CHALLENGES AND SOLUTIONS FOR LOCATION-BASED ROUTING IN WIRELESS SENSOR NETWORKS WITH COMPLEX NETWORK TOPOLOGY

A Dissertation

by

MYOUNGGYU WON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Radu Stoleru
Committee Members,	Jennifer L. Welch
	Anxiao Jiang
	A. L. Narasimha Reddy
Head of Department,	Duncan M. Walker

August 2013

Major Subject: Computer Science

ABSTRACT

Complex Network Topologies (CNTs)–network holes and cuts–often occur in practical WSN deployments. Many researchers have acknowledged that CNTs adversely affect the performance of location-based routing and proposed various CNT-aware location-based routing protocols. However, although they aim to address practical issues caused by CNTs, many proposed protocols are either based on idealistic assumptions, require too much resources, or have poor performance. Additionally, proposed protocols are designed only for a single routing primitive–either unicast, multicast, or convergecast. However, as recent WSN applications require diverse traffic patterns, the need for an unified routing framework has ever increased.

In this dissertation, we address these main weaknesses in the research on location-based routing. We first propose efficient algorithms for detecting and abstracting CNTs in the network. Using these algorithms, we present our CNT-aware location-based unicast routing protocol that achieves the guaranteed small path stretch with significantly reduced communication overhead. We then present our location-based multicast routing protocol that finds near optimal routing paths from a source node to multicast member nodes, with efficient mechanisms for controllable packet header size and energy-efficient recovery from packet losses. Our CNT-aware convergecast routing protocol improves the network lifetime by identifying network regions with concentrated network traffic and distributing the traffic by using the novel concept of virtual boundaries. Finally, we present the design and implementation details of our unified routing framework that seamlessly integrates proposed unicast, multicast, and convergecast routing protocols. Specifically, we discuss the issues regarding the implementation of our routing protocols on real hardware, and the design of the

framework that significantly reduces the code and memory size to fit in a resource-constrained sensor mote. We conclude with a proactive solution designed to cope with CNTs, where mobile nodes are used for "patching" CNTs to restore the network connectivity and to optimize the network performance.

DEDICATION

To my family.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Radu Stoleru, for his endless support, encouragement, and guidance throughout the course of my Ph.D. study. I appreciate that he gladly accepted me to take my first step for becoming a researcher under his guidance. His patience allowed me to make steady but consistent progress. Whenever I was stuck on a problem, he has been always with me to give precious advice to help me move forward. Without his help, I would not have completed this work.

I would also like to thank my committee members, Dr. Jennifer Welch, Dr. Anxiao Jiang, and Dr. Narasimha Reddy for their insightful comments and support. Without the knowledge I learned from their classes and their constructive comments, this work would not have been possible.

My thanks also go to my lab colleagues, Harsha Chenji, Amin Hassanzadeh, Mahima Suresh, Wei Zhang, and Jay Chen, for gladly taking their time for discussion and collaboration and making my life at LENSS Lab pleasurable.

I am also very grateful to my wife, Hyunjung Lim, and my precious princess, Lucy Won, who have been always there for me with their love and understanding. Last, but not least, I thank my parents, Kyungjae An and Hyunseob Won, and my brother, Chulgyu Won, for giving me the strongest support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Motivation

The Wireless Sensor Network (WSN) research community has identified and addressed many important problems for WSNs such as localization, boundary detection, relay-node placement, routing, and data aggregation. A common assumption for many of these work is a "uniform network deployment" where a target area is appropriately covered with sensor nodes. However, in practical deployments, we often encounter regions without deployed nodes (*network holes*). For example, environmental obstacles like buildings and lakes prevent us from deploying nodes; random deployment (e.g., deployment from ground/air vehicles) causes irregular node distribution; in some applications, hostile users may destroy part of deployed nodes, and environmental factors such as wind may relocate deployed nodes, creating network holes. Network holes, when they are large in size, may even disconnect the network, causing *network cuts*. These network holes and cuts are referred to as *Complex Network Topologies (CNTs)*. CNTs are the motivation of this study.

It is well known that CNTs adversely affect the performance of location-based routing protocols – CNTs cause arbitrarily long routing paths, unbalanced energy consumption (i.e., reduced network lifetime), and increased packet loss rates [1, 2]. To address these issues, many researchers have proposed various location-based routing protocols designed for WSNs with CNTs, encompassing three main routing primitives: 1-1 (unicast) [3, 1, 2], 1-n (multicast) [4, 5, 6], n-1 (convergecast) [7]. However, although they aim to address this practical issue caused by CNTs, many proposed protocols are either based on idealistic assumptions, require too much resources when implemented on real hardware, or have poor performance. For example, some algo-

rithms [8, 9, 2] are based on Unit Disk Graph (UDG) radio model. Some protocols are designed for continuous space domain [1]. Some protocols do not consider the limited packet size [5, 4]. Other protocols use a complex geometric algorithm which requires a large amount of memory space [3]. Even the current hardware implementation for a simple geographic forwarding algorithm, with face routing [10], requires relatively large memory space, barely fitting into a contemporary wireless sensor node [11].

Another important set of issues is that existing location-based routing protocols have been developed only for a single routing primitive (i.e., one of unicast, multicast, and convergecast). To the best of our knowledge, there is no unified location-based routing framework that integrates location-based routing protocols developed for each routing primitive. However, the needs for an unified routing framework has ever increased recently. In recent wireless sensor network applications [12, 13, 14, 15], nodes do not just report data to the sink anymore. Peer-to-peer communication between nodes has become an essential communication method for in-network processing, data aggregation, and feedback control. For example, a wireless structural control, one of the promising cyber physical systems [14] uses peer-to-peer communication for feedback control; in disaster response applications [12], sensor nodes communicate with each other for efficiently monitoring victims and buildings. In addition, nodes not only communicate with peer nodes – they perform peer-to-peer communication with heterogeneous devices like mobile phones or even electronic gadgets [13, 15]. Nodes, in some instances, send packets to a particular set of nodes for various purposes such as code-update or mission assignment. For example, in a recent application for Smart Building [16], a multicast routing primitive is used for controlling multiple actuators in a building.

1.2 Dissertation Statement

To address the aforementioned main weaknesses of location-based routing in WSNs, in this dissertation, we propose a suite of algorithms and protocols. Our thesis is that proposed algorithms and protocols focus on achieving the following goals.

- **Energy-efficient**. Developing energy-efficient protocols is of paramount importance for resource constrained WSNs. Therefore, proposed solutions must be energy-efficient. The energy-efficiency of the solutions must be clearly validated.
- **Scalable**. As one of the main advantages of using location-based routing is its scalability, proposed solutions must also be scalable. Proposed solutions must conveniently work for large-scale sensor networks.
- **Practical**. Schemes that have been evaluated through only simulations may have little impact, although they are theoretically interesting. Therefore, proposed algorithms and protocols must be able to run on resource-constrained real hardware.
- **Modularized**. Proposed solutions must be designed such that the integration of the solutions can be easily done to achieve our goal of creation of the first unified location-based routing framework.

1.3 Main Contributions

In light of the objectives described above, the contributions of this dissertation can be summarized as follows: we first present efficient CNT detection and abstraction algorithms. In particular, we focus on the novel concept of peer-to-peer network cut detection that enables nodes to detect a network cut with respect to any destination node; we also address practical issues for contemporary boundary detection

and abstraction algorithms arising when implemented on real hardware. Using these CNT detection and abstraction algorithms, we propose a location-based unicast routing protocol that achieves the smallest path stretch without relying on unrealistic assumptions like UDG radio model [2] and precise time synchronization [1], and too costly assumptions like the "path set-up" process [3]. In particular, we show that our solution significantly reduces the communication and storage overhead by eliminating the path-set-up process and by not relying on the construction of the global visibility graph [3]. We then present a location-based multicast routing protocol which finds near optimal routing paths from a source node to multicast members. Our multicast protocol allows applications to control packet header size as a parameter. The local membership management mechanism of our multicast routing protocol allows more energy-efficient recovery from packet loss. Additionally, our multicast routing protocol considers CNTs in the network – when CNTs are detected, the boundary information of detected CNTs is abstracted. The abstract information is sent to the source node and the source node updates routing paths to multicast members. Next, we present our location-based convergecast routing protocol. The main motivation of the development of our location-based convergecast routing protocol is that existing solutions do not take into account the energy hole problem caused by CNTs in the network. However, the energy hole problem significantly degrades the network lifetime. To improve the network lifetime, our convergecast protocol identifies regions with concentrated network traffic caused by CNTs and distributes the traffic by using a novel concept of virtual boundaries.

We then introduce the design and system implementation of the first unified location-based routing framework that integrates our proposed routing protocols for unicast, multicast, and convergecast. A salient feature of the framework is not only that it has real-world implementation of proposed algorithms and protocols, but more

importantly, it has the design that allows each component for the framework to share as much functionalities as possible so that the resulting code and memory sizes are significantly reduced, enabling us to fit the framework into a resource constrained sensor mote. We implemented the framework on real-hardware and performed extensive experiments on a testbed consisting of 42 TelosB motes.

Finally, we propose a more proactive approach where mobile nodes are used to "patch" CNTs in the network. In particular, we use mobile nodes to restore the connectivity of a partitioned network. Unfortunately, the state-of-the-art solutions [17][18] minimize only the number of used mobile nodes for restoring the connectivity of the network. However, we show that the resulting network performance (i.e., in terms of average path length from all nodes to the sink node) must also be considered, because in some cases minimizing only the number of used mobile nodes gives very poor performance. Thus, we formulate the problem of using mobile nodes to restore the network connectivity focusing on optimizing two criteria: the number of used mobile nodes as well as the resulting network performance. To solve the problem, we propose both centralized and distributed solutions.

1.4 Organization

This dissertation is organized as follows. In Section 2, we present our CNT detection algorithms. We then propose location-based unicast, multicast, and convergecast routing protocols designed for WSNs with CNTs in Section 3, 4, 5, respectively. In Section 5, a CNT-aware location-based unified routing framework is presented that integrates proposed routing protocols for unicast, multicast, and convergecast. In Section 6, we discuss a more proactive approach that uses mobiles nodes to patch CNTs. We conclude our dissertation in Section 7.

2. CNT DETECTION

In this section, we present algorithms and protocols for network cut detection and hole detection (also known as boundary detection). In particular, our novel peer-to-peer cut detection (P2P-CD) scheme is presented in details. For boundary detection, we discuss practical issues encountered during the implementation of the state-of-the-art location-based boundary detection algorithm on real hardware and propose our solutions for the issues.

2.1 Network Cut Detection

One of the challenges in Wireless Sensor Networks (WSNs) is to ensure that the network is connected. The connectivity of the network can easily be disrupted due to unpredictable wireless channels, early depletion of node's energy, and physical tampering by hostile users. Network disconnection, typically referred to as a *network cut*, may cause a number of problems. For example, ill-informed decisions to route data to a node located in a disconnected segment of the network might lead to data loss, wasted power consumption, and congestion around the network cut.

Several centralized algorithms have been proposed to efficiently detect a cut [19, 20, 21, 22]. These algorithms attempt to detect a cut by assigning the task of network connectivity monitoring to a subset of nodes. In particular, Shrivastava et al. [22] proposed an algorithm to detect a linear cut in a WSN, by strategically deploying specially designated nodes, called sentinels. Some researchers have recently proposed distributed cut detection algorithms for WSNs [23, 24, 25]. In these schemes, each

*Reprinted with permission from "A Destination-based Approach for Cut Detection in Wireless Sensor Networks" by Myounggyu Won and Radu Stoleru, 2013. *International Journal of Parallel Emergent and Distributed Systems*, Volume 28, Issue 3, Pages 266-288, Copyright 2013 by Taylor & Francis.

sensor node is able to autonomously determine the existence of a cut. A common aspect of existing cut detection algorithms is that they focus on a "binary problem": is there a cut in the network, or not? However, this may not be sufficient since, in some applications, despite the existence of a cut somewhere in the network, a sender can still communicate with a target node, if they are not disconnected by the cut. For example, some WSN applications adopt the strategy to deploy multiple sink nodes, in order to improve throughput and prolong network lifetime [26, 27]. In these applications, detecting a cut with respect to one sink node does not necessarily mean that nodes in the disconnected network segment should refrain from reporting data, because they may send the data to other connected sink nodes.

In this section, we propose solutions for a more general cut detection problem – the *destination-based cut detection* problem. Unlike the traditional cut detection problem, we attempt to find a network cut between a sender and any node in a set of given destinations. We first propose Point-to-Point Cut Detection protocol (P2P-CD). P2P-CD allows a source node to identify a cut with respect to any destination node. In this protocol, the boundary of a cut is compactly represented as a set of linear segments. The compact representation of a cut allows the information on existing cuts (i.e., the shape and location of the cut) to be efficiently distributed throughout the network with small overhead. A source node, using the distributed information, locally determines whether any given destination is reachable or not.

P2P-CD is a reactive algorithm; in other words, a cut is reactively detected in contrast to the proactive solutions that periodically probe the network for potential cuts; thus, P2P-CD is energy efficient. However, the energy efficiency comes at the cost of overhead: each node has to store a data structure that contains the information on the cuts in the network. Thus, we also propose a lightweight cut detection algorithm, Robust Energy-efficient Cut Detection for Multiple Sinks (RE-

CDM), particularly designed for the scenario, where nodes need to detect a cut with respect to a small number of destinations instead of *any* destination. This scenario typically arises in WSN applications with multiple sinks. RE-CDM allows each sensor node to monitor the connectivity to multiple sink nodes in real time. RE-CDM is a proactive cut detection algorithm, being less energy efficient than P2P-CD. However, it does not require nodes to be localized, and nodes do not need to store data on the partial global topology, which makes a good fit with resource constrained nodes in WSNs.

2.1.1 Related Work

Many researchers stressed the importance of network partition monitoring problem [28, 29, 30, 31]. Chong et al. [29] considered the problem as a security issue, mentioning that cuts can be intentionally created in a hostile environment, and nodes must detect them. Cerpa and Estrin [30], in their self-configuring topology scheme, emphasized that the cut detection problem is potentially crucial in many WSN applications, but left it as future work.

The cut detection problem was first considered in a wired network [19]. Kleinberg et al. [19] introduced the concept of (ϵ, k)-cut, which is defined as a network separation into two sets of nodes, namely $(1-\epsilon)n$ nodes and ϵn nodes (n refers to the total number of nodes), caused by k independently disabled edges. A set of *agents*, denoted by a set D, is strategically deployed in the network to detect the (ϵ, k)-cut. Each agent exchanges a control packet with other agents periodically. A cut is assumed to be present if the control message loss exceeds some threshold. The authors are interested in the size of D, and prove that the size of the set D is $O(k^3 \frac{1}{\epsilon} log \frac{1}{\epsilon} + \frac{1}{\epsilon} log \frac{1}{\delta})$ to detect (ϵ, k)-cut with probability $1-\delta$. Ritter et al. [21] proposed a cut detection algorithm where a sink node broadcasts an *alive message*.

A cut is detected by *border* nodes, which are located on the border of network, if these nodes fail to receive the alive message more than a certain number of times.

Shrivastava et al. [22] recently introduced a protocol to detect a cut in wireless sensor networks. Their work is largely based on [19]. The protocol deploys *sentinels*, a counterpart of *agents* in [19], to detect ϵ-cut, which is defined as a linear cut that separates the network into two parts, where one part has at least ϵ-fraction of total nodes. The paper aims to minimize the number of sentinels based on the assumption that in sensor networks, linear-shaped or other geometric shaped cuts are more likely to happen, rather than the cut with k independent edge failures. They prove that $O(\frac{1}{\epsilon})$ sentinels are required to detect ϵ-cut with $\epsilon < 1$. The limitations of their cut detection algorithm is that they consider only the linear cuts, being unable to detect arbitrarily shaped cuts. Additionally, their algorithm is a centralized solution, requiring global topology information.

Barooah et al. [24, 25] addressed the issues that previous cut detection algorithms have. The Distributed Source Separation Detection (DSSD) algorithm is fully distributed and detects arbitrarily shaped cuts. A positive scalar value, called *state*, is maintained by each node. The state of each node is updated based on the states of its immediate neighbors. If a node is connected to a sink, its state converges to some positive value. Otherwise, its state converges to zero. The DSSD algorithm, however, suffers from control message overhead, since the algorithmic iterations for the convergence depends on the degree of the network. Won et al. [23, 32] introduced an energy efficient solution called Robust Energy-efficient Cut Detection (RE-CD) that minimizes the iteration count for the convergence, thereby minimizing the control message overhead. The main idea is to run the DSSD algorithm on the overlay network consisting of a small number of representative nodes, called *leaders*. The degree of the overlay network is at most 4, allowing the minimal convergence rate.

Table 2.1: Taxonomy for cut detection schemes.

Cut detection scheme	criterion 1	criterion 2	criterion 3
Flooding	k_1	1	proactive
Linear Cut Detection [22]	k_2	1	proactive
DSSD [24]	N	1	proactive
RE-CD [23]	N	1	proactive
RE-CDM	N	k_3	proactive
P2P-CD	N	N	reactive

However, these algorithms detect cuts with respect to only a single sink node.

Systematically organizing the previously introduced cut detection algorithms not only helps better understand our contributions, but also provides guidelines for future research on this topic. We categorize the algorithms based on three criteria. First, we consider which nodes detect a cut. Some algorithms allow only a small subset of nodes to detect a cut, whereas some algorithms allow all nodes in the network to detect a cut. Second, we consider that, with respect to which nodes, a cut is detected. Such "target" nodes might be the sink node, or for some algorithms, all nodes. The last criterion describes whether the cut detection algorithm is proactive or reactive. The proactive solution periodically checks the existence of a cut, whereas the reactive solution runs the algorithm only when there is a cut; thus, a reactive solution is a more energy efficient scheme.

Table 2.1 summarizes the taxonomy for existing cut detection algorithms. As shown, the most basic type of cut detection algorithms is the flooding. In this scheme, a sink node periodically broadcasts a probing packet throughout the network so that the receivers can check the network connectivity to the sink. One drawback of this scheme is that the nodes in the connected network segment cannot detect a cut; only the k_1 number of nodes in the disconnected network segment can detect a cut. The Linear Cut Detection algorithm proposed by Shrivastava et. al [22] significantly

reduces the message overhead caused by broadcasting the probing packet throughout the network, by allowing only a small subset of nodes, called the sentinels, participate in the cut detection process; thus, in this scheme, the k_2 sentinels detect a cut with respect to a sink node. The DSSD algorithm [24] operates in a distributed manner and allows all the N nodes in the network to detect a cut with respect to a sink node. RE-CD [23] algorithm improves the energy efficiency of the DSSD by minimizing the convergence rate of the DSSD algorithm. Despite its better energy efficiency, this algorithm still permits all the nodes in the network detect a cut with respect to a sink node.

At this point, we note that existing algorithms focus on detecting a cut with respect to a single node, the sink node, and furthermore they are all proactive solutions. Two proposed algorithms in Section 2 extend the notion of existing cut detection; specifically, we extend the number of "target" nodes. First of all, RE-CDM is designed to enable nodes to detect a cut with respect to k_3 sink nodes. This algorithm is distributed, but a proactive solution. Our reactive cut detection algorithm, P2P-CD then further extends the second criterion, the number of target destinations, to all the N nodes in a network at the cost of several requirements: 1) each node has to be localized; 2) additional storage space is required; and 3) part of global topology information must be known to the nodes in the network. Despite these requirements, this algorithm, is reactive, thereby being energy efficient.

In sum, P2P-CD realizes the concept of peer-to-peer cut detection. In other words, in P2P-CD, a source node can detect a cut with respect to any destination; the RE-CDM is a more lightweight solution that does not rely on the space and implementation overhead, which suits well for the applications that have a small number of target nodes, such as the applications for WSN with multiple sinks.

2.1.2 Preliminaries and Problem Formulation

We consider a two dimensional network, represented as a connected graph $G_V = (V, E)$, where $V = \{v_1, v_2, \cdots, v_n\}$ is a set of deployed sensor nodes, and E represents a set of links between nodes in V. We denote a set of sink nodes (i.e., base stations) by $S = \{s_1, s_2, \cdots, s_n\}$, $S \subseteq V$. We assume that each node knows its location either from an onboard Global Positioning System (GPS), or by employing node localization protocols [33]. We also assume that a location-based routing protocol is available, such as [10, 34]. A set $N_i \subseteq V$ denotes the immediate neighbors of a node $v_i \in V$. The term $C_v(G)$ represents the connected component of graph G that contains a vertex v. From here on we will use the terms "source" and "destination" for the sender/receiver pair of a unicast communication. As it will become clear later, destination nodes can be either sink nodes (i.e., base station), or peer nodes.

Now we are ready to formally define the "Destination-based Cut Detection" problem: Consider a set of destinations, denoted by $T = \{t_1, t_2, \cdots, t_n\}$, where $T \subseteq V$. How can a source node $v_i \in V$ determine whether any given destination $t \in T$ is in $C_{v_i}(G_V)$, in an energy efficient manner? Informally, we aim to develop energy efficient protocols that allow a node to find its connectivity to any node $t \in T$.

Before presenting our solutions for the destination-based cut detection problem, we briefly describe some background materials. The DSSD algorithm [24] monitors the connectivity of a node to a single sink, say $s_1 \in S$. For ease of presentation, we assume that $v_1 \in V$ is s_1. Each node $v_i \in V$ maintains a positive scalar $v_i(k)$, called the *state*, which is updated at each iteration of the algorithm as the following, where k refers to the iteration counter.

$$v_i(k+1) = \frac{\sum_{v_j \in N_i} v_j(k)}{|N_i| + 1}.$$

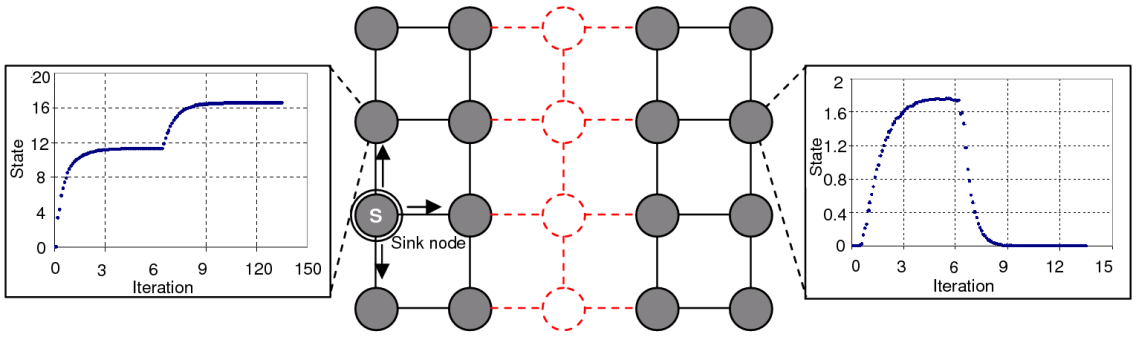


Figure 2.1: An illustration of DSSD algorithm.

The state of a sink node v_1 is updated slightly differently as the following.

$$v_1(k+1) = \frac{\sum_{v_1 \in N_1} v_1(k) + \omega}{|N_1| + 1}.$$

Here the variable ω, called the "sink strength", is a system parameter. The algorithm proceeds in iterations, and each node updates its state. If there is no cut in the network, the state of a node converges to some positive value, otherwise, the state rapidly converges to 0, allowing a node to detect a cut. Figure 2.1 illustrates how DSSD algorithm works. When a node is connected to the sink, its state converges to some positive value. The four nodes in the middle marked as red dotted circles die at iteration 6, creating a cut. After the cut occurs, the state of the node connected to the sink converges to new convergence value (i.e., from approximately 11 to 17); the state of the node that is disconnected from the sink converges to 0, being able to detect the cut.

The RE-CD [23] algorithm was proposed for reducing the overhead of DSSD. In RE-CD, as shown in Figure 2.2, a network is divided into a grid of clusters. In each cluster a leader is elected. In particular, the sink becomes a leader for the cluster that it belongs to. The DSSD algorithm is then executed on the virtual grid network consisting of the leaders (represented as triangles in Figure 2.2) and the virtual

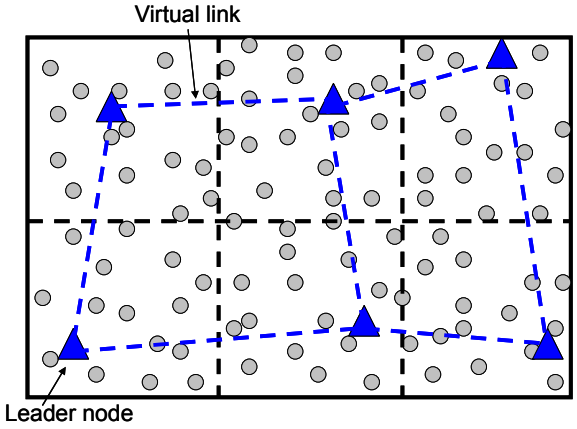


Figure 2.2: An illustration of the virtual grid network.

links between them. RE-CD minimizes the convergence rate of the DSSD algorithm, because the number of neighbors for each leader is at most 4 due to the grid network topology (note that the convergence rate of the DSSD algorithm depends on the maximum degree of the network [24]). RE-CD is energy efficient, because only a subset of nodes participate in the cut detection process.

2.1.3 Main Ideas

Before presenting the details, we first overview the two proposed algorithms with the tradeoffs between them. Both algorithms are designed for detecting cuts with respect to a given set of destinations. Our first protocol, Point-to-Point Cut Detection (P2P-CD), is designed to provide a solution that enables each node to determine connectivity to any destination. Note that a cut partitions a network into multiple network segments; we call such network segment a *cut region*. P2P-CD is based on the knowledge of partial global topology: it uses the shape and location of a cut region. An important issue for this algorithm is thus to compactly, yet precisely, represent the boundary of a cut region. Figure 2.3 illustrates the general idea on how P2P-CD works. There is a cut in the middle of the network separating the

network into two cut regions, denoted by A and B. A packet sent by source node S reaches the boundary of the cut, v_{init} in the middle of the figure. This node initiates the boundary abstraction process. Then, the boundary of a cut region is represented as a set of line segments. By connecting the line segments, P2P-CD yields a set of vertices of a polygon covering the cut region. The locations of the vertices of the polygon are distributed to the nodes in the cut region. Based on the set of received polygons (there might be multiple cuts in the network), nodes determine whether a given destination is reachable or not. The second cut detection protocol we propose, RE-CDM, is suitable for scenarios in which the number of target destinations is small (e.g., a set of a few sink nodes). RE-CDM can be used by sensor nodes to autonomously determine connectivity to multiple sink nodes. This protocol does not require global topology information, nor the location information, thereby reducing the space and implementation overhead. However, it is suitable only for the applications with the small number of target destinations, because its overhead grows with the increasing number of destinations.

2.1.4 Point-to-Point Cut Detection

The point-to-point cut detection (P2P-CD) protocol enables each node in a network to determine the connectivity to any destination. This protocol executes in two main steps. In the first step, the *Cut Boundary Abstraction*, the boundary of a cut region is identified and represented as a polygon $P = \{p_1, p_2, \cdots, p_n\}$, where each element of P is the location of a node that represents the vertex of P. Consider Figure 2.3 for an example. In this figure, the polygon corresponding to the cut region A is $P = \{v_5, v_7, v_8, v_9\}$. After the polygon P is identified, it is broadcast to the nodes in the cut region corresponding to the polygon P. In the second step, the *Cut Detection* phase, nodes determine whether a destination is reachable based on

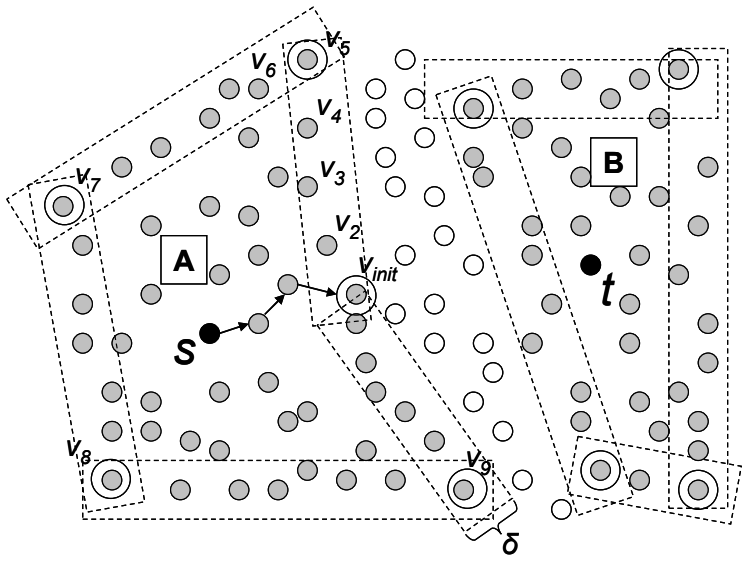


Figure 2.3: An illustration of the P2P-CD algorithm.

the following available information: its location, the location of the destination, and a set of polygons $\mathbb{P} = \{P_1, P_2, \cdots, P_n\}$ that the node has received. Note that a node might receive multiple polygons when it is involved with multiple cuts. Following subsections discuss the details of each step of the protocol.

Step 1. Cut Boundary Abstraction:

The cut boundary abstraction algorithm aims to abstract the boundary information of a cut region. We call the nodes surrounding the boundary of a cut region *boundary nodes*. Our algorithm uses similar technique used in [3] to concisely represent the boundary of a cut region. When a destination is unreachable, a packet would reach one boundary node, say v_{init}. Using the right-hand rule of the face routing, this packet travels along the boundary of the cut region until it reaches again v_{init}, thus detecting the existence of a cut [10, 34]. In particular, we call such node v_{init} the *initiator*. The initiator then sends a probing packet that travels around the boundary of the cut region. The probing packet contains two fields. The first field is

Algorithm 1 Cut Boundary Abstraction (for v_i)

Input: F_1, F_2, δ, and p_i.

```
 1: if v_i ≠ v_init then
 2:     Cut id ← id of initiator.
 3:     // □δ: a rectangle with width δ.
 4:     if ∀p ∈ F_2 ∪ {p_i}, p is in □δ then
 5:         F_2 ← F_2 ∪ {p_i}.
 6:         forward.
 7:     else
 8:         F_2 ← ∅.
 9:         F_1 ← F_1 ∪ {the last element in F_2}.
10:         forward.
11:     end if
12: else
13:     P ← F_1.
14:     broadcast P.
15: end if
```

used to store the locations of the vertices of the polygon representing the boundary of a cut region. We denote the set of such locations by an ordered set F_1. The second field contains all the locations of visited nodes. We denote the locations of the visited nodes by an ordered set F_2.

Algorithm 1 depicts the cut boundary abstraction phase. Upon receiving the probing packet, a node marks the ID of the currently detected cut as the node ID of the initiator. The node then finds a rectangle with width δ, a system parameter, that can cover all the locations in the second field, including the location of the current node. If such a rectangle exists, the location of the current node is appended to the end of the second field of the probing packet, and the packet is forwarded to the next boundary node (Line 2-6). If such a rectangle does not exist, all the locations in the second field are deleted, and the last element in F_2 is appended to the end of the first field (Line 8-9). The current node then forwards the packet to the next boundary node. Note that, in order to keep the size of set F_2 manageable, if the size of F_2 exceeds a threshold, F_2 is emptied and the last element of F_2 is appended to the end of F_1. Note that the threshold is determined based on the maximum packet

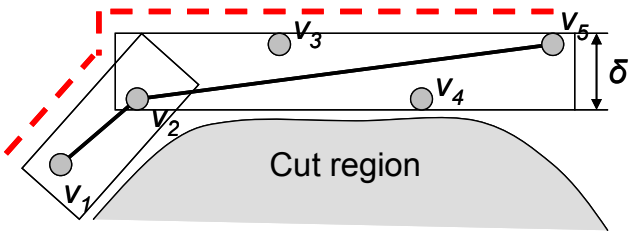
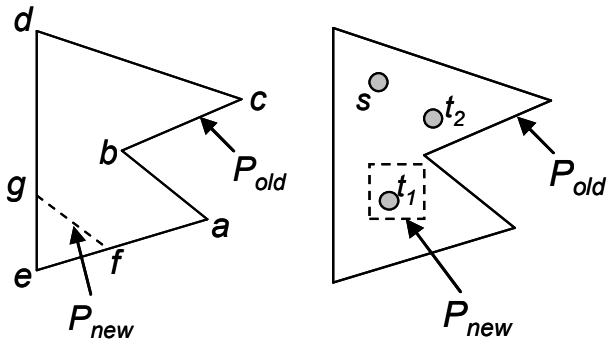


Figure 2.4: An illustration of the false positive.

size in order to make the set fit in a packet. Finally, when the probing packet finishes traversing the boundary, we get a set $P = \{p_1, p_2, \cdots, p_n\}$ in the first field of the probing packet, representing the polygon covering the cut region. This information is then broadcast to the nodes in the cut region (Line 14), and used by the nodes during the second step of the protocol, namely the Cut Detection.

Consider Figure 2.3 for an example. In this figure, source s attempts to send a packet to destination t. This packet then reaches node v_{init} and is routed along the boundary of the cut region A according to the right hand rule of face routing. The packet then returns to node v_{init} starting the cut abstraction process by sending a probing packet to node v_2. When the probing packet reaches node v_6, the first field of the probing packet contains set $F_1 = \{v_{init}\}$ and the second field contains set $F_2 = \{v_{init}, v_2, v_3, v_4, v_5\}$. The node v_6 then examines if there exists a rectangle with width δ that can hold all the locations in set $F_2 \cup \{v_6\}$. Since there does not exist such a rectangle box, the points in F_2 are deleted and F_1 becomes $\{v_{init}, v_5\}$.

As discussed, the boundary of a cut region is compactly represented as a polygon. Despite its concise representation, a polygon, however, might fail to precisely describe the boundary of a cut region, causing false positives. Specifically, a false positive occurs when a polygon fails to cover all the nodes in a cut region. Figure 2.4 illustrates a scenario where the false positive occurs. This figure shows a fraction of



(a) (b)

Figure 2.5: Illustrations of multiple cuts: (a) a new cut sharing boundary (b) an independent cut.

the boundary nodes for cut region A. According to the cut abstraction phase of the P2P-CD algorithm, nodes v_1, v_2, and v_5 serve as the vertices of the polygon representing the boundary of the given cut region. We note that this polygon, however, fails to cover node v_3 that is outside the polygon; thus, when a node attempts to send a packet to node v_3, it will determine that node v_3 is unreachable, although the node is reachable, causing the false positive.

In order to eliminate the false positive, our P2P-CD algorithm is slightly modified at the cost of extra data storage for saving additional vertices. The proposed idea, named the FPE (False Positive Elimination), is simple, yet effective. Instead of constructing a polygon by connecting the two nodes at each end of a bounding box, we build a polygon by connecting the edges of bounding boxes that face the outside of a cut region. This can be easily done by saving the locations of the previous bounding box, and calculating the intersecting points with the currently investigated bounding box in the cut abstraction phase. See Figure 2.4 for an example. In this example, we use a series of dotted lines as the edges of the polygon covering the cut region, rather than using the two line segments, which are $\overline{v_1v_2}$ and $\overline{v_2v_5}$.

Additional cuts might appear after existing cuts have been detected. Such a cut either shares the boundary of previously detected cut(s), or is an independent cut that does not share its boundary with previously detected cuts. Figure 2.5(a) depicts the former case. The old cut P_{old} represented by the polygon $a-b-c-d-e$ shares its boundary with the new cut $a-b-c-d-g-f$. In this case, when the probing packet reaches the boundary nodes of previously discovered cut(s), the cut ID(s) of previously detected cut(s) is recorded in the probing packet, if it has not been done so. When the probing packet returns to the initiator, the set of recorded cut IDs, called the UPDATE_INDEX set, and the set P representing the polygon for the new cut are encoded in the broadcast packet, which then is distributed to the nodes in the cut region. Upon receiving this broadcast packet with the UPDATE_INDEX set, nodes update their database for the detected cuts. The details of this update process is described in the following subsection.

Figure 2.5(b) illustrates the latter case where a new cut does not share its boundary with previously detected cuts. In this figure, the rectangle with dotted lines represents the new cut. In this case, the same boundary abstraction algorithm is used to describe the new cut as a polygon. One difference is that when a probing packet returns to the initiator, the initiator sets the addition bit of the broadcast packet, so that the nodes in the cut region add a new polygon to its database, $\mathbb{P}$.

Step 2. Cut Detection:

When the cut boundary abstraction phase is finished, each node in a cut region recognizes the cut boundary as a polygon, represented as a set $P = \{p_1, \cdots, p_n\}$. Given the locations of source s and destination t, and the collection of polygons, $\mathbb{P}$, the Cut Detection phase determines whether destination t is reachable from source s. To find the connectivity between any pair of source and destination, we borrow an idea from the point-in-polygon (PIP) problem in the computational geometry

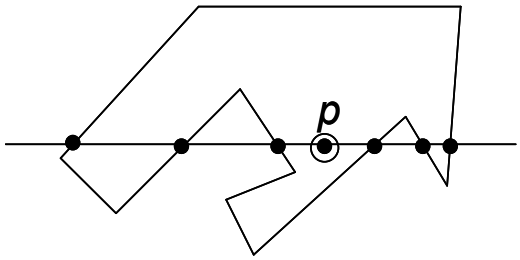


Figure 2.6: Example of the ray tracing algorithm.

that finds whether a point is inside a given polygon or not. There are two well known algorithms to solve this problem: ray casting algorithm and winding number algorithm [35]. We choose to use the ray casting algorithm, because the winding number algorithm involves costly operations [35] that are not feasible for the sensor motes with constrained computational capability.

Ray casting algorithm works as follows. Given a point p, the algorithm finds how many sides of the polygon intersect with the y threshold of the point p. If there are odd times of intersections on each side of p, p is inside the polygon; otherwise, if there are even times of intersections on each side of p, p is outside the polygon. Figure 2.6 illustrates an example. The point p has 3 intersections on its right side and 3 intersections on its left side; thus, p is determined to be inside the polygon. We denote the ray casting algorithm by $\text{PIP}(P, p)$, where P refers to a polygon, and p is the point to be tested. We define that if p is inside P, $\text{PIP}(P, p) > 0$, otherwise $\text{PIP}(P, p) < 0$.

Algorithm 2 depicts the Cut Detection phase of the P2P-CD protocol. As we described, the type of the broadcast packet can be either the update type or addition type. If the packet is the update type, from $\mathbb{P}$, we delete the polygons having the cut IDs specified in the UPDATE_INDEX, and add P to $\mathbb{P}$ (Line 2-4). In our previous example, the polygon $\{a, b, c, d, e\}$ is deleted and a polygon $P = \{a, b, c, d, g, f\}$

Algorithm 2 Cut Detection

Input: $p_s, p_t, \mathbb{P}$ and UPDATE_INDEX
1: upon receiving the broadcast packet:
2: **if** update type **then**
3: $\quad \mathbb{P} \leftarrow \mathbb{P} \setminus \{P_i\}, \forall i \in$ UPDATE_INDEX
4: $\quad \mathbb{P} \leftarrow \mathbb{P} \cup P$.
5: **else**
6: $\quad \mathbb{P} \leftarrow \mathbb{P} \cup P$.
7: **end if**
8: upon having a packet to destination at p_t:
9: **if** $\forall P \in \mathbb{P}, (PIP(P, p_s) \cdot PIP(P, p_t) > 0)$ **then**
10: $\quad t$ is reachable.
11: **else**
12: $\quad t$ is not reachable.
13: **end if**

is added to $\mathbb{P}$. If the control message is the addition type, it simply adds P to collection $\mathbb{P}$ (Line 6). If there is a packet to send, the Cut Detection phase tests if the destination is reachable. Specifically, given the location of the source, p_s, and the location of the destination, p_t, the algorithm checks if the following condition holds for each $P \in \mathbb{P}$: $PIP(P, p_s) \cdot PIP(P, p_t) > 0$. If the condition holds for all P, p_s and p_t are connected (Line 9-13).

In our previous example, we have two cuts, P_{old} and P_{new}, source s, and two destinations t_1 and t_2. Since both s and t_1 are inside P_{old}, $PIP(P_{old}, p_s) \cdot PIP(P_{old}, p_{t_1}) > 0$. However, while s is outside P_{new}, t_1 is inside P_{new}, which gives $PIP(P_{new}, p_s) \cdot PIP(P_{new}, p_{t_1}) < 0$; thus t_1 is not reachable from s. On the other hand, for t_2, $PIP(P_{old}, p_s) \cdot PIP(P_{old}, p_{t_2}) > 0$ and $PIP(P_{new}, p_s) \cdot PIP(P_{new}, p_{t_2}) > 0$, thereby t_2 being reachable from s.

2.1.5 Energy Efficient Cut Detection for Multiple Sinks

As we described, P2P-CD is a suitable protocol for a network with frequent peer to peer communications. However, this protocol maybe an overkill for a network scenario where there are a small number of target destinations, because this protocol requires each node to save the information on partial global topology. For example,

in many sensor network applications, sensor nodes transmit the data with perceived phenomenon only to a sink node, or multiple sink nodes. In other words, the number of target destinations is limited. For these applications, we develop a lightweight cut detection algorithm, which relies on neither location information, nor the knowledge on global topology.

The energy efficient cut detection for multiple sinks, called the RE-CDM, is a more generic solution that builds on our previous work, RE-CD [23]. RE-CDM is based on the virtual grid network consisting of the leaders and the virtual link between them, as in RE-CD. In RE-CDM, however, multiple sink nodes are elected as leaders, and each leader node now maintains a set of states, as opposed to RE-CD, in which a single state is maintained. Each state value represents the connectivity to a sink node. Note that although multiple sink nodes are typically deployed in such a way that they cover as many sensor nodes as possible, thereby being distant with each other, if we have more than one sink node in the same grid cell, the one with higher residual energy becomes the leader. The set of states for multiple sinks are updated at each iteration of the algorithm. The updated states are then encoded in the state message, which is then sent to the neighboring leaders. In essence, RE-CDM overlays the multiple executions of RE-CD for each sink, while using a single state message.

We describe RE-CDM more formally. Consider multiple sink nodes $S = \{s_1, s_2, \cdots, s_n\}$. Let a set $L = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the leaders, and N_i^ℓ be the neighboring leaders of a leader ℓ_i (i.e., $|N_i^\ell| \leq 4$). Each leader node ℓ_i maintains a set of states, each of which corresponds to a sink $s \in S$ and is denoted by $\ell_i(s^k)$, where k is the iteration count of the RE-CDM algorithm. At each iteration of the algorithm, each leader node $\ell_i \notin S$ updates the set S as the following.

