The Pennsylvania State University
The Graduate School

SUPPORTING MULTI-MISSIONS IN WIRELESS SENSOR NETWORKS

A Dissertation in
Computer Science Engineering
by
Changlei Liu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2010

The dissertation of Changlei Liu was reviewed and approved* by the following:

Guohong Cao
Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Thomas F. La Porta
Distinguished Professor of Computer Science and Engineering
Director of Networking and Security Research Center

Sencun Zhu
Assistant Professor of Computer Science and Engineering

Runze Li
Professor of Statistics

Raj Acharya
Professor of Computer Science and Engineering
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

The recent advances in sensing devices, embedded computing and wireless communication technology has sparked the emergence of the wireless sensor networks. However, most of the sensor networks nowadays only target for a single mission, and may not be cost effective from the resource management point of view. Therefore, in this dissertation, we focus on the design of multi-mission sensor network which is envisioned to support multiple applications of diverse requirements with a single network.

To support multiple missions, several challenges naturally arise. The first challenge comes from the fact that the missions or the mission requirements may change over time. Therefore, the network needs to be enriched with the adaptation capability to explicitly handle the mission switch. Second, since sensor network has limited resources, e.g., energy, sensing, bandwidth, etc, these resources have to be shared by the multiple missions. An efficient resource allocation scheme, therefore, should maximize the total profit by taking account into all the mission requirements. Third, since mission-driven sensor network is usually deployed in the harsh, unattended, dynamic environment, it is important to monitor the sensor status, based on which decisions about the mission switch can be made. While more challenges can be listed here, in this dissertation, we primarily focus on these three aspects, namely, mission switch, resource allocation, and network monitoring. Each of the aspects is examined under different contexts within an integrated framework, and briefly summarized in the following.

First, we study the mission switch in a surveillance application. As mission switches, e.g., the network is commanded to last for a longer time, some sensors may have to sleep longer during each duty cycle. Then the original sensor deployment may not be able to satisfy the coverage and lifetime requirements at the same time, and the coverage may need to be traded for network lifetime. To deal with this tradeoff, we propose the concept of spatial-temporal coverage, in contrast to

the traditional area coverage. Our goal is to schedule the sensor activity to maximize the total spatial-temporal coverage during a specified network lifetime. Both centralized and distributed heuristics are proposed, with the approximation factor of the centralized algorithm proved to be $\frac{1}{2}$.

Second, we explore the resource allocation in the landmine networks and investigate how to accomplish multiple missions with the minimum resources usage. Specifically, we have studied a multi-target defense scenario, where our mission is to destroy the multiple intruding targets using the minimum cost pre-deployed landmines. Due to the NP Complexity of the problem, we have proposed a greedy algorithm and a layering algorithm, whose approximation ratios are derived.

Third, we have proposed the multi-poller based monitoring architecture for mission driven sensor networks. Compared with the single poller scheme, the multi-poller scheme significantly reduces the false alarm rate, while keeping the similar bandwidth consumption. To construct the monitoring architecture, we formulate a many-to-many poller-pollee assignment problem and present three distributed algorithms (i.e., random, deterministic, and hybrid). We have also explored the hop-by-hop aggregation opportunity between the poller and pollee, and formulate the optimal aggregation path problem. We solve this NP-hard problem by designing an opportunistic greedy algorithm, which achieves an approximation ratio of $\frac{5}{4}$. As far as we know, this is the first proved constant approximation ratio applied to the aggregation path selection schemes over the wireless sensor networks.

Table of Contents

List of Figures

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor Prof. Guohong Cao, for his consistent support and supervision. During my Ph.D study, he has spent considerable time and effort in training me as a more matured researcher. It is because of his unreserved assistance that I could go through many difficult times as a Ph.D student. Without his illuminating instruction, the completion of my Ph.D study would not be possible.

High tribute shall be paid to Prof. Tom La Porta, Prof. Sencun Zhu, and Prof. Runze Li, who are among my thesis committee. It is my great privilege to have these wonderful professors in my committee. Without their generous help and encouragement, the dissertation could not have reached the present form.

I am also greatly indebted to my fellow labmates who share with me the various resources and give me valuable comments during my Ph.D stage. It is a pure joy to work with them to overcome some difficult research problem. Special gratitude goes to the brothers and sisters in the local and abroad church. Their prayer upholds me and their love impels me. I also thank NSF (CNS-0916171) for funding this work.

My gratitude also extends to my family, who have been assisting and caring me all my life. Rather than giving, I have received too much from them. I owe a special debt of gratitude to them.

Last but not the least, I offer my sacrifice of thanksgiving to the Lord Jesus Christ. He is my creator, saver, and shepherd. His glory is the reason of my existence and my ultimate goal to do the research. Praise Him for what he has done, is doing, and will do in my life. "He restores my soul, and leads me in the path of righteousness for His name's sake." (Psalm 23:3)

Dedication

To my parents.

Chapter 1

Introduction

With the advances in micro-electro-mechanics and wireless communication, wireless sensor technology has become more and more common and permeated the various areas of our life. Although each single sensor is constrained in its capability (i.e., sensing, communication, and processing), numerous sensors can collaborate with each other by forming a network to achieve purposes that a single sensor cannot provide. Compared with other types of networks (e.g., IP network, cellula network, ad hoc network), sensor network has its unique challenges. First, unlike the infrastructure based network, the sensor nodes need to *self-organize* themselves into a multi-hop network, which places critical demand for distributed design. Second, sensor nodes are prone to failure since they usually operate in an unattended, harsh environment, and they may run out of battery. This raises the serious issue on fault tolerance. Finally, since sensors are usually untethered and powered by limited battery, energy efficiency is a crucial and challenging issue.

Most of the sensor networks envisioned so far only target for a single mission and are often designed for some particular application. Therefore, the concept of multi-mission sensor network is developed to capture the notion that multiple missions, each with its own requirement, may share common sensors to achieve different goals. From resource management point of view, it may be more cost effective for the wireless sensor network to support multiple missions instead of a single mission.

The term "mission" can be defined in several ways. A narrow definition is based on the type of applications performed. For example, in a fire disaster relief system,

the mission of the sensor network is to detect the fire event in a timely manner as well as to find a safe route for the people to depart the building. By contrast, a broader definition of mission is based on the objective of the assigned task. The objective, for example, could be the mission duration/lifetime, the degree of coverage, the preciseness of localization, or some other specific metrics.

