covered by the selected sensors) to be able to collaborate to detect the events, and compute and deliver the response, they must be able to communicate with each other directly or indirectly (see Definition 4.3). That is, the logical graph not only needs to satisfy the coverage criterion, it must also be a strongly connected communication graph. The reason for satisfying both criteria is that the coverage is concerned with whether any location is uncovered while connectivity only requires all locations of active nodes are strongly connected. The following example illustrates the issues of coverage and connectivity:

**Example 4.1** Consider the sensor network shown in Figure 4.3.1. Sensors are represented by small circular dots. We have numbered the relevant sensors with $I_1, I_2, \ldots, I_6$ and shown sensing regions (circular $S_i$ disks) associated with some of them (the black nodes/sensors). Let the distance between sensors $I_1$ and $I_2$ be equal to $t$. We assume that any two sensors that are roughly less than $t$ distance apart can directly communicate with each other. Now, let us consider a query over the geographic region represented by the parallelogram $R$ in the figure.

We can see that sensing regions associated with black nodes ($I_1, I_2, I_4,$ and $I_6$) are suf-
efficient to cover the query region, which is the parallelogram $R_Q$. However, the set of black nodes does not form a connected communication graph, as the sensor nodes $I_1$ and $I_4$ cannot communicate. However, if we add the gray nodes ($I_3$ and $I_5$) to the black nodes, we get a set of sensors that also forms a connected communication graph as shown in the figure. Thus the set of six sensors form a connected sensor cover. Our problem is to find a minimum such cover.

In this research, we deal with designing self-organizing algorithm to compute a communication graph, called the connected sensor cover (introduced in [63]), which satisfies both coverage and connectivity conditions. It is obvious from the description of the connected sensor cover problem that it is a global task, meaning nodes cannot locally compute the final response to the query. However, we still require the algorithm to be local in the sense that the nodes collect information from their immediate neighbors (see Definition 4.3). Unlike

Figure 4.3.1: An example showing the construction of a sensor cover.
the solution in [63], no node in the proposed algorithm collects global information, and no node behaves as a special node in any stage of the execution of the algorithm. In our solution, every node can locally decide if it should be an active or passive node in the current computation of the response to the query. In summary, we achieve a global objective by using a local algorithm.

Our goal is to design a self-healing protocol in this research. Our solution is self-healing in the following sense: Under various perturbations (as listed in Section 4.3.2), the system will eventually succeed in computing (or recomputing) the connected sensor cover. We implement the self-organization and self-healing properties using the paradigm of self-stabilization [39].

4.3.3.1 Problem Specification

The specification of the problem is similar to the one introduced in [63]. However, our solution is self-stabilizing while the one in [63] is not.

**Definition 4.5 (Connected Sensor Cover)** Consider a sensor network $G$ consisting of $n$ sensors $I_1, I_2, \ldots, I_n$. Let $S_i$ be the sensing region associated with sensor $I_i$. Given a query $Q$ over a 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 $Q$ if the following two conditions are satisfied:

**Coverage:** $R_Q \subseteq (S_{i_1} \cup S_{i_2} \cup \ldots S_{i_m})$.

**Connectivity:** The subgraph induced by $SC_Q$ is strongly connected in the sense that any two sensors in this set can communicate with each other (Definition 4.3) directly or indirectly.
A set of sensors that satisfies only the first condition above is called a sensor cover for $Q$ in the sensor network.

**Specification 4.1 (Connected Sensor Coverage Problem)** Given a sensor network and a query over the network, the connected sensor coverage problem is to find the smallest connected sensor cover (we will call it $MCSC_Q$). Additionally, we require the algorithm (solving the above problem) to be self-organizing, self-healing, and self-stabilizing.

Computing a minimum sensor cover in its general form is NP-hard [63, 93]. So, the proposed solution makes an attempt to approach an optimal solution by checking and removing redundant sensor nodes from the final cover set. However, the solution although suboptimal in terms of the number of sensors, must cover the query region $R_Q$ accurately. That is, every point in $R_Q$ must be covered (Definition 4.2).

**Note 4.1** On termination, a sensor will know if it should take the role of an active or a passive node for the application being run on the top of Algorithm $MCSC$ in the query region. An active node here means that the sensor will fully participate in the sensing and communicating role in the application. A passive node will not participate in the application. However, it will remain in low-power mode enough to do some local checking to detect faults. Upon detecting faults, it will correct the fault locally, if possible. Otherwise, it may have to change to an active node.

4.4 Connected Sensor Cover Algorithm

In this section, we will present the self-stabilizing solution to the connected sensor cover problem. This work is also reported in [35]. The rest of this section will be as follows: We will state a few simplifying assumptions, and define a few terms to be used in the algorithm.
We then explain the overall approach to the solution in Section 4.4.1. Then we will present the formal algorithm along with the informal explanation. To make the informal description easier to follow, we will split it into two parts. We first describe the normal behavior of the algorithm in Section 4.4.2. That is, we assume that the system starts from a good configuration and no faults occur during the execution of the algorithm. In the following subsection (Section 4.4.3), we point out the type of faults or corruptions which can occur in the system. We also show how they are detected and corrected.

**Assumption 4.2 (No ID's)** *In this research, we assume that the sensors do not have unique identifiers (ID's). However, each sensor i maintains a set of distinct labels, denoted as Ni, such that each label identifies a (unique) neighbor of i (see Definition 4.3). Note that these labels are unique only locally.*

**Assumption 4.3 (Convex Query Region)** *The query region forms a convex region, and its boundary (hence, its center) is known to all sensors.*

**Assumption 4.4** *The sensing and communication regions are the same.*

**Remark 4.1 (Changing Energy Level)** *The energy level of the sensors may change over time due to various reasons. The proposed solution copes up with that.*

We distinguish three types of sensors in or around the query region \( R_Q \). In our algorithm, the rules for these three types of nodes are different.