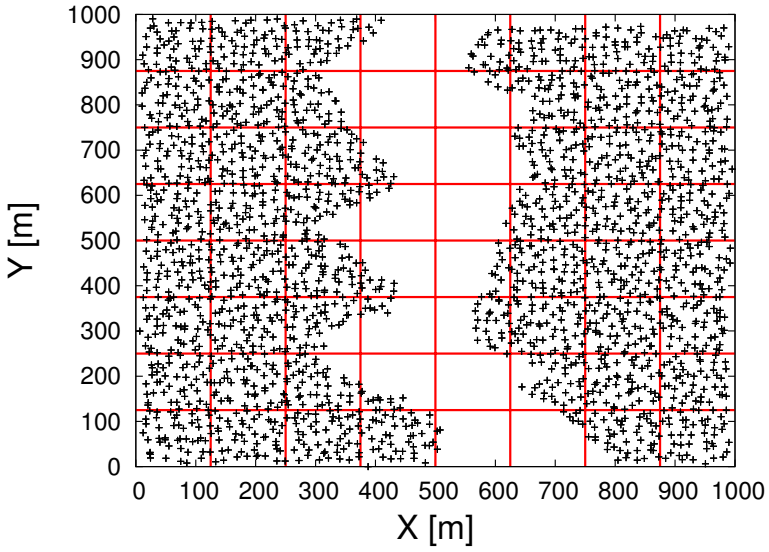


Figure 2.7: Experimental setup depicting 2,500 nodes deployed in a 1,000×1,000m^2 area.

$$\text{For all } s \in S, \ell_i(s^{k+1}) = \frac{\sum_{\ell_j \in N_i^\ell} \ell_j(s^k)}{|N_i^\ell| + 1}.$$

Each node $\ell_i \in S$ update the set S as the following.

$$\text{For all } s \in S, \ell_i(s^{k+1}) = \frac{\sum_{\ell_j \in N_i^\ell} \ell_j(s^k) + \omega}{|N_i^\ell| + 1}$$

Here, ω is a system parameter. A state message is now sequentially encoded with the states for multiple sink nodes (i.e., from $\ell_i(s_1^{k+1})$ to $\ell_i(s_n^{k+1})$), and sent to adjacent leaders.

Despite its scalability and energy efficiency, RE-CDM might not work efficiently for large number of sink nodes, because the state message size grows, and may be split into multiple packets depending on the number of sinks.

2.1.6 Large-scale Simulations

We evaluate the performance of the proposed protocols through simulations. We implement P2P-CD and RE-CDM in C++, mainly focusing on the topological behavior of the protocols. We randomly deploy 2,500 sensor nodes in a network of $1{,}000 \times 1{,}000 m^2$ region with a cut, as shown in Figure 2.7. A superimposed 8×8 grid is used for selecting leader nodes for RE-CDM. The communication radius of a node is varied from 35m to 75m, resulting in an average number of neighbors from 10.32 to 41.36. We ensure that the network is connected. An event occurs every 10sec at a random location, and then a node nearest to the event reports the event data to a random destination.

For accurate energy consumption estimation, we consider an asynchronous duty cycle network, where each node has a randomly generated periodic schedule. Nodes periodically wake up and sleep based on the schedule. We assume that a node knows about the schedules of its neighbors, or the parameters used for the pseudo-random schedule generator like [36] (i.e., these parameters are used to deduce the schedules of its neighbors). In order to coordinate the schedule of a node with its neighbors, we assume that a local synchronization is implemented. The local time synchronization can be implemented by using a Medium Access Control (MAC)-layer time stampingf technique [37]. The MAC-layer time stamping technique achieves the accuracy of 2.24μs at the cost of exchanging a few bytes of packet transmissions with its immediate neighbors every 5 minutes. This accuracy is by far enough for our asynchronous duty cycle scheme. Each period of the schedule consists of 100 time slots. The unit time for each slot is 50ms. The number of time slots during which a node is active varies according to the duty cycle ratio, which is a system parameter.

To simulate the energy consumption, we assume that each node has a radio with

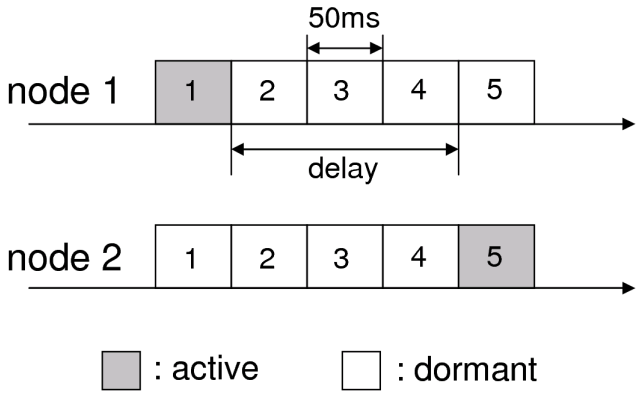


Figure 2.8: an illustration of the source of energy consumption.

250kbps data rate, and maximum packet size of 128B, similar to the Zigbee compliant CC2420 [38]. We consider the following metrics: total energy consumption, network lifetime, false positive rate (a false report of "unreachable destination"), probing packet size, detection delay, and control packet overhead. We vary the following parameters: δ, communication radius, and duty cycle ratio. We use Greedy Perimeter Stateless Routing (GPSR) as the underlying routing protocol, and build different cut detection algorithms on top of it. We compare GPSR+P2P-CD, GPSR+RE-CDM, GPSR+Flooding, and GPSR with no cut detection scheme. The details of experimental results are described in the following subsections.

Experiment 1 - Energy Consumption

In an asynchronous duty cycle network, nodes consume energy when they are awake for either idle listening, or data transmission. Regardless of the protocols run on the network, the same amount of energy is spent for idle listening, because the predefined working schedules determine the energy consumption for idle listening; thus, we consider only the data transmission for the following reasons. First, the current draw for receive mode and transmit mode are different (the difference depends on a radio module; for some radio modules, more current is draw for receive mode).

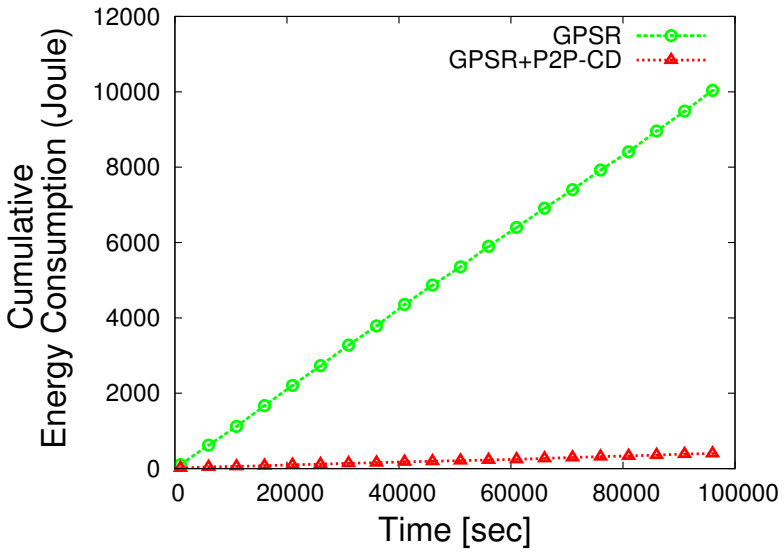


Figure 2.9: Accumulated energy measured for long time.

More important reason is that when a node transmits a packet, it may have to use an additional slot to wake up its radio. Figure 2.8 illustrates this concept. When node 1 attempts to transmit a packet to its neighbor, node 2, it has to turn on its radio at slot 5, which was originally scheduled for sleep. This causes extra energy consumption.

This experiment is designed to see how much energy P2P-CD can save when P2P-CD is coupled with GPSR. The communication radius was set to 70m, δ to 60m, and duty cycle ratio to 1%. We measure accumulated energy consumption in Joules for GPSR, GPSR+P2P-CD, and GPSR+RE-CDM, as a function of operation time. Figure 2.9 depicts the results. P2P-CD significantly reduces the energy consumption by preventing packet transmissions to unreachable destinations. RE-CDM is not much helpful for energy saving in this simulation scenario, because it detects a cut with respect to a small set of destinations; in our network setting of 8 by 8 grids, it detects a cut with respect to 64 nodes. Moreover, the periodic state

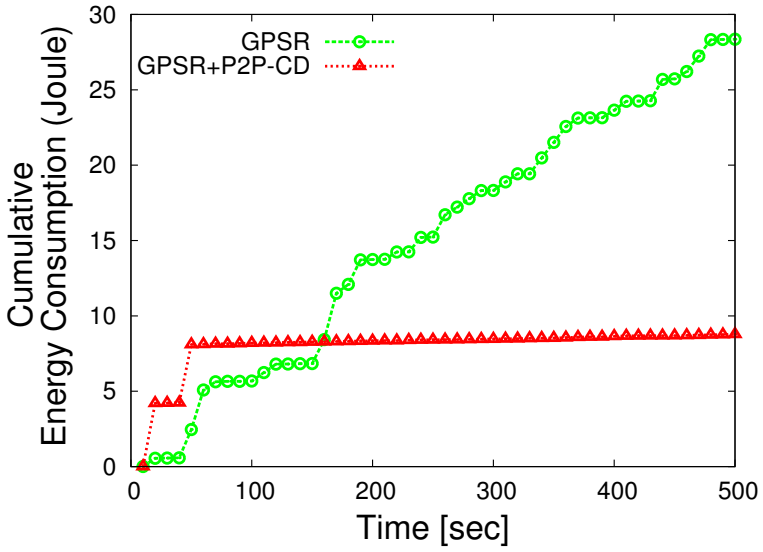


Figure 2.10: Communication overhead for P2P-CD to detect a cut.

message exchanges make RE-CDM less energy efficient than P2P-CD. However, as we prove later, for network scenarios with a small set of destinations, such as a WSN application with multiple sinks, RE-CDM is as energy efficient as P2P-CD.

The P2P-CD protocol incurs communication overhead. Specifically, a probing packet is sent along the boundary of a cut region, and a message containing the obtained set P needs to be flooded to the nodes in the cut region. Figure 2.10 shows how this control packet overhead affects the energy efficiency. The abrupt increases in the energy consumption for GPSR+P2PCD at 10sec and 40sec represent the overhead for identifying the cuts. Due to this overhead, consumed energy for GPSR+P2PCD is higher than GPSR until about 150sec. However, this overhead is compensated by avoiding unnecessary packet transmissions to unreachable destinations; while consumed energy of GPSR+P2PCD gradually increases, GPSR has frequently arising rapid increases in the energy consumption, caused by attempting to send a packet to unreachable destinations. As a result, after 150sec, GPSR exhibits higher energy

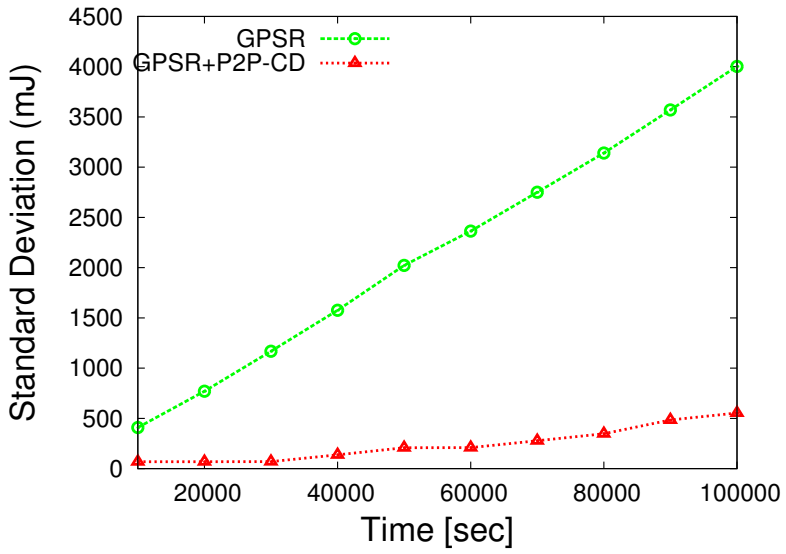


Figure 2.11: Standard deviation of energy consumption.

consumption.

Experiment 2 - Network Lifetime

Another important issue (in addition to high energy consumption) caused by a cut in a network, is unbalanced energy consumption. A packet destined to an unreachable destination always travels around the boundary of a cut region, thereby exhausting the energy of the nodes on the boundary faster than other nodes. In order to verify this unbalanced energy consumption, we measure the standard deviation of energy consumption of all the nodes in the network. Figure 2.11 depicts the results. As expected, the unbalance in energy consumption of GPSR is much worse than GPSR+P2P-CD, and slightly worse than GPSR+RE-CDM. This unbalance deteriorates as operation time elapses.

This unbalanced energy consumption directly affects the network lifetime, which is defined as the elapsed time until a node first dies. In this set of experiments, we are interested in obtaining the expected network lifetimes for GPSR, GPSR+RE-

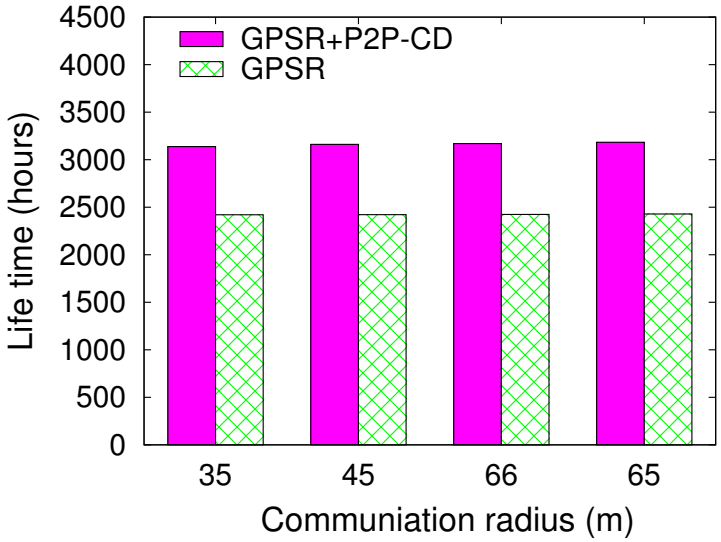


Figure 2.12: Network lifetime for GPSR, GPSR+RE-CDM, and GPSR+P2P-CD.

CDM, and GPSR+P2P-CD. For this experiment, we assume that a sensor node is powered by a single AA battery which has capacity of 2000mWh, which is a usual energy source for modern sensor motes. We measure the network lifetime by varying the communication radius. As shown in Figure 2.12, we observe slight increases in network lifetime for all three algorithms (GPSR, GPSR+P2P-CD, and GPSR+RE-CDM) as we increase the communication radius. One reason is that higher communication radius helps to increase the network lifetime by using less number of packet transmissions, but the effect of communication radius is almost negligible. Comparing network lifetime among these three protocols, we note that GPSR has the shortest lifetime; GPSR+RE-CDM has slightly better lifetime than GPSR; and GPSR+P2P-CD highly improves the network lifetime. These results conform to the distribution of energy consumption in the network.

Higher duty cycle ratios expedite the aging process, because nodes have to be in the wake-up state for longer period of time. Figure 2.13 shows the impact of the

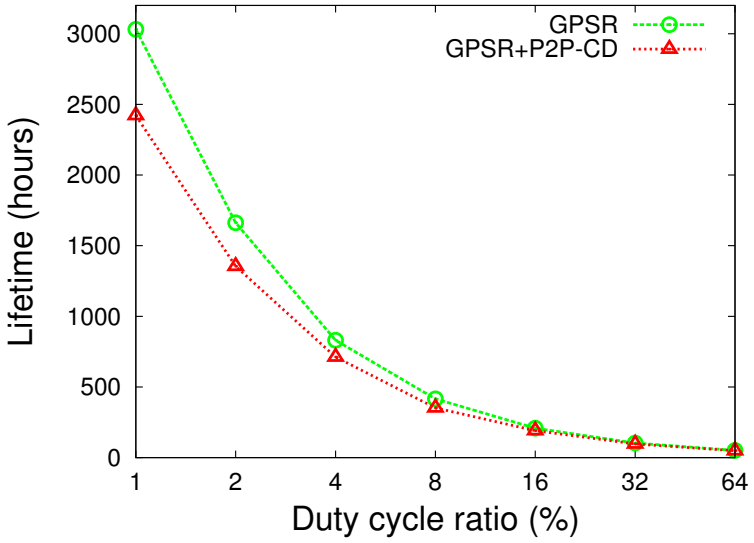


Figure 2.13: Network lifetime for GPSR and GPSR+P2P-CD for different duty cycle ratios.

duty cycle ratio on network lifetime. As shown, as the duty cycle ratio increases, the network lifetime rapidly decreases. One notable observation is that the gap in the network lifetime between GPSR and GPSR+P2P-CD becomes smaller with increasing duty cycle ratio. A reason is that when the duty cycle ratio is large, it is more likely that a node sends a packet to its neighbors by using the predefined working schedule, instead of using an additional wake-up slot.

Experiment 3 - False Positive Rate

A false positive occurs when a node determines that a destination is unreachable, even though the destination can actually be reached. This false positive is caused by the inaccuracy in describing the boundary of a cut region; in other words, when a polygon representing the cut boundary cannot cover all the nodes in the cut region, we observe the false positives. The parameter δ can be used to adjust the accuracy of the boundary description. Smaller δ values permit more precise representation of the boundary, while incurring higher overhead. We measure the false positive rate

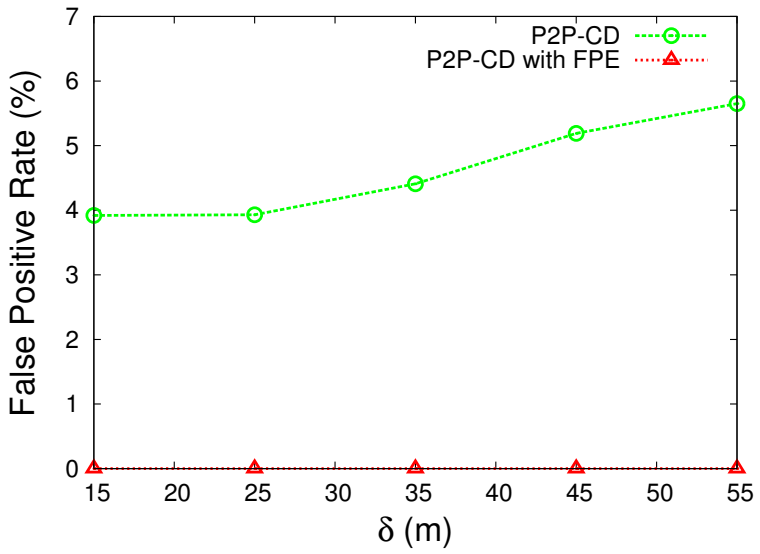


Figure 2.14: Impact of parameter δ on the rate of false positives.

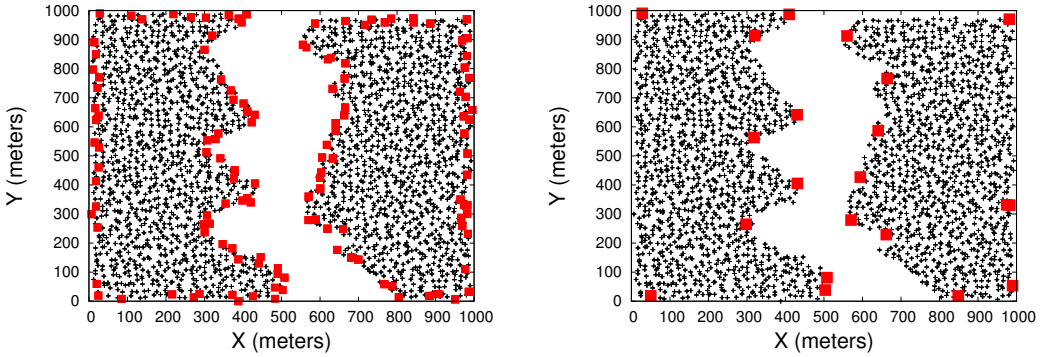
from 10,000 random source and destination pairs. Figure 2.14 shows the results. As expected, the false positive rate increases as the δ value increases; however, even for small δ values, the false positive persists. To address this issue, we introduced a FPE (False Positive Elimination) scheme to eliminate the false positives at the cost of storage overhead. Figure 2.14 depicts the results we obtained after the FPE is applied. As shown, the false positive is 0% when the FPE algorithm is applied.

Experiment 4 - Packet Size

Smaller δ values allow a precise description of a cut region, because a polygon with more vertices is used to describe the cut region. However, the size of the control packet, that is broadcast to nodes in the cut region, must be large to contain more vertices. In contrast, larger δ values permit smaller control packet size, because fewer vertices are used to represent the cut region. This, however, is done at the cost of possible errors, because the bounding box with large width may contain the nodes that do not belong to the cut region. Figure 2.15(a) and 2.15(b) show how

Table 2.2: The size of P, in terms of number of vertices.

δ	Cut Region(L)	Cut Region(R)
10	69	62
30	16	13
50	12	11



(a) (b)

Figure 2.15: Examples of cuts: (a) a cut abstracted with $\delta = 10$ meters; (b) a cut abstracted with $\delta = 50$ meters.

the vertices for the polygon are chosen for different δ values, and Table 2.2 presents the relationship between the δ values and corresponding number of vertices.

Based on the simulation results, we attempt to determine the appropriate δ value for our experiments. We note that the largest δ value that does not cause errors (i.e., finding a bounding box that contains a vertex not within the cut boundary) is $\frac{\sqrt{3}}{2} \cdot r$, where r is the communication radius of a node (See Figure 2.16 for proof: Given any node, say v_1, in a bounding box, we attempt to find the position of its neighboring node, say v_2, such that δ, the height of the intersection area, is minimized. The δ is minimized as $\frac{\sqrt{3}}{2} \cdot r$, when v_2 is located as shown in this figure. Thus the width of a bounding box can be at most $\frac{\sqrt{3}}{2} \cdot r$; otherwise, a bounding box may contain a node that is reachable from neither v_1 nor v_2). Therefore, in our simulation settings, we

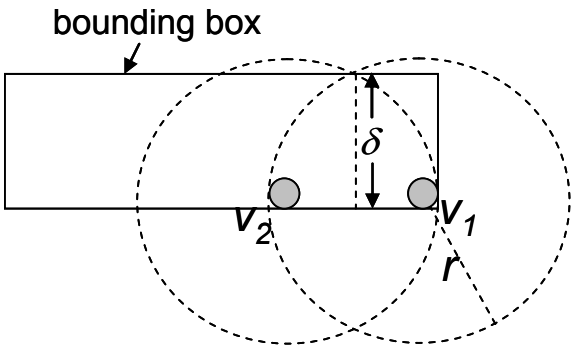


Figure 2.16: An illustration of the proof for choosing optimal δ.

choose $\frac{\sqrt{3}}{2} \cdot 70 \approx 60$ meters as our δ value, because the default communication radius is 70 meters.

Experiment 5 - Detection Delay

Detection delay is defined as the elapsed time between the occurrence of a cut and the detection of the cut. In RE-CDM, a node detects a cut when its state converges either to a new positive value, or 0. As Figure 2.17 illustrates, RE-CDM requires 7 iterations until nodes being able to detect a cut, regardless of the duty cycle ratio. Note that the iteration period of the RE-CDM was set to 100 seconds. These results indicate that if we use a smaller iteration period, RE-CDM would detect a cut faster; but, the control packet overhead would increase with the smaller iteration periods.

The P2P-CD algorithm detects a cut when the probing packet finishes traveling around the boundary of the cut region, and the set of vertices of the polygon representing the boundary is broadcast to the nodes in the cut region. Figure 2.17 shows the detection delay for P2P-CD per duty cycle ratio. For larger duty cycle ratio, a node has more active neighboring nodes. As a result, larger duty cycle ratio allows the probing packet to travel faster, and thus reducing the detection delay, as the figure indicates.

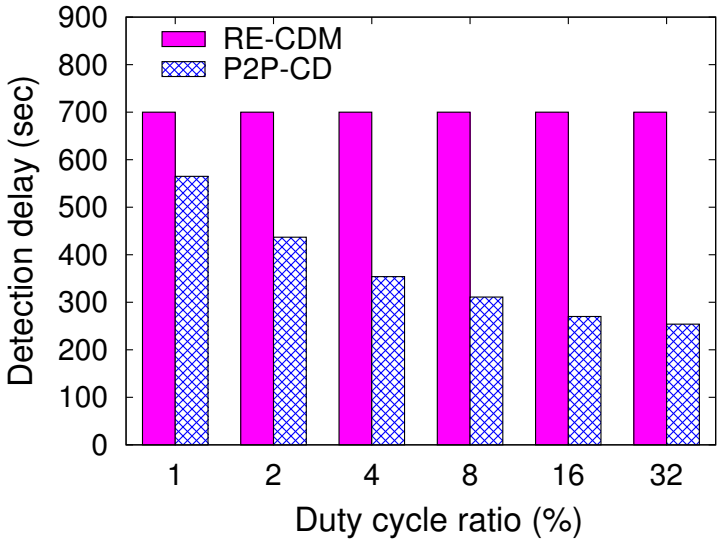


Figure 2.17: Impact of duty cycle ratio on the cut detection delay.

Experiment 6 - Control Packet Overhead

A control packet refers to a packet used to detect a cut. The P2P-CD algorithm has two types of control packets: the probing packet, and the broadcast packet used to distribute the polygon to the nodes in a cut region. In RE-CDM, a control packet is the packet that carries the state message. In this experiment, we measure the overhead of this control packet, in the form of the accumulated number of control packet transmissions. For this experiment, we fix the duty cycle ratio to 1%, and delta to 60m; and we vary communication radius and operation time.

The P2P-CD algorithm is a reactive solution; thus, once it detects a cut, it does not incur additional control message overhead, unless different cuts appear in the network. Figure 2.18 depicts the results. As shown, the control packet overhead for P2P-CD is constant, because the cut has been detected before 100 second (i.e., the time for the first iteration of the RE-CDM algorithm). On the other hand, the accumulated number of control packet transmissions for RE-CDM continuously in-

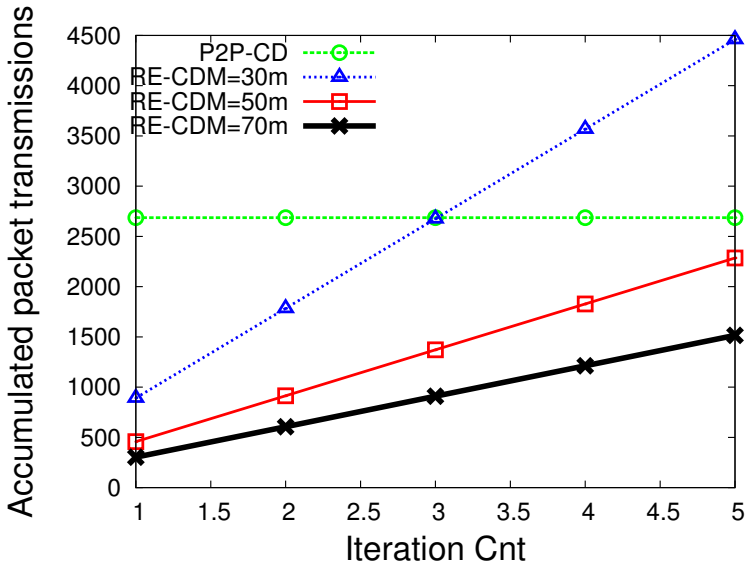


Figure 2.18: Control packet overhead of P2P-CD and RE-CDM.

creases, because RE-CDM is a proactive solution that periodically scans the network for detecting a cut. Figure 2.18 depicts the results. In particular, when the communication radius of a node is smaller, the control packet overhead becomes higher, because a control packet must be transmitted over a larger number of hops to travel the same distance.

Experiment 7 - Number of Sinks

In the previous experiments, we have focused on a scenario with N target destinations, where N is the number of nodes in the network. Now we consider a new scenario with fewer target destinations ranging from 1 to 8. Performing simulations under this new scenario is important because our RE-CDM is specifically designed for a network with a small number of target nodes (e.g., a wireless sensor network with multiple sinks). In this set of experiments, we are interested in how our RE-CDM performs in terms of energy efficiency when there are a small number of target destinations. In particular, we compare RE-CDM with the flooding method, a primitive

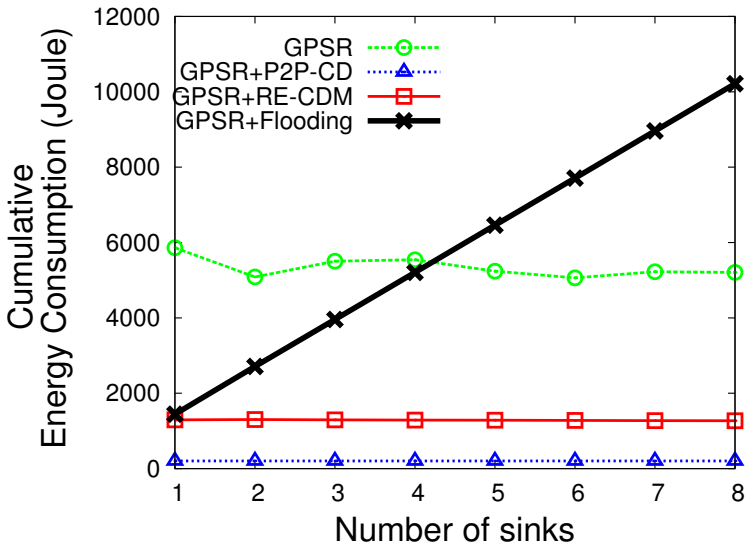


Figure 2.19: Energy consumption for a network with a small number of target destinations.

cut detection algorithm, where each sink node broadcasts a probing packet throughout the network in order to enable nodes detect a cut. We assume that when an event occurs (i.e., every 10 seconds at a random location according to the previous experimental settings), each node reports data to a randomly selected sink node. We measure the accumulated energy consumption in Joules after 10,000 seconds of operation time.

Figure 2.19 depicts the results. Compared with the scenario with N target destinations, which is shown in Figure 2.9, RE-CDM performs significantly better, having as low energy consumption as P2P-CD. Furthermore, unlike P2P-CD, RE-CDM does not require nodes to maintain global topology information in their storages, nor the location information of neighboring nodes. Note that, as RE-CDM does not require location information, it can be coupled with routing protocols that are not based on node localization. Although the energy consumption of RE-CDM would gradually increase as operation time elapses due to the periodic state-message exchanges, we

believe that it is a proper solution for the applications where nodes have limited storage and computational capabilities, and appropriate localization methods are unknown.

Investigating the energy consumption of the flooding method, the simplest cut detection scheme, allows us to understand where our proposed solutions stand in terms of energy efficiency. As shown in Figure 2.19, the flooding method shows as low energy consumption as RE-CDM when there is a single target destination (i.e., a sink). However, if we increase the number of sink nodes, consumed energy for the flooding method linearly increases, because each sink node periodically broadcasts a probing packet throughout the network. For more than 5 sink nodes, the flooding method performs even worse than when no cut detection algorithm is used (i.e., the GPSR). Furthermore, the flooding method suffers from the limitations that only a part of nodes is able to detect a cut; thus, this method can only be useful for a network with very small number of target destinations (e.g., a single sink node), where only the nodes in a disconnected network segment are required to detect a cut.

2.2 Hole Detection

Hole Detection also commonly called boundary detection algorithms for wireless sensor networks are largely categorized into three schemes: location-based, topology-based, and statistical schemes. When node location is known, location-based boundary detection algorithms [8, 9] are useful. Topology-based solutions [39, 40, 41, 42] do not require node-location; however, these solutions often involve non-negligible complexity for implementation. There are several algorithms based on statistical approach [43, 44]. These solutions detect boundaries based on the number of neighbors. However, these heuristic solutions require very high node-density. In this disserta-

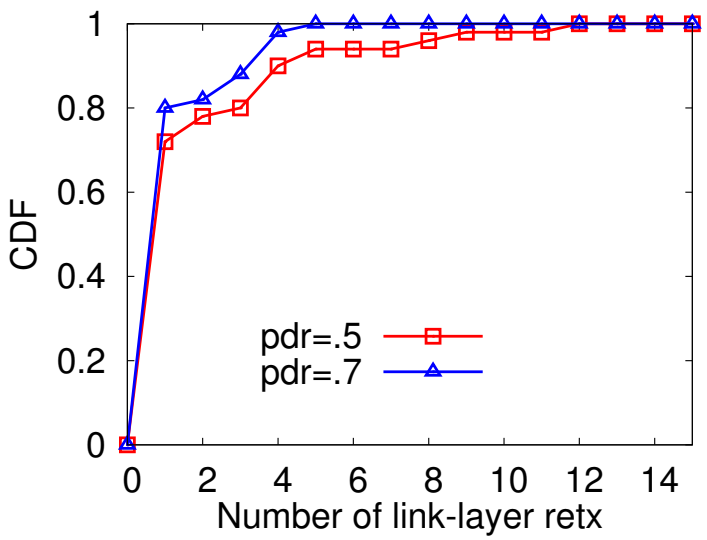


Figure 2.20: Reliability of paths surrounding holes for different pdr.

tion, as node location is known, we adopt the BOUNDHOLE [8], a widely used location-based boundary detection scheme. The main idea of the BOUNDHOLE algorithm is as follows: it first identifies "stuck nodes" where a packet may be stuck in a local minimum during the geographic forwarding process. Once stuck nodes are identified, one of them for each hole sends a control packet that travels along the boundary of the hole. While the control packet traverses the boundary, boundary nodes are detected.

However, running the BOUNDHOLE algorithm on real hardware raises several challenges:

- The BOUNDHOLE algorithm assumes ideal links (i.e., a simple Unit Disk Graph (UDG) radio model).
- The BOUNDHOLE algorithm does not consider the link quality that dynamically changes over time.
- The BOUNDHOLE algorithm identifies even very small holes. However, small

holes do not affect much the performance of location-based routing.

To address the first challenge (i.e., the UDG model), we design a neighbor discovery module such that it take into account link quality in determining 1-hop neighbors and use only the 1-hop neighbors with stable symmetric links for running the algorithm. In other words, we aim for having a closed cycle of boundary nodes that are stably connected with each other. To implement the module, we ensure that nodes maintain a packet delivery ratio (pdr) for each of its one-hop neighbors and chooses as its one-hop neighbor only nodes for which the pdr is greater than a predefined threshold. To choose a proper threshold for the pdr in our experimental environment, we ran experiments where the BOUNDHOLE algorithm was executed with different pdr values. To consider varying link qualities over time, we ran the experiments 3 times a day at 12:00 PM, 6:00 PM, and 12:00 AM. We then counted the number of link-layer retransmissions for nodes used for the operation of the BOUNDHOLE algorithm. The results are depicted in Figure 2.20. As the figure shows, a low threshold for the pdr resulted in a higher number of link-layer retransmissions, because we allowed nodes to select neighbors with unstable links. Especially, when we set the maximum allowable number of link-layer retransmissions to 10, the pdr threshold of 0.5 resulted in some broken links between boundary nodes. We found that in our experimental setting, generally, pdr $> .7$ gave reliable boundary. A scheme for automatically determining a proper value for the pdr threshold for various experimental settings is left as future research.

To address the second challenge (i.e., dynamically changing link quality), we allow nodes to continually monitor the pdr of links for neighboring boundary ndoes. Whenever the pdr for a link between boundary nodes becomes smaller than the threshold, the BOUNDHOLE algorithm reruns and finds new boundaries. The third issue states that the BOUNDHOLE algorithm identifies even a very small hole –

we consider holes with 4 boundary nodes or less as small holes. However, only large holes are of our interest, because small holes have a small impact on the path stretch.

3. CNT-AWARE LOCATION-BASED UNICAST ROUTING

In this section, we present our CNT-aware unified location-based routing protocol called the Local Visibility Graph based Geographic Routing Protocol (LVGR).

3.1 Motivation

Geographic routing is well suited for large-scale wireless sensor networks (WSNs) because of its attractive properties: first, it is simple, i.e., a forwarding node simply sends a packet to the neighbor geographically closest to a destination node, without relying on control packet propagation [45][46], or beacon node selection [47]; second, geographic routing is highly scalable, because its per-node state is independent of network size. These properties make geographic routing protocols play a key role in many applications for WSNs.

Geographic routing, however, fails when a forwarding node has no neighbor closer to the destination node – this situation is called the *local minimum phenomenon*. The local minimum is often caused by complex topological structures such as *network holes*. Kuhn et al. [48] proved that the worst-case path length is $\Omega(c^2)$, where c is the optimal path length. A number of protocols [49][8][50][51] have attempted to address the local minimum problem; however, none of these solutions guarantees bounded stretch.

Recently, Tan et al. [3] proposed a visibility graph-based geographic routing protocol (VIGOR) that achieves a path length of $\Theta(c)$. In this protocol, holes are represented as polygons, where each vertex corresponds to a node - we call such node a VOP (Vertex of Polygon) node. VOP nodes and edges connecting two visible VOP nodes form a visibility graph as shown in Figure 3.1, where VOP nodes are represented as small squares, and line segments connecting two visible VOP nodes

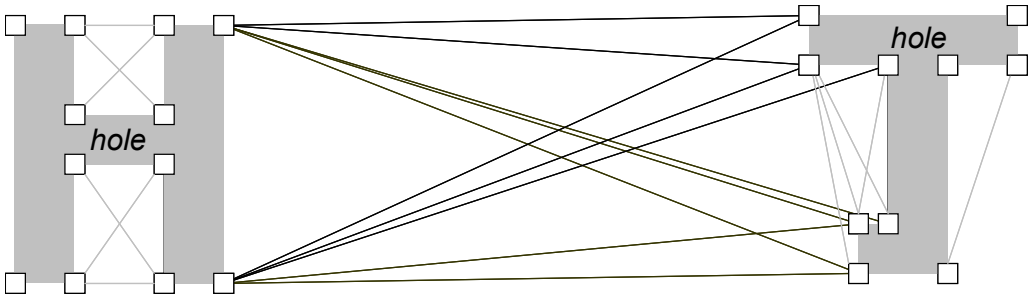


Figure 3.1: An illustration of a visibility graph.

represent the edges of a visibility graph. The visibility graph serves as an overlay network on which a distance vector routing algorithm finds a shortest path. The argued path stretch is, however, only possible with a "path-setup" process: a source node, to join the visibility graph, must first exchange a message with a destination node using a default geographic routing protocol (e.g., GPSR or GOAFER+) prior to beginning its data transmission. Thus, considering the path generated for the "path-setup" process, the argued stretch is debatable. Besides, VIGOR incurs additional communication overhead for running a distance vector routing: VOP nodes must send messages to neighboring VOP nodes until their routing tables converge – if two VOP nodes are distant from each other (especially for large-scale networks; see Figure 3.1), this communication overhead becomes significant. Additionally, storage overhead for maintaining routing table entries is also an important issue for resource-constrained sensor nodes.

In Section 3, based on the boundary detection and abstraction algorithms presented in Section 2, we propose the first routing protocol called Local Visibility Graph-based Geographic Routing (LVGR) that achieves: 1) guaranteed worst-case path stretch of $O(\frac{D}{c})$, where D is the diameter of the network and c is the communication radius of a node (i.e., the minimum allowable communication range for

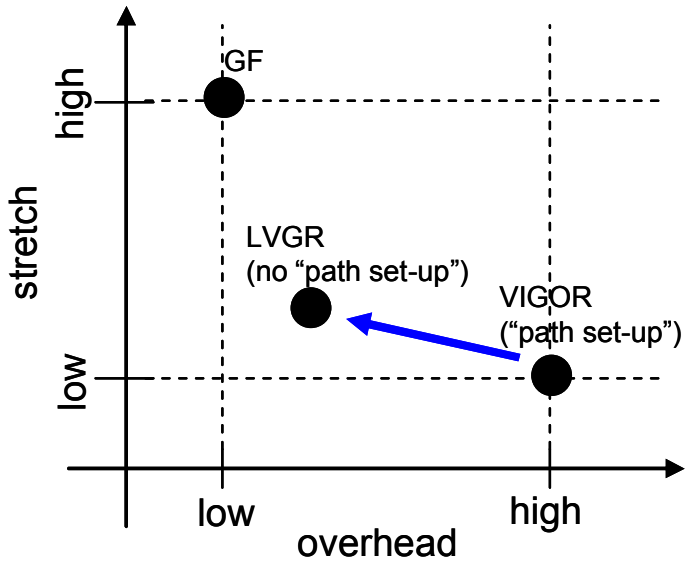


Figure 3.2: Tradeoff between stretch and overhead (GF & VIGOR).

reliable data communication for a realistic communication model); and 2) low communication and storage overhead. *LVGR has the smallest bounded stretch for a large-scale sensor network without an assumption of Unit Disk Graph (UDG) and without relying on the "path-setup" process.* Figure 3.2 shows where LVGR stands compared with state-of-art. LVGR is inspired by an observation that, if both source and destination nodes are *outside the convex hulls* of polygons, representing holes as convex hulls is sufficient for finding a path with constant stretch. This observation allows us to use a visibility graph only when we route packets to/from nodes *inside convex hulls*. Thus, we can reduce the communication and storage overhead for building routing tables by decomposing a global visibility graph into subgraphs. In our scheme, a subgraph, called *a local visibility graph*, is created for each polygon. Messages for building routing tables are only locally exchanged among VOP nodes within the same polygon. In addition, LVGR does not rely on the "path-setup" process. To achieve this, LVGR strategically switches between two routing modes,

namely *outside-convex routing* and *inside-convex routing*, depending on where forwarding and destination nodes are. The outside-convex routing is used as the default routing-mode. The outside-convex routing generates a set of intermediate destinations along the shortest path to the destination node. When either a source node finds that it is inside a convex hull, or when a node at a last intermediate destination finds that a destination node is inside a convex hull, the routing mode is switched to inside-convex routing. We develop a *gateway-based* forwarding algorithm for the inside-convex routing mode, that enables forwarding nodes to either route a packet out of a convex hull, or to deliver a packet to nodes inside a convex hull along an efficient routing path. We prove that, by effectively switching between the two routing modes, LVGR generates paths with guaranteed stretch.

3.2 Related Work

Hierarchical routing protocols and geographic routing protocols are well suited for large-scale wireless sensor networks [47]. A state-of-art hierarchical routing protocol S4 has worst-case path stretch of 3 and requires per-node state of $\mathcal{O}(\sqrt{N})$, where N is the total number of nodes. Compared with hierarchical routing protocols, geographic routing protocols are nearly stateless, thereby being highly scalable. However, a drawback of geographic routing protocols is that, due to the local minimum caused by network holes, the path length can be as long as $\Omega(c^2)$, where c is the optimal path length [48].

A number of protocols have been proposed to efficiently avoid a local minimum. Fang et al. [8] developed the TENT rule that nodes use to determine whether they are in a local minimum. Arad et al. [51] identified the nodes in a local minimum by using the angle between two adjacent neighbors. However, this heuristic approach causes frequent failures of the algorithm. Liu et al. [50] addressed the problem by

dividing a network into k regions, and allowing each node to maintain a vector of size k, where the i-th element of the vector indicates whether this node is at a local minimum with respect to the i-th region. These protocols suffer from the "late reaction problem" [52] – a routing path is corrected only after a packet reaches a local minimum, thereby increasing path stretch.

Some protocols propose to use non-local information to address the late reaction problem and ultimately improve path stretch. Jiang et al. [49] introduced the concept of "unsafe area" defined around a local minimum. Nodes in an unsafe area are notified of the existence of the hole, so that the nodes can make an early detour around the hole before a packet reaches the local minimum. Li et al. [52] introduced a routing protocol that abstracts a hole as an ellipse. The abstract information is then broadcast to nodes within h hops from the hole, and used by the nodes to avoid the hole. Although these protocols succeeded in improving path stretch, they do not offer guaranteed path stretch.

There have been efforts to achieve guaranteed stretch. Flury et al. proposed an embedding algorithm that achieves stretch of $O(logn)$ [2]. However, it is a centralized algorithm based on the Unit Disk Graph (UDG) assumption with the stretch depending on the network size n. Kermarrec and Tan [1] proposed a decomposition-based protocol with worst-case stretch of 7, but the stretch is possible by modeling a network in continuous domain, based on a large number of time-synchronized flooding operations. Tan et al. introduced VIGOR [3], the first geographic routing protocol for large-scale sensor networks that achieves worst-case path stretch of $\Theta(1)$ without the UDG assumption and time-synchronization. This protocol concisely represents holes in a network as polygons. VIGOR then builds a *visibility graph* with VOP nodes as the vertex set, and line segments connecting two visible VOP nodes as the edge set. VIGOR achieves the claimed path stretch by using the visibility graph as a

backbone network, i.e., running a distance vector routing algorithm on the visibility graph. However, the path stretch is possible with a "path-setup" process where a source node first exchanges a message with a destination node using a default geographic routing, prior to data transmission. This implies that the claimed path stretch no longer holds if we take the path built for the "path-setup" process into account. Furthermore, since VOP nodes must iteratively exchange messages with neighboring VOP nodes until their routing tables converge, non-negligible communication overhead is involved.