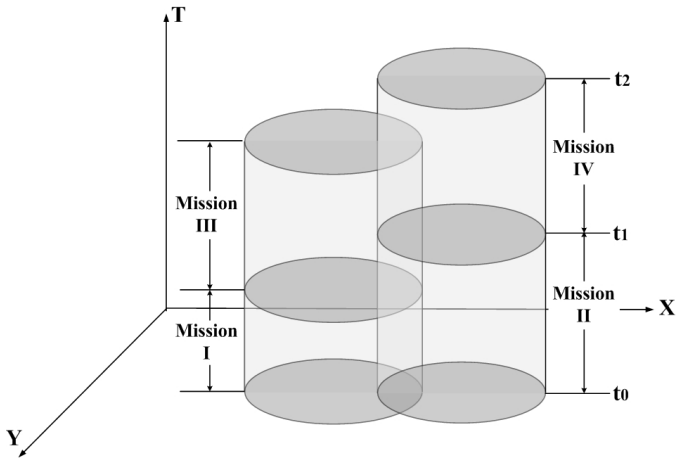


Figure 1.1. Spatial-temporal correlation among multi-missions, with each cylinder denoting a mission in the space and time domain

Depending on whether multi-missions coexist in spatial or temporal domain, they may have different relationship. Fig. 1.1 shows some possible relationship in the 3-D space with X and Y axis denoting the spatial area and T axis denoting the time. Each disk denotes the spatial area of a mission at a particular instant. As mission progresses in time, a cylinder is formed in the spatial and temporal domain. For example, mission II progresses from time t_0 to t_1, and mission IV progresses from t_1 to t_2. It can be seen that mission I and mission II correlate in both spatial and temporal domain because they overlap in both time and space. Mission II and mission IV correlate in spatial domain but not in the temporal domain.

1.1 Challenges of Multi-Mission Wireless Sensor Network

As the multi-mission sensor network is a special type of wireless sensor networks, the aforementioned challenges of sensor network also apply to the multi-mission sensor networks, i.e., distributed operation, fault tolerance, energy constraint. However, due to its mission-oriented design, it also has its own unique challenges, as summarized in the following.

The first challenge is the mission switch. In the example of fire disaster relief system, the initial mission is to detect the occurrence of the fire event. But after a fire is detected, the mission shall switch from the event detection to the safe route discovery. Multi-mission sensor network, therefore, needs to be enriched with the adaptation capability to handle the mission switch. The preplanning kind of strategy is not sufficient here. Another example is in the area surveillance, where the network may be required to last for a longer duration than its initial design. Since it may not be possible to deploy more sensors, some sensors have to sleep longer to extend the network lifetime, which may adversely affect the degree of coverage. Therefore, sometimes the coverage has to be traded for lifetime.

The second challenge is the resource allocation. Because of the limited sensor resources, multiple missions may compete for the same subset of sensors. Each mission has a demand which measures its need of sensing resources, and associates with a profit which represents how important it is. Thus, the objective of the resource allocation is to maximize the total network profit with the limited sensor resources. For example, in a sensor embedded landmine network, a single mission is to destroy one target. But with multiple intruding targets, the multi-mission is to destroy all the targets. Things become much more complicated in the multi-target defense scenario, since each landmine may contribute to the destructive effect of multiple targets, and the destruction of a single target may require the involvement of multiple landmines. It turns out to be a complex task to determine the minimum-cost subset of landmines to be activated through the distributed collaboration among the landmines.

The third challenge comes from the dynamic, unattended, energy constrained nature of the sensor operation. As sensors could fail or run out of battery any time,

the mission goal may not be met all the time. Therefore, it is important to monitor the network condition and sensor status such that we can continually evaluate the progress of mission completion. On the other hand, in a harsh, unattended environment, the monitoring architecture itself should be fault tolerant, and the monitoring overhead should not constitute a significant part of the total bandwidth consumption. Therefore, we need to consider issues such as energy efficiency, fault tolerance, distributed operation when constructing the monitoring architecture.

Fourth, besides the basic sensing and communication task, in many applications, the multi-mission sensor network is often associated with a physical process whose consequence determines how well the mission is accomplished. In such case, there is a close interaction between the sensing process and physical process, where the sensed information (e.g., environmental condition, target attribute) is fed as a parameter into the physical process, based on which the optimal action of the physical process can be decided. For example, in a sensor embedded land-mine network, the physical process refers to the landmine explosion, whose destructive effect depends on many factors such as the type/model of the target and mine, the distance between the target and mine, etc. The sensing process can obtain some of these information for the physical process for the online strategy planning. However, in most cases, to figure out the interaction between the physical process and sensing process and find the corresponding optimal strategy is not easy.

1.2 Focus of This Dissertation

In this dissertation, we have addressed the challenges as elaborated in Section 1, and investigate the issues that are critical to the normal operation of multi-mission sensor network. Specifically, we focus on three important aspects, namely, mission switch, resource allocation, network monitoring, and explore these issues in different application contexts. In the following, we give a brief overview of our problems and contributions, within the context of three case-studies.

1.2.1 Spatial-Temporal Coverage Optimization with Mission Switch

Mission-driven sensor networks usually have special lifetime requirements and the requirement could change on demand. For example in a surveillance application, during the course of its operation, the network may be required to last for a longer time than its initial design. Since it may not be possible to deploy additional sensors, the current sensor density may not be able to satisfy the new lifetime and coverage requirement at the same time. Then, the coverage may need to be traded for network lifetime.

The coverage issue in sensor networks has been studied extensively [1, 2, 3, 4, 5, 6], where scheduling algorithms are proposed to maximize the network lifetime while maintaining some predefined coverage degree. But different from existing works, we study how to schedule sensor nodes to maximize coverage under the constraint of network lifetime. Our work, which is the reverse of the existing formulation, is thus complementary to the current literature.

Specifically, we study how to schedule sensors to maximize their coverage during a specified network lifetime. Unlike sensor deployment, where the goal is to maximize the *spatial* coverage, our objective is to maximize the *spatial-temporal* coverage by scheduling sensors' activity after they have been deployed. Since the optimization problem is NP-hard, we first present a centralized heuristics whose approximation factor is proved to be $\frac{1}{2}$, and then propose a distributed parallel optimization protocol (POP). In POP, nodes optimize their schedules on their own but converge to local optimality without conflict with one another.

1.2.2 Multi-Target Defense in Landmine Networks

With the integration of sensor technology, researchers have developed the modern landmine, so called *smart-mine*, where the landmine is endowed with the capability of sensing, computation, and communication. While most existing work mainly focuses on the single smart-mine design, we investigate the networking opportunities that the sensor technology can bring to the next generation landmine and develop the framework of multi-target defense in landmine networks.