**Definition 4.6 (Boundary Sensor)** \([140]\) *A sensor is termed as a boundary sensor if its sensing region intersects with the boundary of the query region \( R_Q \).*

Note that the boundary of \( R_Q \) is known as per Assumption 4.3.
**Definition 4.7 (Interior Sensor)** [140] A sensor is called an interior sensor if its sensing region is completely inside the query region $R_Q$ and it is not a boundary sensor.

**Definition 4.8 (Exterior Sensor)** All sensors which are neither boundary nor interior are called exterior sensors. In other words, if a sensor's sensing region is completely outside the boundary of the query region, then this sensor is called an exterior sensor.

### 4.4.1 Approach to the Solution

We could take one of the following two strategies to compute the connected sensor cover $MCSC_Q$:

**Approach 1.** The algorithm starts from a special sensor inside the query region $R_Q$ and proceeds towards the boundary of $R_Q$.

**Approach 2.** The algorithm starts from the boundary sensors (see Definition 4.6) and makes progress towards the center of $R_Q$.

Informally, the first approach attempts to construct a spanning tree covering the query region, while the second approach is similar to the computation of the center of the query region. How does one choose one of the above two approaches? There does not seem to be a very clear answer favoring one over the other approach. If we want to follow Approach 1, then the special sensor in the query region must be activated by an external agent (e.g., a laptop, PDA, or satellite). This special sensor must be able to communicate with the external device. Moreover, the external device should have the knowledge of the location of the special sensor. As discussed in Chapter 3, the sensor networks are prone to different types of faults. So, writing a fault-tolerant algorithm based on a special processor for most...
of the applications may not be an ideal way. Moreover, the distance between the initiating
device and the sensor network may be a limiting factor. The communication path between
the external agent and the special sensor may not be secured. Thus, in this research, we
chose Approach 2 which is described below.

We consider very dense networks in this research. Thus, even though some sensors may
fail or become very weak in power, we should have enough sensors to cover the query region
at any time. We state this in the following assumption which we depend on in designing
our algorithm:

**Assumption 4.5** (i) There always exist sufficient number of sensors in the network with
sufficient density to cover the query region if all of them are deployed.

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

The solution to the connected sensor cover problem is given as Algorithm 4.4.1 (referred
in this report as Algorithm \( \mathcal{MCSC} \)). As outlined in Approach 2, the algorithm starts from
the boundary sensors (Definition 4.6) of \( R_Q \), and proceeds outside in, i.e., towards the center
of the region. Starting from any configuration, the algorithm selects a few sensors among
many (due to our assumption of very dense network) to include in the (minimum) cover
set \( \mathcal{MCSC}_Q \). If the system starts from a good initial configuration, it first selects some
boundary sensors, then some interior sensors, and keep repeating the process of selection
moving towards the center of the query region until it covers the whole region \( R_Q \). Our
solution is localized, meaning the decision to be selected in \( \mathcal{MCSC}_Q \) is taken locally by
all nodes. So, unlike the solution in [63], nodes do not collect any global information to
compute \( \mathcal{MCSC}_Q \). Sensors consult only their immediate neighbors to decide if they should
be included in the final set cover. As mentioned earlier, we aim at designing a power-aware (see Section 3.2.3 in Chapter 3) algorithm. We make our algorithm energy efficient by removing redundant sensors as many as we can maintaining the coverage and connectivity as required by Specification 4.1. However, removing redundant nodes is a challenging task, especially since we took a fully localized distributed approach. We also cannot adopt a very complicated and aggressive procedure to improve our solution in terms of the size of the cover because that may waste a lot of sensor energy, which would conflict with the main goal of the research. We will revisit this issue while discussing the implementation of redundancy checking in our solution.

4.4.2 Normal Behavior

Before presenting the informal explanation of the algorithm, we describe the data structures used in the solution.

Data Structures. Three variables \((R_Q, N_i, \text{ and } Pos_i)\) are used as Constants by Algorithm \(MCSC\). That is, the algorithm does not write into these variables. They are used only as inputs to the algorithm. The input query includes the geographical information about the region to be covered. That is represented by \(R_Q\). \(N_i\) represents the neighboring sensors of sensor \(i\). Our solution assumes that there is an underlying self-stabilizing topology maintenance protocol (see [59] for such protocols) which computes \(N_i\). We assume that sensors know their location in the network. The sensors use either some device or/and protocol to know their geometric location. (See Section 3.2.7 in Chapter 3 for discussion about these protocols.)

The program uses two Variables: \(S_i\) and \(Color_i\). \(S_i\) represents the sensing region of
sensor \(i\). The Color variable is used to represent different status of a sensor. Sensors can be either in red or black initially. Eventually, if a sensor turns black and stays in that color, then it is considered to be a member of \(MCSC_Q\). Other sensors will remain in red color.

The solution uses two Macros: \(Dst(i)\) and \(BidirN(i)\). The macros do not represent variables. When referred in the code, they return values. We assume that the sensors know the location of the center (Assumption 4.3). So, they can use their location information (Pos) to compute their distances to the center. \(Dst(i)\) is the distance of Sensor \(i\) from the center of \(R_Q\). We consider directed communication graph of sensors. So, a sensor \(i\) may not have a two-way communication with all its neighbors. Sensor \(i\) may need this knowledge (i.e., which of its neighbors have a two-way communication with it) to check redundancy. The set \(BidirN(i)\) is computed in those situations. It returns the set of neighbors of \(i\) which effectively have a bidirectional communication with \(i\).

**Implementation.** In the following, we assume that the system starts from a good initial configuration, meaning, all sensors are red initially. The computation of the connected sensor cover initiates from the boundary of the given query region \(R_Q\). In the following, we will first describe how some boundary red sensors are selected to turn black to cover the boundary of \(R_Q\). (This description is under the item "Boundary of \(R_Q\".) Then we discuss the general case of covering any uncovered region inside the query region. (This description is placed as item "Inside \(R_Q\".)