3.3 Preliminaries and Problem Formulation

This section presents terms, notations, and definitions used throughout Section 3, and formally describes our problem. We consider a static wireless sensor network consisting of N nodes denoted by a set $V = \{v_1, v_2, \cdots, v_N\}$. We assume that nodes have an unique ID; nodes are localized by using either an on-board GPS, or some node localization mechanism [33]; a source node knows the location of a destination node based on a location service [53], or hash-functions in a data centric storage scheme [54]. Throughout Section 3, we denote a source node by s and a destination node by t.

There are k network holes, denoted by $\{H_1, H_2, \cdots, H_k\}$, in a network. Each hole H_i is represented as a polygon $P_i = \{p_1, p_2, \cdots, p_n\}$, where each element p_i refers to a VOP node that represents a vertex of the polygon. A set $C_i = \{c_1, c_2, \cdots, c_n\}$ denotes the convex hull of polygon P_i (i.e., the convex hull of vertex set P_i), where c_i is an extreme point of the convex hull (i.e., $C_i \subseteq P_i$).

Before we define a visibility graph, we first define "visibility" between two VOP nodes as follows:

Definition 1 *Given two VOP nodes p_i and p_j, they are **visible** with each other if*

and only if line segment $\overline{p_i p_j}$ does not intersect any polygon P_i in the network. ■

A visibility graph and an augmented visibility graph are then defined as follows:

Definition 2 *A **visibility graph** is a graph $G_{vis} = (V_{vis}, E_{vis})$, where the vertex set $V_{vis} = \bigcup_{i=1}^{k} P_i$, and the edge set $E_{vis} = \{(p_i, p_j) |\ p_i$ and p_j are visible; $p_i, p_j \in V_{vis}\}$.* ■

Definition 3 *An **augmented visibility graph** for source s and destination t is a graph $G'_{vis} = (V_{vis} \bigcup \{s, t\}, E'_{vis})$, where $E'_{vis} = \{(p_i, p_j) |\ p_i$ and p_j are visible; $p_i, p_j \in V_{vis} \bigcup \{s, t\}\}$.* ■

The Euclidean distance between two nodes v_i and v_j is denoted by $d(v_i, v_j)$. We denote a path between two nodes v_i and v_j by $v_i \sim v_j$. The length of path $|v_i \sim v_j|$ is defined as the sum of Euclidean distances between two adjacent nodes along the path; for example, the length of path $v_1 - v_2 - v_3$ is $\sum_{i=1}^{i=2} d(v_i, v_{i+1})$.

Having defined all terms, we can formally describe the visibility graph-based geographic routing (VG-G), the problem that VIGOR addresses.

Definition 4 *Given source s and destination t, the **visibility-graph based geographic routing (VG-G)** is to identify a path $s \sim t$ such that $|s \sim t|$ is minimized on the augmented visibility graph for s and t.* ■

A significant issue for VG-G is that, in order to build an augmented visibility graph for a pair s and t, source node s must exchange a control packet with destination node t. As we will show later, using the control packet not only incurs high communication overhead, but also affects the path stretch as it is sent along a suboptimal path. Thus, our first objective is to develop a solution that does not rely on an augmented visibility graph (i.e., a solution using only a visibility graph). Our

Figure 3.3: An illustration of LVG for holes H_1 and H_2.

second objective is to make the size of the visibility graph as small as possible in order to run a distance vector routing on a resource-constrained sensor mote; thus, we decompose the global visibility graph into multiple sub-visibility graphs, called *local visibility graphs*. Figure 3.3 shows an example of local visibility graphs. We formally define a *local visibility graph* as follows:

Definition 5 *A **local visibility graph** for hole H_i is a graph denoted by $G_{vis,i} = (P_i, E_{vis,i})$, where $E_{vis,i} = \{(u,v)|\ u,v \in P_i$; u and v are visible.$\}$* ■

Now we are ready to formally describe our problem, namely the *local visibility graph based geographic routing (LVG-G)*:

Definition 6 *Given source s and destination t, the **local visibility graph-based geographic routing (LVG-G)** identifies a path $s \sim t$ such that $|s \sim t|$ is minimized by using local visibility graphs for holes in a network, without relying on augmented visibility graphs.* ■

LVGR implements technical details to solve LVG-G.

3.4 Local Visibility Graph-based Geographic Routing (LVGR)

In this section, we first present an overview of LVGR and then explain the details of its components.

3.4.1 Overview

Before we present an overview of LVGR, we first precisely define a hole. In a planarized network, each face is surrounded by multiple edges. The size of a face is defined as the number of surrounding edges. Tan et al. [3] define a hole as a face with size greater than a predefined threshold ω, a system parameter. Other faces having fewer edges than the threshold are called small faces and have a relatively small impact on path stretch; thus they are not considered.

The LVGR protocol consists of four phases: *Boundary Node Detection, Polygon Construction, Overlay Network Construction, and Data Forwarding*. In the *Boundary Node Detection* phase, LVGR identifies the nodes on the surrounding edges of holes, which are called *boundary nodes*. We use the boundary detection algorithm discussed in Section 2. During this phase, one boundary node for each hole is selected as a leader node who is responsible for initiating the next phase: *Polygon Construction*. In the *Polygon Construction* phase, a leader node for each hole sends a probing packet that travels along the surrounding edges to identify VOP nodes for the hole. When a probing packet, containing the locations of identified VOP nodes, returns to a leader, the leader node broadcasts the locations. In the *Overlay Network Construction* phase, a local visibility graph is built for each hole. The local visibility graph represents the internal structure of a hole and is used as an overlay network for routing a packet optimally inside the convex hull of a hole. Once the *Overlay Network Construction* phase is finished, nodes have the locations of VOP nodes in the network; if they are VOP nodes, they also have the information about the internal structure of the hole in the form of a routing table. Based on VOP locations and the information about internal structures, in the *Data Forwarding* phase, nodes can send packets along a path with guaranteed path stretch.

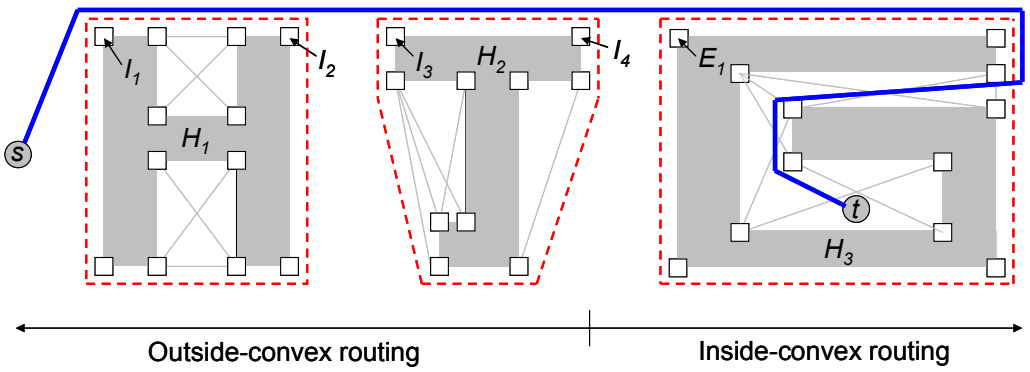


Figure 3.4: An illustration of the data forwarding algorithm.

We develop a data forwarding algorithm (used in the *Data Forwarding* phase) for achieving guaranteed path stretch. It provides two routing modes, namely *outside-convex routing* and *inside-convex routing*. Forwarding nodes use the outside-convex routing as the default routing mode. More specifically, given source node s and destination node t, the outside-convex routing determines a set of intermediate destinations along the shortest path – the details on how intermediate destinations are selected will be discussed in Section 3.4.5. Figure 3.4 shows an example, where I_1, I_2, I_3, and I_4 are intermediate destinations given s and t. If s is inside a convex hull, our forwarding algorithm switches its mode to the inside-convex routing so that it can guide a packet out of the convex hull and resume the outside-convex routing. If s finds that t is inside a convex hull, it first finds intermediate destinations using the outside-convex routing without considering the convex hull containing t; and then when the packet reaches the last intermediate destination, the node at the last intermediate destination switches its mode to the inside-convex routing and delivers the packet to t inside the convex hull along the locally optimal path according to the local visibility graph. In our example (Figure 3.4), intermediate destination I_4, upon receiving a packet, changes its mode to inside-convex routing, finds entry point

E_1, and forwards the packet to E_1. Upon receiving the packet, the VOP node at E_1 routes the packet along the optimal path to destination t according to a distance vector routing running on the local visibility graph.

3.4.2 Boundary Detection

For the Boundary Detection phase, we adopt the boundary detection algorithm introduced in Section 2. Recall that one node sends a control packet that travels along the boundary of a hole. This node is called a leader node. A leader node checks the size of an adjacent face by looking at the number of edges for the face contained in the returned control packet. If a leader node determines that a face is a hole (i.e., the size of the hole is greater than predefined threshold ω), it initiates the next phase; otherwise, it drops the packet.

3.4.3 Polygon Construction

Once the *Boundary Detection* phase ends, leader nodes initiate the *Polygon Construction* phase. Each leader node sends a probing packet, which travels along the closed chain of identified boundary nodes in a clockwise direction and eventually returns to the leader. While traversing the boundary nodes, VOP nodes are identified among the boundary nodes. The probing packet contains three fields: the first field stores the locations of VOP nodes identified so far; the second field contains the locations of visited boundary nodes after the most recently identified VOP node; the third field has parameter δ that specifies the width of a bounding box used to identify VOP nodes. To be more precise, upon receiving a probing packet, a boundary node adds its location to the second field of the packet and checks whether there exists a bounding box with width δ that contains all locations in the second field of the probing packet. If such a bounding box exists, the boundary node forwards the packet to the next boundary node in a clockwise direction; otherwise, the current

boundary node is a VOP node; thus, the location of the node is appended to the first field (i.e., the set of identified VOP nodes) of the packet. Then, the second field is emptied and the packet is forwarded to the next boundary node. The *Polygon Construction* phase also adopts a mechanism [3] for eliminating possible intersecting edges among polygons.

3.4.4 Overlay Network Construction

After the *Polygon Construction* phase, the leader node for each hole has the locations of VOP nodes corresponding to the hole. Leader nodes then broadcast the locations. Once nodes in a network receive the locations of VOP nodes, a polygon for each hole can be found by sequentially connecting the VOP nodes for the hole. Given a polygon, the convex hull representation can be easily computed using a simple convex hull algorithm. These two types of information (i.e., the polygon and convex hull representations) are used for the next phase, *Data Forwarding*. As described in Section 3.4.1, when both s and t are outside the convex hulls of holes, the outside-convex routing mode is used; otherwise, i.e., if either s or t is inside convex hulls, the "internal structure" of a hole is used for the inside-convex routing mode – the internal structure is represented as a "local visibility graph". The *Overlay Network Construction* phase is responsible for building a local visibility graph so that VOP nodes inside convex hulls can run a distance vector routing to optimally route a packet inside convex hulls.

The idea for building an internal structure is the following. Based on a set of polygons (i.e., holes) in a network, a node can check whether it is inside a convex hull or not; a node can also find a set of visible nodes by checking whether the line segment connecting itself and a target node intersects any polygons in the network. VOP nodes inside a convex hull then exchange their routing tables with their visible VOP

nodes within the same convex hull, based on a distance vector routing – we adopt DSDV [55]. After several iterations of the routing-table-exchanges, their routing tables converge, finishing the construction of the internal structure of a hole. The details of how the internal structure is used for optimally routing a packet to a destination node inside a convex hull – or routing a packet from a source node inside a convex hull to the outside of the convex hull – are presented in the next section. One important aspect of this phase compared with the state-of-art is that the overlay network is constructed locally for each convex hull, instead of building a global overlay network, which incurs significant communication overhead as we will show in Section 3.6.

3.4.5 Data Forwarding

This section describes our forwarding algorithm used in the *Data Forwarding* phase. Algorithm 3 depicts the pseudocode. The algorithm has two forwarding modes: *outside-convex routing* and *inside-convex routing*, as follows.

Routing Mode 1. Outside-Convex Routing

A visibility graph is a widely used data structure to find a shortest path for motion-planning of mobile robots [56]. However, computing a shortest path based on a visibility graph *locally* in a sensor node is not feasible, because the data structure requires $\mathcal{O}(N^2)$ memory [57], where N is the number of vertices of a visibility graph. Compared with a visibility graph, *a tangent visibility graph* is a lightweight data structure used to find a shortest path given convex obstacles [58]. One important characteristic of a tangent visibility graph is that it is not dependent on the number of vertices: its memory requirement is $\mathcal{O}(h^2)$, where h is the number of convex hulls. This characteristic of being independent of the number of vertices allows for the local computation of a shortest path in a sensor node. As an extreme example, consider

Algorithm 3 Forwarding in LVGR

```
1: Call outside_convex(s, t). // Default routing mode.
2: outside_convex(s, t):
3: if ∃i s.t. s and/or t ∈ C_i then
4:     Build a tangent visibility graph without C_i.
5: else
6:     Build a tangent visibility graph.
7: end if
8: Find intermediate destinations {I_1, ...I_n} by using Dijkstra's algorithm.
9: Forward to I_1 through I_n.
10: // Routing mode change.
11: if (∃i s.t. s ∈ C_i) || (s = I_n && ∃i s.t. t ∈ C_i) then
12:     Call inside_convex(s, t).
13: end if
14: inside_convex(s, t):
15: if ∃i s.t. s ∈ C_i then
16:     if t ∈ U then
17:         Forward to t.
18:     else
19:         Choose v ∈ U s.t. d_v is minimized.
20:         Forward to v towards the gateway based on LVG.
21:     end if
22: else
23:     if t ∈ U then
24:         Forward to t.
25:     else
26:         if g_1 ∉ U and g_2 ∉ U then
27:             Choose v ∈ U s.t. d_v is minimized.
28:         else
29:             Choose v ∈ U geographically closest to t.
30:         end if
31:         Forward to v towards t based on LVG.
32:     end if
33: end if
```

a network with a large number of holes – say 100 holes with 50 vertices for each hole; in the worst case, the memory requirement for a tangent visibility graph is 10KBytes, while the memory requirement of a visibility graph is 25MBytes, which is not acceptable for a typical sensor node with ∼512KB of RAM [59].

Given source node s and destination node t, s uses the outside-convex routing as the default routing mode (Line 1). When the *Overlay Network Construction* phase ends, nodes have the locations of VOP nodes in the network. Based on the locations of VOP nodes, nodes can compute the convex hull of each polygon. Given the convex hulls and the locations of s and t, the outside-convex routing first considers both s

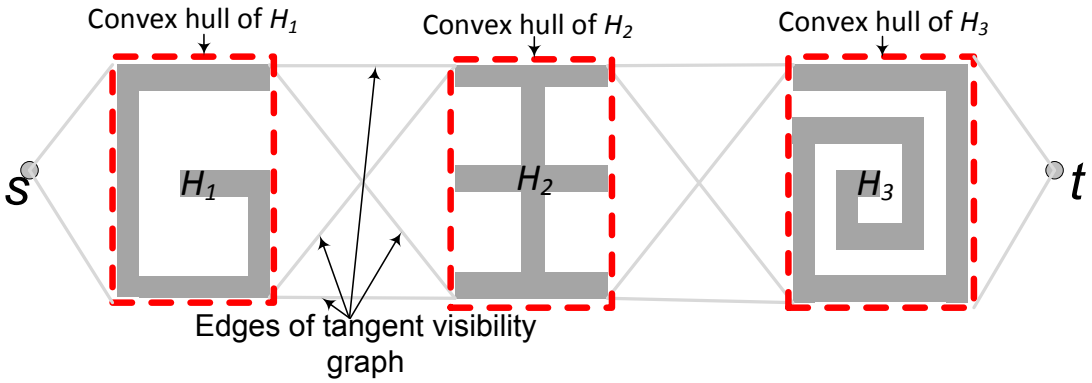


Figure 3.5: An illustration of the outside-convex routing.

and t as degenerate convex hulls. Then it builds a tangent visibility graph (Lines 3-6) with the edges as tangent lines between two convex hulls, and the vertices as tangent points, as shown in Figure 3.5. The time complexity of the construction of a tangent visibility graph is $\mathcal{O}(N \log N + h)$ [58]. To find the shortest path from s to t, the outside-convex routing then applies the Dijkstra algorithm on the computed tangent visibility graph. Consequently, the outside-convex routing produces a set of intermediate destinations $\{I_1, I_2, ..., I_n\}$ which are the tangent points on the shortest path – they are also VOP nodes (Line 7). Source node s then sends a packet to the first intermediate destination, i.e., I_1 (Line 8). If either s finds that it is inside a convex hull, or the last intermediate destination I_n finds that t is inside a convex hull, they change the routing mode (Lines 10-11) to inside-convex routing (which will be discussed in the following section). The outside-convex routing produces a path with constant stretch when s and t are outside convex hulls. We provide the proof in Section 3.5.

Routing Mode 2. Inside-Convex Routing

Nodes change their routing modes to inside-convex routing in the following two cases: 1) the node at the last intermediate destination finds that the destination

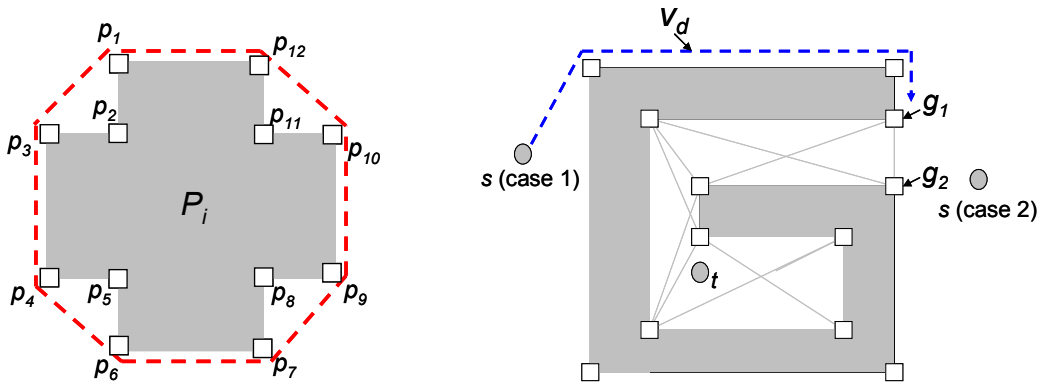


Figure 3.6: Examples of: (a) inner holes; and (b) entry point selection.

node is inside a convex hull; or 2) the source node finds that it is inside a convex hull. We first explain the first case. In order to understand the inside-convex routing algorithm, we need to define several terms as follows:

Definition 7 *A j-th **inner hole** of polygon P_i is a set of sequentially ordered VOP nodes denoted by $P_{ij} = \{p_1, ..., p_n\}$, where $P_{ij} \subset P_i$ such that p_1 and p_n are the extreme points of convex hull C_i. In particular, p_1 and p_n are called **gateway** nodes.*

■

Figure 3.6(a) shows an example with four inner holes and eight gateway nodes. There is a polygon $P_i = \{p_1, ..., p_{12}\}$, and its convex hull representation denoted by $C_i = \{p_1, p_3, p_4, p_6, p_7, p_9, p_{10}, p_{12}\}$; polygon P_i has four inner holes: $P_{i1} = \{p_1, p_2, p_3\}$, $P_{i2} = \{p_4, p_5, p_6\}$, $P_{i3} = \{p_7, p_8, p_9\}$, and $P_{i4} = \{p_{10}, p_{11}, p_{12}\}$. The gateways for P_{i1}, P_{i2}, P_{i3}, and P_{i4} are $\{p_1, p_3\}$, $\{p_4, p_6\}$, $\{p_7, p_9\}$, and $\{p_{10}, p_{12}\}$, respectively.

To describe how the inside-convex routing works for the first case, we let u be the node at the last intermediate destination which sends a packet to destination node t inside convex hull C_i. The objective of node u is to choose a VOP node v (called an *entry point*) from its visibility set in P_i (i.e., the set of visible VOP nodes in P_i)

such that $|u \sim v| + |v \sim t|$ is minimized. Note that the visibility set can be easily determined by checking whether $\overline{uv}$ intersects any edge of P_i. If u finds entry point v, u forwards a packet to it. After reaching v, the packet is transmitted along the shortest path to t based on the local visibility graph for P_i. Next, we describe the details of the entry point selection process.

To determine an entry point, the inside-convex routing first finds the two gateways of the inner hole containing destination node t, which is done based on the sequence of VOP nodes and a simple polygon-in-point algorithm [60]. Denote the two gateways by g_1 and g_2 and let a set U be the visibility set of node u. Now we have to consider two cases as shown in Figure 3.6(b): 1) both g_1 and g_2 are not in U (Lines 23-24); and 2) one of g_1 and g_2 (or both g_1 and g_2) is in U (Lines 25-26). In the first case, we know that the shortest path must pass one of the two gateways. Thus, we find $v \in U$ that minimizes $|u \sim v| + |v \sim t|$ as follows: we compute the lengths of two paths from each $v \in U$ to g_1 and g_2, respectively, by sequentially following the edges of the convex hull of polygon P_i; let d_v denote the shorter distance between the two lengths (see Figure 3.6(b)); we then choose $v \in U$ that minimizes d_v. The second case is a little bit trickier: we cannot simply choose node $v \in U$ that is closer to either of gateways, because in this case, some nodes in U might be inside the inner hole containing t. A problem is that nodes outside a convex hull do not know about the internal structure of the hole (i.e., the local visibility graph of the hole); thus, it is difficult to optimally choose such $v \in U$. To tackle this case, we introduce a simple heuristic: node u chooses $v \in U$ that is geographically closest to t. We will prove in Section 3.5 that this heuristic method yields the stretch of path $u \sim t$ bounded by $\frac{(\pi+2)}{2c}D$, where D is the diameter of the network and c is the communication radius.

Now we explain the second case where the source node is inside a convex hull. This case includes the scenario where both source and destination nodes are inside

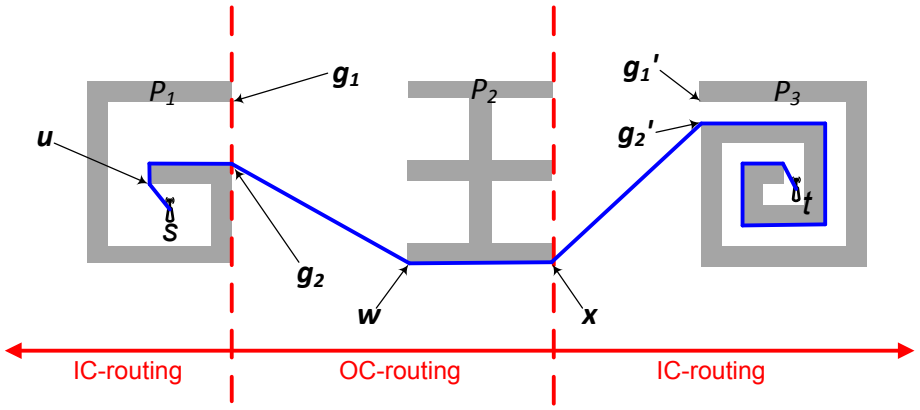


Figure 3.7: An example for the forwarding algorithm.

the same convex hull. Note that if source node s does not have interfering holes (i.e, the line segment connecting s and t does not intersect any polygons), t is the final destination; otherwise t is the first intermediate destination. Source node s can find the inner hole that it belongs to, and the two gateways of the inner hole. The objective of source node s is to choose a node v in its visibility set U that minimizes the distance $|s \sim v| + |v \sim t|$. Note that, after reaching node v, a packet is transmitted out of the convex hull along the shortest path following the local visibility graph. However, source node s cannot make the optimal selection of node v, because it does not have knowledge of hole's internal structure; thus, we introduce a heuristic mechanism. This heuristic algorithm runs as follows. If node t is in U (Line 14), source node s sends a packet to node t using simple geographic forwarding (Line 15); otherwise, s chooses v in U such that d_v is minimized, i.e., the minimum distance to either of gateways (Lines 16-18). We will prove in Section 3.5 that this heuristic algorithm produces paths $s \sim t$ with bounded stretch of $\frac{(\pi+2)}{2c}D$.

Figure 3.7 shows an example describing the operation of our forwarding algorithm. Source node s first runs the default outside-convex routing, identifying intermediate

destinations w and x. When s wants to send a packet to the first intermediate destination w, it finds that it is inside a convex hull C_1. At the same time, since $t \notin U$, s finds a visible VOP node u that minimizes d_u. Then, s sends a packet to u. The packet is forwarded to gateway g_2 along the internal local visibility structure. Once the packet reaches gateway g_2, the outside-convex routing is resumed; the forwarding node at g_2 determines the intermediate destinations w and x again. Upon reaching node x, forwarding node x detects that t is inside a convex hull, so that it changes its mode to inside-convex routing. Node x then identifies the entry point g_2' and forwards the packet to it. Upon reaching the entry point, the local visibility graph for P_2 is used to guide the packet to destination t along the shortest path.

3.5 LVGR Protocol Analysis

3.5.1 Path Stretch

In this section, we prove the worst-case stretch of LVGR. Lemma 1 proves the constant stretch for the outside-convex routing. We then prove the worst-case stretch for the inside-convex routing in Lemma 2 and Lemma 3. Based on the Lemmas, we prove the stretch of LVGR in Theorem 1, considering a general case when both outside-convex and inside-convex routing are used.

Lemma 1 *The outside-convex routing when s and t are outside convex hulls has constant path stretch.*

Proof: Given s, t, and convex hulls, the outside-convex routing computes a tangent visibility graph. The Dijkstra algorithm performed on the tangent visibility graph yields a shortest path [58], say $s \sim I_1 \sim I_2 \sim ... \sim I_n \sim t$; according to Tan et al. [3], the stretch of a path segment between two VOP nodes, i.e., $I_i \sim I_{i+1}$ including $s \sim I_1$ and $I_n \sim t$ is bounded by constant factor $\zeta = 16(\chi+1)(H+1)(H+1+\frac{2}{\pi})$, where H

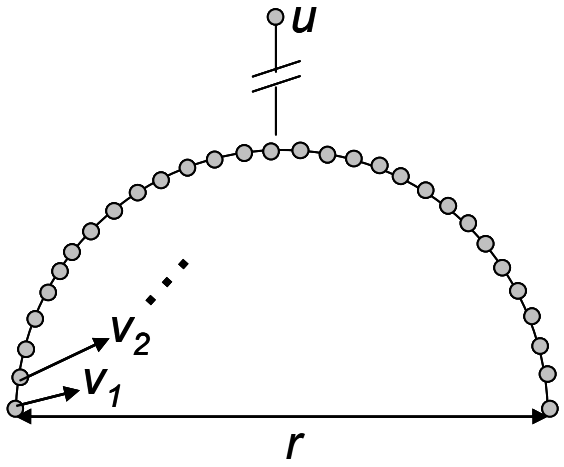


Figure 3.8: An illustration for the bounded stretch of inside-convex routing.

and χ are system parameters (see [3] for details about parameters). Thus, we obtain the following: $|s \sim I_1 \sim I_2 \sim ... \sim I_n \sim t| \leq \zeta \cdot |OPT(s,t)|$, where $|OPT(s,t)|$ is the optimal path length. ■

Recall that nodes change their routing mode to inside-convex routing when: 1) node u at the last intermediate destination finds that destination node t is inside a convex hull; or 2) source node s finds that it is inside a convex hull. We first prove the worst-case stretch for the first case.

Lemma 2 $|u \sim t| \leq (\frac{(\pi+2)}{2c}D) \cdot |OPT(u,t)|$, *where* $|OPT(u,t)|$ *is the optimal path length of path* $u \sim t$, D *is the diameter of the network, and* c *is the communication radius of a node.*

Proof: Let n VOP nodes, for hole P_i with diameter r, that are visible from node u, be denoted by a set $U = \{v_1, v_2, ..., v_n\}, U \subseteq P_i$. Our heuristic method allows node u to select $v \in U$ such that $d(u,v) + d(v,t)$ is minimized. Suppose that node v is not an optimal selection; that is, there exists $v' \in U$ for optimal choice. We are interested in the maximum distance between v and v' (which is the price that our

heuristic method should pay for the wrong decision). Thus, the problem is to arrange nodes in U such that all n nodes are visible from u, and the distance $\sum_1^{n-1} d(v_i, v_{i+1})$ is maximized. As illustrated in Figure 3.8, we can find that, when we arrange n nodes along the semicircle with radius r, $\sum_1^{n-1} d(v_i, v_{i+1})$ is maximized and all n nodes in U are visible from u. In this case, the worst-case path length becomes $|u \sim v'| + |v' \sim v| + |v \sim t|$. And optimal path length is $|u \sim v| + |v \sim t|$. So the path stretch is $\frac{|u \sim v'| + |v' \sim v| + |v \sim t|}{|u \sim v| + |v \sim t|}$. Thus,

$$\frac{|u \sim t|}{|OPT(u,t)|} \leq \frac{|u \sim v'| + |v' \sim v|}{|u \sim v|}$$

$$= \frac{|u \sim v'|}{|u \sim v|} + \frac{|v' \sim v|}{|u \sim v|} \leq \frac{D}{c} + \frac{\pi r/2}{c} \leq \frac{D}{c} + \frac{\pi D/2}{c} = \frac{(\pi + 2)}{2c} D.$$

■

It is easy to note that the worst-case stretch for the second case is the same as Lemma 2, because the proof for the second case is basically to find the maximum cost for the suboptimal selection. Thus, considering both cases, we obtain the following result:

Lemma 3 *Inside-convex routing has stretch of $\frac{(\pi+2)}{2c} D$, where D is the diameter of the network, and c is the communication radius of a node.*

Now finally, when combining the outside-convex and inside-convex routing, we obtain the following result:

Theorem 1 *LVGR has the worst-case path stretch of $O(\frac{D}{c})$, where D is the diameter of the network, and c is the communication radius of a node.*

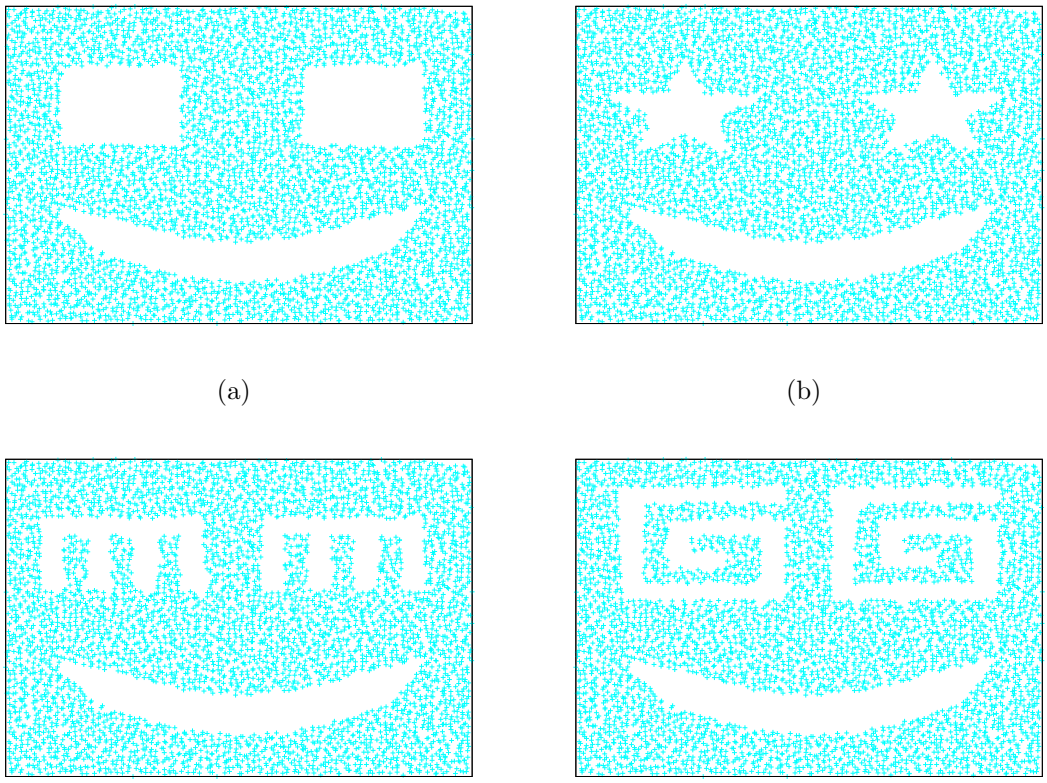
Proof: Given a $s - t$ pair, we denote the optimal path by $OPT(s, t)$. If convex hulls containing s and t are not considered, the outside-convex routing produces interme-

diate destinations $\{I_1, ...I_n\}$ along the shortest path connecting s and t. According to Lemma 1, the outside convex routing produces a path with constant stretch. If we take the convex hulls containing s and t into account, the stretches of path segments $s \sim I_1$ and $I_n \sim t$ are affected. Specifically, Lemma 3 describes that the stretch of such path segment is $\frac{(\pi+2)}{2c}D$, where r' is the diameter of the largest hole in the network. Therefore, the length of path $\mathbb{P}$ after taking the holes containing s and t into account is given as follows: $|\mathbb{P}| \leq (\frac{(\pi+2)}{2c}D) \cdot \zeta \cdot |OPT(s,t)|$. This proves the theorem. ■

3.5.2 Message Complexity

The message complexity of VIGOR is $O(N(V_{VOP}^2 M + \mathcal{C}))$, where N is the number of nodes; V_{VOP} is the number of VOP nodes; M is the number of links (i.e., visible edges) between VOP nodes; and $\mathcal{C}$ is the number of holes in the network. First, it takes $O(N)$ messages to construct VOP polygons. Second, it takes $O(N\mathcal{C})$ messages to broadcast VOP nodes throughout the network. Third, to set up the distance vector routing for a visibility graph with V_{VOP} vertices and M edges, $O(NV_{VOP}^2 M)$ messages are needed, because the message complexity of a general graph with n vertices and m edges for setting up a distance vector routing is $O(n^2 m)$ (i.e., until routing tables converge), and a single message transmission on an edge of a visibility graph for VIGOR requires at most N messages (i.e., an edge between two VOP nodes may have at most N hops). Therefore, VIGOR requires $O(N(V_{VOP}^2 M + \mathcal{C}))$ messages in total.

The message complexity of LVGR can be similarly derived. The messages used for the first and second part (i.e., VOP polygon construction and brodcast of VOP locations, respectively) are the same as $O(N)$ and $O(N\mathcal{C})$, respectively, The difference happens in the third step where only VOP nodes within the same hole are involved in



(a) (b)

(c) (d)

Figure 3.9: Different hole deployment schemes: (a) scenario 1; (b) scenario 2; (c) scenario 3; and (d) scenario 4.

the message complexity; thus, for the third step $O(NV'^2_{VOP}M')$ messages are needed, where N'_{VOP} is the number of VOP nodes for the largest hole in the network, and M' is the number of links between VOP nodes belonging to the largest hole. Therefore, LVGR requires $O(N(V'^2_{VOP}M' + \mathcal{C}))$ messages in total. Since $M' \leq M$ and $V'_{VOP} \leq V_{VOP}$, the message complexity of LVGR is smaller than VIGOR.

3.5.3 Per-Node State

Both VIGOR and LVGR require nodes to keep a routing table. The routing state per node of VIGOR is $O(N_{VOP})$ where N_{VOP} is the number of VOP nodes in

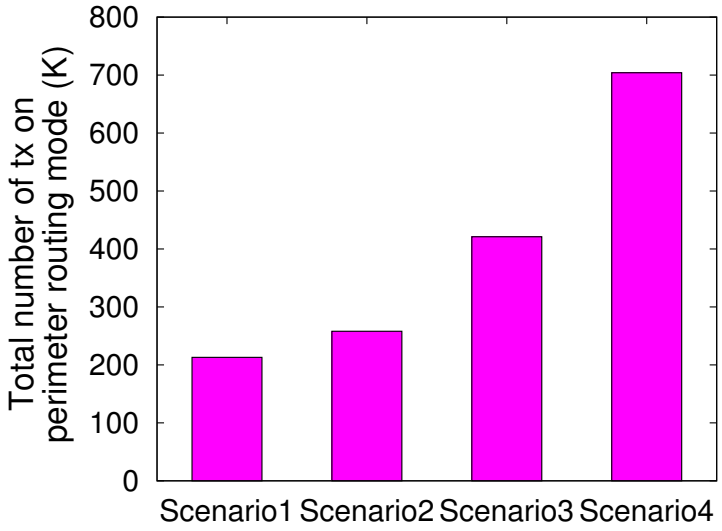


Figure 3.10: Total number of packet transmissions in perimeter-routing mode.

the network, because VIGOR is based on a global visibility graph which requires a routing table with size of $O(N_{VOP})$. Meanwhile, the per-node state of LVGR is $O(N'_{VOP}), N'_{VOP} \leq N_{VOP}$, where N'_{VOP} is the number of VOP nodes of the largest hole in the network. The reason for the smaller per-node state is that LVGR is based on a local visibility graph which requires nodes to store only the local routing information within a hole.

3.6 Simulation Results

We implemented LVGR, VIGOR [3], GPSR [10] and a centralized shortest path routing protocol in C++, mainly focusing on the topological behavior of the protocols. We randomly deployed 6,000 sensor nodes in a 2,000×2,000m^2 area having holes with different *complexities* as shown in Figure 3.9. The *complexity* of a hole was measured as the total number of packet transmissions in perimeter-routing mode for 10,000 randomly chosen source and destination pairs. Measured complexities for different hole-deployment scenarios are depicted in Figure 3.10. As shown, deploy-

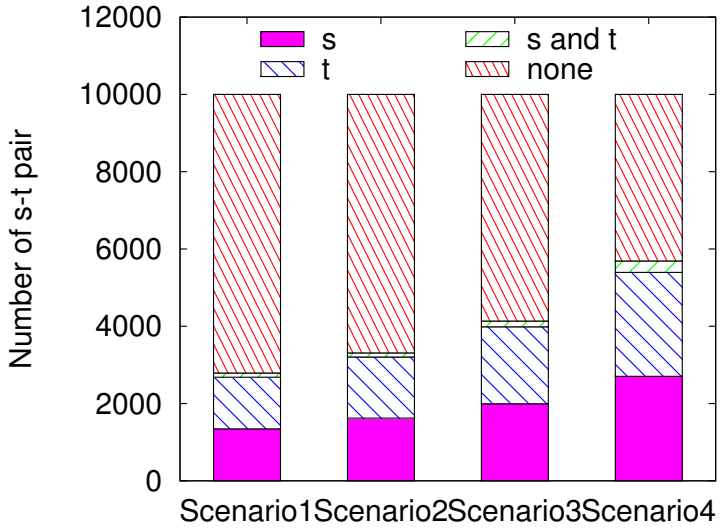


Figure 3.11: The number of $s-t$ pairs for which s and t are inside convex hulls.

ment scenarios are arranged in increasing order of holes complexity. The complexity affects how often LVGR changes its routing mode. Figure 3.11 shows that network scenarios with higher complexity have more source and destination nodes inside convex hulls. To account for realistic communication channels, we employed a radio model [61], which defines the *degree of irregularity* (DOI) as maximum radio range variation in the direction of radio propagation. Figure 3.12 shows radio range for DOI=0.4.

We compared VIGOR with LVGR and GPSR. Our main interests are to show that LVGR has lower communication overhead than VIGOR, while maintaining as good path stretch as VIGOR. In particular, the comparison with GPSR is used to provide the baseline performance. We measured the following metrics: (1) average path stretch, (2) maximum path stretch, and (3) communication overhead. Note that the storage overhead of LVGR and VIGOR can be simply measured as the number of VOP nodes for the largest hole, and the total number of VOP nodes,

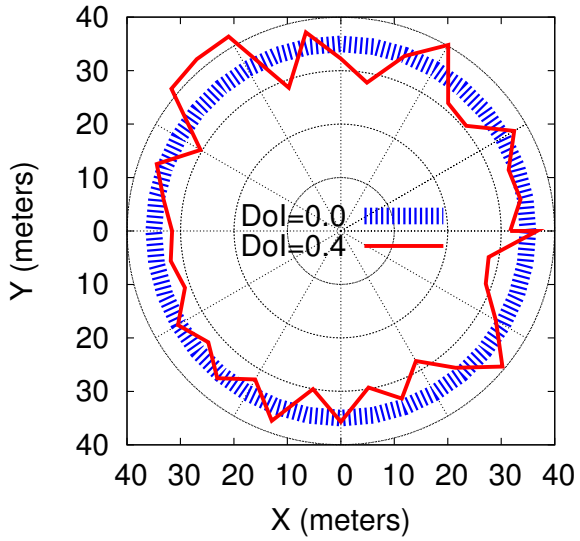


Figure 3.12: Radio range with DOI=0.4.

respectively. The communication overhead consists of two parts: one part measures the total number of packet transmissions for setting up routing tables; the second part measures the total number of control packet transmissions to set up routing paths. We varied the following parameters: (1) δ, (2) DOI, and (3) network density. Each simulation used 10,000 randomly selected source and destination pairs. The default network configuration was: DOI=0.4 with radio range of 50m, $\omega = 15$, and $\delta = 30$. The average degree of the network was approximately 10.

3.6.1 Path Stretch

We measured path stretches for 10,000 randomly selected source-destination pairs in each network scenario. In this set of experiments we used the following definition of path stretch: $path\_stretch = \frac{measured\_hop\_count}{minimum\_hop\_count}$. Given a source and destination pair, $measured\_hop\_count$ refers to the number of hops for the path connecting source s and destination t; and $minimum\_hop\_count$ is the number of hops for the shortest path between s and t. The shortest path was computed using a centralized shortest

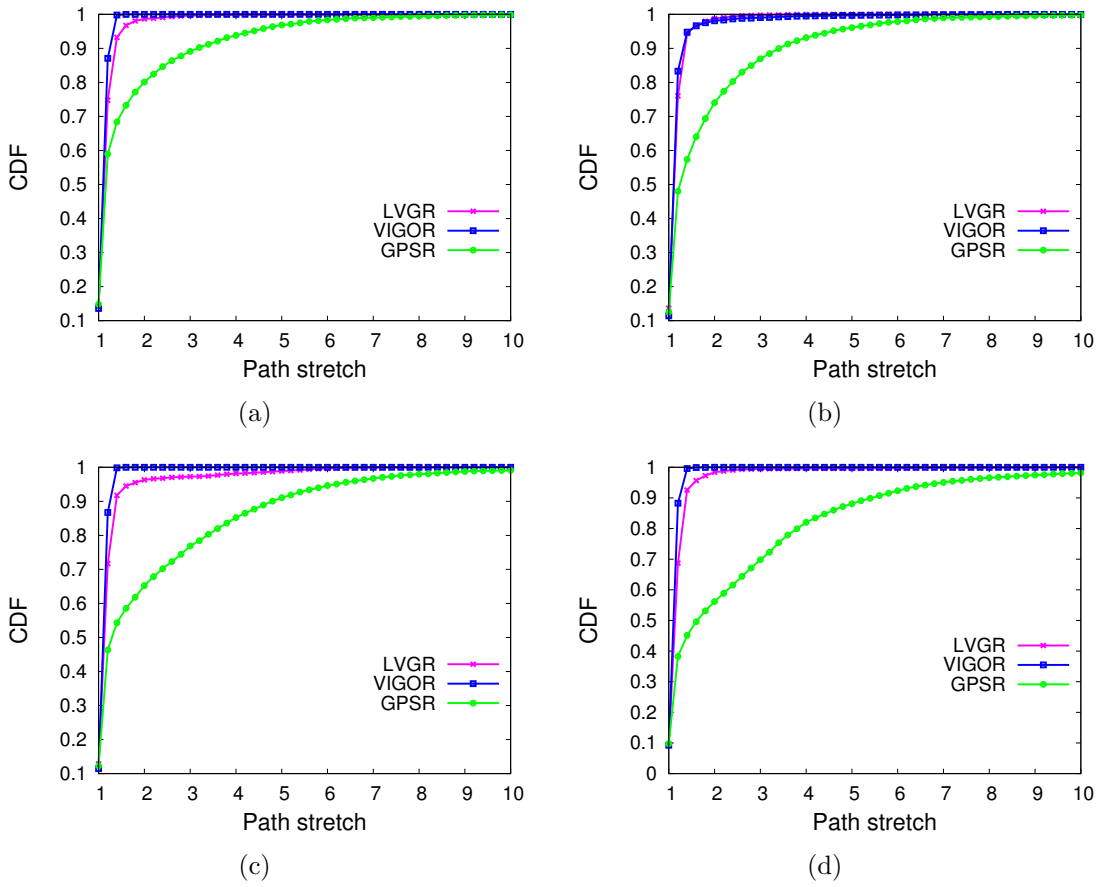


Figure 3.13: CDF graphs of path stretches for each deployment scenarios: (a) scenario 1; (b) scenario 2; (c) scenario 3; and (d) scenario 4.

path routing.