In the single target defense, the mission is to destroy a single target. But in

the multi-mission (i.e., multi-target) defense scenario, the goal is to destroy all the intruding target. Resource allocation in this case is much more complicated since the explosion of a single landmine can affect multiple targets. To address this issue, we utilize the recent result in the impact engineering [7, 8, 9, 10], which states that the destructive effect of a mine explosion on a target depends on many factors such as the type/model of the target and mine, the distance between the target and mine, etc. Based on these inter-disciplinary result, we examine the interaction between the sensing process and physical explosion process, and try to fill the gap between the mine industry and impact engineering via sensor technology.

Specifically, we have formulated a minimum-cost mine selection problem, and transform it using a novel bucket-tub model. Due to the problem complexity, we have proposed two classes of approximation algorithms, i.e., greedy algorithm and layering algorithm, whose approximation ratios are derived. To facilitate different mines to negotiate with each other in a distributed manner, we also present a local greedy algorithm, which produces the same solution set as the global greedy algorithm.

1.2.3 Network Monitoring for Mission Driven Sensor Networks

For multi-mission sensor networks deployed in the unattended, harsh environment, the knowledge of sensor statuses such as liveness, node density and residue energy, is critical for maintaining the normal operation of the network. Based on these information, timely evaluations can be made regarding the ongoing missions, and prompt actions can be taken in response to the status change. As a result, a sound monitoring architecture is fundamental to the multi-mission sensor network, and crucial during the process such as resource allocation, mission switch, etc.

In this dissertation, we have made multifold contributions regarding network monitoring. First, we propose a multi-poller based architecture to monitor the sensor status. Combined with a round robin based scheduling scheme, the multi-poller based scheme can reduce the false alarm rate significantly while keeping similar bandwidth consumption as the single poller scheme. Second, we propose three distributed algorithms to construct the multi-poller monitoring architecture, namely,

random, deterministic, and hybrid algorithm. Third, we explore the hop-by-hop aggregation opportunity during the transmission of status report and propose an opportunistic greedy algorithm. Despite the NP complexity of the problem, the algorithm proposed can achieve an approximation ratio of $\frac{5}{4}$. As far as we know, this is the first proved constant approximation ratio applied to the aggregation path selection schemes over the wireless sensor networks.

1.2.4 Organization

The remainder of the dissertation is organized as follows. Chapter 2 studies the mission switch in a surveillance application. Chapter 3 explores the resource allocation issue in the embedded landmine network. Chapter 4 focuses on the network monitoring for mission driven sensor network. Chapter 5 finally concludes and presents the future work.

Chapter 2

Spatial-Temporal Coverage Optimization With Mission Switch

2.1 Introduction

In wireless sensor networks, there is a tradeoff between network lifetime and sensor coverage. To achieve a better coverage, more sensors have to be active at the same time, then more energy would be consumed and the network lifetime is reduced. On the other hand, if more sensors are put into sleep to extend the network lifetime, the coverage will be adversely affected. The tradeoff between network lifetime and sensor coverage cannot be simply solved at the deployment stage, because it is hard to predict the network lifetime requirement, which depends on the application and may change as the mission changes. For example, in a surveillance application, the initial mission is to monitor the battle field for 6 hours. As the battle goes on, the commander finds that the battle may have to last for 10 hours. Then, the mission of the sensor network is changed, which requires the network to last for 10 hours. Since it may not be possible to deploy more sensors, some sensors have to sleep longer during each duty cycle to extend the network lifetime. As a result, sensor coverage needs to be traded for network lifetime.

The coverage issue in sensor networks has been studied extensively [1, 2, 3, 4, 5, 6], where scheduling algorithms are proposed to maximize the network life-

time while maintaining some predefined coverage degree[1]. However, if the same coverage degree is maintained all the time, the lifetime requirements may not be satisfied as network condition and mission change. For example, the sensor density may drop over time, and the coverage requirement may vary according to the application's demand. Different from existing works, we study how to schedule sensor nodes to maximize coverage under the constraint of network lifetime. This reverse formulation is especially useful when the number of nodes is not enough to maintain the required coverage degree for a specified time period, as shown in the above example.

In this chapter, we aim to resolve the conflict between the static status of sensor deployment and the dynamic nature of mission requirements. As mission dynamically changes, the lifetime and coverage requirement may not be satisfied at the same time. Then the coverage needs to be traded for the network lifetime. Our work is thus complementary to the existing work, which can apply when the sensor density is sufficient to sustain both the lifetime and coverage requirement. To fulfill this goal, we have to consider coverage in both spatial and temporal domain. In particular, we define a new *spatial-temporal coverage* metric, in contrast to the traditional area coverage. The spatial-temporal coverage of each small area is defined as the product of the area size and the length of period during which the area is covered. Then our objective becomes how to schedule the sensor's on-period to maximize the global spatial-temporal coverage, calculated as the sum of individual spatial-temporal coverage over all the areas. This new formulation arises naturally from the mission critical applications with the network lifetime constraint and differentiates itself from most existing works which only consider the spatial domain.

Consider a surveillance example shown in Fig. 2.1(a). Three sensors monitor a rectangular area, where the overlap between sensor 1 and senor 2 is four units, and the overlap between sensor 2 and sensor 3 is one unit. Suppose the network is required to provide full coverage and operate for 10 hours. Since the battery lifetime is 6 hours for each sensor, the coverage and lifetime requirement cannot be satisfied. Most existing coverage-oriented algorithms in such a case would activate the three sensors simultaneously for 6 hours, without considering the network

[1]The area is k-covered if every point in the area is covered by k sensors at the same time.

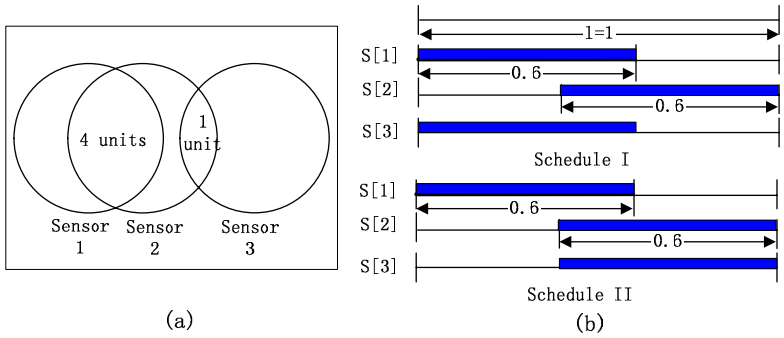


Figure 2.1. A surveillance example with three sensors.

lifetime constraint. However, we trade the coverage for lifetime by dividing the mission duration of 10 hours into ten cycles, and within each cycle, each sensor is active for $6/10 = 0.6$ hour. Then the scheduling issue becomes how to place the 0.6 hour in each cycle so that the spatial-temporal coverage of the overlapping regions are maximized. Fig. 2.1(b) shows two solutions. From the spatial coverage point of view, the two schedules make no difference because each sensor covers the same area for the same length of time in different schedules. But from the spatial-temporal perspective, schedule I is better because the four-unit overlapping region is covered for a full cycle in schedule I but for only 0.6 cycle in schedule II. The spatial-temporal coverage of the overlapping regions in schedule I is $4\times1+1\times1 = 5$ and is $4 \times 0.6 + 1 \times 1 = 3.4$ in schedule II.