**Boundary of \(R_Q\):** The initial task is to select enough sensors to cover the boundary with a communication network of sensors. The selected sensors will be colored black, and the rest will remain red. A sensor is a boundary node only if its sensing region intersects with the boundary of the query region. This is implemented in Action \(A_1\) using the
Algorithm 4.4.1 Self-stabilizing Connected Sensor Cover Algorithm (Algorithm M\textit{CSC})
for Sensor $i$.

**Constants:**
- $R_Q$: Query region; represented as a set in the algorithm;
- $N_i$: Set of sensors within the communication range of Sensor $i$;
- $Pos_i$: Geometric location or coordinates of Sensor $i$;

**Shared Variables:**
- $S_i$: Sensing region of Sensor $i$;
- $Color_i \in \{black, red\}$: Color of Sensor $i$;

**Macros:**
- $Dst(i) \equiv$ Returns the distance of Sensor $i$ from the center of $R_Q$;
  uses $R_Q$ and $Pos_i$ to compute the distance;
- $BidirN(i) \equiv \{j \mid j \in N_i \land i \in N_j\}$;

**Predicates:**
- $Boundary(i) \equiv S_i$ is a boundary sensor;
- $Interior(i) \equiv S_i \cap R_Q = \emptyset$; /* $S_i$ is an interior sensor; */
- $Exterior(i) \equiv S_i \cap R_Q = \emptyset$; /* $S_i$ is an exterior sensor; */
- $ConNbrs(i) \equiv \forall j, k \in BidirN(i): \exists l_1, \ldots, l_n \in BidirN(i): l_1 \in BidirN(j) \land (\forall x < n : l_{x+1} \in BidirN(l_x)) \land l_n \in BidirN(k)$;
- $Redundant_i(i) \equiv \exists j \in N_i : Color_j = black \land ((S_i \cap R_Q) \subseteq (S_j \cap R_Q))$;
- $Redundant_2(i) \equiv \exists j, k \in N_i : Color_j = Color_k = black \land ((S_i \cap R_Q) \subseteq ((S_j \cup S_k) \cap R_Q))$;
- $Redundant_3(i) \equiv \exists j, k, l \in N_i : Color_j = Color_k = Color_l = black \land ((S_i \cap R_Q) \subseteq ((S_j \cup S_k \cup S_l) \cap R_Q))$;
- $Redundant(i) \equiv (Redundant_1(i) \lor Redundant_2(i) \lor Redundant_3(i)) \land ConNbrs(i)$;
- $SameIntrsectn(i, j) \equiv Interior(i) \land Interior(j) \land (\exists x, y \in \{N_i \cap N_j\} : (Color_x = Color_y = black) \land (i \in N_x) \land (i \in N_y) \land (Pos_i \in (S_x \cap S_y) \land Pos_j \in (S_x \cap S_y))$;
- $BestCandidate(i) \equiv \forall j \in N_i : SameIntrsectn(i, j) : Dst(i) \leq Dst(j)$;

**Actions:**
- $A_1 : Boundary(i) \land \neg Redundant(i) \land Color_i \neq black \rightarrow Color_i = black$
- $A_2 : (Redundant(i) \lor Exterior(i)) \land Color_i \neq red \rightarrow Color_i = red$
- $A_3 : BestCandidate(i) \land \neg Redundant(i) \land Color_i \neq black \rightarrow Color_i = black$
predicates *Boundary* and *Redundant*. A boundary node will turn *black* and remain *black* only if it is not a redundant node. The redundant nodes will be eventually marked *red*. The redundancy is checked by using the predicate *Redundant*, and is described in detail in the next paragraph. Action $A_1$ changes a sensor from *red* to *black*. Later, if it turns out to be redundant, Action $A_2$ changes it to *red*. However, we wanted to make the implementation even more energy efficient. As we are using the asynchronous model, some nodes may be slow in executing Action $A_1$, while their neighbors have already changed to *black* by executing the same action. So, the slow nodes soon after turning to *black* may find out that they are redundant. Then, they will have to turn to *red* by executing another action ($A_2$). Instead of wasting energy and time, we added the redundancy check in Action $A_1$ itself. Therefore, a *red* sensor turns *black* only after checking for possible redundancy in the neighborhood.

The algorithm uses three predicates to check for three types of redundancy. The overall idea is to check if any area of the query region is covered by one or more than one sensor nodes. The *Redundant*$_1(i)$ predicate checks if the query region covered by sensor $i$ is covered by another sensor $j$. In case, there is no such sensor $j$, *Redundant*$_2(i)$ verifies if the region covered by $i$ is covered by two sensors $j$ and $k$ together. The third predicate, *Redundant*$_3(i)$ extends this test to a total of four sensors. We could continue this checking for more than four sensors. But, we did not extend the scheme beyond four because (as discussed earlier), the overhead (in terms of energy spent) to test the redundancy may offset the benefit of energy efficiency.

The above tests for redundancy are implemented by a sensor $i$ before it decides to withdraw itself from further consideration into the set cover $MCSC_2$. However, those
tests only verify the coverage of $i$ by other sensor(s). They do not implement the test of connectivity of the neighborhood of $i$. Recall that the set $MCSC_Q$ must be a connected set cover. So, before removing itself, Sensor $i$ wants to secure the connectivity in its neighborhood. This is implemented in the predicate $ConNbrs(i)$ which is a part of redundancy checking. Ideally, Sensor $i$ needs to check if every pair of its neighbors $j$ and $k$, will remain connected if $i$ is marked redundant and removed. However, if the path between $j$ and $k$ contains any node $l \notin BidirN(i)$, then $i$ cannot verify this path because our solution is strictly localized. So, our implementation of $ConNbrs(i)$ verifies if $j$ and $k$ are connected using some intermediate nodes $l_1, \ldots, l_n$ where all the intermediate nodes are neighbors of $i$.