For each network scenario, we present the CDF of path stretches for 10,000 $s-t$ pairs. Figures 3.13(a), 3.13(b), 3.13(c), and 3.13(d) depict the CDFs for network scenarios 1, 2, 3, and 4, respectively. As shown, regardless of the complexity of deployed holes, more than 90% of the path stretches for both VIGOR and LVGR were close to 1. The main reason is that both protocols use non-local information (i.e., the abstract information on holes) to prevent a packet from falling into a local minimum. Another observation is that the path stretch of LVGR is slightly higher

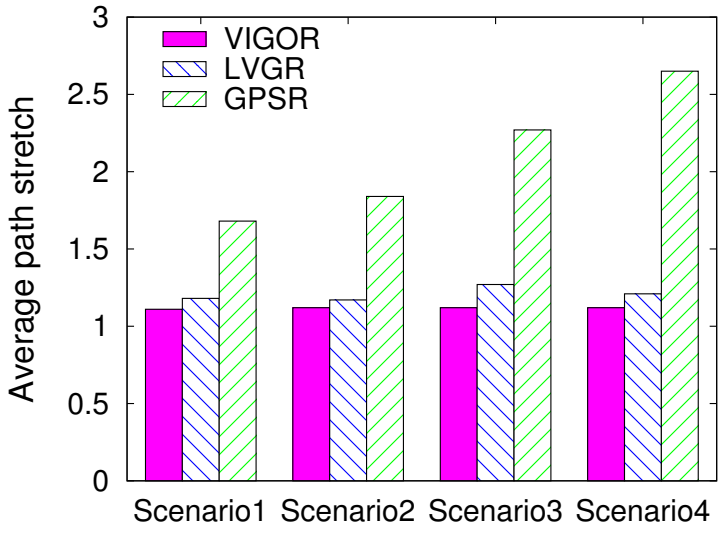


Figure 3.14: Average path stretch.

compared with VIGOR for the four network scenarios. This is because LVGR makes a heuristic decision in forwarding a packet when either the source or destination is inside a convex hull, while VIGOR makes an optimal decision by exchanging a control packet with a destination node using a default geographic routing. However, as we will show in Section 3.6.2, the overhead for the control packet poses a significant problem in large scale deployments. In contrast to the results of VIGOR and LVGR, GPSR is significantly affected by the complexity of holes; that is, as the complexity of holes increases from Scenario 1 to Scenario 4, the path stretch of GPSR severely degrades.

Average and maximum path stretches are statistical measures often used to compare the performance of routing protocols. We summarize the path stretch by means of the average and maximum path stretch. Figure 3.14 shows the average stretches of the three routing protocols. We observe that, while the average path stretch of GPSR increases as the complexity of holes increases, both VIGOR and LVGR show

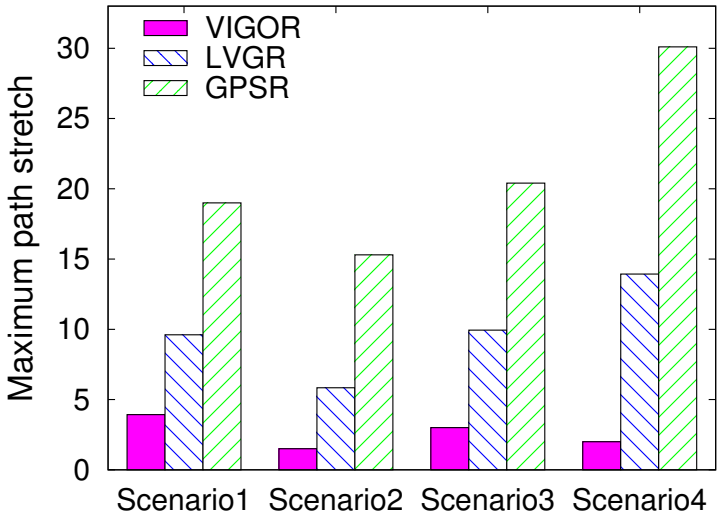


Figure 3.15: Maximum path stretch.

constantly low average path stretches close to 1. In particular, the difference between the average path stretch of VIGOR and LVGR is almost negligible. Figure 3.15 shows the maximum path stretches of the three routing protocols. As shown, the maximum path stretch of VIGOR and LVGR is unexpectedly much greater than 1, despite their optimal route selection. We found that this high maximum path stretch happened to paths with very small hop counts (e.g., $1 \sim 3$); more precisely, even a small increase in a hop count resulted in high path stretch for such paths. We identified that irregular communication range, and small holes that were not considered by the protocols were the main causes for such increases in path stretch. Another observation is that the maximum path stretch of LVGR is relatively higher than VIGOR. The main reason is the use of the heuristic forwarding algorithm to forward a packet to/from a node inside a convex hull. It is worth noting that unlike LVGR, for GPSR, the maximum path stretch is directly influenced by the complexity of holes.

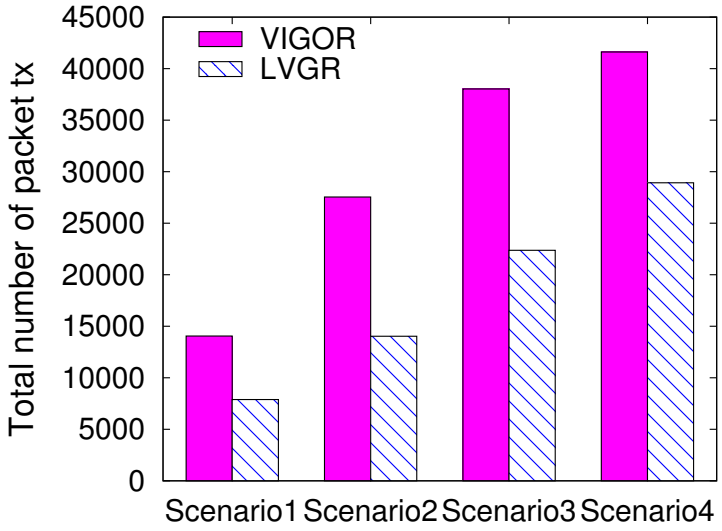


Figure 3.16: Overhead for routing table set up.

3.6.2 Communication Overhead

This section investigates the communication overhead of LVGR and VIGOR. Figure 3.16 shows the communication overhead for the "routing table set-up" of both LVGR and VIGOR. We first observe that the communication overhead for both protocols increases as the complexity of deployed holes increases. The reason is straightforward: the network requires more VOP nodes to represent more complex holes. We also note that, by decomposing the global visibility graph into local visibility graphs, we can reduce the communication overhead; specifically, we can see that LVGR reduces this type of communication overhead by up to 50% compared with VIGOR in our simulations. It is worth remarking that, although this type of communication overhead may not be important for a particular scenario where only a single hole is present, the communication overhead caused by the control packet used for the "path-setup" process still persists and has much higher impact on the performance regardless of hole-deployments.

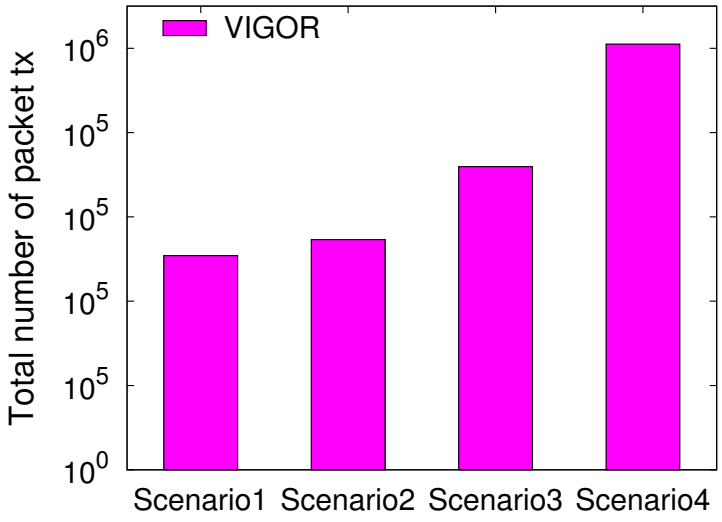


Figure 3.17: Overhead for routing path setup.

Figure 3.17 depicts the communication overhead for the "path-setup" process. We measured this type of communication overhead only for unique $s-t$ pairs, because the same $s-t$ pair can use the previously found routing path. We note that this type of communication overhead for VIGOR increases as the complexity of deployed holes increases. The reason is that VIGOR uses its underlying default geographic routing protocol to deliver the control packet for routing path-setup. As we noted in Section 3.6.1, the path stretches of traditional geographic routing protocols are significantly affected by the complexity of holes; therefore the communication overhead increases with higher complexity of holes. Also note that LVGR does not suffer from this type of communication overhead, resulting in significant improvements in energy efficiency. In summary, when considering both types of communication overhead, LVGR had a total of 28,920 packet transmissions, while VIGOR had 41,614 (overhead for routing table set up) + 1,010,000 (overhead for the "path-setup") in Scenario 4. Thus, LVGR had up to 97% smaller communication overhead.

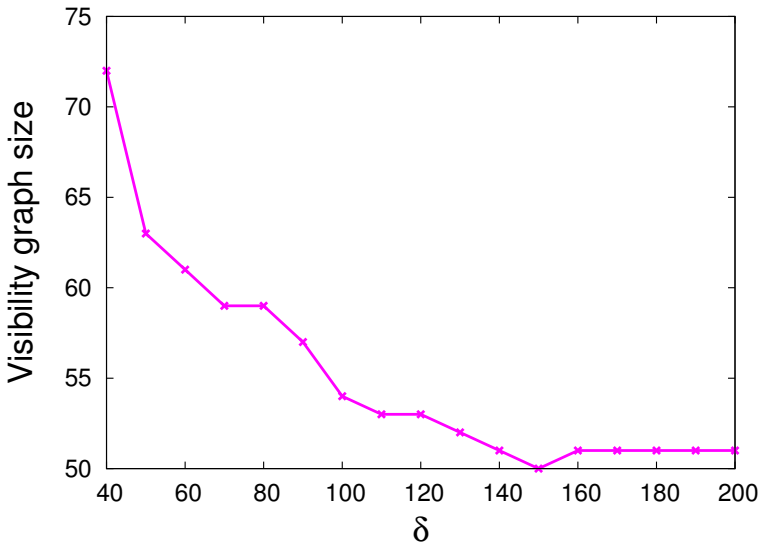


Figure 3.18: Impact of δ on storage overhead.

3.6.3 Impact of δ

The parameter δ determines the width of a bounding box used to find VOP nodes of a given hole. Using a bounding box with small δ allows us to represent holes with more VOP nodes; that is, holes are represented more precisely with smaller δ. We measured the number of VOP nodes for network Scenario 4 by varying δ. Figure 3.18 depicts the results. As shown, the size of the visibility graph decreases with the first few decreasing δ values, and the decrease rate slows down with higher δ. These results indicate that, if δ is sufficiently large, we can represent the visibility graph using a small number of VOP nodes, leading to lower communication overhead for routing table set-up for both VIGOR and LVGR. However, this is only possible at the cost of deteriorated path stretch of the protocol. The reason is that the small number of VOP nodes imprecisely represent holes, which leads to more packets being forwarded in perimeter routing mode. Furthermore, high δ may cause crossing edges among polygons, which also contributes to increased path stretches.

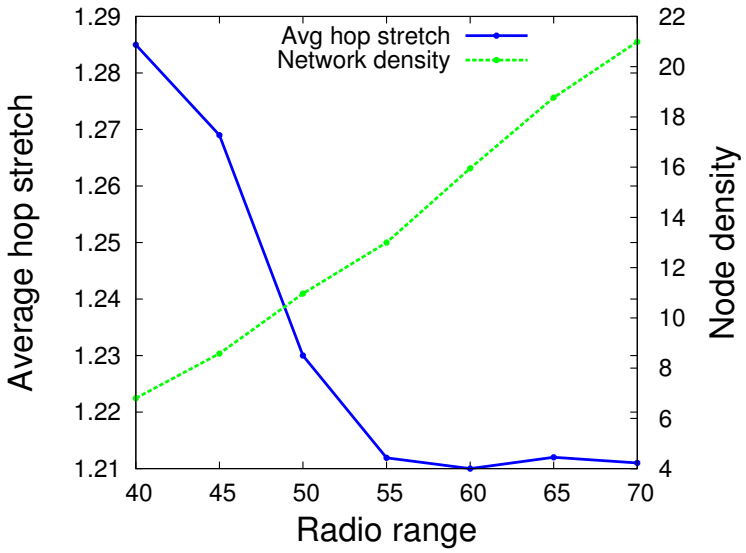


Figure 3.19: Impact of node density.

3.6.4 Impact of Network Density

This section investigates how network density affects our protocol. We varied the radio range for our DOI model from 35m to 60m for Scenario 4. Figure 3.19 shows the average node density for each radio range; it also depicts the path stretches for varying radio ranges (i.e., varying network density). As shown, smaller network density leads to higher path stretches. The reason is that the node density affects the underlying geographic routing in our protocol. More precisely, when a forwarding node has fewer neighbors, it is more likely for the node to choose a neighbor on a suboptimal routing path, thereby increasing the path stretch.

3.6.5 Impact of DOI

As we mentioned in Section 3.6, higher DOI has more irregular communication ranges. This section investigates how irregular communication ranges affect the performance of VIGOR and LVGR. We measured the average path stretches of both

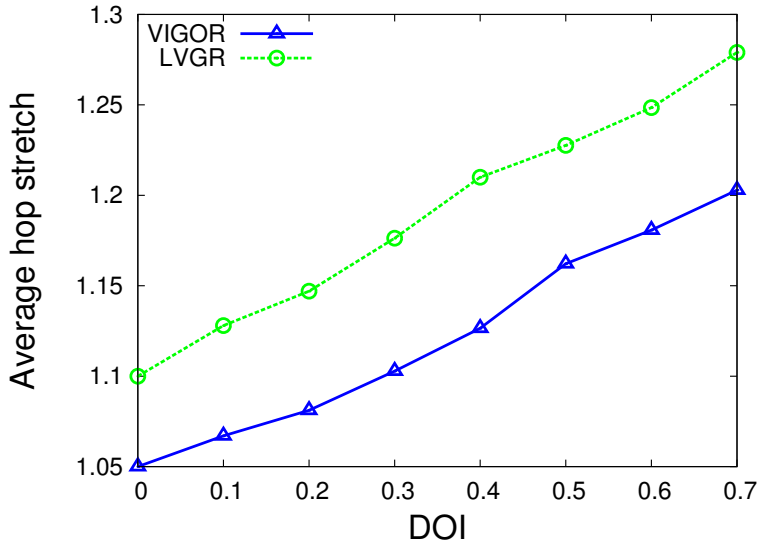


Figure 3.20: Impact of DOI on the average path stretch.

VIGOR and LVGR by varying DOI for Scenario 4. Figure 3.20 shows the results. As shown, if we increase DOI from 0 to 0.6, the path stretches of both VIGOR and LVGR increase by up to 12%. The reason is simple: both VIGOR and LVGR use a geographic routing protocol as its underlying routing protocol; when DOI is high, it is more likely that geographic routing protocols select a suboptimal neighbor, which leads to a suboptimal routing path.

4. CNT-AWARE LOCATION-BASED MULTICAST ROUTING

In this section, our CNT-aware location-based multicast routing protocol called the Robust and Energy-Efficient Multicast Routing protocol (RE^2MR) is presented.

4.1 Motivation

Multicast is an essential component in many Wireless Sensor Network (WSN) applications. Targeted queries, code updates, and mission assignments are well known examples of multicast services. Unfortunately, traditional tree-based [62] and mesh-based [63] multicast protocols, mainly designed for mobile ad hoc networks (MANET), are not suitable to WSNs. Such topology-based protocols require periodic flooding of control messages to maintain the underlying overlay structure up-to-date, thereby causing the early depletion of energy. Additionally, forwarding nodes have to maintain a routing table for each multicast group. Maintaining, possibly large, state information on a sensor node with limited storage capabilities, is an impractical design decision for WSNs, especially in large scale WSN deployments.

Location-based multicast routing protocols, usually referred as Geographic Multicast Routing (GMR), were proposed as a suitable multicast solution for resource constrained WSNs [64][5][65]. In GMR, the locations of all the subscribers in the multicast group are encoded in each multicast packet, so that the routing decision (e.g., whether the path needs to be split or not) is made on the fly, instead of constructing and maintaining the global tree/mesh routing structure. Although GMR addresses the issues encountered by the topology-based multicast protocols, GMR is not a viable solution for large scale WSN deployments in real environments for several reasons. First, in GMR, the packet header size grows significantly as the size of the multicast group increases. Second, GMR incurs significant computational over-

head in forwarding nodes. For example, a forwarding node must compute a heuristic Euclidean Steiner tree [65] or it needs to consider all possible subsets of multicast member nodes [64]–an exponential increase in the computational complexity, with an increase in the multicast group size.

Recently, hierarchical GMR has been proposed as a solution to address the limitations of GMR [4][66][6]. The main idea is to geographically decompose a network into small cells. A *leader* in each cell manages the *subscribers* in that cell. This hierarchical protocol design allows the header size of a multicast packet to be limited. However, the limited packet header size comes at the cost of communication overhead. The control packets are needed for electing a leader in each cell and for managing the local group membership in a cell. Most importantly, the simple network partition into a set of cells results in sub-optimal routing paths from the root node to multicast group member nodes.

To address the aforementioned limitations of the state of art multicast routing solutions, we propose the Robust and Energy-Efficient Multicast Routing (RE^2MR) protocol. RE^2MR is a hybrid multicast protocol that combines the strengths of the topology-based, geographic and hierarchical multicast solutions. RE^2MR, using a solver for the Capacitated Concentrator Location Problem (CCLP), computes the multicast topology that minimizes the sum of path lengths from the multicast root node, to multicast members. To account for realistic WSN deployments, where holes might be present, RE^2MR implements a Trajectory-based Lightweight Hole Detection (TLHD) scheme. TLHD is lightweight since it piggybacks on the regular multicast communication, and efficient since it provides the compact representation of a hole. The information on a hole, discovered by TLHD, coupled with an iterative application of CCLP, enable RE^2MR to refine the multicast topology towards near-optimality. RE^2MR improves its energy efficiency by leveraging the broadcast nature

of the wireless medium and by a careful packet header design. Additionally, the multicast topology that RE^{2}MR produces is ideally suited for implementing reliable multicast packet delivery, through fast and efficient recovery from packet loss.

4.2 Related Work

Conventional multicast protocols can be largely categorized into the tree-based and mesh-based protocols. The tree-based multicast protocols [62][67] build a tree structure, either proactively or reactively, to efficiently deliver a packet to subscribers along this tree. The mesh-based protocols [63][68], to better cope with link failures, build a mesh overlay instead. These topology-based protocols incur overhead for constructing the overlay structures and for maintaining the state information about the overlay structure in each node.

Stateless multicast protocols [64][5][65], based on the locations of nodes, do not require the construction and maintenance of underlying global structures like a tree or a mesh. In these protocols, the locations of subscribers are encoded in a packet. Using this location information, the decision on wether a path needs to be split or not is made on the fly, instead of relying on the global structures. However, these protocols are not scalable, because the packet size grows significantly as the number of subscribers increases. Furthermore, these protocols require high computational overhead in each forwarding node to find the optimal subset of neighbors to forward the packet.

To reduce the packet size overhead and ultimately achieve scalability, several hierarchical geographic multicast protocols have been proposed [4][66][6]. In these protocols, a network is divided into a set of cells. In each cell, a specially designated node is elected for managing the group membership. Instead of sending a packet to all the subscribers, a source sends a packet to the leaders, and the leaders distribute

the packet to its members. This way the packet header size is reduced, because the header contains only the locations of the leaders. However, these protocols incur additional message and computational overhead for electing the leader and for managing the subscribers in a cell. Most importantly, a simple clustering into a group of cells results in sub-optimal path length.

Recent research, mostly related to RE^{2}MR, mitigates the issues faced by different classes of multicast routing protocols, by designing hybrid multicast schemes [69][70]. For example, the hybrid approach of geographic multicasting and topology-based (i.e., tree/mesh-based) multicasting [70] is used to find a good tradeoff between state information storage overhead, communication and computation overhead. Similarly, [69] proposes the hybrid solution of geographic multicast routing and source multicast routing. However, this recent research fails to find the near-optimal multicast routing topology, and does not consider the challenges posed by real deployments (e.g., deployments with obstructions, such as holes).

4.3 System Model and Problem Formulation

We consider a static wireless sensor network consisting of n nodes, denoted by $V = \{v_1, v_2, ..., v_n\}$, uniformly distributed in an area with obstructions, such as holes. There is one sink node denoted by S that collects data from other nodes. A subset of nodes $G \subseteq V$ form a multicast group rooted at S. We assume that each node knows its location and that the locations of multicast members are known to the sink node S. We also assume that existing multicast group management techniques (e.g., *join group*, *leave group*, etc.) are available. These techniques have been studied extensively in the literature. Consequently, in Section 4, we focus exclusively on the energy efficient and robust multicast packet transmissions to the nodes in the multicast group.

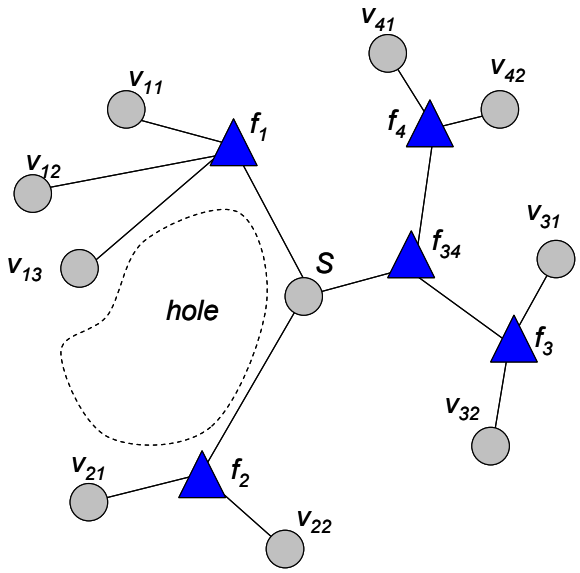


Figure 4.1: An example of facility nodes and multicast members.

The problem we address in Section 4 is two-fold. First, we aim to identify the near-optimal set of routing paths (i.e., near-optimal in the sense of minimizing the sum of path lengths) from the multicast root node to multicast members. Second, given the requirements imposed by large scale, real WSN deployments in complex environments, we aim to limit the packet header size (e.g., make it a system parameter), reduce the state information maintained by forwarding nodes, reduce the computational overhead in forwarding nodes, and achieve reliable packet delivery rate.

Inspired by the *Capacitated Concentrator Location Problem (CCLP)* [71][72], we aim to minimize the total sum of path lengths from the multicast root node node S to each multicast group member, by optimally selecting the locations of *facility* nodes f_i, as depicted in Figure 4.1 (a *facility* node–a term adopted from the operations research–can be thought of as the leader node in the hierarchical multicast routing).

From here on, we will use the term ***source*** node to represent the multicast root

node, the term ***facility*** node to represent nodes aiding in the multicast routing from the root node to multicast group members, and the term ***member*** node to represent multicast group nodes.

For mathematically formulating our problem, we index the possible locations of facility nodes by i, and the possible locations of member nodes by j. We denote the member of a facility node f_i by v_{ij} and the Euclidian distance between two nodes v_i and v_j by $d(v_i, v_j)$. Our problem can then be formulated as a mixed integer program:

$$\text{minimize} \quad \sum_i \sum_j d(f_i, v_{ij}) x_{ij} + \sum_i d(f_i, s) y_i \tag{4.1}$$

$$\text{subject to:} \quad \sum_i x_{ij} = 1 \text{ for all } j, \tag{4.2}$$

$$\sum_j x_{ij} \leq s_i y_i \text{ for all } i, \tag{4.3}$$

$$x_{ij} = \{0, 1\} \text{ for all } i \text{ and } j, \tag{4.4}$$

$$y_i = \{0, 1\} \text{ for all } i \tag{4.5}$$

where x_{ij} and y_i are the indicator variables (i.e., $y_i = 1$ if a facility node is available at location i, and $x_{ij} = 1$ if a member node at location j is connected to a facility node at location i); and s_i is the storage capacity of a facility node at location i, specifying the maximum number of members it can handle. The first constraint guarantees that a member node is connected to only one facility node. The second constraint specifies that no facility node can handle more than its storage capacity. The third and fourth constraints specify the integrality of x_{ij} and y_i.

Several challenges remain to be solved when CCLP is applied to large scale WSNs deployed in realistic environments. First, the *holes* with various shapes in real world

deployments must be efficiently abstracted and taken into account when aiming for the optimal solution. The reason for including the holes in the protocol design is that they change the end-to-end communication cost (i.e., $d(f_i, v_{ij})$ and $d(f_i, S)$ in CCLP). The classical CCLP formulation assumes no *holes* in the target region. Thus, we need to reduce the multicast communication costs by: efficiently and proactively detecting the holes in the network; abstracting the hole information; and by recomputing the optimal solution. Second, CCLP assumes a high-speed and zero-cost communication medium between the source node and each facility; thus, CCLP typically ignores the communication cost for delivering a packet along the path between the source node and a each facility node. In our problem, however, the path from a source node to each facility node consists of multi hop wireless links, having similar characteristics to the links between facility nodes and their members. Consequently, by iteratively solving the CCLP problem (i.e., finding the new facility nodes with the existing facility nodes as new members) one can further optimize the multicast routing paths. Lastly, a reliable multicast packet delivery is typically required in real world WSN deployments. The reliability must be ensured with reduced recovery time and the small number of control packets. The design of RE^{2}MR addresses these problems in the sections that follow.

4.4 Proposed Solution

In this section we provide an overview of RE^{2}MR, followed by the designs of its components.

4.4.1 Main Ideas

A key observation is that the topology obtained by solving the CCLP has the properties that satisfy our goals. In the topology obtained, the sum of Euclidean distances between a source node and members is near-optimal; the packet header size

is limited to the capacity of a facility node (which can be set as a system parameter); the majority of computation happens at the source node, because the source node solves the centralized approximation algorithm for the CCLP and finds the set of facility nodes; facility nodes can be used for implementing reliable multicast packet delivery with small control packet overhead.

In order to obtain the near-optimal multicast routing path, it is important to detect/identify holes and provide the information on them (i.e., the size, shape, and location of the hole) to the source node (which solves the CCLP). Detecting the holes, however, requires high message overhead. Furthermore, the size of the packet must be large for precisely describing the information on the holes. One important observation we make is that only the holes affecting the optimal multicast routing paths must be detected. Based on theis observation, we propose a Trajectory-based Lightweight Hole Detection (TLHD) algorithm as part of our multicast protocol.

More aggressive energy savings can be achieved by finding a new set of facility nodes that serve existing facility nodes. However, to enable this multi-level facility system, a new message passing mechanism must be developed. Our Energy efficient Packet Forwarding (EPF) scheme provides an efficient way to deliver a packet to the facilities in multiple levels with reduced number of packet transmissions, by using the broadcast nature of wireless communications and careful packet header design.

4.4.2 Energy Efficient and Robust Multicast Routing (RE^{2}MR)

The proposed RE^{2}MR protocol, presented in Algorithm 4, consists of four main components: the CCLP solver, the TLHD algorithm, the EPF scheme, and Multi-level Facility computation. As shown, the $CCLP$ solver is used to compute the locations of the facility nodes in the first level (denoted by F_1). Subsequently, a packet is forwarded to each facility $f_i \in F_1$ (Line 2-6). The EPF algorithm is used to

Algorithm 4 RE^{2}MR Protocol

```
 1: Init: k ← 0, F_k ← G, r ← TRUE
 2: while (packets to send) > 0 do
 3:     if r = TRUE then
 4:         F_{k+1} ← CCLP(F_k)
 5:     end if
 6:     Forward a packet to each f_i ∈ F_{k+1} using EPF
 7:     // Hole detection (TLHD) started
 8:     if Feedback received then
 9:         Update hole info.
10:         r ← TRUE
11:         continue (i.e., goto Line 3) // Recompute solution
12:     else
13:         // Multi-level facility
14:         if k + 1 < FACILITY_LEVEL then
15:             k++
16:             r ← TRUE
17:             continue (i.e., goto Line 3)
18:         end if
19:     end if
20:     r ← FALSE
21: end while
```

forward a packet to reduce the total communication costs and to enable the multi-level facility computation. During the packet transmission to each facility node, TLHD is used for detecting any holes that interfere with the path from the root node to the facility node. If a hole is found, a feedback packet is sent immediately to the root node. This feedback packet is used to efficiently estimate the size, shape, and location of the hole. If the feedback packet is received, the root node updates its database of the detected holes and recomputes the solution reflecting the newly discovered holes (Line 8-11). Otherwise, RE^{2}MR checks if the multi-level facility computation is enabled (Line 14). If it is enabled, RE^{2}MR recomputes a new solution F_2, in a similar way that it computed previous facilities in F_1. Otherwise, RE^{2}MR keeps forwarding the next packet to the facilities in the first level F_1 (Line 14-17). This recomputation process is repeated until RE^{2}MR finds the facility set F_k, where k equals FACILITY_LEVEL.

In the following subsections, we present the four main components of RE^{2}MR in

Algorithm 5 Trajectory Based Hole Detection (TLHD)

```
1: if feedback_bit then
2:     if (| ⊥ (v_i, st)| > τ) or (local_minimum) then
3:         Org_x ← x_i Org_y ← y_i
4:         Force_Face_Routing()
5:     end if
6:     if |x_i − x_{i−1}| > I then
7:         find index idx_i corresponding to | ⊥ (v_i, st)|
8:         if x_i > x_{i−1} then
9:             encode idx_i
10:        else
11:            encode −idx_i
12:        end if
13:    end if
14: else
15:    if (| ⊥ (v_i, st)| > τ) or (local_minimum) then
16:        detect_bit ← 1
17:    end if
18:    Forward()
19: end if
```

detail.

4.4.3 Trajectory-based Lightweight Hole Detection (TLHD)

The TLHD algorithm, presented in Algorithm 5, consists of the hole detection phase and hole identification phase.

The hole detection phase is implemented as part of multicast packet transmission; thus, this phase does not require additional packet transmissions. Figure 4.2 illustrates the hole detection phase (A packet, routed around the hole, measures the distance d to the line connecting the source S with facility node f_i. If the distance d is greater than a user defined threshold τ, a hole is detected). As shown, if a forwarding node v_i, finds that it is in local minimum, or that the perpendicular distance to the line $\overline{Sf_i}$ is greater than a given threshold, then node v_i sets the *hole detection bit* in the packet header and forwards the packet (Lines 15-17). When the packet reaches facility node f_i, the facility node f_i checks the hole detection bit. If this bit is set, then the facility node f_i starts to execute the second phase, the hole identification phase.

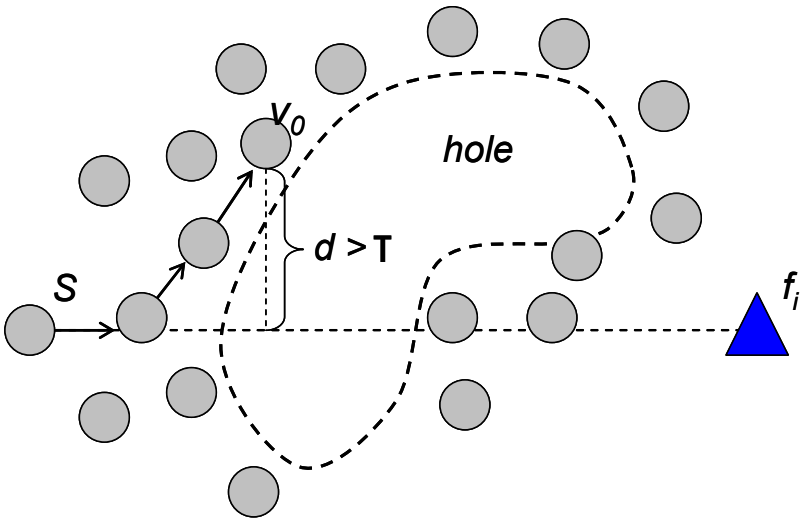


Figure 4.2: An illustration of hole detection.

In the hole identification phase, the information about the detected hole (i.e., the location, size, and shape of the hole) is obtained and concisely represented. Figure 4.3 illustrates the hole identification phase. The facility node f_i that has received a packet with the *hole detection bit* set, starts this phase by sending a feedback packet to the source S. If the feedback packet reaches a forwarding node (v_0 in Figure 4.3) either in local minimum, or having a perpendicular distance to the line $\overline{Sf_i}$ greater than a given threshold (Algorithm 5: Line 2), the forwarding node v_0 encodes its location, (Org_x, Org_y) in the packet and, using face-routing, forwards the packet in the clockwise direction (Algorithm 5: Line 3-4). We call this forwarding node, an *origin node*. Additionally, the origin node sends a copy of the feedback packet in counter-clockwise direction. These two feedback packets will be routed in opposite directions around the hole, collect the information about the hole (described below), and meet at one boundary node of the hole. The collected information is combined into a one feedback packet, and transmitted back to the source node.

More specifically, while the nodes along the boundary of the hole route the packet,

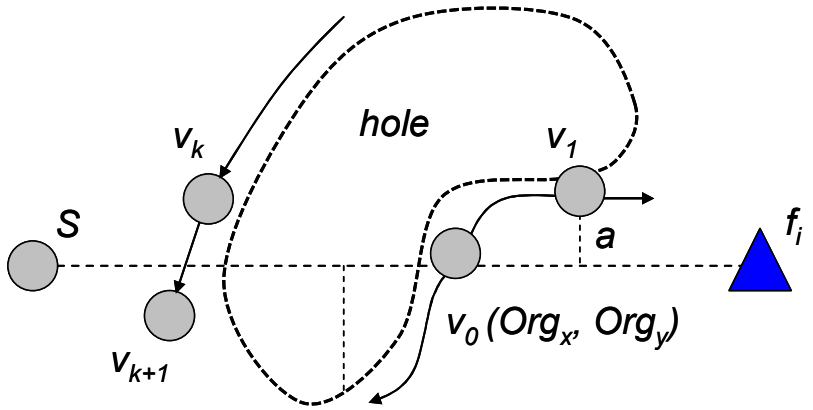


Figure 4.3: An illustration of hole identification.

they measure the distance between them and the line $\overline{Sf_i}$. For example, consider Figure 4.3, where a packet from node v_0 reached the next boundary node along the face, node v_1 with location (x_1, y_1). Node v_1 computes the projected distance to previous node v_0 (i.e., $|x_i - x_0|$) and checks if this projected distance is greater than I. If it is greater, node v_1 calculates the perpendicular distance to line $\overline{Sf_i}$, represented by a in Figure 4.3. The representation of this distance is further abstracted as a simple index in a table, in which each entry of the table represents a range of distances. (Algorithm 5: Line 6-7). The matching index is then encoded in the feedback packet. In order to differentiate the packet forwarding directions (i.e, either towards the origin node or not), we use a negative representation of the index in the packet when the packet travels towards the source node (Algorithm 5: Line 8-12). The feedback packet is kept forwarded to the next boundary node along the hole, until it crosses line $\overline{Sf_i}$. For example in Figure 4.3, when the packet is forwarded from node v_k to node v_{k+1}, the line $\overline{v_k v_{k+1}}$ crosses the line $\overline{f_i S}$. And then, node v_{k+1} stops forwarding the feedback packet and waits for the feedback packet coming from the opposite direction. If that packet arrives, node v_{k+1} combines the collected data

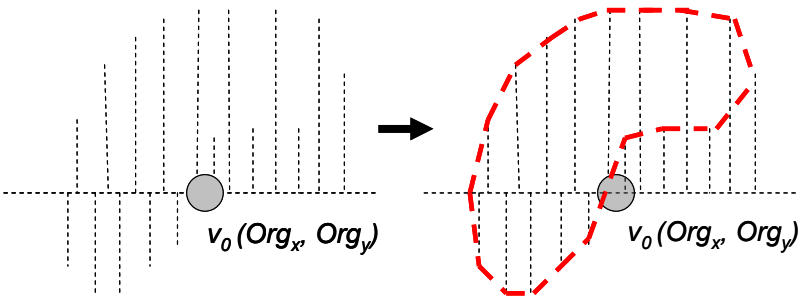


Figure 4.4: An illustration of hole reconstruction.

and sends the packet to the source S.

A special case occurs when a facility node is located in a hole. The TLHD algorithm efficiently handles this situation. Assume that a packet is sent from the source S to a facility node f_i, and that this facility node is inside a hole. This packet would traverse along the hole, cross the line defined by S and f_i at a point p (for which $d(S, f_i) < d(S, p)$), and then be received by a node, say v'. Finally, node v' sets its feedback_bit, becomes an origin node, and initiates the hole identification phase to find a new facility node that is not inside a hole.

Upon receiving the feedback packet, the source node S uses the information on the detected holes for recomputing the optimal path to each member. Source node S firsts reconstructs a hole by using the data in the feedback packet. Figure 4.4 illustrates this process. The hole is represented as an origin (org_x, org_y) and a set of perpendicular distances to the line $\overline{f_i s}$ as shown in Figure 4.4. Consequently, source node S is able to represent the hole as a polygon by sequentially connecting all the end points of the perpendicular lines starting from the origin. This polygon representation of a hole is used to recompute the shortest path between a facility node and its members and between the facility node and the source node S. To compute such shortest path, RE^{2}MR exploits the *Visibility Graph*, a well known

mechanism to compute the shortest path in the presence of polygonal obstacles [73].

4.4.4 Energy-efficient Packet Forwarding (EPF) and Multi-Level Facility Computation

There are two types of nodes in our protocol, namely facility nodes and non-facility nodes. In order to save energy by reducing the number of packets transmitted, RE^{2}MR uses different packet forwarding schemes for different node types. A facility node forwards a data packet to multiple destinations, either to its members, or to the facility nodes in the lower level. A naive forwarding scheme for a facility node is to use multiple unicasts to each destination. However, this method not only incurs high energy consumption but also causes unbalanced energy distribution. To solve this problem, our EPF scheme exploits the broadcast nature of wireless transmission and a careful packet header design. Specifically, a facility node first computes the best neighbor for each destination (the best neighbor refers to the closest node to a given destination). The facility node then puts the locations of the destinations in the header and sequentially inserts the corresponding node id of the best neighbor for each destination. The first bit of the header is set to 1 so that a receiver treats this packet differently from a simple forwarding.

Upon receiving a packet, a node checks the first bit of the header. If this bit is set, the node checks if its node id matches any node ids in the header. In case of no-match, it just forwards this packet. If its node id matches the i-th node id in the header, it sets the destination location as the i-th location in the header. Figure 4.5 illustrates a packet forwarding scenario and the packet header. In this figure, a facility node has four member nodes with the locations A, B, C, and D respectively. The facility node first computes the best neighbors: b for C, c for D, and a for both A and B. It then sets the first bit of header to 1, puts the locations

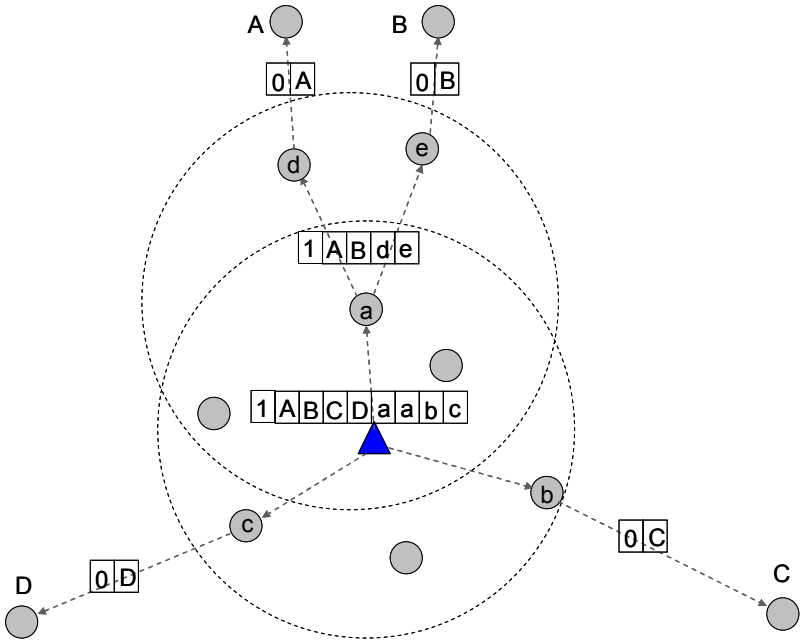


Figure 4.5: An illustration of packet forwarding by a facility node.

of the destinations, and inserts corresponding ids of best neighbors. Consequently, we have the header $1, A, B, C, D, a, a, b, c$. The facility node broadcasts this packet, so that all of its neighbors receive this packet. Upon receiving this packet, a node b finds that it has to forward this packet to C by looking at the header. Similarly, c forwards the packet to D. However, a node a finds that it has two destinations, A and B. Node a then applies the same logic to split the packet.

One other type of a node in our protocol is the non-facility node. Unless the first bit of a packet header is 1, a non-facility node simply forwards the packet to a destination using simple geographic forwarding, or face-routing for escaping from the local minimum.