The above example shows that different schedules may result in different coverage redundancy. Although the optimal solution for this example can be easily found (schedule I is actually the optimal solution), in a complex network setting where thousands of sensors are arbitrarily deployed, we need a systematic way to address the problem. Our contribution in this chapter can be summarized as follows. First, we formalize the sensor scheduling problem in the spatial and temporal dimension, with the objective to maximize the spatial-temporal coverage with network lifetime constraint. We further prove that it is equivalent to minimize the coverage redundancy under certain conditions. Second, we prove the problem is NP-hard and propose a centralized greedy algorithm with an approximation factor of $\frac{1}{2}$. Third, we propose a distributed heuristic, POP (parallel optimization protocol), where nodes optimize their schedules on their own but converge to local

optimality without conflict with one another.

The rest of the chapter is organized as follows. Section II presents the problem formulation. Section III shows the problem complexity and proposes a centralized approximation algorithm. Section IV presents the distributed heuristic. Performance evaluations are done in Section V. Section VI gives related work and Section VII concludes the chapter.

2.2 Problem Formulation

When the sensor density is not sufficient to satisfy both the lifetime and coverage requirements, the coverage has to be traded for lifetime. In such a case, the sensors have to make their best efforts to provide the coverage while meeting the lifetime constraint. To achieve this, we divide the network lifetime L into cycles and turn on each sensor within each cycle for a period proportional to its battery life. We further designate that the same schedule repeats in each cycle, such that the sleep schedule can be implemented, e.g., using the Power Saving Mode (PSM) of 802.11. Then the purpose of the scheduling is to place the on-periods within each cycle, such that the total spatial-temporal coverage can be maximized. We formalize it as a $maxCov$ problem in subsection 2.2.1, and then transform it to a $minRed$ problem in subsection 2.2.2 whose objective is to minimize the overall coverage redundancy.

2.2.1 Maximize The Spatial-Temporal Coverage

Problem MaxCov*: Given a unit-disk graph $G(N, E)$ with n nodes, the battery life of each sensor $B_i, i = 1...n$, and a mission lifetime of L, where $B_i \leq L$, we want to calculate an "on" schedule per cycle for each sensor such that the overall spatial-temporal coverage is maximized.*

To quantify the overall spatial-temporal coverage (or coverage, for short), we first define *elementary region* as the minimum region formed by the intersection of a number of sensing disks. Notice that different points belonging to the same elementary region are covered for the same length of time. Therefore, the spatial-temporal coverage of each elementary region can be calculated as the product of

its area size and the length of time during which the region is covered by *at least* one sensor. Note that for each elementary region, the area size is fixed after the sensors are deployed, but the coverage time varies depending on the different sensor schedule.

Further define the *k-redundant elementary region* as the elementary region formed by the intersection of k sensors, where $k \geq 2$. For example in Fig. 2.2(a), there are seven elementary regions and four of them are redundant elementary regions, whose area sizes are $a_1 = a_2 = a_3 = a_4 = 1$. a_1, a_2, a_3 refer to the 2-redundant elementary region and a_4 refers to the 3-redundant elementary region. The *non-redundant elementary regions* are covered by only a single sensor, such as those elementary regions other than a_1, a_2, a_3, a_4 in Fig. 2.2. Since the coverage time of the non-redundant elementary region is the same as the "on" period of that sensor, its spatial-temporal coverage is constant irrespective of the sensors' schedule. Therefore, to devise a better "on" schedule per cycle for each sensor, we only need to focus on the redundant elementary regions to maximize their total spatial-temporal coverage.

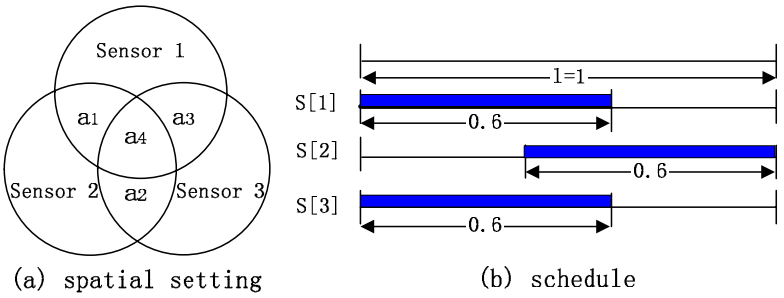


Figure 2.2. An example to illustrate how to calculate the redundancy for k-redundant elementary regions.

Given the schedule in Fig. 2.2(b), the spatial-temporal coverage of the 2-redundant elementary region can be calculated similar to that of Fig. 2.1. For example, the spatial-temporal coverage for a_1 is the product of the area of a_1 and the time during which a_1 is covered by either s_1 or s_2, or both, i.e., $1 \times 1 = 1$. Similarly, the coverage for a_2 and a_3 are $1 \times 1 = 1$ and $1 \times 0.6 = 0.6$, respectively. For the 3-redundant elementary region a_4, we need to find out the length of time during which it is covered by at least one of the three sensors, which is 1 time unit

in Fig. 2.2. Therefore, the total spatial-temporal coverage over all the redundant elementary regions in Fig. 2.2 is $1 + 1 + 0.6 + 1 = 3.6$.

In general, we can formalize the problem in the form of mathematical programming. Before giving the formulation, we first define some notations that will be used throughout the chapter.