At this point, the boundary of the query region is partially or completely covered by a network of sensors without much redundancy. Thus, the boundary of the query region is pulled towards the center of the query region. That is, the uncovered area of the query region $R_Q$ is reduced.

**Inside $R_Q$:** Current black nodes (their creation is discussed in the next paragraph) are used to create more black nodes to gradually cover the uncovered region. Future black nodes are selected from the intersections of pairs of existing black nodes. The algorithm considers every intersection of two black nodes. It chooses one or more red nodes from the intersection as the new members of the cover set $MCSC_Q$ (using the predicate $BestCandidate$).

Note that the current black nodes may have been created using the boundary sensors selected earlier. Or, they were created by using the predicate $BestCandidate$ among some sensors inside an intersection of two other black nodes.
Newly selected nodes for the cover set create a new (virtual) boundary of the uncovered query region. Thus, the algorithm reduces the uncovered portion of the query region $R_Q$ by effectively pulling the (virtual) boundary towards the center of the query region.

A sensor $i$ is a possible candidate if it is located inside the intersection of the sensing regions of two other black sensors. As we are dealing with dense sensor network, lots of redundant sensors are expected to be in every intersection of two black sensors. The algorithm uses some checking (Redundant) (explained above), and marks the best candidates as black. The best candidates are those which are closest to the center of $R_Q$ among all the candidates in a particular group. The reason for using this distance to remove redundancy is that the algorithm covers the query region starting from the boundary towards the center. The candidates compare their distances to the center using the predicate BestCandidate. Eventually, every intersection system between two black nodes will be in one of the following two situations:

(i) Node $i$ is the best candidate in its neighborhood inside the intersection because it is the nearest node to the center of $R_Q$ (see BestCandidate). If $i$ is not a redundant node, it will execute Action $A_3$ and turn black.

(ii) Sensor $i$ is one of the sensors (let us refer them by a set $B$) (all in the same intersection) at the same shortest distance from the center of $R_Q$. In this case, all nodes in $B$ will satisfy BestCandidate and turn black (provided they are not redundant) by executing Action $A_3$. Then $B$ is the set of best candidates to be included into $MCSC_Q$.

We will now explain the predicate $SameIntrsectn(i, j)$ in more detail because this tests two critical conditions in the algorithm. As shown below, it tests if both sensors $i$ and
$i$ and $j$ are in the same intersection, the one created by two black sensors $x$ and $y$. This is implemented by using the location information (the variable $Pos$) of the sensors. This first condition is shown below as “Location”. The second condition (marked “Connectivity”) tests the connectivity of $i$. Recall that the main purpose of this predicate is to select the best candidate in the neighborhood (see BestCandidate). We want to ensure that every possible candidate for the best candidate has a bi-directional communication with the black sensors (neighbors) forming the intersection.

$\text{SameIntersect}(i, j) \equiv \text{Interior}(i) \land \text{Interior}(j) \land$

$\left( \exists x, y \in \{N_i \cap N_j\} : (\text{Color}_x = \text{Color}_y = \text{black}) : \text{Pos}_i \in (S_x \cap S_y) \land \text{Pos}_j \in (S_x \cap S_y) \land$

$(i \in N_x \land i \in N_y) \right)$

$\text{Location}$

$\text{Connectivity}$

Termination: As mentioned earlier, the proposed solution is distributed and local. Therefore, no node can directly (locally) verify if $R_Q$ has been fully covered yet. The localized termination detection is implemented in our solution in the following manner: When all actions are disabled at all sensor nodes, the connected sensor cover has been computed, and the algorithm has terminated. All red nodes satisfy the Redundant predicate, so do not satisfy the guard of Action $A_2$. The black nodes satisfy BestCandidate and are not redundant, so will not be able to execute Action $A_3$ any more. Thus, we can claim that our local distributed algorithm is also silent. Note that our solution does not include any explicit termination detection scheme as described above. We explained the closure and convergence of the algorithm just to demonstrate how the algorithm arrives at the final result.
4.4.3 Faults and Recovery

In this section, we focus on the fault handling features of the proposed algorithm (Algorithm 4.4.1). There are two variables used in the solution: $S_i$ and $Color_i$ for a Sensor $i$. So, we need to show that our solution can cope up with all possible corruptions associated with these two variables. As discussed before, the proposed solution is based on a fully localized distributed approach. That made the fault tolerant implementation simple. In the following, we will make an attempt to list all or most important types of faults, and show how they are dealt with in Algorithm $MCSC$.

1. **Wrong initialization of the color variable.**
   
   As discussed in Section 4.4.2, all sensors, if properly initialized start as red.

   (a) **Boundary Sensor.**
   
   Assume that a boundary sensor $i$ starts in black color. If $i$ is not a redundant node, then $i$ remains black (see Action $A_1$). That is, no correction is necessary. If $i$ is redundant, then it will satisfy the predicate $Redundant$, hence the guard of Action $A_2$. Node $i$ then executes $A_2$ and changes its color to red.

   (b) **Interior Sensor.**
   
   Assume that an interior node is initialized as a black colored sensor. If $i$ is not a redundant node, then $i$ remains black (see Action $A_2$). So, no correction is necessary. If $i$ is redundant, then it will satisfy the predicate $Redundant$, executes Action $A_2$ which will change its color to red.

   (c) **Exterior Sensor.**
   
   All exterior sensors must be eventually colored red. If any exterior sensor starts
as black initially, then it will execute Action $A_2$ to change its color to red.

2. *Best candidate sensor's color is corrupted from black to red.*
   Action $A_3$ corrects the color back to black.

3. *A redundant sensor's color changes from red to black.*
   The node, regardless of whether boundary or interior, will satisfy $\text{Redundant}$ and hence, the guard of Action $A_2$. So, it will change its color to red.