As shown in Lines 14-17 of Algorithm 4, EPF also provides an efficient message passing mechanism for multi-level facility system. We illustrate the concept in Figure 4.6 where f_i^1 represents a facility node in the first level, f_i^2 means a facility

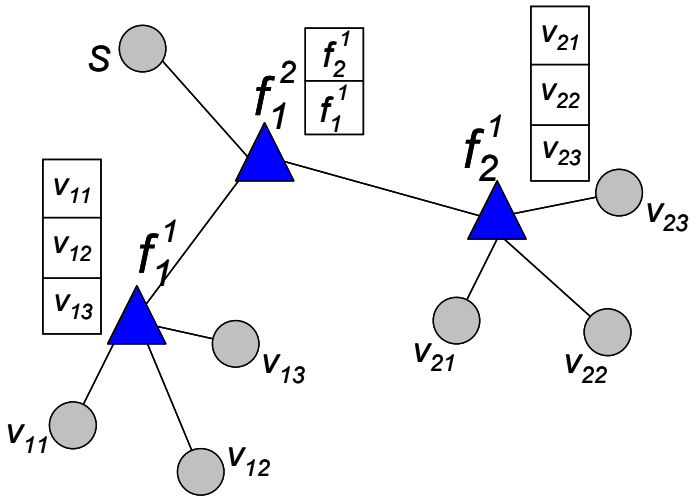


Figure 4.6: An illustration of multi-level facility nodes.

node in the second level, and so on. Recall that in the single level facility system, a source node S sends a multicast packet only to the facility nodes in the first level. In the multi level facility system, a source node S first sends a multicast packet to the facility nodes f_i^1 in the first level; then each facility node f_i^1 saves the locations of its members (which are encoded in the header of the received packet). A source node S then sends the second multicast packet to the facility nodes f_i^2 in the second level. Similarly, each second-level facility node f_i^2 saves the locations of its members (in this case, the members are the facility nodes in the first level), and forwards the packet to its members. Upon receiving the second multicast packet from the facility node in the second level, the facility node in the first level forwards the packet to its members, using locally stored locations. This process is repeated for higher facility levels. Consequently, a source node sends a packet to the facility nodes at the highest level without encoding the locations in the packet header.

Note that adding more facility levels permits higher energy savings. However, in order to use higher facility levels, more nodes need to maintain state information (i.e.,

Algorithm 6 RPD: Code for facility node f

```
1: Init: B ← ∅
2: On received packet p_i:
3: B ← B ∪ {p_i}
4: if p_i in sequence then
5:     Forward to members
6:     Start timer T_i
7: else
8:     Send NACK_j to the root (j:the index of missed packet)
9: end if
10: On timer T_i fired:
11: if NACK not received then
12:     B ← B \ {p_i}
13: else
14:     Retransmit p_i
15:     Reset timer T_i
16: end if
```

the locations of members). RE^{2}MR allows the user of a WSN to make the tradeoff decisions between more aggressive energy savings and smaller state information.

4.4.5 Reliable Packet Delivery

To improve the packet delivery ratio, we employ a slightly modified version of NACK based retransmission scheme, which is particularly useful for RE^{2}MR. Specifically, the recovery time from a packet loss is reduced, since the retransmission request is directly sent to the facility node, not to the source node. Additionally, since the facility nodes handle packet retransmission requests, we eliminate the bottleneck that the source node would have been, had it handled all retransmission requests.

Algorithm 6 presents the Reliable Packet Delivery (RPD) scheme implemented by facility nodes. As shown, upon receiving a packet p_i from a source node (or from a facility node in one level higher than itself, when multi-level facility are used), a facility node f, checks if the packet p_i is in sequence. If it is in sequence, f simply forwards p_i to its members and sets a timer T_i, specifying the waiting time for NACK packets from its members (Line 4-5). If the facility node identifies a lost packet p_j, then it sends NACK$_j$ to the root node (Line 7-9). When the timer T_i fires, f examines

if any NACK packets have been received from its members. If no NACK packets were received, f deletes the packet p_i from buffer $\mathcal{B}$ (Line 11-12). If a NACK$_i$ was received, f retransmits the packet p_i and resets the timer T_i (Line 13-15).

4.5 Theoretical Analysis

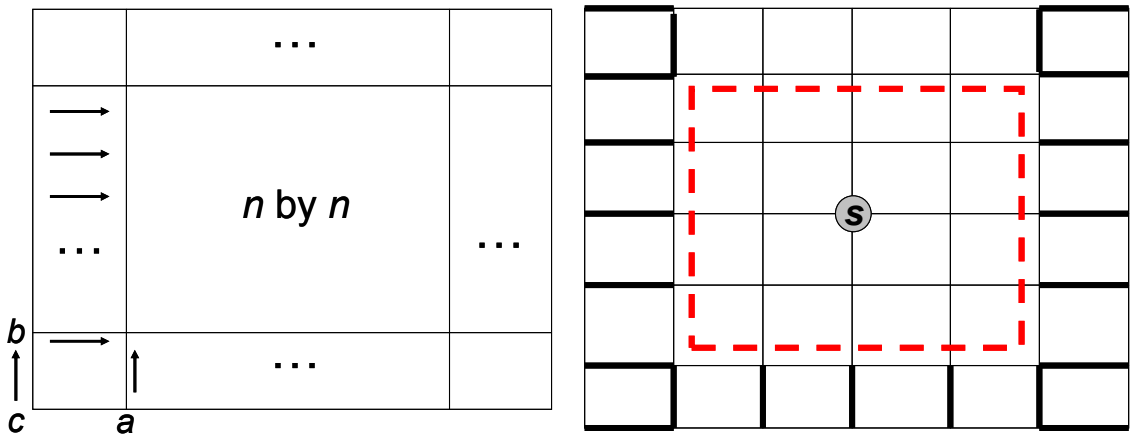
In this section, we aim to theoretically investigate our proposed solution before we perform extensive testbed experiments and TOSSIM simulations. The theoretical results would provide insights into the experimental and simulation results. We present analysis of RE^{2}MR and two state of art multicast routing protocols, namely RSGM [6] and MRBIN [70]. For our analysis, we consider a square-shaped two dimensional network with a grid topology of varying size, $n \times n$, where n is an even positive number. The internode distance is a unit distance. Member nodes are located along the four edges of the network (i.e., $4n$ members are positioned in the network of size $n \times n$).

4.5.1 Total Number of Branch Nodes

The total number of branch nodes, denoted by N_b, is an indicator for the amount of state information maintained in a network. The following results show the total number of branches when there are N members.

Lemma 4 *For MRBIN, maximum* $N_b = N - 6$.

Proof: We prove this by induction on the length of side n of our network.
Base step ($n = 2$, i.e., 2×2 network): it is trivial to see that $N_b = 2$ for a 2×2 network, where the number of members is 8.
Inductive Step: assume that for $n = k$, $N_b = 4k - 6$, and consider Figure 4.7(a). Paths from two members a and b that are adjacent to member c at the corner of the $(n + 1) \times (n + 1)$ network meet at the corner of $n \times n$ network, forming a branch.



(a) (b)

Figure 4.7: An illustration of inductive step for: a) Lemma 1 and b) Lemma 3.

There are four such branches at each corner of $n \times n$ network. A path from each member at the corner of $(n+1) \times (n+1)$ network meets with the path from one of its adjacent members, making four more branches. Except for these members (i.e., the members at the corner and their adjacent members in $(n+1) \times (n+1)$ network), all the other members send packets to the adjacent member that is located on the edge of $n \times n$ network as shown in Figure 4.7(a). Thus, the problem is reduced to counting N_b for $n \times n$ network. Therefore, the total number of branches for the $(n+1) \times (n+1)$ network is given by $N_b = (4k-6)+8 = 8(k+1)-6$. ■

Lemma 5 *For RSGM, maximum* $N_b = \frac{N}{2} - 4$.

Proof: In RSGM, a branch node is a leader in a cell. Thus, we need to consider the total number of cells that have members in them. If the cell size is C_s, in a $n \times n$ network, the total number of cells is $\frac{n^2}{C_s}$. Next, we need to consider the cells that contain members (i.e., the cells that are adjacent to the four sides of the network). The number of cells that do not contain members can be computed as $\frac{(n-2\sqrt{C_s})^2}{C_s}$. Thus, the total number of branches is $N_b = \frac{n^2}{C_s} - \frac{(n-2\sqrt{C_s})^2}{C_s}$. Next, by replacing n by

$\frac{N}{4}$ and solving the equation for N_b, we obtain $N_b = \frac{N}{\sqrt{C_s}} - 4$, where $4 \leq C_s < \frac{N^2}{16}$. Therefore, the maximum $N_b = \frac{N}{2} - 4$. ■

Theorem 2 *RE²MR has the lowest N_b, when compared with RSGM and MRBIN.*

Proof: In RE²MR, facility nodes are the only branch nodes. Thus, we count the maximum number of facility nodes. Let F_c be the facility capacity. Since members are uniformly located along the four sides of the network, $\lceil \frac{N}{F_c} \rceil$ facilities are selected such that members around the corners of the network are first covered, thereby $\lceil \frac{N}{F_c} \rceil \geq 4$. The maximum number of facilities is obviously N when the facility capacity $F_c = 1$. However, for fair comparison, we must determine the value for F_c. Note that for RSGM, L_b is minimum when C_s=4, which yields that the capacity of a leader, specifically the leader of a cell located at the corner of network, is at most 5. For MRBIN, the capacity is at most 4 due to the grid topology. Thus, we choose 5 for F_c and get $N_b = \lceil \frac{N}{5} \rceil$. By Lemma 1 and Lemma 2, MRBIN $(N - 5)$ > RSGM $(\frac{N}{2} - 4)$ > RE²MR $(\lceil \frac{N}{5} \rceil)$. ■

4.5.2 Sum of Path Lengths

The sum of path lengths is indicative of how much energy is consumed for each multicast packet transmission. The following analysis estimates the sum of path lengths as a function of N members.

Lemma 6 *The sum of path lengths for MRBIN is $\frac{N^2}{16} + \frac{N}{2}$*

Proof: We prove this by induction on the length of side n of n by n network.

Basis step ($n = 2$): the path length is trivially 8.

Inductive step: assume that the claim holds for $n = k$. Then, the path length of $k \times k$ network is $k^2+2k = k(k+2)$, since $N = 4k$. Now we consider the case with $n = k+2$,

since n is an even positive number. Figure 4.7(b) depicts this case. Note that each member can reach the inner square, $k \times k$ network, in one hop. In particular, the members at the four corners reach the inner square by forming a branch with one of its two neighboring members. Thus, we get $4(k+2)$ additional path lengths, which yields that the total path length for $n = k+2$ is $k(k+2)+4(k+2) = (k+2)(k+4)$. ■

Lemma 7 *The sum of path lengths for RSGM is* $\frac{3}{2}(\frac{N^2}{16} - \frac{N}{2} + 8)$.

Proof: In order to compute the sum of path lengths, we consider a Cartesian coordinate system in which the member at the left bottom corner of the network is the origin. We first compute the path lengths from the source node S to each leader. The longest path is the one that connects the source node with the leader of the cell located at the corner of the network. The coordinates of this leader are $(\frac{\sqrt{C_s}}{2}, \frac{\sqrt{C_s}}{2})$, and the coordinates of S are $(\frac{N}{8}, \frac{N}{8})$. The Euclidean distance between them is $\sqrt{2}(\frac{N}{8} - \frac{\sqrt{C_s}}{2})$, and thus the actual path length is $2(\frac{N}{8} - \frac{\sqrt{C_s}}{2})$. The shortest path is of length $\frac{N}{8} - \frac{\sqrt{C_s}}{2}$. Thus, the median path length is $\frac{3}{2}(\frac{N}{8} - \frac{\sqrt{C_s}}{2})$. When taking into account the total number of leaders, the sum of lengths for the paths connecting a source to leaders is $\frac{3}{2}(\frac{N}{8} - \frac{\sqrt{C_s}}{2})(\frac{N}{\sqrt{C_s}} - 4)$. Now we compute the path lengths from a leader to members. The longest path connects a leader to its members located at the left bottom corner of its cell. This member has coordinates $(0, 0)$. The Euclidean distance between them is $\frac{\sqrt{2C_s}}{2}$, and thus the actual distance is $\sqrt{C_s}$. The shortest length of such path is $\frac{\sqrt{C_s}}{2}$. Thus, the median length is $\frac{C_s+\sqrt{C_s}}{4}$. Since the total number of members is N, the sum of lengths of such paths is $(\frac{C_s+\sqrt{C_s}}{4})N$. Therefore, after combining the results for the two cases above, the total sum of path lengths becomes $\frac{3}{2}(\frac{N}{8} - \frac{\sqrt{C_s}}{2})(\frac{N}{\sqrt{C_s}} - 4) + (\frac{C_s+\sqrt{C_s}}{4})N$. When considering C_s=4, for maintaining a minimum number of branch nodes for MRBIN, we obtain $\frac{3}{2}(\frac{N^2}{16} - \frac{N}{2} + 8)$. ■

Theorem 3 *RE²MR has the shortest sum of path lengths, when compared with RSGM and MRBIN.*

Proof: The maximum distance between a source node S and the facility node is $2(\frac{N}{8}-2)$, when the facility node is located at the point closest to the member positioned at the corner of network. By Lemma 2, we choose F_c=6. Thus, the sum of path lengths connecting S and facilities is at most $2(\frac{N}{8}-2)\frac{N}{6}$. The maximum path length between the facility node and its member is thus $\sqrt{10}$. Since there are N members, the sum of path lengths connecting a facility node to each member is at most $\sqrt{10}N$. Thus the total sum of path lengths is at most $2(\frac{N}{8}-2)\frac{N}{6}+\sqrt{10}N$.

■

4.6 Performance Evaluation

We implemented RE²MR in nesC for the TinyOS operating system. The protocol (implemented in 4,916 lines of code) occupies 7,289B of RAM, and 25,466B of program memory. We adopted GPSR [10] for the underlying geographic routing protocol. We compared the performance of RE²MR with two recent, state of art multicast protocols for WSN: one hierarchical geographic multicast–RSGM [6], and one hybrid multicast–MRBIN [70]. Due to the relatively large scale network deployment we need for our evaluation, we performed simulations using TOSSIM. For our simulations, we deployed 400 nodes in a 20×20 grid, with an inter-node distance of 20m. The radio range of a node was between 30m and 70m. The metrics we used for our performance evaluation are: the total sum of path lengths (PL), the total number of packets transmitted (PC), the average end-to-end delay ($E2E$) and the packet delivery ratio (PDR). We vary the following parameters: node density (ND), facility level (FL), and hole size (HS). Each experimental point represents the mean of five runs.

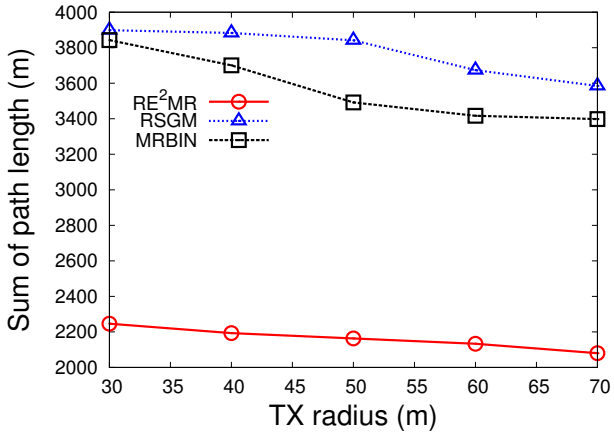


Figure 4.8: Impact of node density on sum of path lengths.

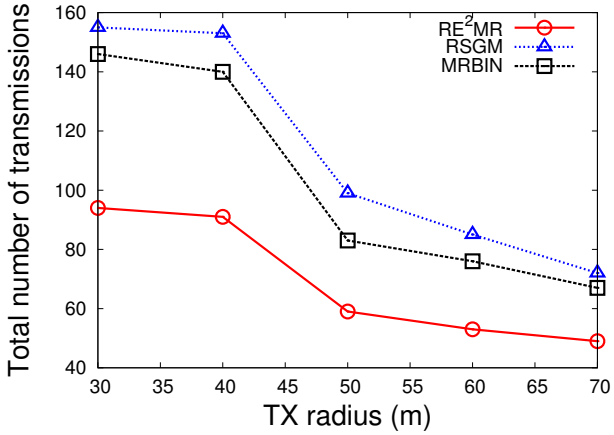


Figure 4.9: Impact of node density on total number of packets.

4.6.1 Impact of Node Density

We expect that node density (ND) affects the performance of RE^{2}MR, RSGM and MRBIN, because these protocols are based on geographic routing that is known to be sensitive to node density. We measured PL, PC and $E2E$ by varying ND from 30 to 70. For this experiment, we fixed FL=1, FC=3, and NM=4%. Figure 4.8 depicts the results for PL. One can observe that RE^{2}MR yields shorter path lengths,

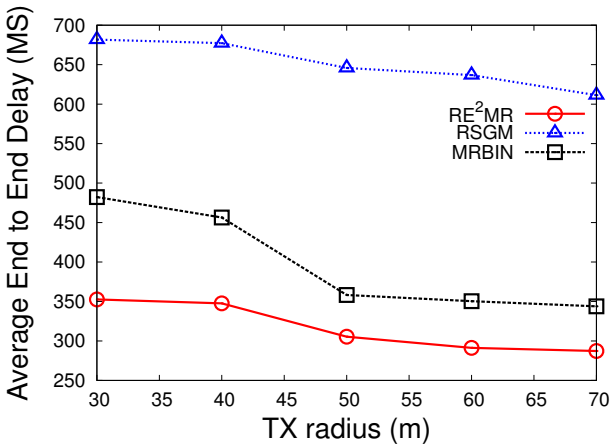
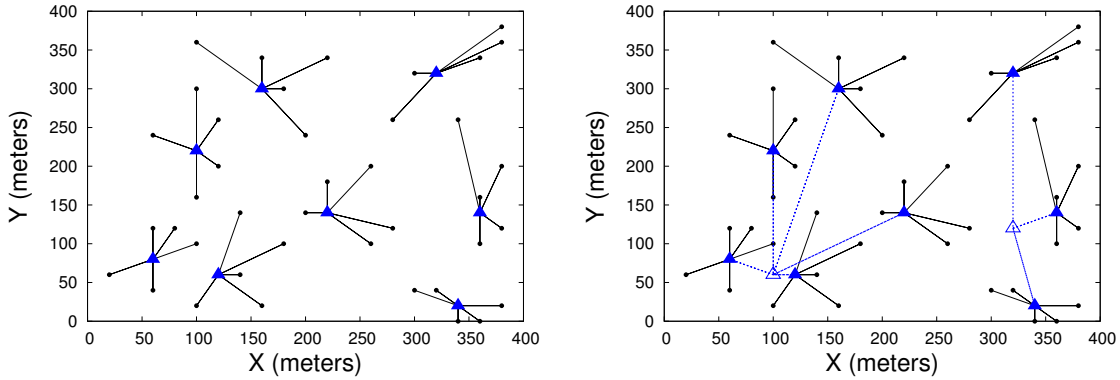


Figure 4.10: Impact of node density on average end-to-end delay.

by as much as 57%, when compared with both RSGM and MRBIN. As shown, when ND increases, PL for all the protocols decreases slightly. The explanation for this is that at higher node densities, geographic routing protocols are able to identify routing paths more closer to the Euclidian distance between a source and a destination, and, hence, shorter. Figure 4.9 shows PC as a function of ND. As expected, for larger communication ranges, the total number of packets exchanged decreases. One can observe that the PC for RE^{2}MR is the lowest for all the ND values. Figure 4.10 depicts the end-to-end delay $E2E$ for the three protocols as a function of ND. The results indicate that protocols with lower PL exhibit a lower end-to-end delay. As shown, RE^{2}MR has an average end-to-end delay shorter by up to 8% when compared with MRBIN, and by up to 50% when compared with RSGM.

4.6.2 Impact of Level of Facilities

In this subsection, we investigate how the facility level (FL) affects the performance of our protocol. Specifically, we measured PL, PC and $E2E$ by varying FL in different ND settings. For this experiment, we fixed NM=6% and FC=3 and



(a) (b)

Figure 4.11: An example of RE^{2}MR topologies for (a) a single level facility and (b) two level facilities.

did not consider the holes. Example topologies for RE^{2}MR with single level and two level facilities are depicted in Figure 4.11.

Figure 4.12 shows PL for different FL values. As the figure shows, PL slightly decreases as ND increases, for all FL values. As expected, higher FL results in shorter PL. The reason is that the path lengths from a source node to facility nodes are reduced when higher level facility nodes are used. Figure 4.13 depicts PC as a function of ND for different FL. Note that PC becomes smaller for longer communication ranges. We also observe that PC for higher FL was lower than for lower FL. The reason is that higher FL essentially aggregates more paths. However, as Figure 4.14 depicts, $E2E$ for higher FL is actually higher than for lower FL. Although the total sum of path lengths is reduced by aggregating more existing paths using facility nodes in higher levels, the direct path to a facility node in lower level was no longer used, causing higher end-to-end delay for higher facility level.

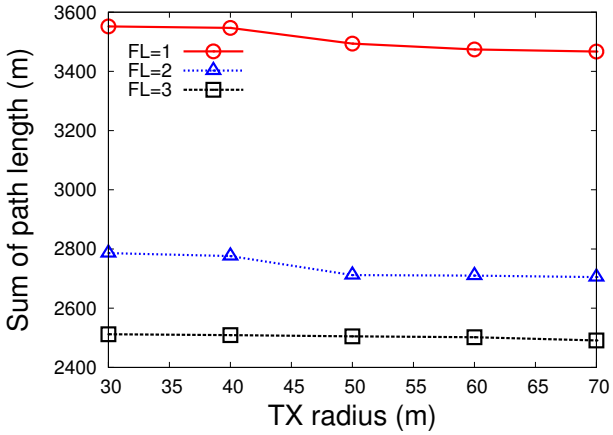


Figure 4.12: Impact of facility level on sum of path lengths.

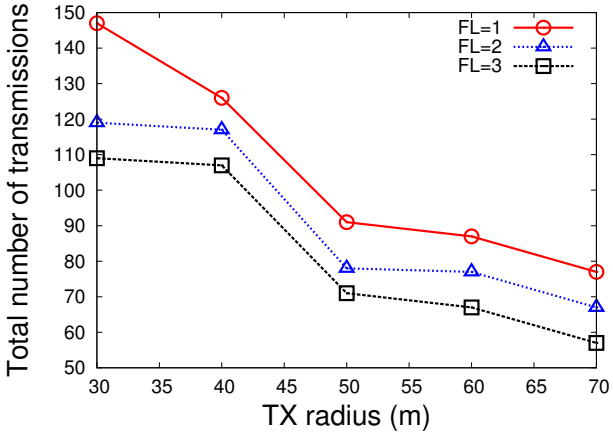


Figure 4.13: Impact of facility level on total number of communications.

4.6.3 Impact of Holes

To investigate the impact of a hole, we created a hole in the middle of our network. Specifically, the hole is a square shape with the length of a side varying from 80m to 240m. In this experiment we fixed ND=60, FC=3, FL=1 and we uniformly deployed 12 members along the upper and right sides of our network, allowing space for the hole. We measured PL and $E2E$ by varying the size of the hole.

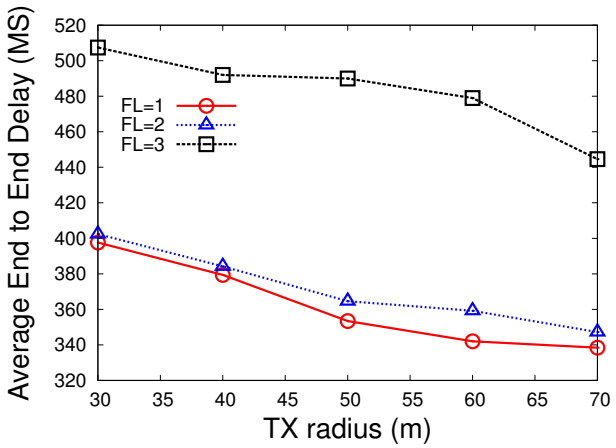


Figure 4.14: Impact of facility level on average end-to-end delay.

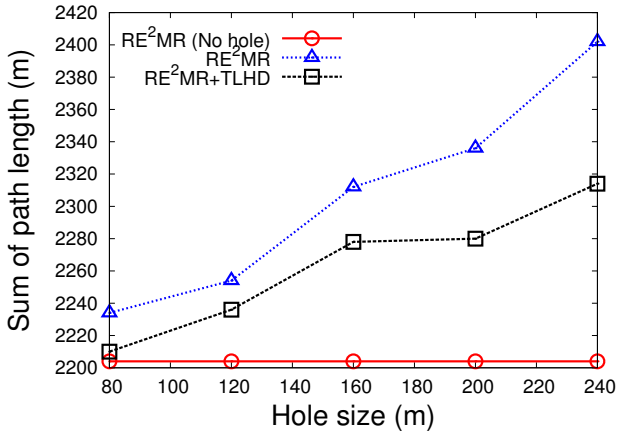


Figure 4.15: Impact of hole size on sum of path lengths.

Figures 4.15 and 4.16 depict the results. When a hole is present, the performance in terms of PL and $E2E$ degrades, when compared with the scenario when a hole is not present. As we increase the size of the hole, the performance degradation increases. The reason for this is that a hole affects the routing cost between a facility node to the source node, and to members, by making a packet travel along the face of the hole. Evaluation results demonstrate how our proposed TLHD algorithm

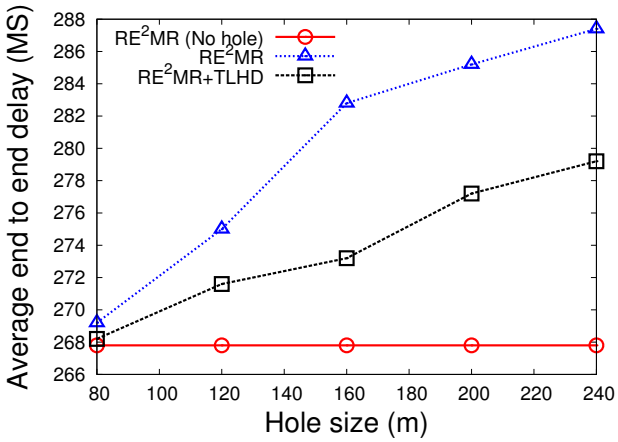


Figure 4.16: Impact of hole size on average end-to-end delay.

improves the performance by relocating the facilities. The TLHD algorithm allows the source node to recalculate the locations of the facility nodes. As we increase the size of the hole, PL and $E2E$ for RE^{2}MR with TLHD also increase. The impact of the hole, however, is mitigated by the new set of facilities.

4.6.4 Reliability

In this section, we investigate the reliability of RE^{2}MR by measuring the packet delivery ratio (PDR). We compared the PDR of RE^{2}MR to that of RSGM and MRBIN for different ND and NM settings. We fixed NM=6%, FC=3, FL=1 and measured PDR by varying ND from 30m to 70m. The source node sent 100 packets, at a rate of 2 packets per second. Each member node computed its own PDR. The reported PDR was then calculated as the average PDR of all members.

Figure 4.17 depicts our results. As shown, as we increase ND, the PDR values of all three protocols increase. The reason is that an increased transmission range reduces the number of packet transmissions, thus decreasing the probability of packet collisions. Additionally, a higher node density increases the probability of packet

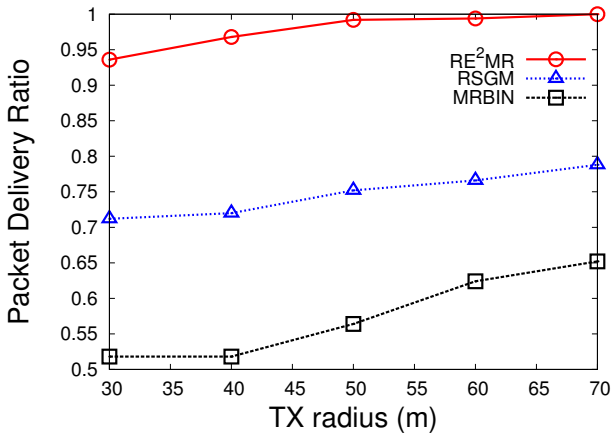


Figure 4.17: Reliability measurements as a function ND.

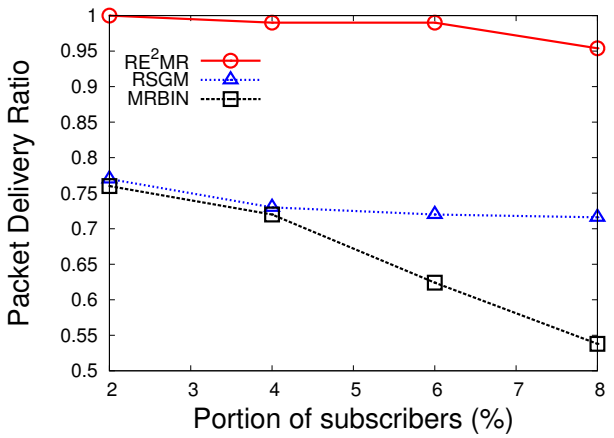


Figure 4.18: Reliability measurements as a function of NM.

delivery. Note that RE^{2}MR's PDR approaches almost 100%, greatly outperforming MRBIN and RSGM. The low PDR of state of art multicast protocols suggests that a packet recovery mechanism must be employed for reliability. An additional observation is that MRBIN's PDR is worse than that of RSGM. The number of branch nodes in RSGM depends on the size of the cell. In our setting, RSGM has more branch nodes, when compared with MRBIN; thus MRBIN shows relatively better

performance in terms of PL, PC and $E2E$. However, the smaller number of branch nodes means the higher chance of packet loss, because a single packet carries data to more members. This also explains why MRBIN's PDR rate increase is higher than that of RSGM.

Next, we fixed ND=60, FC=3 and FL to 1, and measured PDR for RE^{2}MR, RSGM, and MRBIN by varying NM from 2% to 8%. Figure 4.18 shows the result. As we increase NM, PDR for all protocols decreases. A simple explanation for this is that higher traffic increases the chance of collisions and interference. Similar to the result for different ND setting, RE^{2}MR shows the best performance, when compared with RSGM and MRBIN. Similar to the result shown in Figure 4.17, MRBIN's decrease rate in PDR for increasing NM was higher than that of RSGM, because MRBIN has the smaller number of branch nodes.

5. CNT-AWARE CONVERGECAST AND UNIFIED ROUTING FRAMEWORK

In this section, we present the design, implementation details, and experimental results for our unified location-based routing framework called CoLoR that integrates our CNT-aware location-based unicast, multicast, and convergecast routing protocols. In particular, the details of our CNT-aware convergecast routing protocol is also presented.

5.1 CNT-Aware Unified Location-based Routing (CoLoR): Overview

5.1.1 Software Architecture

We start by presenting an overview of CoLoR. We first give a brief description of the software architecture depicted in Figure 5.1. We then explain how various pieces of the architecture work together by using an example of a typical deployment scenario.

The software architecture for CoLoR consists of three main components: CNT Support, Routing Engine, and Packet Forwarder, as denoted by dotted (green) boxes in Figure 5.1. The CNT Support component has three modules: Boundary Detection, Boundary Abstraction, and Cut Detection modules. As shown in Figure 5.1, the Boundary Detection module uses the Boundary Abstraction module to abstract detected boundaries into polygons. This abstract information is used by the modules in Routing Engine as well as the Cut Detection module, thereby significantly simplifying the implementation of Cut Detection. The boundary detection/abstraction process accesses information on neighboring nodes through the Neighbor module. When the CNT detection/abstraction process is completed, the abstract CNT information is broadcast throughout the network using the Bcast module.

The Routing Engine component is responsible for determining a routing path for

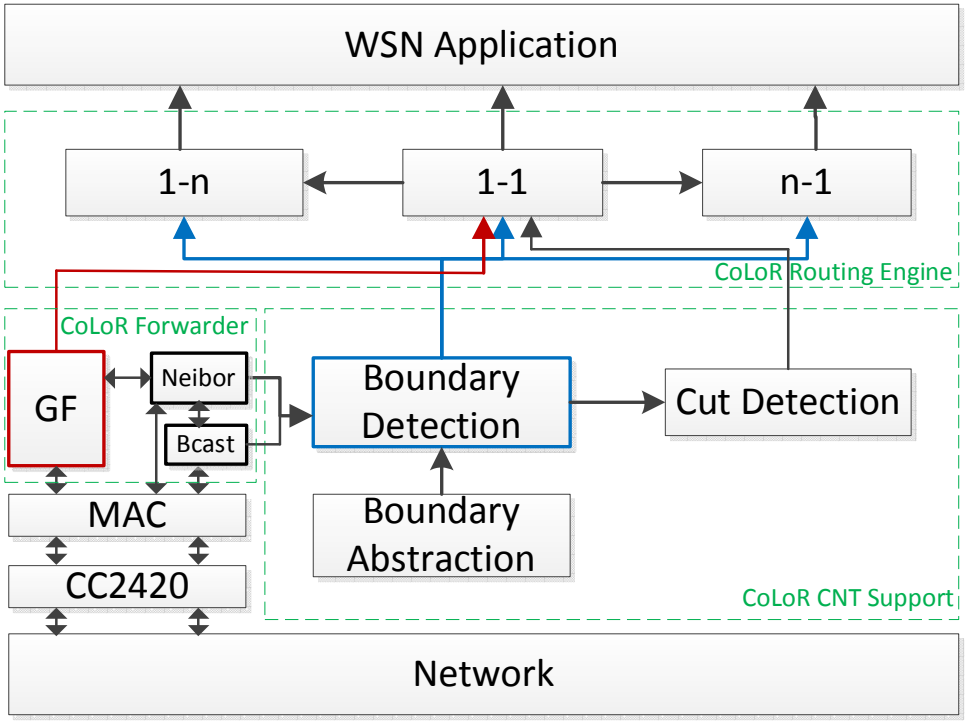


Figure 5.1: Software architecture of CoLoR routing framework

Table 5.1: Sizes of the three main components.

	Forwarder (GF+Nei+Bcast)	**CNT Support +Nei+Bcast**	**Routing Engine +CNT Support+GF+Nei+Bcast**
ROM	26,120 Bytes	29,112 Bytes	46,594 Bytes
RAM	1,863 Bytes	4,490 Bytes	8,654 Bytes

forwarding a packet. The Routing Engine integrates routing protocols for three major routing paradigms: 1-1 (unicast), 1-n (multicast), and n-1 routing (convergecast). As shown in Figure 5.1, all three modules rely on the abstract CNT information provided by the CNT Support component. In particular, 1-1 Routing module also uses the reachability information received from the Cut Detection module. It is important to note that both 1-n and n-1 Routing modules reuse the functionalities of 1-1

Routing module, thereby reducing the memory size for implementation. The Packet Forwarder component includes a Neighbor Discovery module, Geographic Forwarding (GF) module, and broadcasting (Bcast) module which directly interact with an underlying MAC protocol. Table 5.1 shows the RAM/ROM sizes of the components for the CoLoR architecture, when implemented on a TelosB mote running TinyOS 2.1.2. As shown, the entire framework fits in a TelosB mote having very limited memory space with the ROM size of 48 KBytes and RAM size of 10 KBytes.

5.1.2 Use Case Scenario

Having presented an overview of the CoLoR software architecture, we now consider a simple deployment scenario to explain how various pieces of the routing framework work together. When sensor nodes are first deployed, nodes run the Neighbor Discovery module and select their neighbors with reliable links. The sink node also performs the neighbor discovery and joins the network. Based on discovered neighbors, nodes detect CNTs in the network. As the CNT Detection module identifies boundary nodes, the CNT Abstraction module abstracts the boundary information into polygons. More specifically, the CNT Abstraction module finds the *vertex nodes* for each boundary, i.e., the nodes corresponding to the vertices of a polygon enclosing the boundary. Once the vertex nodes are identified, their locations are broadcast throughout the network. Note that we refrain from broadcasting the locations of all boundary nodes to prevent the broadcast-storm problem and also to reduce the overhead for nodes to store the locations of boundary nodes. Consequently, when the broadcast is finished, all nodes in the network have the abstract information about holes in the form of polygons. Using the abstract boundary information, the Cut Detection module can find whether a given destination is reachable or not by using a point-in-polygon algorithm. It should be noted that if the sink node knows all

the locations of deployed nodes, the sink node can compute the boundary of CNTs in a centralized manner, thus not having to install the CNT Detection/Abstraction modules. However, users who are interested in autonomous operations of CNT detection/abstraction to cope with unexpectedly arising CNTs, e.g., destroyed sensors due to environmental factors or hostile users, may install the CNT Detection/Abstraction modules.

Now assume that a source node, say s, sends a packet to a destination node denoted by t. When source and destination nodes are outside the convex hulls of holes in the network, we use the convex hulls of holes to determine a path with guaranteed stretch. More specifically, given the locations of source and destination nodes, the 1-1 Routing module computes a set of intermediate destinations $I_1, I_2, ...$ for node s; node s then sends the packet first to I_1; upon the packet reaching I_1, node I_1 sends the packet to the next intermediate destination I_2, and so on, until the packet reaches the destination t. This way the 1-1 Routing module guides the packet along a path with guaranteed stretch. We will discuss the details of the 1-1 Routing module in Section 5.4.1. When either source or destination (or both) is inside the convex hull of a hole, a network infrastructure, called *local visibility graph*, is used for guiding a packet optimally (i.e., in terms of path length) to the destination node. We defer the details on the local visibility graph until Section 5.4.1.

When source node s wants to send a packet to a multicast group, say M, node s uses the 1-n Routing module. Due to the limited memory space of a sensor mote, and assuming that the sink has abundant resources, we choose a design in which the sink node manages the multicast group – nodes join/leave a multicast group by sending a short control packet to the sink node. So in our 1-n routing design, the computation of paths to multicast members is done at the sink node. Our 1-n Routing module ensures that nodes find optimal routing paths to multicast members

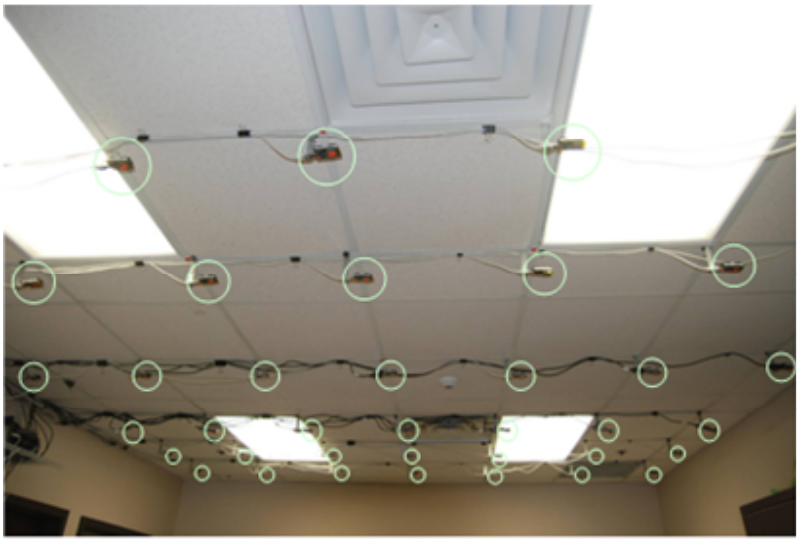


Figure 5.2: Our testbed with 42 Telosb motes.

without requiring the nodes to encode all locations of multicast members in a header. The 1-n Routing module also allows for energy-efficient recovery from packet loss and offers functionality for achieving higher energy efficiency at the cost of more storage overhead. The details of the 1-n Routing module will be discussed in Section 5.4.2.

One distinctive characteristic of wireless sensor networks compared to other networks is the traffic pattern called convergecast (i.e., n-1 routing) where all nodes report data to the sink. This unique traffic pattern, when there are holes in a network, creates regions with higher traffic called *hot zones*. The motivation behind designing the n-1 Routing module is to avoid hot zones, thereby increasing the network lifetime. Thus, when source node s wants to send a packet to the sink, it uses the n-1 Routing module that reduces the impact of the hot zones. The details on the n-1 Routing module will be discussed in Section 5.4.3. Note that we did not adopt the current implementation of a widely used convergecast algorithm like CTP [74] because of limited memory space – instead we reuse part of functionalities for our

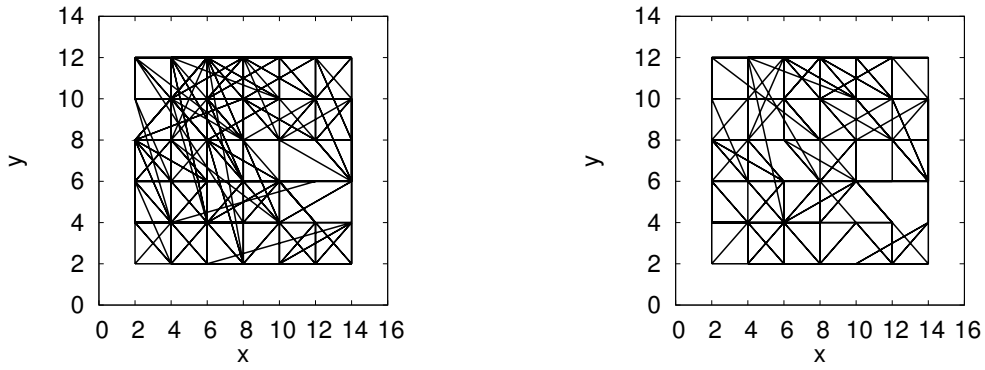


Figure 5.3: The topology of testbed with (a) pdr threshold = .7; (b) pdr threshold = .9; Unit distance equals 15 cm.

1-1 Routing module for implementing our n-1 Routing module.

5.2 Experimental Setup

Before we describe the details of each component for CoLoR framework, we present our experimental setup used for evaluating our implementation. Our testbed consists of 42 TelosB motes attached to the ceiling of an office as shown in Figure 5.2. The ceiling is about 2.7 meters high and the size of the office is 6 by 4.5 meters. The motes form a 7 by 6 grid network, where inter-node distance is approximately 30 cm. The motes are programmed and powered via USB cables connected to a main PC running Ubuntu 12.04 with AMD Opteron Processor 252 and 16 GBytes of RAM. The transmit power of motes is fixed to 1 (i.e., CC2420_DEF_RFPOWER is set to 1 for TinyOS). Consequently, the network is at most 7 hops (without holes). We debugged our work on the central PC by allowing motes to send debug messages through USB interfaces.

Figure 5.3(a) and Figure 5.3(b) depict the testbed topology with different *pdr thresholds*. The *pdr threhold* means the minimum allowable packet reception ratio

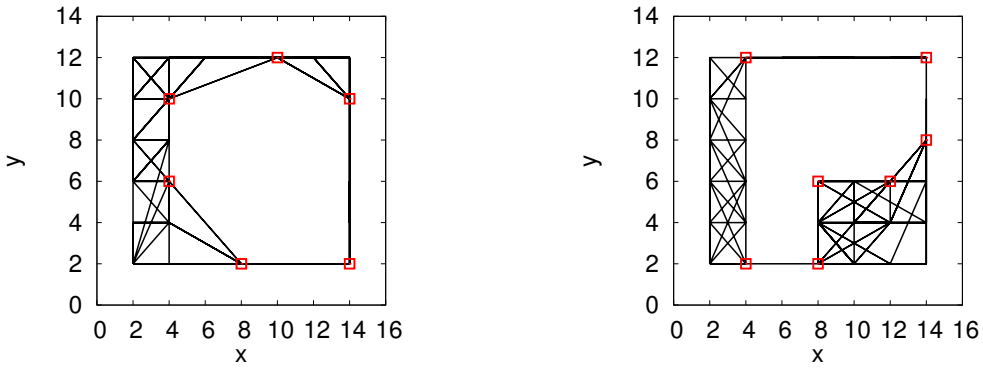


Figure 5.4: The Topology of the testbed with a large "convex" hole and a large "concave" hole, with pdr threshold=.8.

over a given link. In other words, a link with smaller pdr than a threshold is not considered as a neighbor. We created two types of holes – convex and concave holes. The topologies with holes are shown in Figure 5.4(a) and Figure 5.4(b) for convex and concave holes, respectively. To create holes, we artificially disabled links crossing the holes. For the experiments we set the pdr threshold to .8. The reason for this particular threshold is specified in Section 5.3.

5.3 CoLoR CNT Support

This section describes the implementation details of the CNT Support component. The CNT Support component detects CNTs in the network and abstracts the boundary of CNTs as abstract data types (i.e., polygons and convex hulls). The component also provides reachability information on whether a given destination node is reachable or not. The CNT Support component consists of three modules: Boundary Detection, Boundary Abstraction, and Cut Detection. Following subsections present the details of each module.