- N, E: the node set and the edge set of the graph
- $s_i, N(i)$: sensor i and the neighbor set of sensor i, i.e., $N(i)$=$\{j \mid$ sensor s_j is the neighbor of sensor $s_i\}$
- I: the index set of the redundant elementary regions, i.e., I=$\{i \mid$ region i is a redundant elementary region$\}$
- $I(m)$: the index set of the redundant elementary regions within the sensing disk of sensor m
- $A(i)$: the index set of sensors whose intersection of sensing disks forms the ith redundant elementary region, with $|A(i)|$ denoting its cardinality.
- a_i: the area size of the ith redundant elementary region
- T_i: the time during which the ith redundant elementary region is covered by at least one sensor
- L, l: L is the network lifetime, and l is the length of the cycle, so there are $\frac{L}{l}$ number of cycles, assuming $\frac{L}{l}$ is an integer. l should not be too small so that the switch overhead between the on/off state is negligible.
- B_i, b_i: B_i is the battery life of s_i, and b_i is the length of s_i's on-period per cycle. Since there are $\frac{L}{l}$ number of cycles, we have $b_i \times \frac{L}{l} = B_i$.
- $s_i.start$, $s_i.end$ is the start and end of s_i's on-period respectively, where $s_i.end - s_i.start = b_i$
- C: the total coverage of all the redundant elementary regions.

With these notations, we can calculate the total coverage as the sum of the product of area size and coverage time, over all the redundant elementary regions. Thus our objective function and constraints are:

$$\text{Max } C = \sum_{i \in I} a_i \times T_i \tag{2.1}$$

$$\text{ST: } 0 \leq s_i.end \leq l, i \in N \tag{2.2}$$

$$b_i = B_i \times \frac{l}{L}, i \in N \tag{2.3}$$

$$s_i.end - s_i.start = b_i, i \in N \tag{2.4}$$

The purpose of the above optimization is to determine the variables $s_i.end$ (or $s_i.start$) to maximize the spatial-temporal coverage subjecting to the constraints 2.2, 2.3, 2.4. In the objective function, a_i is the area size of the ith redundant elementary region, which could be in arbitrary shape, and T_i is the time during which the ith redundant elementary region is covered by *at least* one sensor, which depends on the schedules of all the neighboring sensors. Constraint (2.2) shows that the on-period may fall on the boundary of the cycle, so $s_i.end$ ranges from 0 to l. Constraint (2.3) requires that each node's on-period within a cycle is proportional to its battery life. Constraint (2.4) establishes the relationship between $s_i.end, s_i.start$ and the length of its on-period.

2.2.2 Minimize The Coverage Redundancy

In this section, we consider the coverage maximization problem from another perspective and propose a new formulation. In the previous section, the objective is to maximize the total spatial-temporal coverage, which desires the total coverage time of each redundant elementary region to be as large as possible. Alternatively, we can achieve the same goal by minimizing the schedule overlap of the sensors that monitor the same redundant elementary region. Toward this direction, we propose another metric, *spatial-temporal coverage redundancy*, whose value depends on the area size, the overlapping "on" periods, and the the number of sensors that monitor the area in each period. With the concept of spatial-temporal coverage redundancy (or *coverage redundancy* for short), the problem of "maximizing cov-

erage under the constraint of network lifetime" becomes "minimizing the coverage redundancy under the constraint of network lifetime" (called *minRed* problem). We can prove that the two objectives are equivalent under certain condition.

We first use Fig. 2.2 as an example to illustrate how to calculate the coverage redundancy of the redundant elementary regions. For instance, the coverage redundancy for a_1 is the area of a_1 times the schedule overlap of s_1 and s_2, i.e., $1 \times 0.2 = 0.2$. Similarly, the redundancy for a_2 and a_3 are 0.2 and 0.6, respectively. The coverage redundancy of a_4 consists of two parts, i.e., the part of time when a_4 is covered by *exactly* two sensors, and the part of time when it is covered by *exactly* three sensors. Intuitively, the two parts should have different contribution to the coverage redundancy, because more resources will be wasted as more sensors overlap in time. To reflect this, we assign different *weight* to different periods during which the same region is monitored by different number of sensors. In particular, a_4 is *solely* monitored by s_1 and s_2 for 0 unit of time, by s_1 and s_3 for 0.4 unit of time, by s_2 and s_3 for 0 unit of time, all of which are assigned weight 1. On the other hand, a_4 is solely monitored by s_1, s_2 and s_3 for 0.2 unit of time, and it is assigned weight 2. Then, the total coverage redundancy is the weighted sum of the product of area size and time overlap over all the redundant elementary regions. For example, in Fig. 2.2, the total redundancy is $(0.2) + (0.2) + (0.6) + (1 \times 1 \times 0 + 1 \times 1 \times 0.4 + 1 \times 1 \times 0 + 3 \times 1 \times 0.2) = 2$.

To study the problem from the perspective of coverage redundancy, we need to define more notations in addition to those used in the previous section.

- $t_i^j(S), S \subseteq A(i)$: the time during which the ith redundant elementary region is covered by *exactly* j sensors that include all the sensors in S. S can be empty set $\emptyset$.
- $\overline{a_i a_j}$: the area overlap (i.e., the size of the overlapping area) between s_i and s_j
- $\overline{s_i s_j}$: the time overlap (i.e., the length of the overlapping on-period) between s_i and s_j
- R: the coverage redundancy of the whole network

With these notations, the problem *minRed* is formulated as follows, with the objective to minimize the total coverage redundancy.

$$\text{Min } R = \sum_{i \in I} \sum_{j=2}^{|A(i)|} w(j) \times a_i \times t_i^j(\emptyset) \tag{2.5}$$

$$\text{ST: } 0 \leq s_i.end \leq l, i \in N \tag{2.6}$$

$$b_i = B_i \times \frac{l}{L}, i \in N \tag{2.7}$$

$$s_i.end - s_i.start = b_i, i \in N \tag{2.8}$$

In the objective function, the total coverage redundancy is calculated as the weighted sum of the product of area overlap and time overlap, first over the possible coverage degrees within each region and then over all the redundant elementary regions. Specifically, $t_i^j(\emptyset)$ is the time during which the ith redundant elementary region is covered by *exactly* j sensors, which depends on the schedules of the j sensors. Intuitively, a larger j contributes more redundancy, so the weight factor $w(j)$ is used in the objective function to reflect this, which should be a monotonically increasing function of j. In the following, we can prove that this new objective function (Eqn. 2.5) is equivalent to the previous objective function (Eqn. 3.1) with a properly set weight factor.

Theorem 1. *With the same graph, network lifetime requirement, and battery constraints, the objective to maximize the total spatial-temporal coverage is equivalent to minimize the total spatial-temporal coverage redundancy when setting the weight factor to be* $w(j) = j - 1$.