4. *Wrong initialization of constants.*
   As discussed in Section 4.4.2, the constants ($R_Q$, $N_i$, and $Pos_i$) are computed by some other protocols. If Algorithm $MCSC$ starts when any of these constants has not been stabilized yet, the connected sensor cover $MCSC_Q$ produced may be incorrect. However, assuming the protocols computing the constants are stabilizing, eventually, the constants would be corrected. Starting from that configuration, in finite steps, Algorithm $MCSC$ will compute $MCSC_Q$ correctly. (Refer to the description in Section 4.4.2.)

5. *Change of values of constants.*
   Assume that the cover set $MCSC_Q$ has been correctly computed with respect to a particular set of input values of the constants. Then due to changes in topology or environment, new obstacles, or occurrence of faults, the input values change. The proposed algorithm will adapt to those changes dynamically in the same manner as described above in Case 4.

6. *Weakening of sensors, both in terms of communication and sensing ability.*
   The power level of sensors will change in time. If we assume that the power is not
replenished, then this change will affect the sensing and communication range of the sensors. In other words, the constant set $N$ and the variable $S$ will change. Recovering from changes of $N$ was described in Cases 4 and 5. Change of $S$ may change the values of $\text{Redundant}$ and $\text{BestCandidate}$. All these changes will be reflected in the change of values of the guards. So, eventually, the color of the affected node will change due to the execution of the actions. All change of colors have already been discussed in earlier cases above.

7. Failure of sensors.

Assume that a sensor $i$ fails due to any reason. This will remove $i$ from the sets $N$ and $\text{Bidir}N$ of its neighbors. Failure of $i$ may also change the coverage and connectivity of its neighboring sensors. This may eventually affect the predicates $\text{Redundant}$ and $\text{BestCandidate}$. So, this situation is similar to Case 6.

4.5 Correctness of the Solution

In this section, we will prove the correctness of Algorithm $\text{MCSC}$ (presented in Section 4.4). That is, our task is to prove that the proposed solution to the connected sensor cover problem satisfies the given specification. The outline of this section is as follows: We will first define a legitimacy predicate of Algorithm $\text{MCSC}$ with respect to the specification of the proposed problem. Then in Section 4.5.1, we will show that any configuration satisfying the legitimacy predicate satisfies the specification of the connected sensor cover problem. We will prove that the set $\text{MCSC}_Q$ produced by the algorithm when the system is in a legitimate state satisfies the coverage and connectivity properties as defined in Specification 4.1. The silent property of our solution will also be shown. Therefore, the set $\text{MCSC}_Q$ does not change
once the system reaches a legitimate configuration. In the next subsection (Section 4.5.2), it will be shown that the algorithm is guaranteed to arrive at a legitimate state regardless of the initial configuration or type of faults occurring in the system. We will use the results established in Sections 4.5.1 and 4.5.2 to prove the self-stabilization and self-* properties of Algorithm $MCSC$ in Section 4.5.3.

**Definition 4.9** The system is considered to be in a legitimate state (i.e., satisfies the legitimacy predicate $L_{MCSC}$) if the following conditions are true with respect to a query region:

(i) All non-redundant boundary sensors are black.

(ii) All non-redundant best candidate sensors are black.

(iii) All other active sensors — exterior, boundary, and interior — are red.

### 4.5.1 Proof of Closure

**Lemma 4.1 (Coverage)** In any legitimate configuration, the connected set cover $MCSC_Q$ computed by Algorithm $MCSC$ completely covers the query region $R_Q$.

**Proof.** By contradiction. Assume that there is an area $A$ which intersects $R_Q$ is uncovered by $MCSC_Q$. By Assumption 4.5, $A$ must contain at least an active sensor $i$ which is obviously red. Then according to the predicate $L_{MCSC}$, $i$ must be a redundant sensor. By the definition of predicate $Redundant(i)$ in Algorithm $MCSC$, $i$ must be covered by some black nodes. Since $i$ was chosen to be any node in the uncovered area $A$, we can claim that all active red sensors in $A$ are covered by some black sensors. Therefore, $A$ is covered, and we arrive at the contradiction. \[\square\]

**Lemma 4.2 (Connectivity)** In any legitimate configuration, the connected set cover $MCSC_Q$ computed by Algorithm $MCSC$ forms a connected graph.
**Proof.** By contradiction. Assume that there exist two node-disjoint connected components in the set $MCSC_Q$. It is obvious from Assumption 4.5 that all active sensors initially form a connected graph. So, the only way for the set being disconnected is by marking some active sensor (say $i$) as redundant such that not considering $i$ as part of the final set $MCSC_Q$ disconnected $i$’s neighborhood.

However, per predicate $ConNbrs(i)$, $i$ is considered to be a redundant node only after ensuring the complete bidirectional connectivity of its neighborhood. That is, if $i$ was marked redundant and colored red by Action $A_2$, all neighbors of $i$ remained connected. In other words, if the set $MCSC_Q$ was connected before $A_2$ was executed, it would remain connected after the execution of the action as well. We reach the contradiction. \hfill $\Box$

**Theorem 4.1 ($L_{MCSC}$ satisfies specification)** Any system configuration satisfying the legitimacy predicate $L_{MCSC}$ (per Definition 4.9) satisfies the specification of the connected sensor cover problem (as given by Specification 4.1).

**Proof.** The coverage and connectivity properties are proved in Lemmas 4.1 and 4.2, respectively. The definition of $L_{MCSC}$ implies that in a legitimate configuration, there exists no redundant black sensor, meaning that all redundant sensors have been identified and marked red. Therefore, the connected cover set $MCSC_Q$ computed at this point is the smallest possible by Algorithm $MCSC$. \hfill $\Box$

**Property 4.1** The system defined by the legitimacy predicate $L_{MCSC}$ is silent.

**Proof.** In any configuration satisfying $L_{MCSC}$, all actions of Algorithm $MCSC$ are disabled. \hfill $\Box$
Lemma 4.3 (Closure) The legitimacy predicate $L_{MCSC}$ is closed.