5.3.1 Boundary Detection

For the implementation of the Boundary Detection module, we use our boundary detection algorithm introduced in Section 2. The following is the nesC interface for the Boundary Detection module:

```
interface BoundaryDetection {
  command error_t initBoundaryDetection();
  command location_t* getVertices(uint8_t idx);
  command location_t* getVirtualVertices(uint8_t idx);
  command uint8_t getNumHoles();
  command error_t setVirtualVertices(location_t*,
                uint8_t idx);
  event void boundaryDetectionDone();
}
```

The command initBoundaryDetection() implements the boundary detection algorithm called the BOUNDHOLE algorithm. As described in Section 2, the BOUNDHOLE algorithm identifies "stuck nodes" where a packet can be stuck in a local minimum during the geographic forwarding process. Once stuck nodes are identified, one of them for each hole sends a control packet that travels along the boundary of the hole. While the control packet traverses the boundary of a hole, boundary nodes are detected. As we will show in the following section, as the control packet traverses, the boundary abstraction is also performed. The boundary is abstracted as a set of vertices of a polygon that surrounds the hole. Once the boundary detection completes (i.e., the abstraction is also done), the boundaryDetectionDone() event is triggered and the locations of discovered vertices are broadcast throughout the network using the Bcast module. After nodes obtain the vertices of polygons for

Table 5.2: Linear-regression-based abstraction method.

	Code Size (Lines)	**Time**	**Memory**
Linear-regression	88	$O(N)$	$O(N)$
State-of-the-art [3]	578	$O(N^2 \log N)$	$O(N)$

holes, modules in CNT Routing Engine component can use various commands such as getVertices(), getVirtualVertices(), setVirtualVertices(), isReachable() and getNumHoles() for their operations. What these commands do and when they are called will be clarified as we present the details of each module for our framework that uses the commands.

As mentioned in the beginning of this section, while the control packet is forwarded by the boundary nodes of a hole, the boundary abstraction process is also executed. The details for the boundary abstraction process are presented in the following section.

5.3.2 Boundary Abstraction

While the control packet for the BOUNDHOLE algorithm traverses the boundary nodes of a hole, the boundary nodes receiving the control packet perform the CNT abstraction process (i.e., a process for abstracting the boundary into a polygon). We, however, noted that the state-of-the-art CNT abstraction algorithm [3] cannot be directly applied to a real-world setting. Its operation is conceptually simple (see [3] for details; due to limited space, we omit the details of their protocol), but it requires complex implementation with much higher time complexity mostly due to non-trivial geometric algorithms (i.e., a problem of finding arbitrarily oriented minimum bounding box). As Table 5.2 shows, it is interesting to note that when compared with our solution, the state-of-the-art abstraction algorithm requires nearly 7 times longer code size – thus taking a lot more program memory – and higher

time complexity for computation. For a resource-constrained sensor mote, we found that the implementation of the algorithm on a mote was a very challenging task. Furthermore, the state-of-the-art CNT abstraction requires the control packet to potentially store all locations of visited boundary nodes (while traversing the boundary nodes for a hole). This requirement poses a significant limitation as the maximum packet size for 802.15 is only 133 Bytes including all headers [75]. Therefore, to make the CNT abstraction properly work in a real-world setting, we have to control the packet size and simplify the implementation to reduce the ROM size. To address these challenges, we develop a linear-regression-based CNT abstraction. It requires the control packet to contain only k (in our experimental setting, $k = 10$) locations of visited boundary nodes, where k is a user-specified parameter. When receiving the control packet, nodes perform a linear regression on at most k locations. When the correlation coefficient for the linear regression is larger than a predefined threshold, it selects the last node as a vertex and stores the location of the vertex in the control packet. Thus, when the control packet finishes traversing the boundary nodes and returns to the initiator, all vertices are identified and stored in the control packet. The following nesC code shows the interface of the Boundary Abstraction module:

```
interface BoundaryAbstraction {
  command bool leastSqrRegression
    (location_t* visitedNodes, uint16_t nodeSize);
}
```

Recall that the CNT Abstraction module is used by the CNT Detection module – the leastSqrRegression() command is called by a boundary node receiving the control packet for the BOUNDHOLE algorithm to see if it is a vertex node. The parameter visitedNodes contains the set of nodes in the control packet to be tested for linear

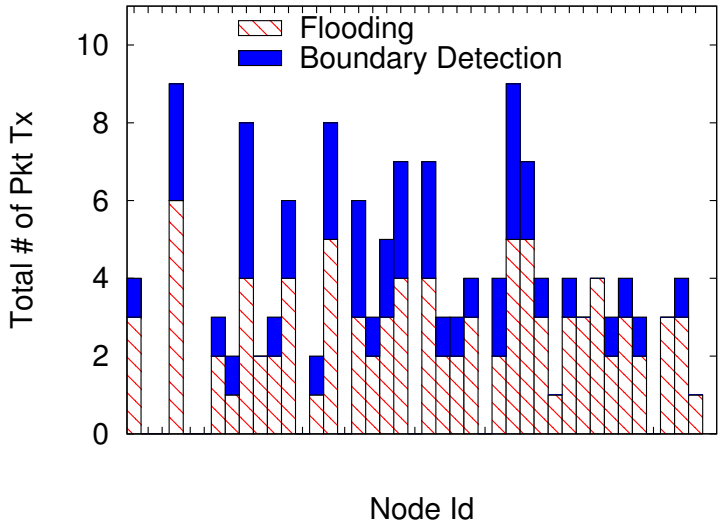


Figure 5.5: Overhead for CNT detection/abstraction.

regression and the parameter nodeSize is the number of the nodes ($< k$).

Figure 5.5 shows the overhead for CNT detection/abstraction in terms of the number of packet transmissions including the link-layer retransmissions. As shown, the overhead consists of two parts: overhead for forwarding the control packet and overhead for flooding the locations of discovered vertices. Remarkably, the per-node packet transmissions was 2.26 on average in our experimental setting, which we believe is a reasonable overhead.

5.3.3 Cut Detection

The Cut Detection module is used by the 1-1 Routing module to determine whether a given destination is reachable or not. The Cut Detection module allows nodes to find a cut with respect to *any node*, i.e., realizing the peer-to-peer cut detection. The implementation of peer-to-peer cut detection is based on the P2P-CD algorithm presented in Section 2. *One important reason why we choose the P2P-CD algorithm is that we can significantly simplify the implementation as the peer-to-peer*

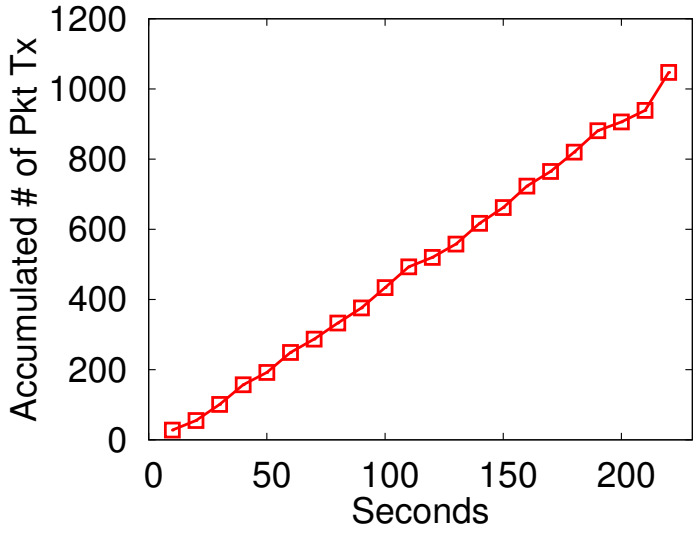


Figure 5.6: The accumulated packet transmissions when no cut detection mechanism was used.

cut detection algorithm relies on abstract CNT information, and the CNT Detection module already implements the CNT detection/abstraction.

More specifically, the Cut Detection module uses the CNT Detection/Abstraction module, i.e., it uses the boundary information in the form of a polygon. Given the locations of source node s and destination node t, the decision on whether the two nodes can reach each other is based on an application of a point-in-polygon (PIP) problem, which finds whether a point is inside a given polygon or not. The following nesC code is the interface of the Cut Detection module:

```
interface CutDetection {
  command bool isReachable(location_t dest);
}
```

The command isReachable() uses the getVertices() command to get the abstract CNT information; based on the abstract CNT information, it performs cut detection

and returns TRUE if the destination at location dest is reachable. As mentioned, the implementation of the Cut Detection module is significantly simplified, taking only 100 lines of codes, because it implements much of its functionality by reusing code from the CNT Support component. However, the benefits are significant. Figure 5.6 depicts the accumulated number of packet transmissions when no cut detection mechanism is used. More specifically, we sent a packet from a fixed source node to a randomly selected destination node every 10 seconds. We observed that when the destination node is unreachable, packets traveled around the outer boundary of the disconnected segments of the network until the maximum TTL is reached, causing a significant number of unnecessary packet transmissions.

5.4 CoLoR Routing Engine

This section presents the details of the Routing Engine component – 1-1, 1-n, and n-1 Routing modules.

5.4.1 1-1 Routing

The focus of unicast location-based routing has been on achieving guaranteed small stretch. The implementation of the 1-1 Routing module is based on our LVGR protocol presented in Section 3. The 1-1 Routing module provides the following nesC interface:

```
interface Unicast {
  command error_t initLocalVis();
  command error_t sendPacket(location_t dest,
            void* msg, uint16_t msg_size, bool, virtual);
  event error_t packetReceived(void *msg,
            uint16_t msg_size);
}
```

Table 5.3: Comparison of required memory space.

	ROM	RAM
Forwarding w/o Face Routing	26,120 Bytes	1,863 Bytes
GPSR [10]	30,291 Bytes	3,371 Bytes
CLDP [11]	38,278 Bytes	4,338 Bytes

The command sendPacket() sends a message to the destination node at location dest using outside and/or inside convex routing modes (details on the routing modes will be explained shortly). The command first obtains the abstract CNT information (i.e., information about holes in the form of polygons) by calling CNT Support's getVertices(). If the abstract CNT information is not available, the sender uses the underlying Geographic Forwarder component to send a packet. Otherwise, the sender first checks whether the destination is reachable or not by calling CNT Support's isReachable() command. Then, if the destination is reachable, based on the abstract CNT information, it finds out whether the sender is inside or outside the convex hulls of holes in the network. Depending on its location, the sender uses either the inside-convex or outside-convex routing. The parameters of the sendPacket() command are self explanatory, except for virtual, for which we defer the details until Section 5.4.3. The packetReceived() event is signaled on a destination node when a unicast packet reaches it. The command initLocalVis() is used to initiate the construction of a visibility graph.

By the time the 1-1 Routing module runs, the CNT support component may or may not have completed the CNT detection/abstraction process. When the abstract CNT information is unavailable, the 1-1 Routing module uses the Geographic Forwarder module to send a packet. In particular, for the implementation of underlying geographic forwarder for the 1-1 Routing module, we reduce the RAM and

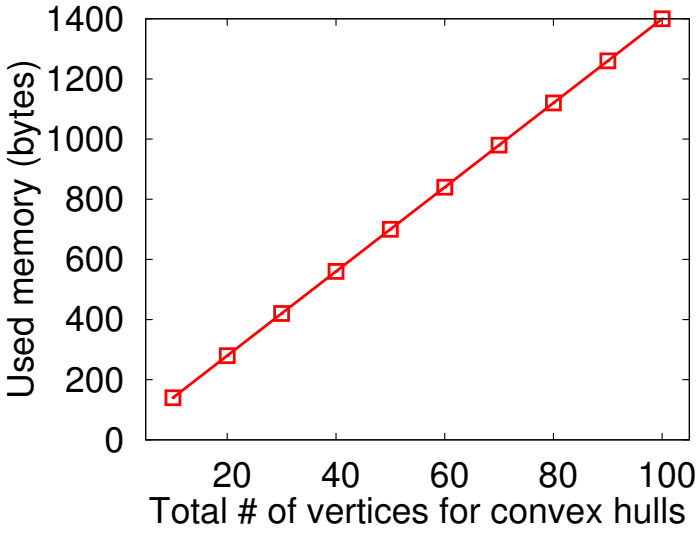


Figure 5.7: Max memory used for the outside-convex + Dijkstra computation.

ROM sizes of the Forwarder module compared with other well-known implementations [10, 11] by not using Face Routing [10]. Instead, as proposed in [8], we use the cycle of discovered boundary nodes for routing around holes. More specifically, when a packet is stuck at a local minimum, the packet is forwarded to the neighboring boundary node according to the right-hand rule [8] until greedy routing can be resumed. Table 5.3 compares the ROM and RAM sizes for the implementations of different forwarding schemes. As shown, our implementation reduced the ROM and RAM sizes, in particular the RAM size significantly, by not implementing the face routing.

To prove the feasibility for running the outside-convex routing locally in a mote, we measured the maximum used memory for the outside-convex Routing module by varying the number of vertex nodes. We depict the results in Figure 5.7. It is interesting to note that our results match the theoretical bound [58] – the maximum used memory linearly increases as the number of vertices increases. As the figure

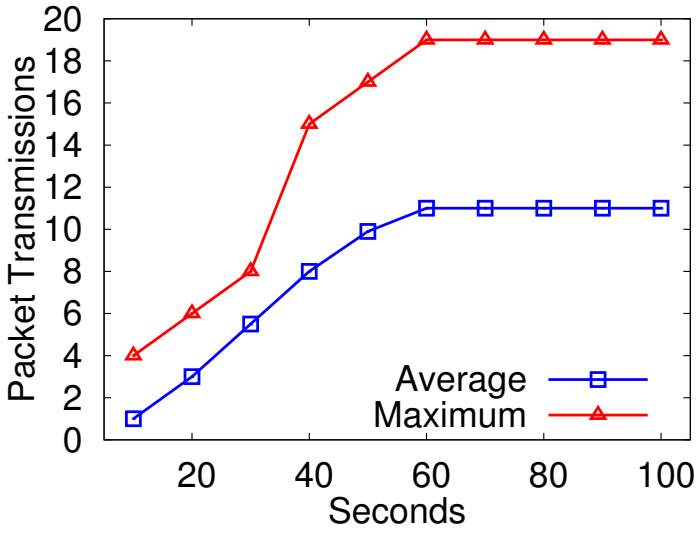


Figure 5.8: Overhead for the construction of local visibility graphs.

shows, the memory overhead for the real-world implementation is reasonable, taking up about 1 KBytes for a very large number of vertices of about 100.

We implement a distance vector routing to construct a local visibility graph. More specifically, for each hole, vertex nodes in the convex hull of the hole set their visible vertex nodes in the same convex hull as their virtual neighbors. Each node maintains a routing table where each entry contains a destination vertex node, the next vertex node to which a packet should be sent to reach the destination vertex node, and the cost in terms of Euclidean distance to reach the destination vertex node. This routing table is periodically – in our setting, every 10 seconds – exchanged with virtual neighbors. Since two visible neighbors might be multiple hops away from each other, to send a routing table to neighbors, we used our Geographic Forwarder. The routing table exchanges are continued until the routing table converges. This local visibility graph construction is initiated by initLocalVis() command, which is called by the application when the boundary detection finishes.

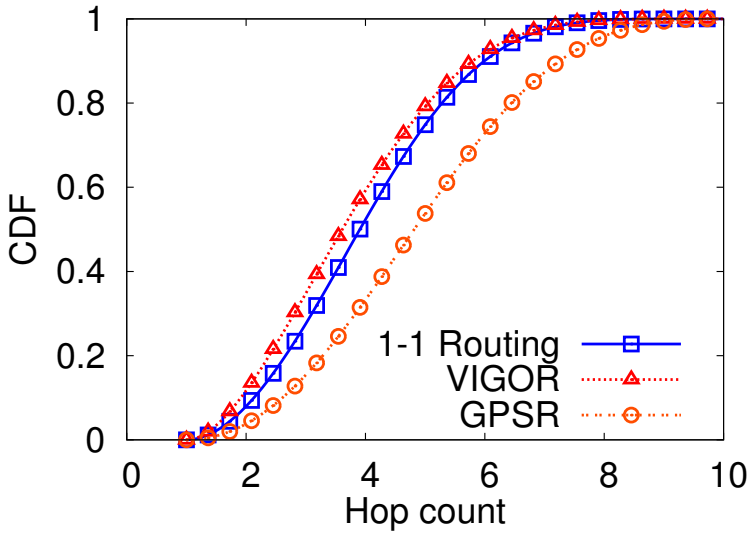


Figure 5.9: Path length in hop count for the convex hole scenario.

To measure the overhead for building a local visibility graph, we ran our 1-1 Routing module in the concave-hole scenario. Figure 5.8 depicts the per-node communication overhead – the average accumulated number of packet transmissions and the maximum accumulated packet transmissions (including link-layer retransmissions) for building the local visibility graph. We found that in our experimental environment the convergence of the routing tables was achieved relatively quickly, i.e., with small amount of overhead. More specifically, all nodes finished constructing the local visibility graph at the 6-th iteration, and the average per-node communication overhead was about 10 packets.

Once vertex nodes have routing tables that converged – local visibility graphs are constructed and nodes can use the inside-convex routing. Given the visibility graphs and the mechanisms for switching routing modes, our 1-1 Routing module achieves a bounded path stretch of $O(r)$, where r is the diameter of the largest hole in the network. Due to space constraints, we omit the proof for the analytical bound.

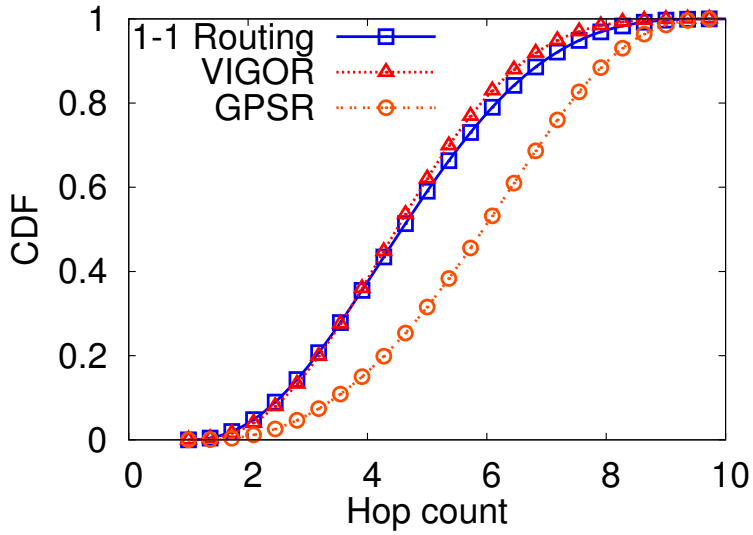


Figure 5.10: Path length in hop count for the concave hole scenario.

To show that our 1-1 Routing module generates a path with small stretch, we compared the path lengths (in terms of hop counts) for our 1-1 Routing module with state-of-the-art geographic unicast routing protocol called VIGOR [3]. Note that to obtain routing paths for VIGOR, we ran VIGOR in a C++ simulator and obtained intermediate destinations off-line, and then ran VIGOR on our testbed. We also measured the path length for GPSR as a base line. We performed experiments for both scenarios for convex and concave holes. We randomly selected 20 pairs of source and destination nodes. For the same set of source-destination pairs, we measured hop counts for different routing protocols. Figures 5.9 and 5.10 depict the results for convex-hole scenario and concave-hole scenario, respectively. Interestingly, for both scenarios, our 1-1 Routing module achieved very close path lengths to the state-of-the-art protocol, without requiring the source node to send a control packet to a destination node using a default geographic routing protocol (e.g., GPSR), which is the main drawback of the state-of-the-art protocol [3]. Another interesting

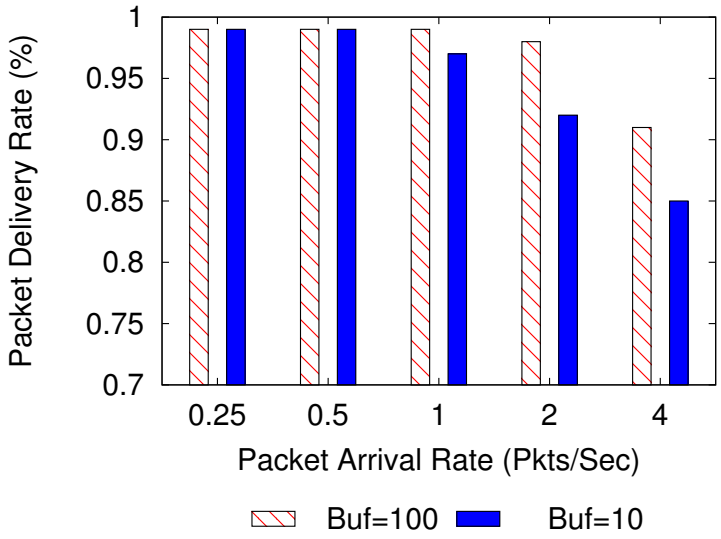


Figure 5.11: Packet delivery ratio of the 1-1 Routing module for the concave hull scenario.

observation was that GPSR performed worse in the concave-hole scenario compared with convex-hull scenario. The reason is that for the concave-hole scenario, GPSR relied more on routing along the boundary of a hole than the convex-hole scenario, especially when sending a packet to nodes inside a convex hull.

We now show that our 1-1 Routing module has reliable packet delivery ratio. To measure the packet delivery ratio, we fixed our packet size to 90 Bytes (including the header) and varied the buffer size. Figure 5.11 shows the results. We observed that our 1-1 routing achieved reliable packet delivery ratio, mostly because it is based on underlying geographic forwarder, although with a small buffer size of 10, the packet delivery ratio decreased due to dropped packets. Interestingly, this result also proves that the underlying geographic forwarder works quite well without Face Routing.

Figure 5.12: Multicast packet format: (a) output packet to the requested node; (b) output packet to a facility node.

5.4.2 1-n Routing

This section presents the details of our 1-n Routing module. Nodes use 1-n routing when they send a packet to a group of nodes (i.e., a multicast group). Based on abstract CNT information received from the CNT Support component, the 1-n Routing module finds a path to each *multicast member* (i.e., a node belonging to a particular multicast group) such that the total sum of path lengths to all multicast members, for a multicast group, is minimized. The implementation of the 1-n Routing module is based on our REGMR algorithm introduced in Section 4.

Protocol Design and Implementation

Given the limited memory space of a sensor node, we choose a design in which the sink node manages multicast members. In other words, the sink node stores the IDs and locations of multicast members for each multicast group. When a node wants to join a multicat group, it sends a *Join Message* containing the multicast group ID, and its ID and location, to the sink node; similarly when a node wants to leave from a multicast group, it sends a *Leave Message* with the multicast group ID and its ID to the sink node. When a node needs to send a packet to a set of nodes in a multicast group, it sends a *Request Message*, containing its location, the multicast group ID, and the level (we will discuss the details on "levels" shortly) to the sink node. Upon receiving the request packet, the sink node computes the optimal path to each multicast member and sends the result to the requested node.

The 1-n Routing module has the following nesC interface:

```
interface Multicast {
  command error_t sendPacket(uint8_t mcastID,
                uint8_t level);
  command error_t join (uint8_t mcastID,location˙t myLoc,
                uint16_t ID);
  command error_t leave (uint8_t mcastID, uint16_t ID);
  event uint8_t mcastPktReceived (void *msg,
                uint16_t msg_size);
}
```

A node joins a multicast group by calling the join() command which sends a *Join Message* to the sink. Similarly, a node leaves a multicast group by calling the leave() command. In the leave() command, the *Leave Message* is sent to the sink. A node calls sendPacket() command to send a packet to a multicast group. The sendPacket() command obtains the locations of facility nodes from the sink node by sending the *Request Message* to the sink. When the sink node receives the request message, it computes the locations for facility nodes based on abstract CNT information obtained by calling CNT Support's getVertices() command. Upon finishing the selection of facility nodes, the sink, using the 1-1 Routing modules's sendPacket() command, sends the locations of facility nodes to the requested source node, and transmits the locations of member nodes to corresponding facility nodes. Figure 5.12 shows the structure for this output packet. The packet of type (a) containing the locations of facility nodes denoted by f_i is sent to the source node, and the packet of type (b) containing the locations of the member nodes for facility f_i, denoted by v_{ij} is sent to the facility node. Here MID represents multicast group ID. Note that the packet size

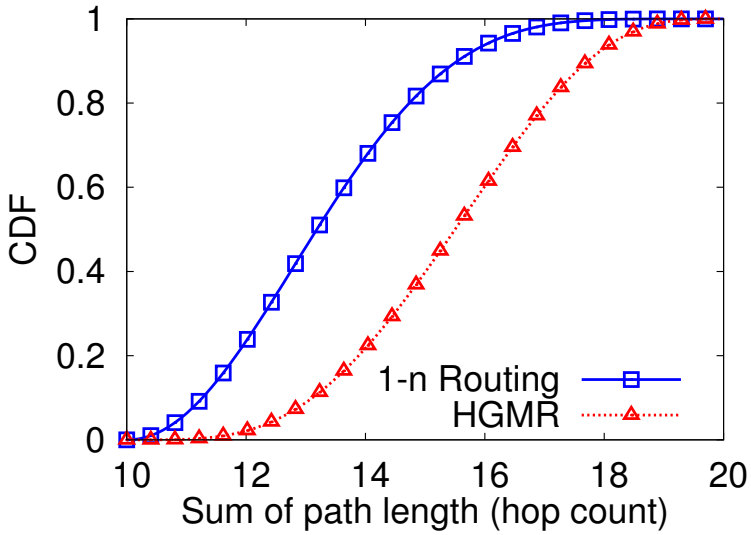


Figure 5.13: Total sum of path length in terms of hop count for 1-n routing and HGMR.

is limited to the capacity of a facility node, i.e., the maximum number of multicast members that can be assigned to a facility node. When receiving the locations of facility nodes from the sink (i.e., the 1-1 Routing modules's packetReceived() event is triggered), the source node sends the packet to the facility nodes using the 1-1 Routing module's sendPacket() command. If facility nodes receive the locations of assigned member nodes, the facility nodes store the locations so that they can later distribute a multicast packet to their members. The event mcastPktReceived() is triggered when a multicast packet reached the destination node (i.e., either a facility node or a multicast member node). If the receiving node is a multicast member, the multicast transmission process is complete. If the receiving node is a facility node, the facility node distributes the received packet to its multicast members using the 1-1 Routing module's sendPacket() command.

Facility Node Selection

We implemented an exact solver for the problem of optimally locating facility

Table 5.4: Execution time of the facility-node-location solver.

	100	**200**	**300**	**400**
Execution time for 0 hole (sec)	1.7	6.5	21.7	54.2
Execution time for 1 hole (sec)	1.2	21.3	27	45.5
Execution time for 3 hole (sec)	2.1	13.5	35	41.7

nodes using MATLAB 2011b 64 bit on a PC running Windows7 64 Bit with Intel i7 920 Processor and 24 GBytes of RAM. Table 5.4 shows the computation time for different number of multicast members and holes. As shown, our exact solver computes facility locations quite quickly for a relatively large number of multicast members. We also advise that, for a very large network with thousands of member nodes, users may adopt a heuristic solvers for capacitated facility location problem [76, 77]. We then evaluated the performance of the 1-n routing by measuring the total sum of path lengths for a given set of multicast members and compare it to the state-of-the-art hierarchical geographic multicast routing called HGMR [4]. For this set of experiments, we used 6 different sets of randomly selected multicast members and ran a multicast routing 20 times for each set in a concave-hole scenario. Figure 5.13 shows the results. As shown, our 1-n Routing module produced paths with smaller sum of path lengths compared with HGMR. Interestingly, we observed that HGMR produced much worse path lengths than expected. The reasons are: first, HGMR does not optimize the path length when determining APs (i.e., the corresponding concept for our facility node) to multicast members; second, maybe more importantly, HGMR does not consider holes in the network.

Multi-level Facility Node Selection

The 1-n Routing module employs the multi-level facility mechanism to achieve higher energy savings – by reducing the total sum of path length to multicast mem-

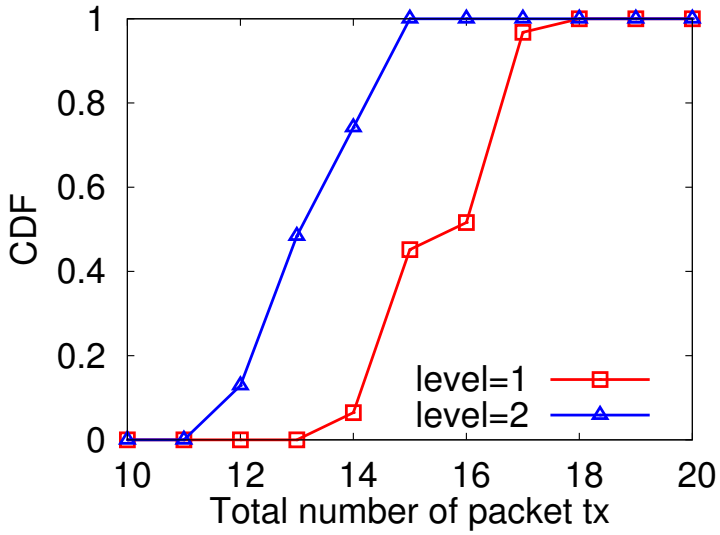


Figure 5.14: Communication overhead for different facility levels.

bers. Recall that the basic idea is that after running our solver for finding the first-level facility nodes, we run our solver again, with the selected facility nodes in the first-level as multicast members, obtaining the facility nodes in the second level, and so on. The sink node then sends the locations of selected facility nodes in the highest level to the source node (packet type (a)) and sends the locations of either assigned member nodes or facility nodes in the lower levels to selected facility nodes (packet type (b)). A source node, when calling the command sendPacket(), specifies the desired level as a parameter, so that the sink computes the multi-level appropriately.

Using higher facility levels permits higher energy savings by reducing the number of packet transmissions. However, in order to use higher facility levels, more nodes need to maintain state information (i.e., the locations of members). To see the benefits of using higher facility levels, we allowed a source node to send a packet to 8 pre-selected multicast members. We then measured the total number of packet

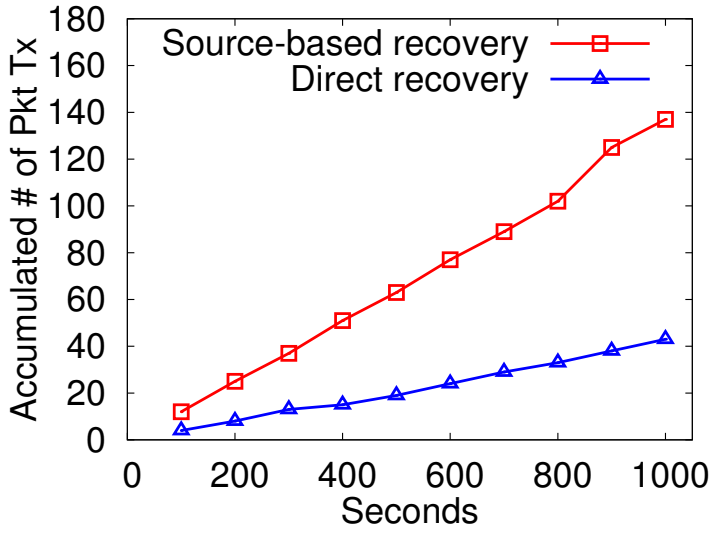


Figure 5.15: Comparison of overhead for packet-loss recovery.

transmissions for both facility level of 1 and facility level of 2. Figure 5.14 shows the results. As expected, the figure shows that when we used a higher level, we could reduce the number of packet transmissions by aggregating some paths to multicast members. However, we must note that higher facility levels require more nodes to work as facility nodes, increasing the storage overhead in the network.

Packet Recovery

The design of 1-n Routing module has an additional benefit for packet-loss recovery. For recovery from packet losses, we adopt a widely used NACK-based algorithm. More specifically, assume that a node received packets with sequence numbers 1, 2, and 4. The receiver finds that packet 3 is missing and may request retransmission to a source node. The 1-n Routing module makes this retransmission mechanism more energy efficient (in terms of number of packet transmissions used for packet-loss recovery) by allowing the receiver to request retransmissions directly to a nearby facility node. To verify the advantage of the "direct packet-loss recovery" from a

facility node, we artificially caused packet losses at multicast members with probability .9. A source node sent a packet to four multicast members every second. We then counted the total number of accumulated packet transmissions for both the "direct packet loss recovery" and the "packet loss recovery from the source node". Figure 5.15 shows the results. As the result shows, the direct packet-loss recovery from a facility node significantly reduced communication overhead. An interesting observation is that this performance difference becomes larger and larger as time elapses – at only 1000 second, we reduced the number of packet transmissions by up to 70% even in our small test-bed environment.

5.4.3 n-1 Routing

This section describes the details of our n-1 Routing module. The n-1 routing, also called convergecast, is an important routing primitive for wireless sensor networks because major part of network traffic is for nodes to report their observations to a single point, the sink node. One can observe that we can implement our n-1 Routing module by simply using the 1-1 Routing module. However, as briefly mentioned in Section 5.1, there are issues to be addressed when implementing our n-1 Routing module using our 1-1 Routing module. One important issue is the *energy hole problem* [78]: since all nodes send packets to the sink, the nodes around the sink consume higher energy because they handle higher network traffic. Fortunately, this type of energy hole problem has received sufficient attention, and many solutions have been proposed [78, 7]. However, when there are holes in the network, especially large ones, the energy hole problem occurs in overlooked regions other than around the sink node. To illustrate this scenario, see Figure 5.16, where there is a large hole in the network. In this scenario, nodes in region denoted by A, when they use our 1-1 Routing module, will send packets along the paths that are designed to optimally

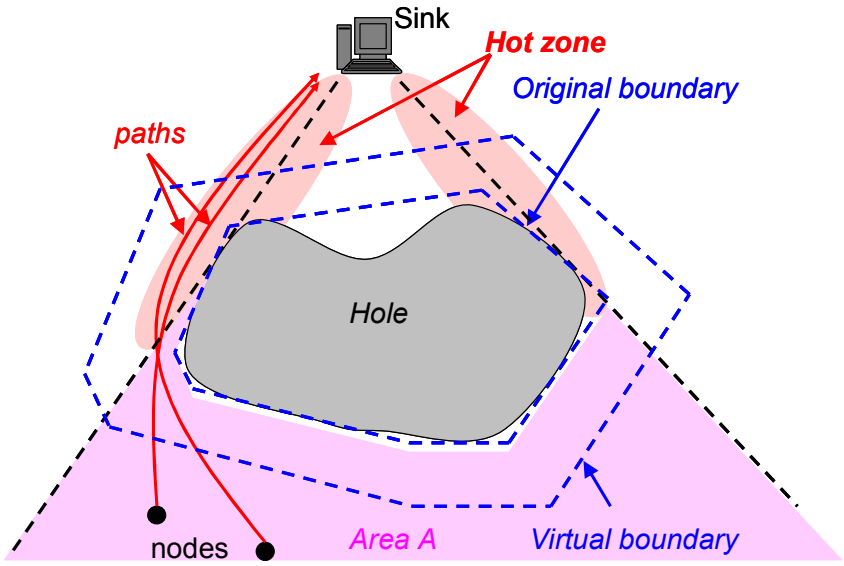


Figure 5.16: The energy hole problem in "hot zones".

detour the hole. A problem is that all such paths pass through common areas denoted by red zones called "hot zones". An energy hole problem arises in these hot zones.

Therefore, the design of our n-1 Routing module is focused on addressing the energy hole problem around "hot zones". The main idea is to allow nodes to use *virtual boundaries* when they send a packet to the sink. The virtual boundary is formed by extending the original boundary of a hole – more precisely, the convex polygon of the hole, which is obtained by calling the underlying CNT Detection module's getVertices() command. Figure 5.16 shows an example of the 2-level virtual boundary, which extends the original boundary also called level-1 boundary. When the computed virtual boundary intersects level-1 boundaries (i.e., original boundaries), the virtual boundary is not used, to prevent concentrated traffic on the level-1 boundaries; however, intersection with other virtual boundaries with levels greater than 1 is allowed. Also, depending on the locations of virtual boundaries, it is possible that

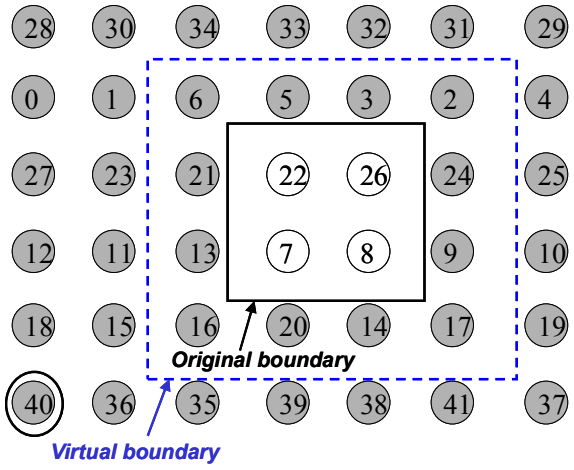


Figure 5.17: A scenario for evaluation of n-1 Routing.

there is no node at the location of the vertex for a virtual boundary. In this case, our geographic forwarder sends a packet to the closest node to the location of the vertex. The maximum level for virtual boundary is given as a system parameter. The following code is the nesC interface for our n-1 Routing module:

```
interface Convergecast {
  command error_t sendPacket(void* msg,
                uint16_t msg˙size);
  event error_t packetReceived(void* msg,
                uint16_t msg˙size);
}
```

The sendPacket() command is used to send a packet to the sink. This command allows nodes to send a packet to the sink by using different levels of boundaries alternatively, thereby distributing the network traffic. More specifically, to implement this command, we first compute the virtual boundary by calling the command setVirtualBoundary() of the CNT Detection module. Then the command sendPacket() of

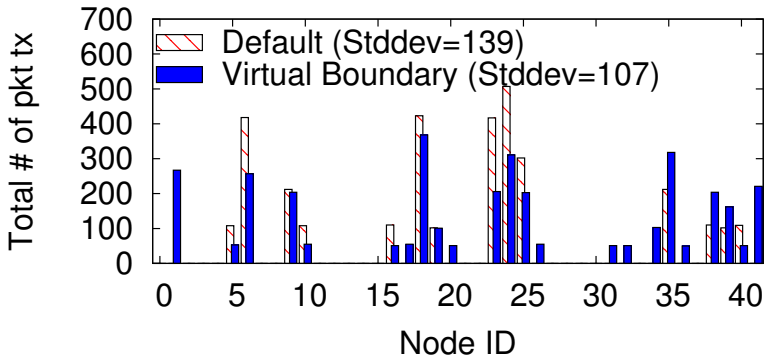


Figure 5.18: Energy consumption for convergecast and convergecast based on virtual boundary.

the 1-1 Routing module is invoked with the VIRTUAL flag set to TRUE. The 1-1 Routing module, upon receiving the command with the VIRTUAL flag being TRUE, obtains the virtual boundary by calling the command getVirtualBoundary() of the CNT Detection module and performs its unicast routing process based on the obtained virtual boundaries. Once the 1-1 Routing module determines a next intermediate destination, it forwards the decision to the underlying Forwarder component with the VIRTUAL flag set to TRUE. As the packet is forwarded, the forwarder checks whether there is a node at the location of given intermediate destination. When there is no such node, it forwards the packet to the closest node to the location. When the packet reaches the sink node, the 1-1 Routing module's packetReceived() event is triggered and the 1-1 Routing module signals the sendPacket() event for the n-1 Routing module.

To evaluate the performance of our n-1 Routing module, we placed a hole in the network as shown in Figure 5.17. In this figure, the virtual boundary for the hole is represented as red dotted convex polygon. We made nodes with IDs {28, 30, 34, 33, 32, 31, 29, 4, 25, 10, 19, 37} send a packet to the sink node with ID 40. We measured the total number of packet transmissions including link-layer retransmissions.

Figure 5.4.3 shows the results. As shown, when our virtual boundary mechanism was not used, the network traffic was concentrated on the boundary of the hole and on the path to the sink. However, when the virtual boundary mechanism was used, the energy consumption was more evenly distributed, resulting in the reduction of the standard deviation for the total number of packet transmissions from 139 to 107 – achieving a 23% decrease.

5.5 Lessons Learned

It was a challenging task to fit the entire framework in a mote. To achieve this, first, each module is designed to implement only its core functionalities, while sharing as much code as possible with other modules. For example, we offloaded codes for the selection of facility nodes and management of multicast members to the sink node, thereby simplifying the implementation on a regular node. Also, the n-1 Routing module implemented only the virtual boundary mechanism and reused the unicast functionality from the 1-1 Routing module. Similarly, the 1-1 Routing module, except for its functionalities for switching routing modes and computing intermediate destinations, reused the code from Geographic Forwarder module. Another useful technique we adopted to reduce the code size was to use the latest version of TinyOS (2.1.2 as of March 2013). This way we could reduce the total ROM size by up to 25%.

When we deployed TelosB motes on our testbed, we had to estimate the communication range of a mote (with power level 1) to get an idea about where to place motes. For this measurement, we powered a mote with 2 AA batteries. However, when we finished placing motes and powered them via USB interfaces, they had smaller communication ranges. We found that the reason was because multiple motes were connected to a single USB interface, because we used a USB hub to

connect many motes to a central PC. We also found that the orientation of TelosB motes affect the communication ranges. More specifically, we found that motes communicate longer when they are located back to back than when they are located side by side.

6. CNT RESTORATION

In this section, we present a more proactive approach to handle CNTs in the network – we use mobile nodes to "patch" CNTs for better performance. More specifically, we use mobile nodes to restore the connectivity of the network and to maximize the network performance in terms of average path length from all nodes to the sink node.

6.1 Motivations

As the cost and form factor of wireless sensor nodes shrink, we envision significant growth in the demand for *enterprise-scale* wireless sensor networks (WSNs). An enterprise-scale WSN consists of disconnected subnetworks called segments, each serving its own purpose. One example application is an enterprise-scale WSN for disaster management [12], in which one sensor subnetwork identifies victims under a rubble pile, while another subnetwork monitors the stability of a damaged building. An enterprise-scale WSN may also appear in typical WSN applications. An example is a volcano monitoring application. Since it is difficult to cover the entire area of a target mountain with nodes, a plausible design option is to deploy a number of disconnected sub-sensor networks in only critical regions. To enable a system-wide analysis, *data generated in each subnetwork must be efficiently transmitted* to a remote base station. Consequently, mechanisms for optimally connecting segments are of paramount importance for enterprise-scale WSNs.

Besides segmentation in enterprise-scale WSNs because of sparse deployments, networks can often be unexpectedly segmented if many sensors become disabled. For example, unexpected network segmentation may occur when hostile users destroy sensors; when parts of the network are destroyed after a disaster, or even when

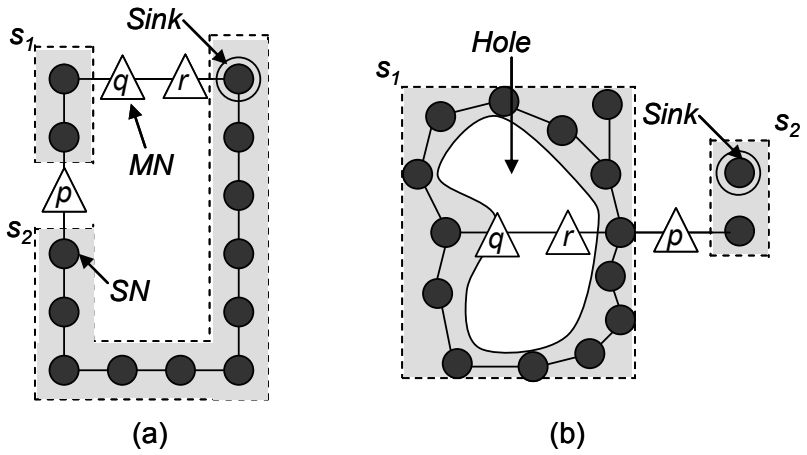


Figure 6.1: The effects of (a) segment shape, and (b) holes on connectivity restoration.

environmental factors, such as wind, may arbitrarily relocate/disable sensors. It is important that the connectivity of these segmented networks must be immediately restored for correct operation.