Proof. We first rewrite the objective of total coverage, i.e., Eqn. 3.1, by decomposing the coverage time T_i into a multitude of sub-periods according to the different coverage degree.

$$C = \sum_{i \in I} a_i \times T_i = \sum_{i \in I} \sum_{j=1}^{|A(i)|} a_i \times t_i^j(\emptyset) \tag{2.9}$$

Then we add the two objective functions together, i.e. Eqns. 2.5 & 2.9. Since $w(j) = j - 1$, we have

$$C + R = \sum_{i \in I} \sum_{j=1}^{|A(i)|} j \times a_i \times t_i^j(\emptyset) \tag{2.10}$$

Note that a set of sensors $A(i)$ monitor the elementary region a_i, and each sensor $k \in A(i)$ is active for b_k period of time per cycle. So each active period b_k can be decomposed into sub-periods according to the different coverage degrees of region a_i, i.e., $b_k = \sum_{j=1}^{|A(i)|} t_i^j(s_k)$. Therefore, if we sum up $b_k, k = 1, ..., |A(i)|$, each $t_i^j(\emptyset)$ would be counted exactly j times. Then we have

$$\sum_{k \in A(i)} b_k = \sum_{j=1}^{|A(i)|} j \times t_i^j(\emptyset) \tag{2.11}$$

Combining Eqn. 2.10 and Eqn. 2.11, we have $C + R = \sum_{i \in I} a_i \sum_{k \in A(i)} b_k$, which is a constant value. This implies that maximizing the total coverage C is equivalent to minimizing the total coverage redundancy R.

□

2.3 Centralized Algorithm Design

In this section, we first prove that the *maxCov* problem is NP-hard and then propose a centralized greedy algorithm whose approximation factor is $\frac{1}{2}$.

Theorem 2. *The problem maxCov is NP-hard. (The proof is given in Appendix A.)*

Despite the problem complexity, we propose the greedy heuristics for the problem *maxCov*, with the pseudo code listed in Algorithm I. Assume the area size a_i is known for each elementary region i. The algorithm is executed in iterations. During each iteration, it calculates the maximal additional spatial-temporal coverage ΔC_i that each sensor s_i can provide based on the existing schedule and picks the sensor with the largest ΔC_i. Each time a new sensor is picked, the existing schedule within a cycle is augmented by placing the active period of the selected sensor in the optimal position of the cycle (i.e., where the sensor can provide the

maximum coverage). This process will be repeated iteration after iteration, until no more sensors can be picked to increase the total spatial-temporal coverage.

Define $\Delta C_i = \sum_{j \in I(i)} a_j \times \Delta T_j$, as the sum of the product of area size a_j and additional coverage time ΔT_j over all the redundant regions covered by sensor s_i. The process of deriving an optimal schedule is to calculate the maximum ΔC_i for each sensor s_i by choosing the right $s_i.start$. Given an existing schedule among sensor i's neighbors whose on-periods are determined by its $s_i.start$ (or $s_i.end$), it turns out that it takes just $O(|N(i)|)$ computational effort to derive $max\{\Delta C_i\}$, where $|N(i)|$ is the number of the neighbors of sensor s_i. The linear computational efficiency results from the observation that the maximum ΔC_i is achieved when $s_i.start$ is at one of the end points of the active periods of s_i's neighbors. This is because when $s_i.start$ moves between two neighboring end points the value of ΔC_i changes (i.e., increases or decreases) linearly. Therefore, we first identify the end points, i.e., $s_j.start$ and $s_j.end$, for each neighbor j of s_i, and then select $s_i.start$ among those end points where ΔC_i can be maximized.

Algorithm 1 Greedy Algorithm

Input: graph $G(N, E)$, mission lifetime L, the battery life $B_i, i = 1...n$, the area of each redundant elementary region $a_i, i \in I$

Output: a subset of sensors S_{gre} with their schedules determined, the overall spatial-temporal coverage C_{gre}

Procedure:

1: $S_0 = N, S_{gre} = \emptyset$
2: **while** $S_0 \neq \emptyset$ and $\Delta C_k \neq 0$ **do**
3: **for** each $i \in S_0$ **do**
4: identify all the end points of the active periods among the neighbors whose schedule has been determined. Use S_e to denote the collection of these end points. $S_e = S_e + \{0\}$
5: calculate $\Delta C_i = \max_{s_i.start \in S_e} \sum_{j=2}^{|A(i)|} a_j \times \Delta T_j$, and the corresponding $s_i.start$ that achieves the optimal schedule
6: **end for**
7: find the sensor s_k that can provide the maximum coverage, i.e., $\Delta C_k = \max_{i \in S_0} \Delta C_i$
8: **if** $\Delta C_k \neq 0$ **then**
9: $S_0 = S_0 - \{s_k\}$, $S_{gre} = S_{gre} + \{s_k\}$, $C_{gre} += \Delta C_k$.
10: update the current schedule by adding the on-period of s_k
11: **end if**
12: **end while**
13: Output S_{gre}, and C_{gre}.

To analyze the performance of the algorithm, we cannot simply borrow the

techniques from the traditional coverage model in the spatial domain. Due to the unique challenges of coverage issue in the spatial-temporal domain, the algorithm not only needs to select sensors but also needs to determine their corresponding schedules. In addition, the scheduling decision needs to be made in a continuous space while there are infinite possibilities to place the on-period in a cycle. As a result, some traditional modeling approach such as the set cover model [12] cannot be simply applied. Therefore, we need new techniques to analyze the algorithm complexity.

Theorem 3. *Algorithm 1 achieves an approximation factor of $\frac{1}{2}$ for the maxCov problem.*

Proof. Suppose there are total n sensors picked in the greedy algorithm. Sensor s_i^g is picked in the ith iteration, which provides ΔC_i^g additional coverage with its active period placed at the optimal position of the cycle. Use the vector $V_i^g = (v_{i1}^g, v_{i2}^g ..., v_{i|I|}^g)$ to denote the coverage provided by s_i^g, with each component corresponding to each redundant elementary region. Since there are total $|I|$ redundant elementary regions, there are $|I|$ components for each vector. Each v_{ij}^g denotes the time periods during which the region j is covered by s_i^g. For example, $v_{ij}^g = \{[1,3],[5,6]\}$ means region j is covered by sensor s_i^g during the periods [1,3] and [5,6]. It can be seen that each period in v_{ij}^g is determined by a start time and an end time, with no overlap between the different periods. If region j is not covered by sensor s_i^g, set $v_{ij}^g = [0,0]$. Further use $|v_{ij}^g|$ to denote the total length of time during which region j is covered by s_i^g. For example, if $v_{ij}^g = \{[1,3],[5,6]\}$, then $|v_{ij}^g| = 3$.