**Proof.** Property 4.1 asserts the closure of $L_{MCSC}$.

4.5.2 Proof of Convergence

Our obligation in this section is to prove that starting from any arbitrary configuration of the system of sensors, Algorithm $MCSC$ guarantees that in finite steps, the system will reach a configuration that satisfies the legitimacy predicate $L_{MCSC}$. The proof outline is as follows: We first show that starting from an arbitrary configuration, the boundary of $R_Q$ will be covered. Then we establish the progress towards covering the whole region by proving that every black node creates two more black nodes to cover some other uncovered area of $R_Q$. This process is repeated until $R_Q$ is completely covered.

Lemma 4.4 Starting from any arbitrary configuration, the boundary of the input query $R_Q$ will be covered.

**Proof.** By contradiction. Assume that there is an area $A$ which intersects the boundary of $R_Q$ is not covered. By Assumption 4.5, $A$ must contain at least an active sensor $i$. Then if Action $A_1$ is enabled at $i$, $i$ will turn black. Considering other sensors in $A$, $A$ will eventually be covered. That is a contradiction.

Let us assume that $i$ is not enabled to execute Action $A_1$. Then per guard of $A_1$, there are two possibilities:

1. Sensor $i$ is black. So, considering other sensors (like $i$) in $A$, $A$ is covered. That is a contradiction.
2. The predicate $\text{Redundant}(i)$ is true. By the definition of $\text{Redundant}(i)$, it follows that $i$ is covered by black sensors. Again, considering other active sensors in $A$, we obtain that $A$ is covered, hence the contradiction. 

$\square$

**Lemma 4.5** In any configuration, if all boundary and interior red nodes are redundant, then the region $R_Q$ is completely covered.

**Proof.** Assume the contradictory, i.e., although all boundary and interior red nodes satisfy $\text{Redundant}$ predicate, $R_Q$ is not completely covered yet. Consider an area $A$ intersecting $R_Q$ which is not covered. By Assumption 4.5, $A$ must contain at least an active sensor $i$. The color of $i$ cannot be black since $A$ is assumed to be uncovered. So, $i$ is red. Since $A$ is not covered, $i$ will not satisfy $\text{Redundant}(i)$. That contradicts the lemma hypothesis. $\square$

**Lemma 4.6** Every black node covering a region of $R_Q$ will eventually add two neighboring black nodes unless the new nodes are found to be redundant.

**Proof.**

1. Consider a black boundary node $r$. That is, $r$ covers a region on the boundary of $R_Q$. The existence of at least one such node is implied by Lemma 4.4. Assume that $r$ has two black neighbors, $r_1$ and $r_2$. This is a valid assumption because only one black sensor covering a region is very unlikely. Let $I_1$ be the area of intersection between the sensing regions of $r_1$ and $r$. Then in $I_1$, there must exist a node $p$ that satisfies $\text{Best.candidate}(p)$. So, unless $p$ is redundant (i.e., satisfies the predicate...
Redundant(p)), it will execute Action $A_3$ to turn black. Similarly, let $I_2$ be the intersection between the sensing regions of $r$ and $r_2$. So, there must be a sensor $q$ in $I_2$ which will satisfy Best candidate($q$), perform $A_3$, and change its color to black if it is not a redundant node. Note that both $p$ and $q$ are interior nodes.

2. Now, consider a black interior node $r$ covering a region in $R_Q$. By Lemma 4.5 and Case 1 above, a node like $r$ exists unless $R_Q$ is completely covered. Following the same reasoning as in Case 1, we can show that $r$ will add two more black nodes unless the new nodes are marked to be redundant.

□

Lemma 4.7 Starting from an arbitrary configuration, the input query region $R_Q$ will eventually be completely covered by black nodes.

Proof. Assume that $Covered Region_i$ and $Uncovered Region_i$ represent the current covered and yet to be covered region of $R_Q$, respectively. By Lemmas 4.5 and 4.6, there must exist at least one black node $r$ in $Covered Region_i$, which will generate two more black sensors. These new black sensors will cover some portion of $Uncovered Region_i$, effectively reducing the area of $Uncovered Region_i$. Therefore, repeated application of Lemma 4.6 progressively reduces the area of $Uncovered Region_i$. Since $Uncovered Region_i$ is finite, eventually the system reaches a configuration which satisfies one of the following two conditions:

1. $Uncovered Region_i$ becomes an empty set. That is, $R_Q$ is completely covered.

2. $Uncovered Region_i$ is nonempty. By the lemma hypothesis, there are some black nodes in $Covered Region_i$. By Lemma 4.6, a black node will create two more black
nodes. If the newly created nodes are redundant, then by Lemma 4.5, $R_0$ is already covered. But, that is a contradiction.

□

**Theorem 4.2 (Convergence)*** Starting from an arbitrary configuration, Algorithm MCSC reaches a configuration that satisfies the legitimacy predicate $\mathcal{L}_{MCSC}$.

**Proof.** By Lemma 4.7, $R_0$ will be eventually covered. Starting from this configuration, we now prove that the system will reach a configuration satisfying $\mathcal{L}_{MCSC}$. In the following, we will consider the three conditions to be satisfied to satisfy $\mathcal{L}_{MCSC}$.

(i) *All non-redundant boundary sensors are black.*

The proof follows from Action $A_1$.

(ii) *All non-redundant best candidate sensors are black.*

The proof follows from Action $A_3$.

(iii) *All other active sensors — exterior, boundary, and interior — are red.*

Exterior nodes will turn red by applying Action $A_2$. Other nodes if redundant will turn red by executing Action $A_2$. The nodes which are not best candidates (if not already redundant) will eventually become redundant when they will be covered by black nodes, and will turn red (if not already red).