Proactive protocols for connectivity restoration of a segmented sensor network have recently received attention [17][18]. These protocols use more powerful nodes, called mobile/relay nodes, to build "bridges" among segments, so that the network becomes connected. These mobile nodes can be of various forms – simple wifi switches, or devices that can even fly [79]. Paying attention to the cost of mobile nodes, these schemes have focused on minimizing the number of mobile nodes. However, building bridges with the minimum number of mobile nodes may lead to suboptimal routing paths between nodes and the sink (i.e., the path length). In fact, *bridges must be carefully placed by considering several aspects of a segmented network – the sizes and shapes of segments, and even possible holes in segments.*

Two examples depicted in Figures 6.1(a) and 6.1(b) show how the geometric information of segments, and holes in segments affect the solution of connectivity

restoration obtained based on the minimum number of mobile nodes, respectively. For ease of presentation, we denote static nodes by SNs, and mobile nodes by MNs hereafter. If we are to minimize the number of MNs, a single MN (denoted by triangle p) can be deployed, as shown in Figure 6.1(a). In this case, the average hop count for all SNs in segment s_1 to reach the sink is 9.5. However, if we connect segments s_1 and s_2 through a bridge consisting of two MNs, denoted by triangles q and r, the average hop count to reach the sink is reduced to 3.5, at the cost of one more MN. Furthermore, existing connectivity restoration schemes do not consider possible holes in a network, which may negatively influence the average hop count. Figure 6.1(b) illustrates an example. A connectivity restoration scheme based on the minimum number of MNs will place a single MN denoted by triangle p. A notable fact is that some packets may have to unnecessarily travel along the perimeter the hole. However, by deploying two more MNs, denoted by triangles q and r, the average hop count can be reduced (i.e., packets can now be routed over the shortcut, to reach MN p).

Additionally, protocols for connectivity restoration must be able to cope with unexpected network segmentation. More precisely, such protocols must provide mechanisms to autonomously identify network segmentation, abstract the information about segments and utilize it for optimal connectivity restoration. State-of-art protocols [17][18] do not offer such mechanisms.

To address the above issues, first, we define a problem called the Optimal Connectivity Restoration Problem (OCRP) for a segmented WSN. OCRP minimizes both the number of deployed mobiles and the average path length from nodes to the sink such that the connectivity of the segmented network is restored. This problem is formulated as a multi-objective optimization problem. Based on the observation that the problem is NP-Hard, for solving it, we propose a centralized heuristic algorithm

called the Connectivity Restoration Genetic Algorithm (CR-GA). The algorithm is designed for fast convergence towards the Pareto Optimal set by using a novel scheme for efficiently generating initial solutions, fast evaluation of solution validity (based on the concept of *virtual sensor*) and a reduction of the solution search space. Furthermore, in order to handle scenarios when the global network topology, i.e., the locations of nodes and their neighbors, is not known (e.g., when a network is unexpectedly segmented) we propose a Distributed Connectivity Restoration (DCR) algorithm. DCR autonomously detects network segmentation and establishes bridges to an adjacent segment without relying on the global topology. The distributed algorithm has lower computation overhead than CR-GA, at the cost of a suboptimal solution, i.e., longer average path length from nodes to the sink and/or more mobile nodes used – through a theoretical analysis, we demonstrate that DCR has a bounded worst case performance, when compared with the globally optimal solution. Lastly, we demonstrate the efficiency and feasibility of proposed solutions through extensive simulations and a proof-of-concept system implementation, respectively.

6.2 Related Work

6.2.1 Relay node placement

The *relay node placement problem* (RNP) determines where to deploy relay nodes, RNs in short, in order to achieve various objectives. These objectives include providing *connectivity* [80][81], *fault tolerance* [82][83][84], and *network lifetime* [85][86][87].

Lin and Xue [88] proved the hardness of the relay node placement problem *for connectivity* and proposed 5-approximation algorithm. Cheng et al. [81] proposed a faster randomized 2.5-approximation algorithm. Lloyd and Xue [80] then studied a more general problem with $R \geq r$, where R is the communication radius of RNs, and r is the communication radius of sensors. These algorithms, however, focus only on

minimizing the number of relay nodes.

Some prior work pursued *fault tolerance* by ensuring that a given network is k-connected after deploying RNs [82][83][84]. Bredin et al. [83] presented an $\mathcal{O}(1)$-approximation algorithm for $k \geq 2$. Kashyap et al. [82] studied the fault tolerance with $k = 2$ and proposed 10-approximation algorithm. For more general case with $R \geq r$, Zhang et al. [84] proposed 14-approximation algorithm when $k = 2$.

Some researchers [85][86][87] focused on improving the *network lifetime* by deploying RNs. Hou et al. [85] jointly considered the energy provisioning and relay node placement with the objective of prolonging network lifetime. Wang et al. [86] studied the performance of dense WSNs when RNs are mobile. They showed that, with one mobile RN, the network lifetime can be increased by up to a factor of four. Wang et al. [87] considered the case with varying traffic, and provided an algorithm to deploy RNs such that the network lifetime is maximized with traffic considerations. These algorithms, however, do not consider a disconnected (segmented) network.

6.2.2 Segmented WSNs

Abbasi et al. [89] proposed two decentralized algorithms for solving the connectivity restoration problem caused by *single node failure*. The algorithm coordinates the movement of mobile nodes in a cascading manner with the objective of minimizing the distance moved. Several work proposed to restore the connectivity of a segmented network caused by *multiple nodes' failure*. Almasaeid and Kamal [90] designed a scheme that models the movement of a mobile agent to make a segmented network connected over time. However, it is infeasible to assume that mobile nodes continuously move, because mobility consumes significant energy. Lee and Younis [17][18] considered the problem of federating disjoint segments. Especially, they focused on minimizing the number of relay nodes required to restore the connec-

tivity. Noting that the connectivity-restoration problem is NP-hard, they provided a heuristic algorithm. Senel et al. [91] tackled the same problem by establishing a bio-inspired spider-web topology. However, these schemes focus only on minimizing the number of relay nodes.

6.3 System Model and Problem Formulation

We consider a disconnected wireless sensor network consisting of a set of segments denoted by $\mathcal{S} = \{s_1, s_2, ..., s_n\}$. Each segment may have holes (defined as regions without deployed nodes). In a segmented network, there are two types of deployed nodes: static nodes (SNs) denoted by the set $\mathcal{X} = \{SN_1, SN_2, ..., SN_N\}$, and mobile nodes (MNs) denoted by the set $\mathcal{Y} = \{MN_1, MN_2, ..., MN_M\}$. We assume that each node knows its location. The MNs are uniformly distributed in all segments. As we will clarify in Section 6.5, *we uniformly distribute MNs in segments, so that the WSN can autonomously cope with unexpected network segmentation.* Considering deployed SNs and MNs, we represent our segmented network as a HCG (Hybrid Communication Graph), formally defined as follows:

Definition 8 *A hybrid communication graph* $\mathsf{HCG}(r, \mathcal{X}, \mathcal{Y})$ *is an undirected graph with vertices* $\mathcal{X} \bigcup \mathcal{Y}$*, and edges defined as follows. Edge* e_{xy}*, where* $x, y \in \mathcal{X} \bigcup \mathcal{Y}$*, exists if and only if* $d(x, y) < \mathcal{R}$*, where* $d(x, y)$ *is the Euclidean distance between nodes* x *and* y*, and* $\mathcal{R}$ *is the communication range of a node.* ■

Each SN_i periodically senses the area of interest. Sensed data from each sensor is transmitted to the sink through the shortest path. We denote by P_i the path from SN_i to the sink and by $|P_i|$ the length of path P_i. We assume that MNs have significantly higher energy in comparison with SNs. Having defined our system model, we now formally describe the Optimal Connectivity Restoration Problem (OCRP), as follows:

Definition 9 *Given a set of SNs $\mathcal{X}$ and a set of segments $\mathcal{S}$ with holes,* OCRP *places a set of mobiles $\overline{\mathcal{Y}}$ ($\overline{\mathcal{Y}} \subseteq \mathcal{Y}$) satisfying the following three conditions: 1) $\frac{\sum_{i \in \mathcal{X}} |P_i|}{|\mathcal{X}|}$ (i.e., the average path length of all SNs) is minimized; 2) $|\overline{\mathcal{Y}}|$ is minimized; and 3) induced* HCG *is connected.* ■

In particular, the second condition of OCRP ensures the robustness against unexpected network segmentation; more specifically, by keeping more spare MNs ($i.e., \mathcal{Y} - \overline{\mathcal{Y}}$) uniformly distributed in segments, we improve the chance of autonomous network connectivity restoration (as it will be described in Section 6.5).

We discretize the problem by dividing the network into grid regions, where each grid is a square with side $\frac{\mathcal{R}}{2\sqrt{2}}$ ensuring that a MN in a grid can reach MNs in neighboring grids. Grids can be created by pre-computing a rectangular region that wraps a target area and dividing the rectangular region. Each node then easily determines in which grid it is located based on its location. Now the OCRP problem is to decide the grid regions where MNs will be deployed. This decision is represented by a binary variable y_{ij}, where $y_{ij} = 1$ means a MN is placed and $y_{ij} = 0$ means no MN is placed, on the grid located at (i, j). OCRP is then formulated as a multi-objective optimization problem as follows:

$$\text{Minimize} \quad \left[\frac{\sum_{i \in \mathcal{X}} |P_i|}{|\mathcal{X}|}, \sum_{i,j} y_{ij} \right].$$

$$\text{HCG is connected.} \tag{6.1}$$

$$y_{ij} \in \{0, 1\}; \sum_{i,j} y_{ij} \leq M. \tag{6.2}$$

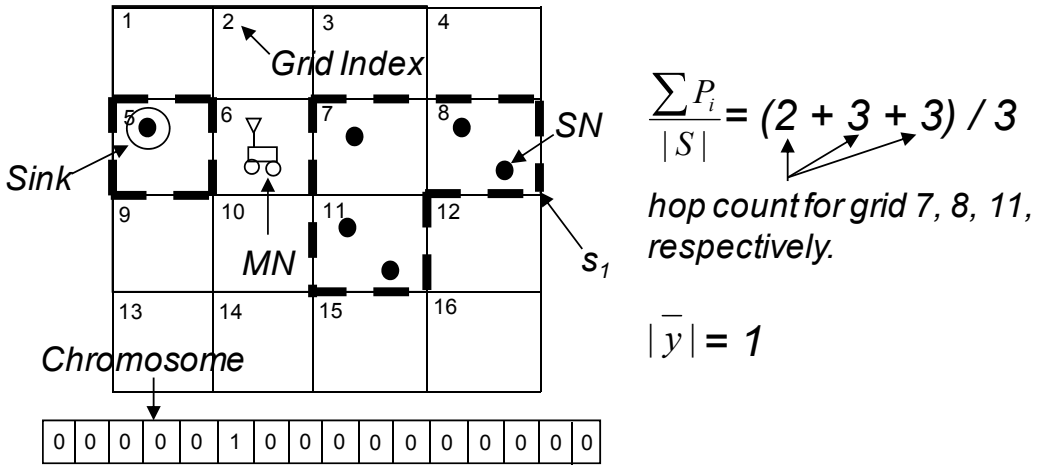


Figure 6.2: A representation of a chromosome.

The first constraint ensures network connectivity, and second constraint specifies the ranges of variables. Finding the minimum number of MNs for restoring network connectivity is NP-Hard [17]. Hence, OCRP is NP-Hard.

6.4 Centralized Connectivity Restoration

In this section we present a centralized algorithm, called Connectivity Restoration Genetic Algorithm (CR-GA), for solving OCRP. Given global topology information, CR-GA finds a near-optimal set of locations for MNs. Genetic algorithms are well suited for solving multi-objective optimization problems, because they can find a set of non-dominated solutions in parallel by maintaining a population of solutions [92] and they can efficiently approximate NP-Hard problems [93]. Since our problem is an NP-Hard multi-objective optimization problem, we propose a genetic algorithm called CR-GA. CR-GA is designed for fast convergence to a close-to-optimal solution and uses a novel initial solution generation scheme, a virtual sensor-based solution evaluation scheme, and solution search space limitation.

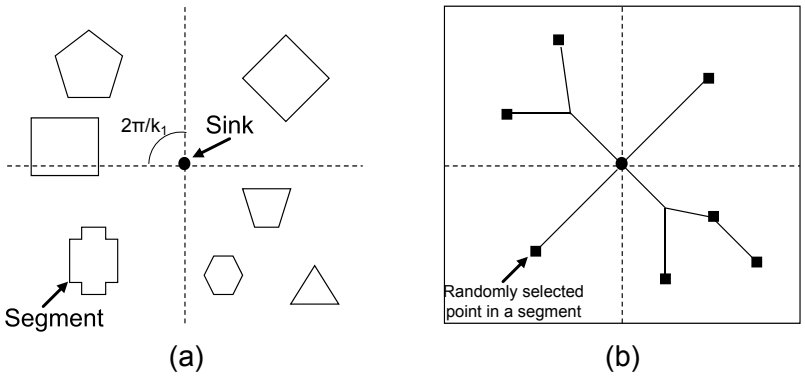


Figure 6.3: Examples of: (a) initial population generation; (b) generated bridges.

6.4.1 Initial Population

Genetic algorithms represent solutions to given problems as chromosomes. A chromosome is encoded as a bit string. In our problem, each bit represents a grid in the network. A bit is set to 1 when a MN is placed in the corresponding grid; otherwise, the bit is set to 0. Given global topology information, CR-GA computes the average path length and the number of used MNs for each chromosome (i.e., chromosome's fitness or solution's optimality). However, computing the shortest paths for all SNs for each chromosome to obtain the average path length is computationally intensive. CR-GA thus uses an optional scheme for reducing the computation overhead, when nodes are relatively uniformly distributed. Consider Figure 6.2, which shows two segments (one containing the sink, and the other one containing five SNs) and a MN connecting the two segments. *CR-GA represents the SNs in each grid as a virtual SN at the center of the grid.* CR-GA then calculates the average path length by considering the shortest paths only for the virtual SNs in the grid network.

Having explained how the chromosome is constructed and how it's fitness is evaluated, we introduce a scheme for generating initial population of k chromosomes, where k is a system parameter. Producing *high-quality, yet diverse*, initial popu-

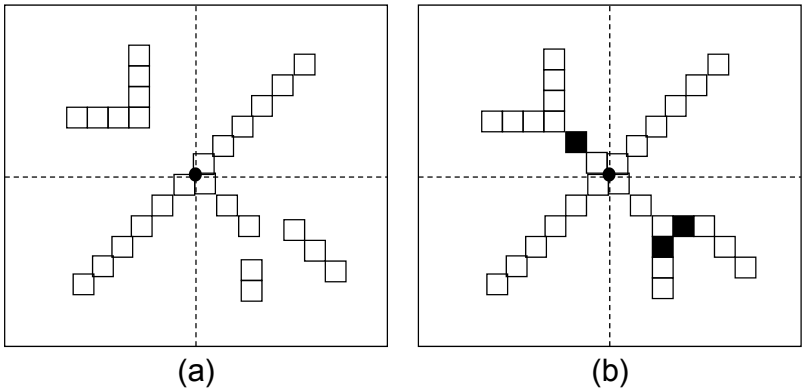


Figure 6.4: An example of correction of a chromosome: (a) before; and (b) after.

lation is critical for fast convergence. We propose a scheme which consists of two steps. In the first step, we randomly choose a point from each segment. In the second step, we select $k_1 \in \mathbb{N}$, a parameter, and divide the 2π angle around the sink into $2\pi/k_1$ sets. Figure 6.3(a) shows an example with $k_1 = 4$, where different polygons represent segments. We then apply a heuristic Minimum Steiner Tree algorithm for each subregion. The first step of the scheme ensures diversity, i.e., diverse bridge locations are considered; the second step of the scheme aims to obtain high-quality initial population, i.e., the average path length from the randomly selected points to the sink are locally minimized in subregions. Figure 6.3(b) shows the results as a tree. We then set the bits of a chromosome corresponding to the grids intersecting with the resulting tree, if the grids are either outside segments, or inside holes in segments. We repeat the above process k times, obtaining k chromosomes – our initial population.

6.4.2 Evolution and Correction

A sequence of evolutionary processes – selection, crossover, and mutation – are applied to the initial population to produce a higher quality population. We apply

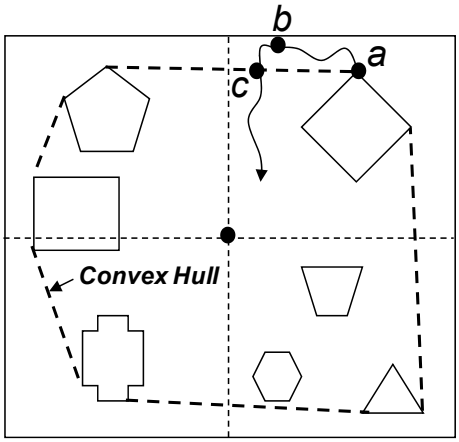


Figure 6.5: An example of search space limitation.

the well-known rank-based selection algorithm [92] to implement the selection; more specifically, we rank each chromosome based on the number of dominations, e.g., if a chromosome is dominated by three chromosomes (i.e., both the number of used mobile nodes and the average path length are smaller than the three chromosomes), its rank is 3, and chromosomes with rank 0 are called the *non-dominated* chromosomes. We sort all k chromosomes in increasing rank order and select the first half. After the selection process, we randomly choose two chromosomes, say p_1 and p_2, from the selected chromosomes to perform a crossover operation. We select a position uniformly at random in a chromosome, say r. We then build a new chromosome by taking the first r bits from p_1, and the remaining bits from p_2. We repeat this operation $\frac{k}{2}$ times, creating a new set of k chromosomes. We then perform the mutation for the generated chromosomes, where we randomly select k_2 bits and switch them. After evolutionary processes are applied, some chromosomes might not satisfy our constraints. As shown in Figure 6.4(a), some segments are not connected to the sink. To address this problem, we first identify disconnected grids. For each such grid, we find the closest disconnected grid and connect them. Figure 6.4(b) shows

the chromosome after the patching process. This evolution and correction process iterates until the set of non-dominated solutions converges, e.g., the algorithm stops when the set does not change for k_3 consecutive iterations.

6.4.3 Search Space Limitation

In order to reduce the convergence time of our algorithm, we propose to limit the search space. More precisely, we consider the placement of MNs only within the convex hull of all segments (see Figure 6.5 for an example) based on the following theorem:

Theorem 4 *The optimal solution does not place MNs outside the convex hull of network segments.*

Proof: Assume the optimal solution placed MNs outside the convex hull. Say one such grid outside the convex hull is b as shown in Figure 6.5. Then, there must be some grid that intersects the convex hull, because bridges are built toward the sink which is inside (or on the boundary of) the convex hull; say such grid is c. A contradiction arises, because there always is a better path connecting a directly to c, instead of connecting a to b, and then to c.

As described, if information about global topology is given, CR-GA obtains a set of non-dominated solutions for OCRP. However, such information may not be available, especially when a network is unexpectedly segmented due to, for example, a large number of disabled sensors by hostile users. The following section describes distributed heuristic algorithms that allow for autonomous connectivity restoration.

6.5 Distributed Connectivity Restoration

This section presents a distributed heuristic algorithm called the Distributed Connectivity Restoration (DCR) algorithm. The DCR algorithm establishes locally

optimal bridge(s) between two adjacent segments without considering all segments in a network; thus, DCR has lower overhead (when compared with CR-GA), at the cost of a suboptimal solution (possibly longer paths from nodes to the sink and/or more MNs used), allowing any MN to compute the solution for OCRP. We first describe an overview of the algorithm.

The DCR algorithm consists of mainly three phases. In the first phase, nodes autonomously detect network segmentation and find the *boundary information* of the segment they belong to. The second phase delivers the boundary information to an adjacent segment. Since this information can not be delivered via packet transmissions because the network is segmented, our protocol uses the concept of *ferrying* – one MN in a disconnected segment stores the boundary information and moves towards the sink until it meets an adjacent segment. It is important to observe that since mobility involves very high energy consumption, it may be difficult to move all the way to the sink, especially for large scale networks. Upon reaching an adjacent segment, the ferry performs the third phase, where it finds the locally optimal (i.e., between two adjacent segments) solution for OCRP. The following sections describe the details of each phase.

6.5.1 Detection and Abstraction of Segments

Nodes can detect segmentation through various methods, such as distributed network cut detection algorithms [23]. Once a node detects network segmentation, it broadcasts a control packet to nodes in the disconnected segment it belongs to. Upon receiving this control packet, nodes in the disconnected segment execute a boundary detection algorithm, e.g., [94] to find the boundary nodes of the segment. When the boundary node detection phase is finished, the boundary node with the largest ID becomes the leader. This leader node stores the locations of the boundary nodes and

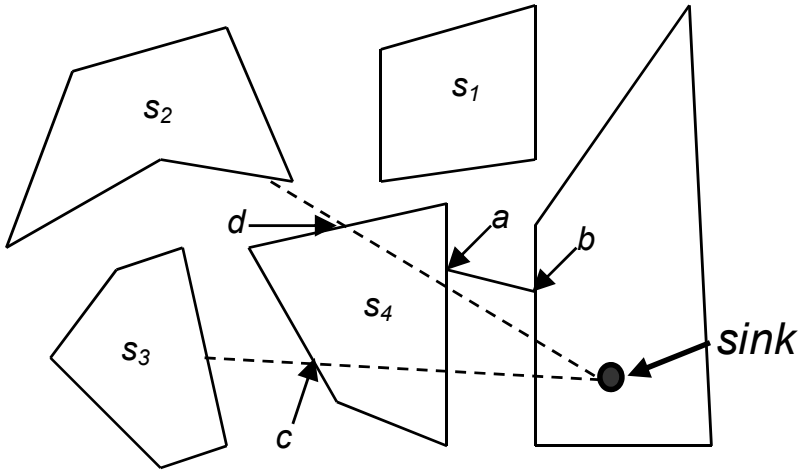


Figure 6.6: An example of ferry movement in DCR.

then broadcasts its ID to the MNs in the segment.

6.5.2 Movement of a Ferry

Upon receiving the ID of the leader, MNs inform the leader of their remaining energy. The leader then selects a MN with the largest remaining energy and sends the locations of the boundary nodes to the selected MN. The selected MN, after receiving this information, starts the ferrying process, by traveling towards the sink until it meets an adjacent segment.

When a ferry reaches an adjacent segment, it checks *the state of the segment* – A segment's state is *disconnected* when all nodes in the segment are disconnected from the sink; otherwise, *connected*. If the state is *connected*, then the ferry executes the third phase of the DCR algorithm, which finds a locally optimal set of locations for MNs, that connects the two adjacent segments. If the state is *disconnected*, the ferry waits until the state changes to *connected*.

Consider Figure 6.6 for an example. Assume that network segmentation resulted in four segments denoted by $\{s_1, s_2, s_3, s_4\}$. Assume that s_2 first sends a ferry along

Algorithm 7 DCR: code for ferry f

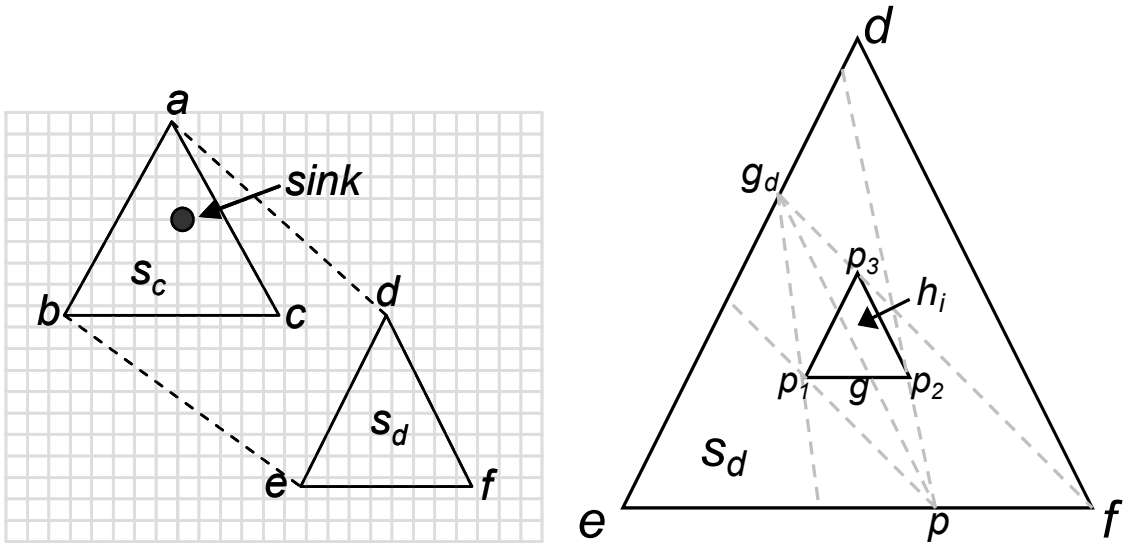
```
1: if s_c reached then
2:     // Step 1; VisEdge(s_c, s_d): Return grids on visible edges of s_c and s_d.
3:     {V_c, V_d} ← VisEdge(s_c, s_d).
4:     // Step 2
5:     for each g_d ∈ V_d, compute h(g_d).
6:     if s_c is connected then
7:         // Step 3
8:         for each (g_d, g_c) pair, g_d ∈ V_d, g_c ∈ V_c,
9:           compute (nm, pl).
10:        for each nm, find pl_min.
11:        for each nm, compute μ.
12:        find (g_d, g_c, h(g_d)) s.t. μ is maximized.
13:    else
14:        wait until s_2 is connected.
15:    end if
16: end if
```

the dotted line towards the sink. This ferry meets a node at point d and checks the state of segment s_4, which is *disconnected*, because it is not yet connected to the segment containing the sink. Thus, this ferry waits until the state changes to *connected*. Next, assume that segment s_4 sends a ferry. This ferry reaches the segment containing the sink, and decides the location of bridge $\overline{ab}$. The state of segment s_4 changes to *connected*; and the waiting ferry sent from segment s_2 now builds a locally optimal bridge by running the third phase of the DCR algorithm, described in the following section.

6.5.3 Computation of Locally Optimal Solution

This section explains the details of the third phase, summarized in Algorithm 1. The computation of a locally optimal solution involves three major steps: 1) candidate grids selection; 2) bridge placement on holes; and 3) bridge selection.

We are given two adjacent segments: one in a *disconnected* state denoted by s_d, and the other one in a *connected* state denoted by s_c. A ferry sees a network as a set of grid regions, as explained in Section 6.3 (See Figure 6.7(a)). Define a set of grids that are contained in segments s_d and s_c by G_d and G_s, respectively. In particular,



(a) (b)

Figure 6.7: Illustrations of (a) visible edges; (b) bridge placement on holes.

one grid in s_c is called the *destination grid* and denoted by t. The destination grid is either a grid containing the sink when the sink is in s_c, or a grid containing the entry point of a bridge that connects to other connected segment when the sink is not in s_c.

The first step of the algorithm is to find a set of *target grids* for adjacent segments s_c and s_d. Given s_d and s_c, we first find edges visible to each other. Consider Figure 6.7(a) for an example. The two segments are represented by triangles $\triangle abc$ and $\triangle def$. In this example, the visible edges are $\{\overline{bc}, \overline{ca}\}$ for segment s_c, and $\{\overline{de}\}$ for segment s_d. Target grids are the grids that are located on the visible edges of the two segments. We denote the set of target grids for s_d by V_d, and for s_c by V_c.

In the second step, the algorithm places bridges over holes in the segment. For each $g_d \in V_d$ and each hole h_i, invisible edges of hole h_i from g_d are identified. See Figure 6.7(b) for an example. By drawing two tangent lines from g_d to hole h_i, we can find that line segments $\overline{p_1p_2}$ and $\overline{p_2p_3}$ are the invisible edges. Define the set of

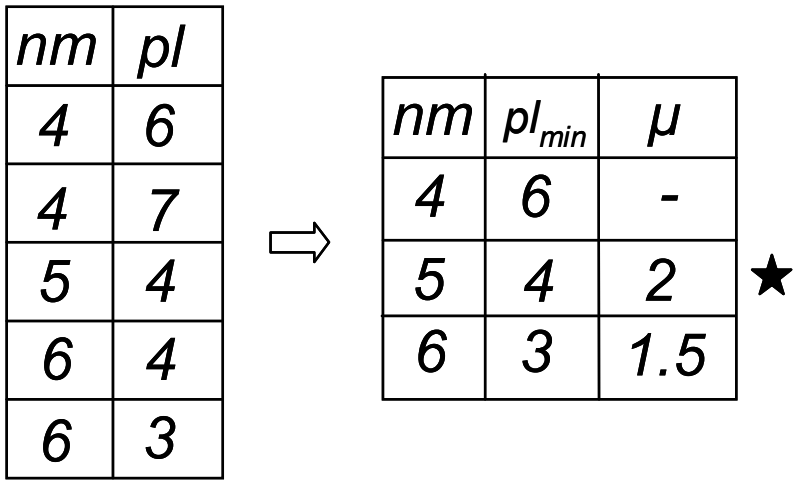


Figure 6.8: An example for marginal utility computation.

grids on the invisible edges for hole h_i by V_{h_i}. Now for each $g \in V_{h_i}$, we consider a line starting from g_d, passing through g. We denote the farthest intersection with the edges of segment s_d by p as shown in Figure 6.7(b) (there may exist multiple such intersections). If line segment $\overline{gp}$ intersects other holes, h_i is not considered. We then consider two tangent lines from p to hole h_i. These two tangent lines, with the edges of hole h_i and possibly with the edges of segment s_d, create a region A_g, which represents the number of grids that will contribute to the reduction of the average path length by placing the bridge on that hole. For example, the two tangent lines from p (i.e., $\overrightarrow{pp_1}$ and $\overrightarrow{pp_2}$) create a region $A_g = \{p, p_1, p_2\}$. We then select g' from V_{h_i} such that A_g is maximized. We denote such grid g' for each $g_d (\in V_d)$ by $h(g_d)$.

In the third step, we consider line segment $\overline{g_d g_c}$ for each $g_d (\in V_d)$ and $g_c (\in V_c)$ as a bridge connecting two adjacent segments s_d and s_c. If line segment $\overline{g_d g_c}$ intersects any of the visible edges, the line segment is not considered. Now for each pair (g_d, g_c), representing a bridge, we compute the average path length denoted by pl and the number of used MNs denoted by nm as follows: $pl = \frac{\sum_{g \in G_d}(d(g,g_d)+d(g_d,g_c)+d(g_c,t))}{|G_d|}$,

where nm represents the number of grids on $\overline{g_d g_c}$. Here the term $d(p, q)$ refers to the length of the shortest path connecting p and q. In particular, for computing $d(g, g_d)$, we consider two cases: (1) placing bridges on holes according to pre-computed $h(g_d)$ (i.e., placing MNs on line segment $\overline{g_d h(g_d)}$ that is within hole(s)); (2) not placing bridges on holes. After computing pl and nm for all (g_d, g_c) pairs, we have a set of (nm, pl) pairs (the table on the left-hand side of Figure 6.8 gives an example). Different from CR-GA (which produces a Pareto Frontier), due to the lack of computational capabilities, the DCR algorithm chooses one pair that maximizes the marginal utility. Marginal utility shows the incremental contribution of each added MN to the average path length. Choosing the solution with maximum marginal utility thus leads to the most economic decision. For example, for each nm, we first find the minimum pl, denoted by pl_{min}. The table on the right-hand side shows pairs (mn, pl_{min}). For each pair (mn, pl_{min}), we then compute the marginal utility, denoted by μ, as follows: $\mu = \frac{pl - pl_{min}}{nm - nm_{min}}$. In our example, from all the pairs (mn, pl_{min}), our DCR algorithm selects $(5, 4)$, the most economic decision.

6.6 Algorithms Analysis

As presented in Section 6.5, the DCR algorithm finds a locally optimal solution for *two adjacent* segments. If we consider *all* segments in a network, however, a simple combination of locally optimal solutions may not guarantee optimal performance. Thus, in this section, we address the following research question: *how much worse is the performance of the DCR algorithm, when compared with the centralized CR-GA?*

For answering the question, we consider a network with a circular shape centered at the sink. The radius of the network is r, where $r \gg 1$. As mentioned in Section 6.3, there are n segments in the network. Considering a very large network (i.e., $r \gg 1$) for deriving worst-case bounds, segments and MNs are represented as points in the

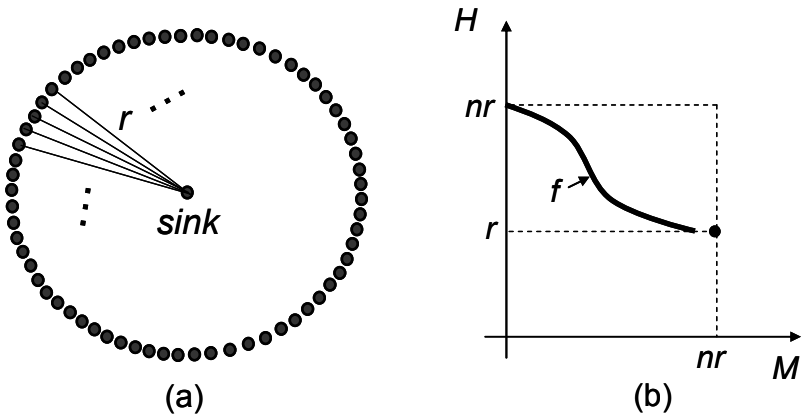


Figure 6.9: Illustrations for (a) worst case scenario; (b) the domain and codomain of Pareto Frontier.

network.

We first identify the worst case scenario for the DCR algorithm and analyze how much worse it is, when compared with the globally optimal solution. The following lemma proves the worst-case average path length and number of used MNs for the DCR algorithm.

Lemma 8 *The DCR algorithm shows the worst performance when n segments are uniformly positioned on the circumference of the network.*

Proof: Figure 6.9(a) shows the worst case scenario. It is easy to note that, for this scenario, the average path length is r, and the number of used MNs is nr. Assume by contradiction that there is a scenario with either the average path length greater than r, or the number of used MNs greater than Nr. In order to have the average path length greater than r, there must be at least one segment with its path length greater than r. Since, for this scenario, the DCR algorithm places a bridge as a straight line towards the sink, the path length cannot be greater than r, i.e., a contradiction. Similarly, in order to have the number of used MNs greater than nr, we must have

at least one bridge with more than r MNs; a bridge with more than r MNs is no longer a straight line. ■

We define a two dimensional Cartesian coordinate system with the domain (i.e., X-axis, and denoted by M) representing the number of used MNs and the codomain (i.e., Y-axis, and denoted by H) representing the average path length. Then, the Pareto frontier, i.e., *the solution of CR-GA*, is a curve represented by a function $f : M \rightarrow H$. We are interested in the maximum distance between any point on the curve (a CR-GA solution) and the point that represents the worst-case DCR solution (i.e., as obtained by Lemma 8). We call this distance *performance gap*. The main idea for obtaining the maximum performance gap is to bound the domain and codomain of function f. The following two lemmas find the bounds for the domain and codomain of function f, respectively.

Lemma 9 *The domain M of f is bounded by $0 < M \leq nr$.*

Proof: Since the path length from any segment to the sink for CR-GA is greater or equal to r, the average path length for CR-GA is greater or equal to r, i.e., $H \geq r$. Assume by contradiction that $M > nr$. Then, we have $M > nr$ and $H \geq r$, which means that any solution for CR-GA (i.e., points on the curve f) is worse than the solution obtained by the DCR algorithm (i.e., both the number of used MNs and average path length are greater than the DCR algorithm). ■

Lemma 10 *The codomain M of f is bounded by $r \leq H \leq nr$.*

Proof: By Lemma 9, we know that $H \geq r$. Since the domain of f is bounded by nr, the average path length is maximized when all paths from segments are aggregated into a single path of length nr. ■

Theorem 5 *The performance gap is bounded by $nr\sqrt{1 + (\frac{n-1}{n})^2}$*

Proof: Based on Lemma 9 and Lemma 10, the Pareto optimal curve f can be one of any possible curves defined in $0 < M \leq nr$ and $r < H \leq nr$, as shown in Figure 6.9(b). Thus, the maximum distance from point (nr, r) to curve f is the distance from point (nr, r) to point $(0, nr)$, which is $nr\sqrt{1 + (\frac{n-1}{n})^2}$. ■

Theorem 5 shows that the performance of DCR degrades asymptotically linearly with the number of segments n and the network diameter r. The interpretation of this result is that, since DCR builds bridges based on adjacent segments without taking into account all segments in the network, the overall performance degrades when there are more segments. Besides, if the diameter of a network is large, the distances between segments and the sink are more likely to be longer; thus, the performance degrades, because a better solution may be found by aggregating such long paths. However, this result also proves that the performance does not degrade arbitrarily, only linearly with the number of segments and the network diameter.

6.7 Simulation Results

For performance evaluation, we consider a 2,000m × 2,000m area with randomly generated segments of different sizes and shapes. Sensor nodes are uniformly deployed in each segment. To account for more realistic wireless communication, we adopt the radio model [61], which defines the *degree of irregularity* (DOI) as the maximum radio range variation in the direction of radio propagation. In our experiments, the radio range of a node is 40m with DOI=0.4, resulting in a network density of approximately 8 nodes/radio range.

We implemented DCR, CR-GA, and the state-of-art cut restoration scheme called Cell-based Optimized Relay node Placement (CORP) [17] in C++. CORP is a state-of-the-art centralized heuristic algorithm for restoring network connectivity by using the fewest MNs possible. For fair performance comparison between DCR and CR-

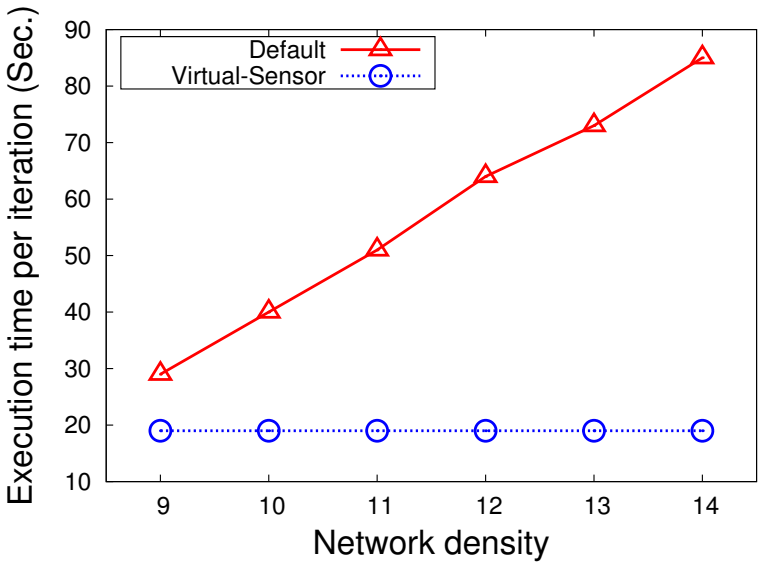


Figure 6.10: Computation speed w/ and w/o VS.

GA, we select a CR-GA solution on the Pareto Optimal set (i.e., an average path length and the corresponding number of used MNs) with the largest marginal utility. We used the following values for CR-GA: $k_1 = 4, k_2 = 10$ % of total bits, and $k_3 = 15$. CR-GA was executed on a PC with 64bit Ubuntu, Intel Core i7 CPU, and 8 GByte memory.

For our evaluation, we measured the average path length in hops and the number of used MNs by varying several properties related to a segmented sensor network: Segment Size (SS), Number of Segments (NS), Location of Sink (LS), and Hole Size (HS). A segment was represented as a polygon. The vertices of the polygon were selected within a randomly located circle with radius SS. SS is thus used to control the size of a segment. We ensure that the area covered by a segment is at least 20% of that of a circle with radius SS. The parameter LS represents the distance between the sink and the center of the network. The default values for our parameters were: SS=200, NS=4, LS=0, HS=0.

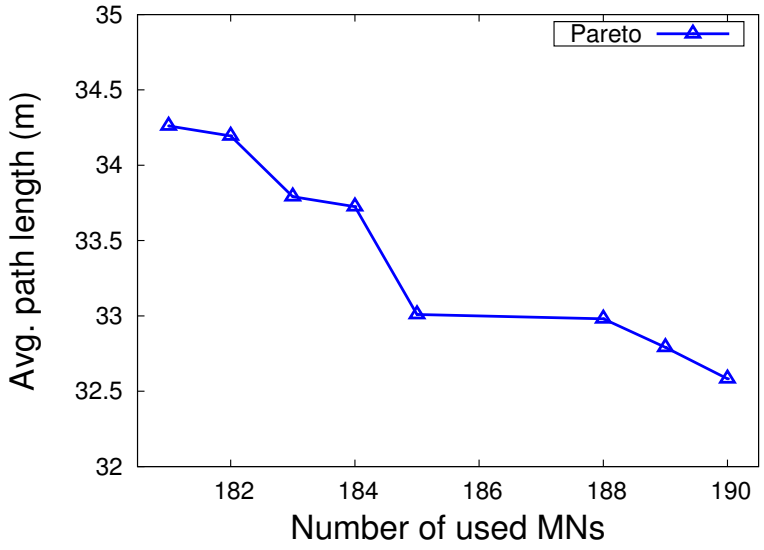


Figure 6.11: Pareto Optimal set for default settings.

6.7.1 Evaluation of CR-GA

In each iteration of CR-GA, a set of chromosomes are generated. CR-GA computes the rank of each chromosome based on the average path lengths and numbers of MNs of the set of chromosomes. Our proposed virtual sensor (VS) can significantly reduce the time to compute chromosome's rank, when nodes are uniformly distributed. To verify this, we compared the time taken by CR-GA for computing ranks when VS is used, with the time taken when VS is not used. Figure 6.10 presents the results. As shown, the average rank-computation time increased linearly with the number of nodes. In contrast, CR-GA that uses VS showed a constantly small average rank computation time.

Figure 6.11 shows the Pareto Optimal Set obtained for a network with the default setting. Each point of the graph represents a feasible solution. As the graph shows, when we can afford a large number of MNs, we can achieve a better average path length by placing more MNs; in contrast, a solution that uses a small number of

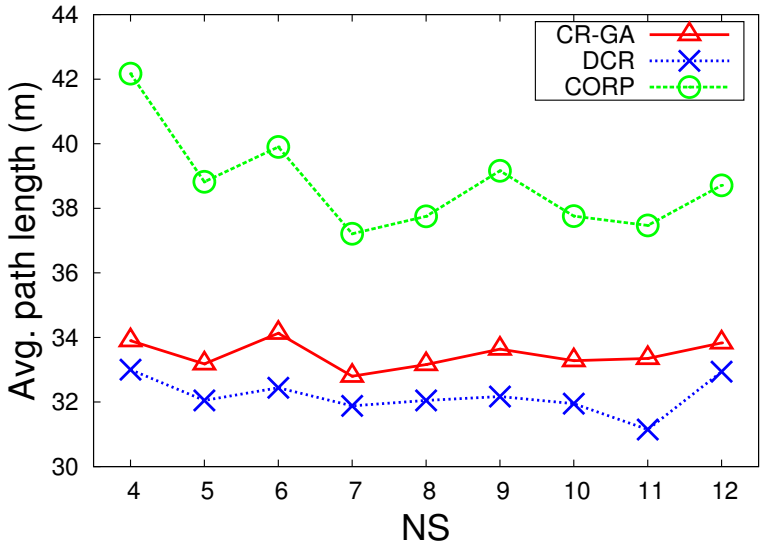


Figure 6.12: Effect of NS on average path length.