Define the norm of the vector V_i^g in the form of a weighted sum, i.e., $|V_i^g| = \sum_{j=1}^{|I|} a_j \times |v_{ij}^g|$, where the weight is the area size a_j. For example, if $V_i^g = (\{[1,3],[5,6]\}, [0,0], \{[2,4]\})$, then $|V_i^g| = 3a_1 + 2a_3$. It can be seen that $|v_{ij}^g|$ denotes the total spatial-temporal coverage provided by s_i^g.

Further suppose V_i^g and V_j^g are two different vectors, then define the vector addition/substraction as follows:

$$V_i^g + V_j^g = (v_{i1} \cup v_{j1}, v_{i2} \cup v_{j2}, ..., v_{i|I|} \cup v_{j|I|}) \tag{2.12}$$

$$V_i^g - V_j^g = (v_{i1} - v_{i1} \cap v_{j1}, ..., v_{i|I|} - v_{i|I|} \cap v_{j|I|} \tag{2.13}$$

Thus, the total coverage vector generated by the greedy algorithm is $V^g = \sum_{i=1}^{n} V_i^g$, and the total coverage is

$$C_{gre} = \sum_{i=1}^{n} \Delta C_i^g = |V^g| = |\sum_{i=1}^{n} V_i^g| \tag{2.14}$$

Suppose there are total m iterations in the optimal algorithm, and during each iteration one sensor is picked. Then we rearrange the order of sensors selected in the optimal algorithm, such that if a sensor is also chosen by the greedy algorithm, it is chosen in the same iteration in both the algorithms. Since the schedule of each sensor remains intact, changing the order of sensors selected will not affect the outcome of the optimal algorithm. Similar to the greedy algorithm, we also define the notations $s_i^o, V_i^o, |V_i^o|, v_{ij}^o, |v_{ij}^o|, \Delta C_i^o, V^o, C_{opt}$ for the optimal algorithm.

In the ith iteration, the greedy algorithm picks sensor s_i^g and enhances the total coverage by ΔC_i^g. Thus we have $\Delta C_i^g = |V_i^g - \sum_{j=1}^{i-1} V_j^g|$. According to the definition of the greedy algorithm, during each iteration it picks the sensor that can provide the maximum additional coverage, thus $|V_i^g - \sum_{j=1}^{i-1} V_j^g| \geq |V_i^o - \sum_{j=1}^{i-1} V_j^g|$, where V_i^o is the coverage vector of the sensor s_i^o picked in the same iteration by the optimal algorithm. In other words, the greedy algorithm picks s_i^g instead of s_i^o because s_i^g can provide more additional coverage than s_i^o. On the other hand, $V^g \supseteq \sum_{j=1}^{i} V_j^g$ for $\forall i = 1...n$. Therefore we have

$$\Delta C_i^g \geq |V_i^o - \sum_{j=1}^{i-1} V_j^g| \geq |V_i^o - V^g|, \quad \forall i = 1...n \tag{2.15}$$

Since n sensors are picked in the greedy algorithm and m sensors are picked in the optimal algorithm, to make the above equation also hold when $m < n$, let $V_i^o = ([0,0], ..., [0,0])$ for $\forall i \in (m, n]$. Then adding all the inequalities denoted by Eqn. 2.15 gives

$$\sum_{i=1}^{n} \Delta C_i^g = C_{gre} \geq \sum_{i=1}^{n} |V_i^o - V^g| \tag{2.16}$$

Note that the spatial-temporal coverage of each sensor is denoted by a vector and each component of a vector is a collection of time periods. Then based on the vector analysis and the addition/substraction defined in Eqns. 2.12 & 2.13, it is not hard to see

$$\sum_{i=1}^{n} |V_i^o - V^g| \geq |\sum_{i=1}^{n} (V_i^o - V^g)| \geq |\sum_{i=1}^{n} V_i^o| - |V^g| \tag{2.17}$$

As $C_{opt} = |\sum_{i=1}^{n} V_i^o|$ and $C_{gre} = |V^g|$, combining Eqn. 2.16 and Eqn. 2.17 gives

$$C_{gre} \geq C_{opt} - C_{gre} \tag{2.18}$$

The above relationship shows that the approximation ratio of Algorithm 1 is $\frac{1}{2}$.

□

The centralized algorithm has theoretical favor, as it gives a constant factor performance bound. However, it is not practical as it is difficult to enumerate and compute each a_i and let each node have such global knowledge. Thus in the next section, we propose the distributed heuristics based on the local coverage redundancy.

2.4 Distributed Algorithm Design

From the above discussion, we know that in a complex network of large scale, it is computational infeasible to enumerate each elementary area a_i and list each period $t_i^j(\emptyset)$ during which area i is covered by exactly j sensors. Therefore, in the distributed design, we focus on the pairwise sensors and let each node minimizes its own *local coverage redundancy*, defined as the sum of pairwise redundancy with its neighbors. Although the global optimal is computationally infeasible to achieve, we can design a class of algorithms in which each node is able to achieve the local optimal if certain conditions can be satisfied. The basic idea is to let each node first generate a random schedule independently. Then, each node adjusts its schedule individually to minimize the local coverage redundancy with its neighbors, until

everyone converges to its local optimality. The seemingly simple idea has several challenges.

- How to do the local optimization? Does it have polynomial time algorithms to achieve the local optimal?
- If each sensor adjusts the schedule individually, is the algorithm able to converge?
- How to eliminate conflicts caused by simultaneous adjustments of the neighboring nodes?

The following subsections will address these challenges one by one.

2.4.1 Local Optimization

Without loss of generality, suppose sensor s_0 has $|N(0)|$ neighbors. The local optimization problem at s_0 can be formalized as follows:

Given the area overlap between s_0 and its neighbors (i.e., $\overline{a_0a_i}, i \in N(0)$), the individual schedule of its neighbors, we want to decide s_0's own schedule, such that the local sum of the coverage redundancy with its neighbors $R[0] = \sum_{i \in N(0)} \overline{a_0a_i} \times \overline{s_0s_i}$ can be minimized.