□
4.5.3 Proof of Self-*

We want to conclude the proof of Algorithm $MCSC$ in this section by showing how our solution satisfies some of the self-* properties [158] discussed in Chapter 2. Algorithm $MCSC$ is distributed, self-configuring, self-healing, and scalable. Thus, the proposed self-* solution made the goal of the ubiquitous/pervasive computing a reality since two of the main requirements for this type of large ubiquitous sensor networks are low-power and self-configuring.

4.5.3.1 Self-configuring

The low per node cost will allow the wireless networks of sensors to be densely distributed. The large number of nodes deployed in these systems will preclude manual configuration, and the environmental dynamics will preclude design-time pre-configuration. Therefore, nodes will have to self-configure to establish a topology that provides communication and sensing coverage under stringent energy constraints. Algorithm $MCSC$ is self-configuring in the sense that starting from any initial configuration, it configures itself to form a network topology where all the sensors (members of the connected sensor cover) are able to communicate with each other either directly or indirectly. We also proved that the given query region will eventually be covered starting from any arbitrary state. Hence this algorithm is self-configuring.

4.5.3.2 Self-healing

Self-reconfiguration (self-healing) is a concept inherent to the nature of wireless sensor networks. The proposed algorithm is self-healing under various perturbations, such as node joins, failures (due to crash and energy loss), state corruptions, and weakening of power. If
a node fails, Algorithm MCSC heals itself in the following manner: If it is not a redundant node, then a part of the query region $R_Q$ may become uncovered. In that situation, a subset of its (active) neighbors will take over by executing Action $A_1$ or $A_3$. Similar process may be initiated, if necessary, when a node's power weakens to the point that it affects the node's communication ability. On the other hand, if a node joins the network (after recovering, being repaired, or being re-energized in power), it would be considered as a redundant node since the query region is already covered by the existing nodes. So, the node joining event will not change the connected set cover. Arbitrary corruptions of state variables of the nodes are also dealt with in the solution — change of Color variable due to faults is fixed in a very simple manner.

4.5.3.3 Self-*

We have implemented the self-configuring and self-healing features in our solution (Algorithm MCSC) using the concept of self-stabilization [39, 158]. As explained in Section 2.3 in Chapter 2, the paradigm of self-stabilization subsumes all other self-* properties. Thus, our solution is truly fault-tolerant in terms of self-* feature.

**Theorem 4.3** Algorithm MCSC is self-stabilizing.

**Proof.** The proof follows from Theorem 4.1, Lemma 4.3, and Theorem 4.2. ☐

4.6 Complexity and Simulation

We are yet to perform an extensive cost analysis of our solution. Here, we will sketch on our current findings. In terms of space, Algorithm MCSC is an optimal solution. Recall the problem specification. Per Note 4.1, upon termination of the sensor cover algorithm,
a node will know if it should be active or passive. So, it must use at least two states to distinguish its two possible roles. Our solution uses exactly two states in the Color variable to implement this. In Algorithm MCSC, every node maintains its neighborhood set $N$. The problem requires implementing a strongly connected graph. So, maintaining $N$ is essential. This costs an $O(\Delta)$ states per node, where $\Delta$ is the maximum degree of a node in the network.

We have not completed the stabilization time analysis. However, once the system is stabilized, meaning the connected sensor cover has been correctly computed, a fault (like a crash, power failure, memory corruption, etc.) can be corrected in the immediate neighborhood of the faulty node. In this sense, the proposed solution is self-containing [55], meaning the faults do not spread more than one hop away. This shows a very important and desirable property for large scale wireless sensor networks.

As mentioned earlier, computing a minimum sensor cover is NP-hard. Theoretical analysis of suboptimality is not done yet. Philippe Raipan-Parvédy at IRISA/Université Rennes 1, France [35] has simulated our algorithm using Java. Some parameters can be provided to the simulator such as the number of nodes and ranges like sensing, communication. The region used for testing is a $12 \times 12$ grid deployed with 244 sensors. The distance between any two sensors is 4 units, and the sensing region of each sensor has a radius of 9 units. In the screenshot (shown in Figure 4.6.2), the circle with the white boundary indicates the query region $R_Q$. The graph shown in the screenshot represents the connected sensor cover computed by Algorithm MCSC. So, the sensors at the vertices of the graph are the only black nodes selected by the algorithm after eliminating some redundant sensors. Simulation results look very good — the size of the connected sensor cover computed is not very high,
i.e., the algorithm successfully removed a good percentage of the redundant nodes.

Figure 4.6.2: A screenshot from the simulation of Algorithm $MCSC$.

4.7 Scalable and Localized

Scalability and localized are two highly desirable properties of large scale wireless sensor networks. Algorithm $MCSC$ is scalable and localized in the following terms (see Section 4.6): Firstly, it is a space optimal solution in terms of the number of states per node — only two states per node. Secondly, faults are contained within the immediate neighborhood. Thus, the solution is not only self-healing and self-stabilizing, but is self-containing. Thirdly, the
solution approach is strictly local. Nodes communicate only with their immediate neighbors during both self-configuring and self-healing.

4.8 Extensions

Sensing coverage characterizes the monitoring quality provided by a sensor network in a designated region. Different applications may require different degrees of sensing coverage. While some applications may only require that every location in a region be monitored by one node, other applications like distributed detection require significantly higher degrees of coverage [140]. Algorithm $MCSC$ was designed with the goal of achieving optimality in terms of the number of sensors while maintaining the degree of coverage of only one. The impact of the above assumption (of one-coverage) is the following: Our solution looks for every opportunity to remove any active redundant node from the cover set. This may cause some areas in the query region covered by a single (black) sensor. Although this is an ideal solution from the point of view of the minimum possible sized cover set, but this makes the query region very weak in terms of sensing failures. That is, even if a single active sensor fails, some area may become uncovered right away. Recall that Algorithm $MCSC$ is self-stabilizing with respect to all types of transient faults. So, our solution will cover the uncovered area very quickly. Thus, no area will remain uncovered forever. However, if we had designed out solution with the aim of maintaining higher degree of coverage, a single fault may not have caused any area uncovered even temporarily. In this regard, we can extend our solution in a couple of ways. Firstly, we may write a parametric solution where the input query will include the degree of coverage expected. The predicate $Redundancy$ will be relaxed to allow the corresponding higher degree of coverage. Secondly, we can simply assume a particular degree ($>1$) of coverage in our algorithm. Unfortunately, higher degree
of coverage would require more communication cost, i.e., consuming more power. We can conduct a study on the tradeoff between cover size optimality vs. robustness.