MNs has a longer average path length, but the spare MNs can be used for detecting unexpected network separation, increasing robustness.

6.7.2 Effect of Number of Segments

In this section we investigate how the Number of Segments (NS) affects the performance of the three protocols. We varied NS from 4 to 12, while having all other parameters set to default values. Figure 6.12 and Figure 6.13 show the average path length and the number of used MNs for the three protocols, respectively. Comparing CR-GA and CORP, we observed that both used a similar number of MNs regardless of NS. However, CR-GA produced much smaller average path length up to 20%. The reason is that, while CORP tries to minimize only the number of used MNs, CR-GA minimizes both the average path length and the number of used MNs. It should be noted that CR-GA involves higher computation than CORP, because it is based on a genetic algorithm. However, what really important is higher network performance achieved by optimizing both the number of MNs and average path length, because

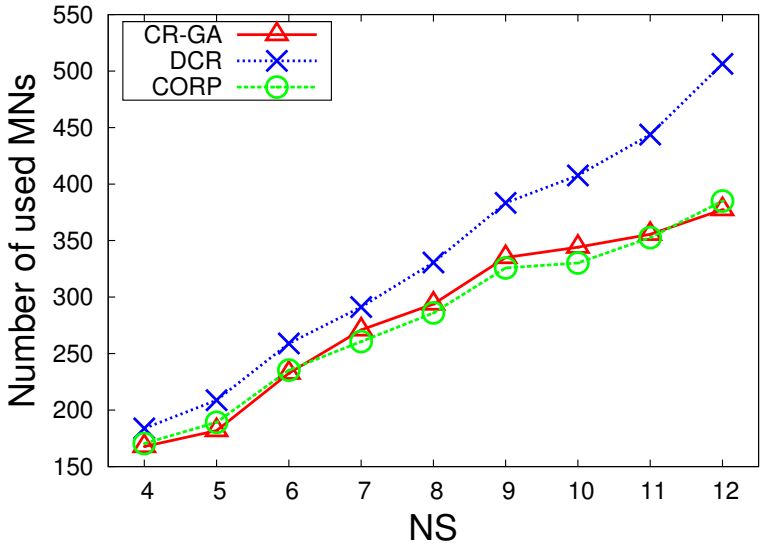


Figure 6.13: Effect of NS on the number of mobile nodes.

the computation of CR-GA is performed only once in a powerful device. Comparing CR-GA and DCR, we observe that DCR produced a slightly smaller average path length, i.e., about 4%. The reason is that DCR builds multiple bridges towards the sink without merging them. The smaller average path length, however, required a significantly higher (about 35%) number of used MNs. An interesting observation was that the difference in the number of MNs used by the two protocols increased as NS increased. We believe this result confirms our theoretical analysis which shows that the performance gap between CR-GA and DCR increases with the number of segments.

6.7.3 Effect of Sink Location

In this section we investigate how the Location of the Sink (LS) affects the performance of the three protocols. We select sink locations to be LS meters away from the center of the network, towards one corner of the network. We set all other parameters to their default values. Figure 6.14 and Figure 6.15 show the average

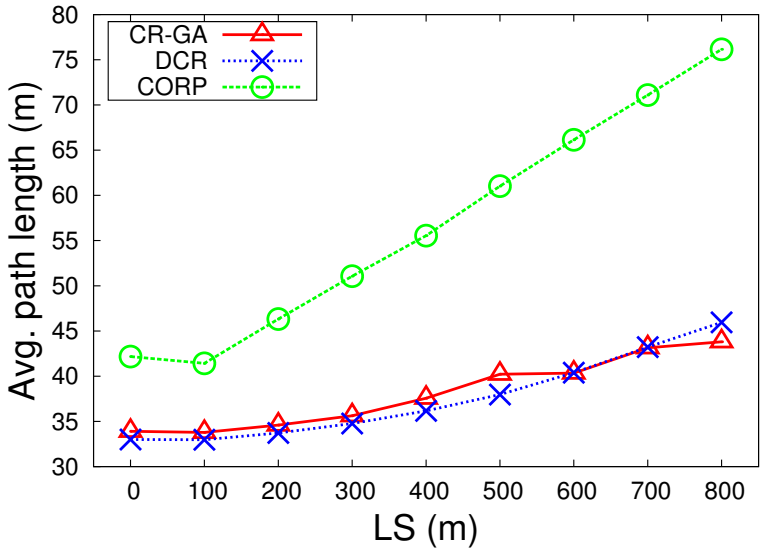


Figure 6.14: Effect of LS on average path length.

path length and the number of used MNs for the three protocols, respectively. One immediate observation was that the average path length and the number of used MNs for all three protocols increased as we increased LS. This is simply because when the sink is located far from the center of the network, a packet must travel longer distance to reach it. Comparing CR-GA and CORP, we found that the two protocols used a similar number of MNs. However, CR-GA produced much smaller average path lengths up to 75%. An interesting observation was that CORP's performance was more significantly affected by LS, than CR-GA; more precisely, while the average path length of CR-GA gradually increased with increasing LS, the average path length of CORP increased more steeply. The reason is that CORP is designed to build bridges towards the center of the network. Next, we compared CR-GA with DCR. While they achieved similar average path lengths, DCR used more MNs up to 20%. The reason is the same as above, namely that DCR builds bridges towards the sink without merging them. In fact, CR-GA finds a balance between the number of

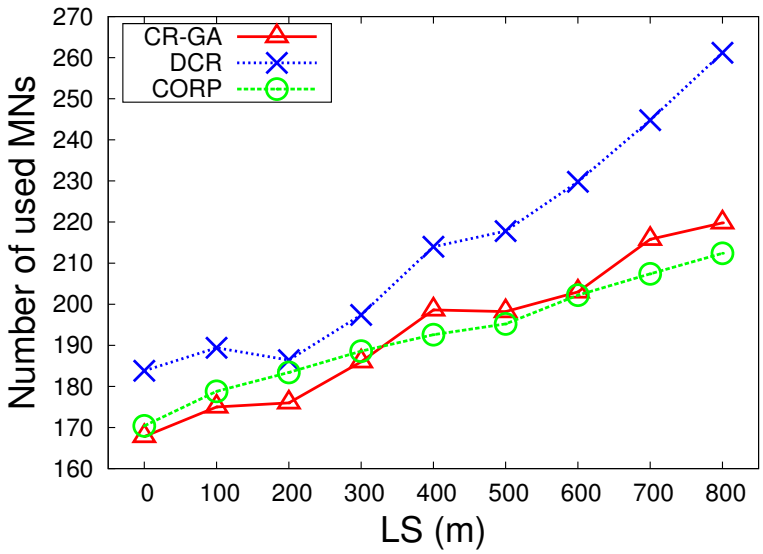


Figure 6.15: Effect of LS on the number of mobile nodes.

MNs and the average path length by appropriately merging bridges. An interesting observation was that the difference between the number of MNs used by DCR and CR-GA increased with increasing LS. The reason is that, since DCR favors building a direct bridge towards the sink, if the sink is far, it requires more MNs to connect the segment to the sink.

6.7.4 Effect of Segment Size

In this section we investigate the effect of Segment Size (SS). Figure 6.16 and Figure 6.17 depict the average path length and the number of used MNs for the three protocols, respectively. An immediate observation was that as we increased SS, both the average path length and the number of used MNs decreased for all protocols. The reason is simply that larger segments take more space in the network, leaving fewer empty spaces. Thus fewer MNs are required for connectivity restoration. Comparing CR-GA and DCR, we observed that both showed a similar performance in terms of the average path length. The difference comes from the number of used MNs, as CR-

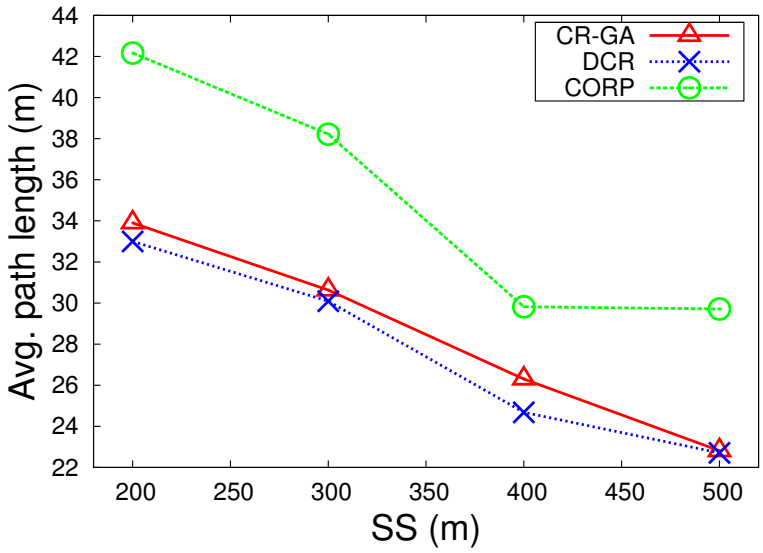


Figure 6.16: Effect of SS on average path length.

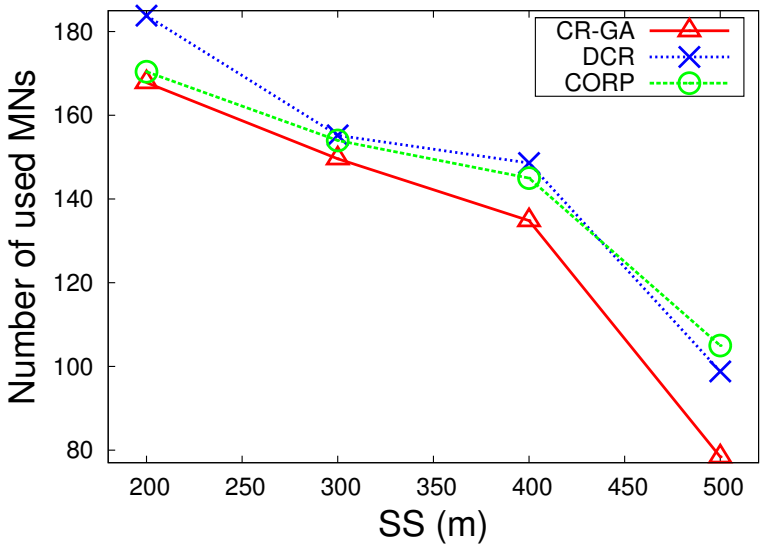


Figure 6.17: Effect of SS on the number of mobile nodes.

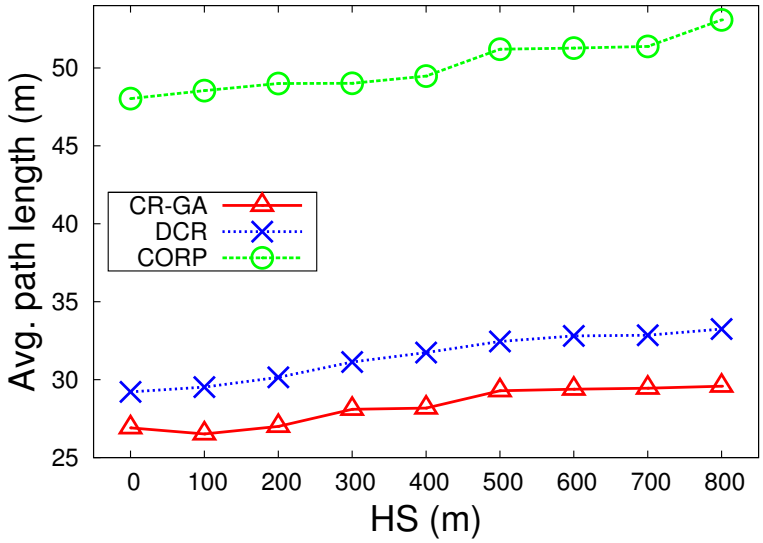


Figure 6.18: Effect of HS on average path length.

GA used about 10% fewer MNs on average, a relatively small improvement. This result is not surprising since the number of segments was the default value of 4. With few segments, DCR builds bridges quite well. Second, we compared CR-GA and CORP. We observed that the average path length of CORP was worse than CR-GA by up to 25%. The reason is that CORP does not consider the average path length. An interesting observation was that the difference in the number of MNs between CORP and CR-GA became larger as SS increased. We believe the reason is that CORP chooses a *representative node* (i.e., the starting point of bridges, selected for each segment) without considering the size and shape of segments in a network. Therefore, the impact of sub-optimally selected representative nodes becomes greater as the segment size becomes larger (i.e., more possible locations for selecting representative nodes).

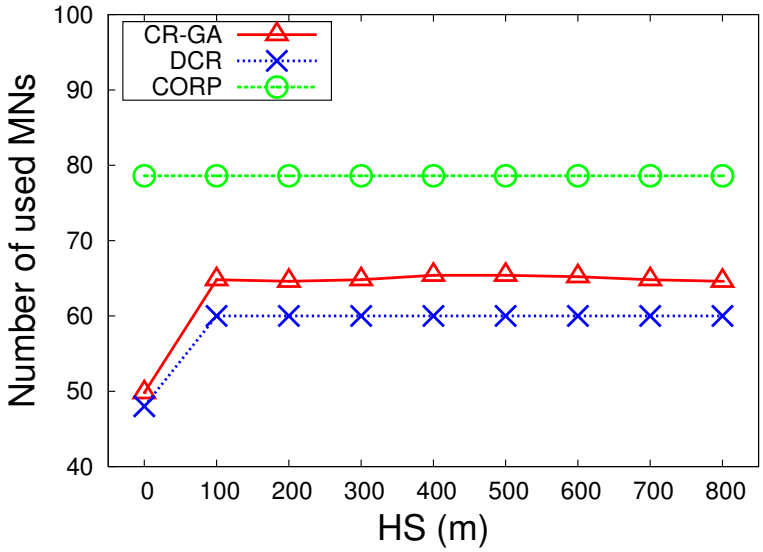


Figure 6.19: Effect of HS on the number of mobile nodes.

6.7.5 Effect of Hole Size

To evaluate the impact of holes on protocols' performance, we consider a scenario with two large rectangular-shaped segments (200m × 1,000m), and place holes of varying sizes in them. In each segment we create a bar-shaped hole, of varying heights (i.e., bars with different sizes of 100m × [0, 800]m). Figure 6.18 and Figure 6.19 depict the average path length and the number of used MNs for the three protocols, respectively. As shown in Figure 6.18, for all three protocols, the average path length increased with an increasing hole size. The reason is that larger holes result in longer, detoured routing paths. It should be noted that, although DCR and CR-GA place bridges on holes, depending on the locations of bridges, there are still nodes that use detoured paths, thereby showing small increases. Comparing CR-GA and DCR, we observed that CR-GA produced about 9% smaller path length by using more MNs. The reason is that, while DCR places only a single, straight-line bridge over a hole, CR-GA places multiple bridges, or even merge bridges. Comparing CR-GA

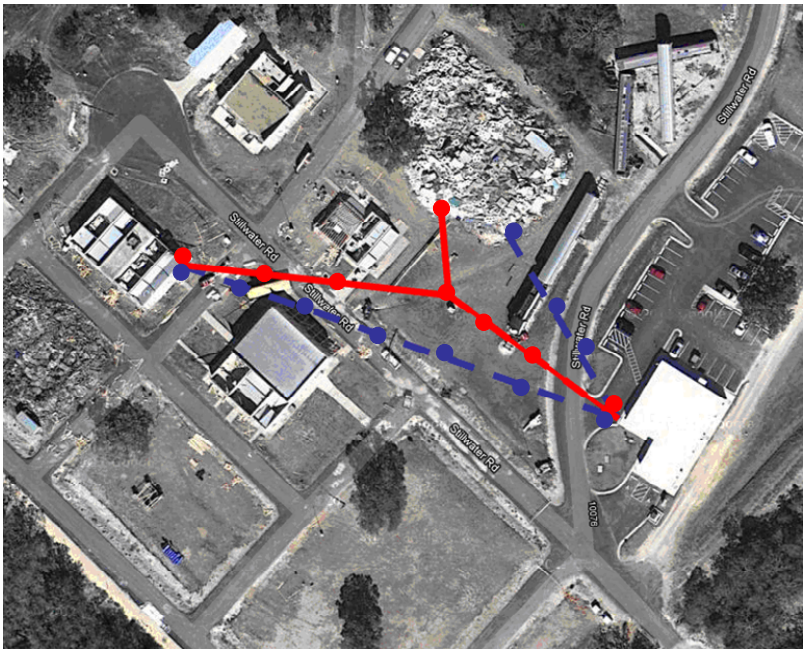


Figure 6.20: A deployment area at a Disaster Training Facility.

and CORP, we observed that although CR-GA places MNs on a hole, CORP used more MNs for restoring the connectivity. The reason is the large sizes of the two segments used in this experiment. As we mentioned previously, CORP more likely to choose representative nodes far from the sink, when the size of a segment is large. This suboptimal selection of representative nodes results in a large number of MNs. Furthermore, since CORP does not handle holes, it has longer average path length.

6.8 System Evaluation

As a proof-of-concept system, we implemented our DCR algorithm in TinyOS 2.1.1 for the TelosB platform and compared it with CORP. We deployed 10 TelosB motes in each of two segments in a disaster training facility of approximately $150m$ by $150m$, as shown in Figure 6.20. Routing paths from motes to the sink were obtained using CTP [74]. Motes reported events, with varying reporting rates, i.e., 250msec,

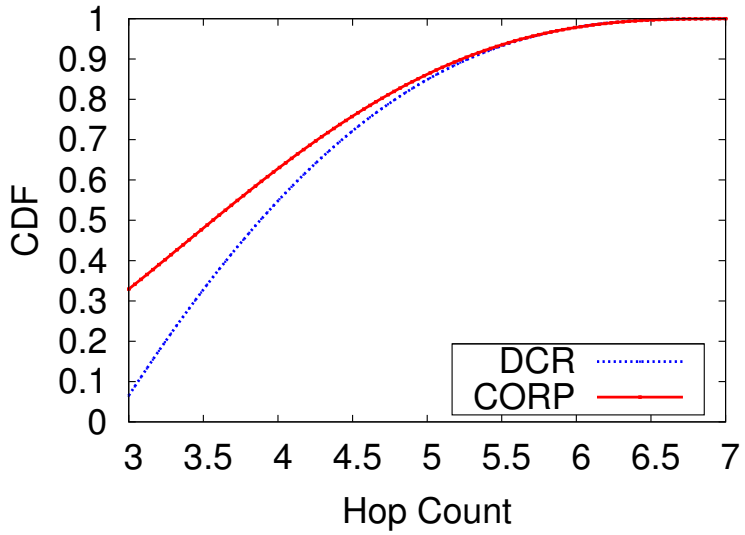


Figure 6.21: CDF of hop count.

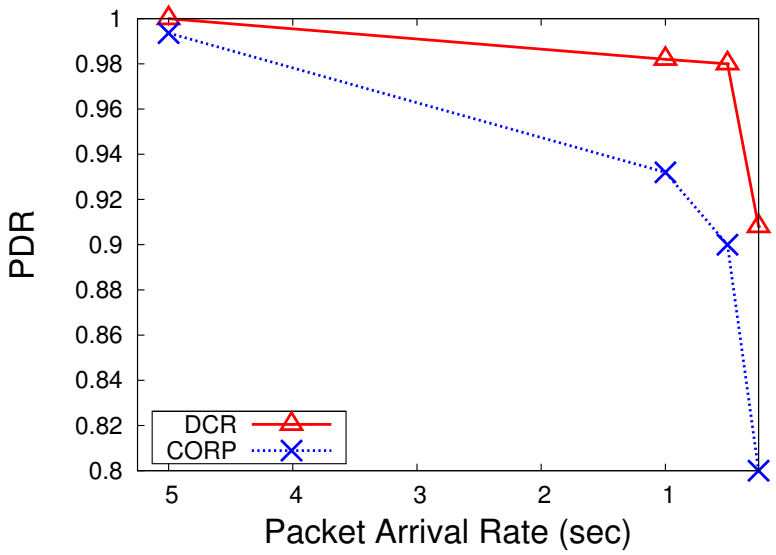


Figure 6.22: Packet delivery ratio.

500msec, 1sec, and 5sec, by sending a packet. We manually placed TelosB motes on the computed bridges based on DCR and CORP, using TelosB motes as MNs. As Figure 6.20 shows, the solid line represents the bridges for CORP, and the dotted line the bridges for DCR; filled circles represent MNs. We measured the path length in hops and computed the packet delivery ratio at the sink.

Figure 6.21 shows the cumulative distribution function for path length (in hops) for DCR and CORP. As shown, DCR has a smaller hop count. The reason is that CORP, to reduce the number of MNs, merged two bridges, resulting in longer paths. We then compared the Packet Delivery Ratio (PDR) of DCR and CORP. Figure 6.22 shows the results. For both protocols, the PDR decreased as the packet arrival interval decreased. A notable observation was that the PDR of CORP more rapidly decreased. We believe that the reason is because the merged paths increased the chance of collisions and possible congestion.

7. CONCLUSIONS

7.1 Conclusions

In this dissertation, we presented challenges and solutions for location-based routing in wireless sensor networks with complex network topologies. Efficient CNT detection/abstraction algorithms are developed to find the boundary information of CNTs in the network and abstract the boundary information into user-specified data types. Using proposed CNT detection/abstraction algorithms, we developed CNT-aware location-based routing protocols for main routing primitives: unicast, multicast, and convergecast. These routing protocols are then seamlessly integrated into an unified location-based routing framework to cope with the recent calls for a routing protocol that can handle diverse traffic patterns. We presented the system design and implementation details of the framework. Finally, we considered a more proactive approach where mobile nodes are used to restore the connectivity of the network such that resulting performance is maximized.

7.2 Future Work

7.2.1 Segmented Sensor Networks

As the cost and form factor of sensor nodes shrink, we envision that demand for enterprise-scale wireless sensor networks will increase. Enterprise-scale sensor networks consist of a multitude of smaller, potentially disconnected, sensor networks. An example is a WSN application for disaster response. In this application, a sensor network may be deployed on a rubble pile for monitoring victims, while another sensor network may be deployed for monitoring the stability of a building. These potentially disconnected sub-sensor networks, called segments, are connected by plac-

ing "bridges" (consisting of stronger devices, e.g., Wi-Fi switches) in-between. This new type of network consisting of segments and bridges is called a segmented sensor network. A segmented sensor network may also appear in typical WSN applications. An example is a volcano monitoring application. Since it is difficult to cover the entire area of a target mountain with nodes, a plausible design option is to deploy a number of disconnected sub-sensor networks in only critical regions, creating a segmented sensor network. Sometimes a segmented sensor network appears because of a large number of node failures. For example, environmental factors such as wind may relocate nodes; hostile users may destroy part of deployed nodes. For correct operations, the connectivity of the network must be immediately restored by placing bridges.

Segmented sensor networks offer a vast amount of research opportunities. For example, the capacity of a segmented sensor network is unknown. The capacity analysis of a segmented sensor network is fundamentally different from the traditional capacity analysis, because the capacity of a segmented network, unlike general wireless sensor networks, depends on the sizes and shapes of segments, as well as the properties of bridges, e.g., the number, length, and bandwidth of bridges. Energy provision to nodes comprising bridges is also a potential research problem, because the nodes handle higher traffic and thus consume higher energy. Possible approaches include building redundant bridges, employing self-recharging schemes, and allowing for duty-cycling. However, no optimal method is known. In addition, segmented sensor networks require fundamental changes to existing algorithms. For example, in a segmented sensor network, even a simple greedy routing shows poor performance, because it does not properly handle the scenario where a packet crosses a bridge.

7.2.2 Practical Aspects of Node Deployment

Random deployment (e.g., from a ground/air vehicle) is an often used method for deploying nodes. When nodes are randomly deployed, it is difficult to expect that all nodes stand in an upright position for the best performance. In other words, the antennas of nodes tend to have different orientations. Given randomly deployed nodes, a potential research question is: "If we consider the orientation of an antenna as a random process in a WSN, what would be the performance of the WSN?" More specifically, we are interested in analyzing various performance metrics such as the capacity, throughput, and delay, given a WSN consisting of nodes with their antenna-orientations modeled as a stochastic process. A more interesting aspect is to verify the analytical results by comparing them to the real-world experimental results obtained by actually deploying nodes randomly. Eventually, this future work may propose a novel hardware design that guarantees the best performance under the circumstances of random deployment.

7.2.3 Local Minimum-Aware Duty Cycling

The lifetime of a WSN is dramatically improved when nodes are duty-cycled. However, few geographic routing protocols adopt duty-cycling. Only recently, geographic routing in a duty-cycled WSN is considered, in which a scheduling algorithm is developed to allow as many nodes as possible to be in the sleep mode, while maintaining desired connectivity and routing latency. The motivation behind this algorithm is that the uniform node degree is essential for the optimal performance of a geographic routing protocol. However, as long as a local minimum is involved, especially when a network has holes, maintaining the uniform node degree is no longer an efficient solution. The main reason is because if we reduce the degree of nodes near holes, the probability for a packet to be stuck in a local minimum increases.

Thus, this future work proposes a new approach for strategically putting nodes into the sleep mode considering the impact on the probability that a packet is stuck in a local minimum.

7.2.4 Convergecast as Inverse Multicast

Intuitively, convergecast routing can be seen as inverse multicast routing. Based on this intuition, our future work is to apply the main idea for our geographic multicast routing protocol, i.e., the concept of the facility node, to the development of our convergecast routing protocol. More specifically, given a source node and all other nodes, a set of facility nodes are selected such that the total sum of path lengths from all nodes to the source node is minimized. This way, nodes, when they report data to the source node, they will first send a packet to a facility node; the facility node will then forward the packet to the source node. A more interesting aspects are: first, in designing the inverse multicast routing protocol, i.e., for selecting facility nodes, the concept of virtual boundary nodes can be used; in other words, different sets of facility nodes will be selected based on different sets of boundary nodes. Second, a facility node can be used as a data aggregation point, which will significantly reduce the communication overhead.

7.2.5 Evaluation with Realistic Network Holes

In a real-world deployment, a network may have various numbers of holes with different shapes. Thus, in order to obtain more credible results, it is important to evaluate our proposed protocols for a sufficiently large number of "randomly generated" hole shapes and placements to approximate realistic network holes.

REFERENCES

[1] A.-M. Kermarrec and G. Tan, "Greedy geographic routing in large-scale sensor networks: a minimum network decomposition approach," in *Proc. of ACM MobiHoc*, Chicago, Illinois, USA, 2010, pp. 161–170.

[2] R. Flury, S. Pemmaraju, and R. Wattenhofer, "Greedy routing with bounded stretch," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 1737–1745.

[3] G. Tan, M. Bertier, and A.-M. Kermarrec, "Visibility-graph-based shortest-path geographic routing in sensor networks," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, 2009, pp. 1719–1727.

[4] D. Koutsonikolas, S. M. Das, Y. C. Hu, and I. Stojmenovic, "Hierarchical geographic multicast routing for wireless sensor networks," *Wireless Network*, vol. 16, no. 2, pp. 449–466, 2010.

[5] J. Sanchez, P. Ruiz, and I. Stojmnenovic, "Gmr: Geographic multicast routing for wireless sensor networks," in *Proc. of IEEE SECON*, Reston, Virginia, USA, 2006, pp. 20–29.

[6] X. Xiang, X. Wang, and Y. Yang, "Stateless multicasting in mobile ad hoc networks," *IEEE Transactions on Computers*, vol. 59, pp. 1076–1090, 2010.

[7] H. Zhang and H. Shen, "Balancing energy consumption to maximize network lifetime in data-gathering sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 10, pp. 1526–1539, 2009.

[8] Q. Fang, J. Gao, and L. J. Guibas, "Locating and bypassing holes in sensor networks," in *Proc. of IEEE INFOCOM*, Hong Kong, 2004, pp. 2458–2468.

[9] M. Fayed and H. T. Mouftah, "Localised alpha-shape computations for boundary recognition in sensor networks," *Ad Hoc Networks*, vol. 7, no. 6, pp. 1259 – 1269, 2009.

[10] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Proc. of ACM MOBICOM*, Boston, Massachusetts, USA, 2000, pp. 243–254.

[11] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic routing made practical," in *Proc. of USENIX NSDI*, Berkeley, California, USA, 2005, pp. 217–230.

[12] S. M. George, W. Zhou, H. Chenji, M. Won, Y. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, "DistressNet: a wireless AdHoc and sensor network architecture for situation management in disaster response," *IEEE Communications Magazine*, vol. 48, no. 3, Mar. 2010.

[13] U. Park and J. Heidemann, "Data muling with mobile phones for sensornets," in *Proc. of ACM Sensys*, Seattle, Washington, USA, 2011, pp. 162–175.

[14] B. Li, Z. Sun, K. Mechitov, G. Hackmann, chenyang Lu, chirley Dyke, G. Agha, and B. Spencer, "Realistic case studies of wireless structural control," in *Proc. of ACM/IEEE ICCPS*, Philadelphia, Pennsylvania, USA, 2013, pp. 112–121.

[15] *M2M, http://en.wikipedia.org/wiki/Machine_to_machine*, 2013.

[16] L. Schor, P. Sommer, and R. Wattenhofer, "Towards a zero-configuration wireless sensor network architecture for smart buildings," in *Proc. of ACM BuildSys*, Berkeley, California, USA, 2009, pp. 31–36.

[17] S. Lee and M. Younis, "Optimized relay placement to federate segments in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 5, pp. 742–752, 2010.

[18] ——, "Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree," *Journal of Parallel Distributed Computing*, vol. 70, no. 5, pp. 525–536, 2010.

[19] J. Kleinberg, "Detecting a network failure," in *Proc. of IEEE SFCS*, Redondo Beach, California, USA, 2000, pp. 31–36.

[20] J. Kleinberg, M. Sandler, and A. Slivkins, "Network failure detection and graph connectivity," in *Proc. of ACM SODA*, New Orleans, Louisiana, USA, 2004, pp. 76–85.

[21] H. Ritter, R. Winter, and J. Schiller, "A partition detection system for mobile ad-hoc networks," in *Proc. of IEEE SECON*, New Orleans, Louisiana, USA, 2004, pp. 76–85.

[22] N. Shrivastava, S. Suri, and C. Tóth, "Detecting cuts in sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 2, pp. 1–25, 2008.

[23] M. Won, M. George, and R. Stoleru, "Towards robustness and energy efficiency of cut detection in wireless sensor networks," *Elsevier Ad Hoc Networks*, vol. 9, no. 3, pp. 249–264, 2011.

[24] P. Barooah, "Distributed cut detection in sensor networks," in *Proc. of IEEE CDC*, Cancun, Mexico, 2008, pp. 1097–1102.

[25] P. Barooah, H. Chenji, R. Stoleru, and T. Kalmar-Nagy, "Cut detection in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 483–490, 2012.

[26] E. Oyman and C. Ersoy, "Multiple sink network design problem in large scale wireless sensor networks," in *Proc. of IEEE ICC*, Paris, France, 2004, pp. 3663–3667.

[27] A. Das and D. Dutta, "Data acquisition in multiple-sink sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, pp. 82–85, July 2005.

[28] V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. of IEEE ICC*, Kobe, Japan, 1997, pp. 1405–1413.

[29] C.-Y. Chong and S. Kumar, "Sensor networks: evolution, opportunities, and challenges," *Proceedings of the IEEE*, vol. 91, no. 8, pp. 1247 – 1256, 2003.

[30] A. Cerpa and D. Estrin, "ASCENT: Adaptive self-configuring sensor networks topologies," *IEEE Transactions on Mobile Computing*, vol. 3, no. 3, pp. 272–285, 2004.

[31] M. Jorgic, M. Hauspie, D. Simplot-Ryl, and I. Stojmenovic, "Localized algorithms for detection of critical nodes and links for connectivity in ad hoc networks," in *Proc. of IEEE Med-Hoc-Net*, Bodrum, Turkey, 2004, pp. 360–371.

[32] M. Won, M. George, and R. Stoleru, "RE^2-CD: Robust and energy efficient cut detection in wireless sensor networks," in *Proc. of WASA*, Boston, Massachusetts, USA, 2009, pp. 80–93.

[33] R. Stoleru, J. Stankovic, and S. Son, "On composability of localization protocols for wireless sensor networks," *IEEE Network*, vol. 22, no. 4, pp. 21 –25, 2008.

[34] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, no. 6, pp. 609–616, 2001.

[35] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A characterization of ten hidden-surface algorithms," *ACM Comput. Surv.*, vol. 6, pp. 1–55, March 1974.

[36] L. Tang, Y. Sun, O. Gurewitz, and D. Johnson, "Pw-mac: An energy-efficient predictive-wakeup mac protocol for wireless sensor networks," in *Proc. of IEEE INFOCOM*, Shanghai, China, 2011, pp. 1305–1313.

[37] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proc. of ACM SenSys*, Baltimore, Maryland, USA, 2004, pp. 39–49.

[38] *CC2420 Data Sheet, http://www.chipcon.com*, 2009.

[39] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer, "Deterministic boundary recognition and topology extraction for large sensor networks," in *Proc. of ACM SODA*, Miami, Florida, USA, 2006, pp. 1000–1009.

[40] Y. Wang, J. Gao, and J. S. Mitchell, "Boundary recognition in sensor networks by topological methods," in *Proc. of ACM MobiCom*, Los Angeles, CA, USA, 2006, pp. 122–133.

[41] D. Dong, Y. Liu, and X. Liao, "Fine-grained boundary recognition in wireless ad hoc and sensor networks by topological methods," in *Proc. of ACM MobiHoc*, New Orleans Lousiana, USA, 2009, pp. 135–144.

[42] S. Funke, "Topological hole detection in wireless sensor networks and its applications," in *Proc. of ACM DIALM-POMC*, Cologne, Germany, 2005, pp. 44–53.

[43] S. P. Fekete, A. Kroeller, D. Pfisterer, S. Fischer, and C. Buschmann, "Neighborhood-Based Topology Recognition in Sensor Networks," in *Proc. of ALGOSENSORS*, Turku, Finland, 2004, pp. 34–55.

[44] S. P. Fekete, M. Kaufmann, A. Kroeller, and K. Lehmann, "A New Approach for Boundary Recognition in Geometric Sensor Networks," *eprint arXiv:cs/0508006*, pp. 135–144, 2005.

[45] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proc. of ACM MobiCom*, Boston, Massachusetts, USA, 2000, pp. 56–67.

[46] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proc. of USENIX WMCSA*, New Orleans, Lousiana, USA, 1999, pp. 90–100.

[47] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith, "S4: Small state and small stretch compact routing protocol for large static wireless networks," *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 761–774, 2010.

[48] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric ad-hoc routing: of theory and practice," in *Proc. of ACM PODC*, Boston, Massachusetts, USA, 2003, pp. 63–72.

[49] Z. Jiang, J. Ma, and W. Lou, "An information model for geographic greedy forwarding in wireless ad-hoc sensor networks," in *Proc. of IEEE INFOCOM*, Phoenix, Arizona, USA, 2008, pp. 825–833.

[50] C. Liu and J. Wu, "Destination-region-based local minimum aware geometric routing," in *Proc. of IEEE MASS*, Pisa, Italy, 2007, pp. 1–9.

[51] N. Arad and Y. Shavitt, "Minimizing recovery state in geographic ad hoc routing," *IEEE Transactions on Mobile Computing*, vol. 8, pp. 203–217, 2009.

[52] P. Li, G. Wang, J. Wu, and H.-C. Yang, "Hole reshaping routing in large-scale mobile ad-hoc networks," in *Proc. of IEEE GLOBECOM*, Honolulu, Hawaii, USA, 2009, pp. 1–6.

[53] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proc. of ACM MOBICOM*, Boston, Massachusetts, USA, 2000, pp. 120–130.

[54] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: a geographic hash table for data-centric storage," in *Proc. of WSNA*, Atlata, Georgia, USA, 2002, pp. 78–87.

[55] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing for mobile computers," in *Proc. of ACM SIGCOMM*, London, United Kingdom, 1994, pp. 234–244.

[56] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[57] Y. Gao and B. Zheng, "Continuous obstructed nearest neighbor queries in spatial databases," in *Proc. of ACM SIGMOD*, Providence, Rhode Island, USA, 2009, pp. 577–590.

[58] M. Pocchiola and G. Vegter, "Topologically sweeping visibility complexes via pseudotriangulations," *Discrete and Computational Geometry*, vol. 16, no. 4, pp. 419–453, 1996.

[59] *Wikipedia, http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes*, 2008.

[60] M. Won and R. Stoleru, "Destination-based cut detection in wireless sensor networks," in *Proc. of IEEE/IFIP EUC*, Melbourne, VIC, Australia, 2011, pp. 55–62.

[61] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "Range-free localization schemes for large scale sensor networks," in *Proc. of ACM MobiCom*, Sandiego, California, USA, 2003, pp. 81–95.

[62] J. G. Jetcheva and D. B. Johnson, "Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks," in *Proc. of ACM MobiHoc*, Pittsburgh, Pennsylvania, USA, 2001, pp. 90–100.

[63] S.-J. Lee, M. Gerla, and C.-C. Chiang, "On-demand multicast routing protocol," in *Proc. of IEEE WCNC*, New Orleans, Louisiana, USA, 1999, pp. 1298–1302.

[64] M. Mauve, H. Fussler, J. Widmer, and T. Lang, "Position-based multicast routing for mobile ad-hoc networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 53–55, 2003.

[65] S. Wu and K. Candan, "GMP: Distributed geographic multicast routing in wireless sensor networks," in *Proc. of IEEE ICDCS*, Lisboa, Portugal, 2006, pp. 49–58.

[66] S. M. Das, H. Pucha, and Y. C. Hu, "Distributed hashing for scalable multicast in wireless ad hoc networks," *IEEE Transactions on Parallel Distributed System*, vol. 19, no. 3, pp. 347–362, 2008.

[67] X. Zhang and L. Jacob, "Multicast zone routing protocol in mobile ad hoc wireless networks," in *Proc. of IEEE LCN*, Bonn, Germany, 2003, pp. 150–159.

[68] C.-C. Chiang, M. Gerla, and L. Zhang, "Forwarding group multicast protocol (FGMP) for multihop, mobile wireless networks," *Cluster Computing*, vol. 1, no. 2, pp. 187–196, 1998.

[69] S. Song, D. Kim, and B.-Y. Choi, "AGSMR: Adaptive geo-source multicast routing for wireless sensor networks," in *Proc. of WASA*, Boston, Massachusetts, USA, 2009, pp. 200–209.

[70] S. Song, B.-Y. Choi, and D. Kim, "MR.BIN: Multicast routing with branch information nodes for wireless sensor networks," in *Proc. of IEEE ICCCN*, Zurich, Switzerland, 2010, pp. 1–6.

[71] R. Sridharan, "The capacitated plant location problem," *European Journal of Operational Research*, vol. 87, no. 2, pp. 203–213, 1995.

[72] H. Pirkul, "Efficient algorithms for the capacitated concentrator location problem," *Comput. Oper. Res.*, vol. 14, no. 3, pp. 197–208, 1987.

[73] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside simple polygons," in *Proc. of SCG*, Yorktown Heights, New York, USA, 1986, pp. 1–13.

[74] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," in *Proc. of ACM SenSys*, Berkeley, California, USA, 2009, pp. 1–14.

[75] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis, "Surviving wi-fi interference in low power zigbee networks," in *Proc. of ACM Sensys*, Zurich, Switzerland, 2010, pp. 309–322.

[76] D. B. Shmoys, E. Tardos, and K. Aardal, "Approximation algorithms for facility location problems," in *Proc. of ACM STOC*, New York, USA, 1997, pp. 274–296.

[77] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," in *Proc. of ACM SODA*, San Francisco, California, USA, 1998, pp. 1–10.

[78] X. Wu, G. Chen, and S. Das, "Avoiding energy holes in wireless sensor networks with nonuniform node distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 710–720, 2008.

[79] A. Pruhit, Z. Sun, F. Mokaya, and P. Zhang, "Sensorfly: Controlled-mobile sensing platform for indoor emergency response applications," in *Proc. of ACM/IEEE IPSN*, Chicago, Illinois, USA, 2011, pp. 223–234.

[80] E. Lloyd and G. Xue, "Relay node placement in wireless sensor networks," *IEEE Transactions on Computers*, vol. 56, no. 1, pp. 134–138, 2007.

[81] X. Cheng, D.-Z. Du, L. Wang, and B. Xu, "Relay sensor placement in wireless sensor networks," *Wireless Networks*, vol. 14, no. 3, pp. 347–355, 2008.

[82] A. Kashyap, S. Khuller, and M. Shayman, "Relay placement for higher order connectivity in wireless sensor networks," in *Proc. of IEEE INFOCOM*, Barcelona, Spain, 2006, pp. 1–12.

[83] J. L. Bredin, E. D. Demaine, M. Hajiaghayi, and D. Rus, "Deploying sensor networks with guaranteed capacity and fault tolerance," in *Proc. of ACM MobiHoc*, Urbana-Champaign, Illinois, USA, 2005, pp. 309–319.

[84] W. Zhang, G. Xue, and S. Misra, "Fault-tolerant relay node placement in wireless sensor networks: Problems and algorithms," in *Proc. of IEEE INFOCOM*, Anchorage, Alaska, USA, 2007, pp. 1649–1657.

[85] Y. Hou, Y. Shi, H. Sherali, and S. Midkiff, "Prolonging sensor network lifetime with energy provisioning and relay node placement," in *Proc. of IEEE SECON*, Santa Clara, California, USA, 2005, pp. 295–304.

[86] W. Wang, V. Srinivasan, and K.-C. Chua, "Extending the lifetime of wireless sensor networks through mobile relays," *IEEE/ACM Transactions on Networking*, vol. 16, no. 5, pp. 1108–1120, 2008.

[87] F. Wang, D. Wang, and J. Liu, "Traffic-aware relay node deployment: Maximizing lifetime for data collection wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1415–1423, 2011.

[88] G.-H. Lin and G. Xue, "Steiner tree problem with minimum number of steiner points and bounded edge-length," *Information Processing Letters*, vol. 69, no. 2, pp. 53–57, 1999.

[89] S. Wang, X. Mao, S.-J. Tang, M. Li, J. Zhao, and G. Dai, "On movement-assisted connectivity restoration in wireless sensor and actor networks," *IEEE*

Transactions on Parallel and Distributed Systems, vol. 22, no. 4, pp. 687–694, 2011.

[90] H. M. Almasaeid and A. E. Kamal, "Data delivery in fragmented wireless sensor networks using mobile agents," in *Proc. of ACM MSWiM*, Chania, Crete Island, Greece, 2007, pp. 86–94.

[91] F. Senel, M. Younis, and K. Akkaya, "Bio-inspired relay node placement heuristics for repairing damaged wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 4, pp. 1835–1848, 2011.

[92] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation discussion and generalization," in *Proc. of ICGA*, Berkeley, California, USA, 1993, pp. 86–94.

[93] H. Kim and K. Shin, "Asymmetry-aware real-time distributed joint resource allocation in ieee 802.22 wrans," in *Proc. of IEEE INFOCOM*, San Diego, California, USA, 2010, pp. 1–9.

[94] F. Li, J. Luo, C. Zhang, S. Xin, and Y. He, "Unfold: Uniform fast on-line boundary detection for dynamic 3d wireless sensor networks," in *Proc. of ACM MobiHoc*, Paris, France, 2011, pp. 1–11.