In solving the connected sensor cover problem in this thesis, our main goal was to achieve the optimality in terms of the cardinality of the cover set. However, that was not the only criterion we used in writing our solution. It was kept in mind that the redundancy checking itself should not be very power intensive computation. In other words, the overhead in terms of energy spent in computing the connected cover set should not offset the benefit of finding the small cover set. One possible extension of our research is to study the impact of the power consumption of the sensor cover algorithm versus the size of the cover set. For example, in Algorithm MCSC, the redundancy check was limited to the finding the answer to the following question: “Is a node covered by any combination of three other nodes?” If we had extended the test to more than three nodes, we probably could see a better solution.

Another interesting way to extend our work is to implement different degrees of connectivity in the final communication graph produced by the connected sensor cover algorithm. In the current solution, the predicate ConNbrs implements the connectivity. It guarantees that all the neighbors of a node i are strongly connected before marking i as redundant. Similar to the implementation of higher degree of coverage to achieve better robustness, we may also require higher degree of connectivity for the same purpose (i.e., to increase the level of fault-tolerance). We can extend the neighborhood connectivity checking to k-node (k > 1) disjointness in the communication graph. As in the case of better coverage, we cannot ignore the power consumption. So, the above study would involve investigating the impact of better fault-tolerance on the energy spent.

The Steiner tree problem is a minimum interconnection problem. Given a finite set $V$
of points (called terminals), the Steiner tree problem is to find a minimum length inter-
connection of those terminals according to some geometric distance metric. The resulting
interconnection is a tree, called a Steiner minimal tree. The Steiner problem has a wide
variety of applications ranging from efficiently organizing power grids to designing multicast
routing systems. Though this problem is NP-complete, several heuristic algorithms have
been designed to approximate the results [16]. Another approach to solving the connected
sensor cover problem is the following which is based on Steiner tree: The algorithm would
require two phases. In the first phase, a sensor cover is computed without considering the
connectivity criterion. In the next phase, a Steiner tree is constructed on the sensor cover
to connect the nodes in the cover set. This approach may have two negative impacts. First,
the set cover produced finally at the end of the second phase may have more redundant
nodes than the approach used in our solution. Second, computing the Steiner tree may
incur higher communication cost. However, we can study implementation of various types
of heuristic methods used in constructing Steiner trees in our common setting — local,
distributed, and self-stabilizing.
CHAPTER 5

CONCLUSION AND FUTURE RESEARCH

This research started with studying the ubiquitous/pervasive computing. Wireless sensor networks made the ubiquitous computing a reality. So, our research continued with the study of large scale wireless sensor networks. We studied many applications of the sensor networks. We also extensively surveyed existing protocols written for sensor networks.

Constraints of designing sensor networks are well-known. The most important of which is the energy. However, designing reliable sensor network is still highly desirable. Considering the size, frequency of topology changes, and changes of power level in sensor networks, designing self-organizing, self-configuring or self-healing sensor networks is essential. This influenced us to research and write an extensive survey on self-* systems.

The main motivation of our research was to design a self-* query response system in sensor networks. We presented the first local, distributed, scalable, self-* design of the connected sensor cover problem introduced in [63]. We presented a stabilizing solution to the problem and showed how the solution is self-organizing and self-healing as well. Throughout the design process, we followed a power-aware approach. Although our goal was to design an optimal size sensor cover, but we used the power-awareness as a strong guide in our design, and accepted a slight degree of suboptimality. Our simulation results show that the size of the cover set is very close to the optimal size. Algorithm $MCSC$ is
space optimal — only two colors are used. The solution is fully localized and distributed, and highly scalable. Once the system is stabilized, the faults can be corrected in their neighborhood. Hence the system is self-containing.

This research showed that the concept of self-stabilization subsumes many other self-\* properties. We were also able to demonstrate the power of self-stabilization to achieve quite an elegant and efficient solution to a very practical problem in a large scale wireless sensor network. Our research should also encourage the pervasive community and MANET community to look into designing large networks using self-stabilization.

There are ample opportunities to explore several issues related with the topic of this thesis. As said earlier, we wanted to design a strictly localized algorithm for its obvious benefit. One can study if relaxing the locality assumption to some extent can produce a more optimal connected sensor cover. However, it would probably consume more energy.

The connected sensor cover problem is just one example of the query response system. A future topic of research would be to find other types of query handling problems in sensor networks, and see if it is possible to design self-\* solutions to those problems. We used the shared memory model to write our algorithm. It would be interesting to see how efficiently we can write the corresponding message passing solution.

Although the early simulation results look good, we would like to run a thorough simulation study and extensive theoretical analysis of our algorithm. One of the most important results to be obtained is the ratio of size of the computed sensor cover vs. that of an optimal solution.
BIBLIOGRAPHY


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


VITA
Graduate College
University of Nevada, Las Vegas

Preethi Linga

Home Address:
9721 Northern Dancer Drive
Las Vegas, NV 89117

Degrees:
Bachelor of Science, Computer Science, 1999
Andhra University, India

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

Thesis Examination Committee:
Chairperson, Dr. Ajoy K. Datta, Ph.D.
Co-Chairperson, Dr. Maria Gradinariu, Ph.D.
Committee Member, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. John T. Minor, Ph.D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.
Committee Member, Dr. Emmanuelle Anceaume, Ph.D.