It can be seen that the local coverage redundancy (i.e., $R[i] = \sum_{j \in N(i)} \overline{a_ia_j} \times \overline{s_is_j}$) is much easier to calculate than the global redundancy (Eqn. 2.5). Suppose two sensors s_i and s_j, whose sensing radius are r and the distance between them is d. Their area overlap and time overlap can be simply computed by:

$$\begin{cases} \overline{a_ia_j} = 2r^2 \arccos(\frac{d}{2r}) - d\sqrt{r^2 - \frac{d^2}{4}}, \text{ if } d < 2r; \\ \overline{a_ia_j} = 0, \text{ if } d \geq 2r; \\ \overline{s_0s_j} = \max\{\min(s[i].end, s[j].end) \\ \quad - \max(s[i].start, s[j].start), 0\} \end{cases} \tag{2.19}$$

Each node has its own reference cycle. The cycles at different nodes are not required to be synchronized. Each node only needs to know the relative position of its neighbor's on-period. This can be easily achieved via exchange of hello packets with its neighbors.

Note that s_i's schedule per cycle is solely determined by the start of its on-period $s_i.start$ and the end of its on-period $s_i.end$, where $s_i.end - s_i.start = b_i$. Then the objective of local optimization at s_0 is to decide $s_0.end$ (or $s_0.start$) within its own reference cycle such that $R[0]$ is minimized. However, because $s_0.end$ could be any value between 0 and l, it is not realistic to enumerate all the possibilities. In our solution, we only focus on some crucial points, which could jointly determine the redundancy $R[0]$ at every possible value of $s_0.end$.

Line Traversal Algorithm: s_0 first selects its own reference cycle and places each neighbor's schedule (i.e., on-period) in the cycle. Then s_0's on-period traverses from the left of the cycle (i.e., $s_0.end = 0$) to the right of the cycle (i.e., $s_0.end = l$), during which the local redundancy $R[0]$ over the whole range can be recorded. In the end, the points corresponding to the minimum $R[0]$ are identified and selected as s_0's schedule. For example, in Fig. 2.3, s_0 has s_1, s_2, s_3 as neighbors, whose schedules are given. For ease of illustration, assume $b_i = b_j, i, j = 0...3$. However, the algorithm is not limited in the homogeneous case but allows the heterogeneous battery states at different nodes. Fig. 2.3 shows the relationship between $R[0]$ and $s_0.end$ by executing the line traversal algorithm.

The algorithm is based on the observation that the coverage redundancy $R[0]$ increases/decreases linearly as $s_0.end$ traverses from left to right, and the slope k shifts only at some *crucial points*, which corresponds to the following four cases.

- Case I: the end of s_0's on-period enters the start of s_i's on-period, i.e., $s_0.end = s_i.start$, then the slope increases by $\overline{a_0 a_i}$, i.e., $k = k + \overline{a_0 a_i}$.
- Case II: the end of s_0's on-period leaves the end of s_i's on-period, i.e., $s_0.end = s_i.end$, then the slope decreases by $\overline{a_0 a_i}$, i.e., $k = k - \overline{a_0 a_i}$.
- Case III: the start of s_0's on-period enters the start of s_i's on-period, i.e., $s_0.start = s_i.start$, then the slope decreases by $\overline{a_0 a_i}$, i.e., $k = k - \overline{a_0 a_i}$.
- Case IV: the start of s_0's on-period leaves the end of s_i's on-period, i.e., $s_0.start = s_i.end$, then the slope increases by $\overline{a_0 a_i}$, i.e., $k = k + \overline{a_0 a_i}$.

We use Case I as an example to illustrate why the slope k is updated in such a way. When the end of s_0's on-period enters the start of s_i's on-period (such as

at point P_3 in Fig. 2.3), the time overlap between s_0 and s_i starts to increase as $s_0.end$ traverses to the right. Thus, the slope of $R[0]$ will increase by $\overline{a_0a_i}$, i.e., $k = k + \overline{a_0a_i}$, where $\overline{a_0a_i}$ is the area overlap between s_0 and s_i.

Since the on-period may fall on the boundary of the cycle, we let $s_0.end$ traverse from 0 to l, and count the coverage redundancy $R[0]$ over the range $[-b_0, l]$. Because each crucial point corresponds to one of the above four cases and s_0 has $N(0)$ neighbors, there are $4 * N(0)$ crucial points, denoted as $P_j, j = 1...4N[0]$. Adding two points $s_0.end = 0$ and $s_0.end = l$, denoted as P_0, $P_{4N[0]+1}$, there are total $4N[0] + 2$ crucial points. Since the slope k only changes at the crucial points, the relationship between $R[0]$ and $s_0.end$ can be presented by a piecewise curve, as seen from Fig. 2.3. Note that some crucial points may overlap. For example in Fig. 2.3, P_1 and P_2 overlap because $s_0.end$ enters b_2 and $s_0.start$ leaves b_1 at the same time; P_4 and P_5 overlap because $s_0.end$ leaves b_2 and $s_0.start$ enters b_2 at the same time. We use $R[0][j]$ to denote the coverage redundancy at P_j and use $k[j]$ to denote the slope between points P_j and P_{j+1}, then we have the following recursive relationship.

$$\begin{cases} R[0][j+1] = R[0][j] + k[j](P_{j+1} - P_j) \\ R[0][0] = \frac{1}{2} \sum\limits_{i \in N(0)} \overline{a_0a_i} \times \overline{s_0s_i}. \end{cases} \tag{2.20}$$

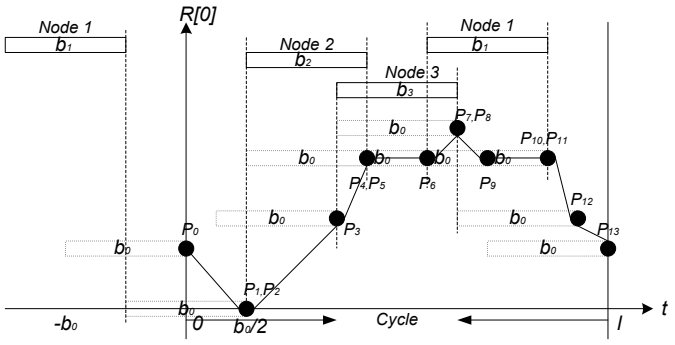


Figure 2.3. An example to illustrate the line traverse algorithm. The piecewise curve depicts the relationship between $R[0]$ and $s[0].end$. There are total 14 crucial points, at which the slope k of the curve changes.

The above recursive relationship shows that the value of $R[0]$ at the current crucial point can be determined by its value at the previous point and the slope in

Algorithm 2 Line Traverse Algorithm

Input: Graph G, the schedules of s_0's neighbors $s_i, i \in N(0)$;
Output: Node 0's schedule, $s_0.end$;
Procedure:

enumerate the set of crucial points in terms of the value of $s_0.end$, $\chi = \{x | x \in \sum_{i \in N(0)} \cup \{s_i.start, s_i.end, s_i.start + b_0, s_i.end + b_0\}, x \in [0, l]\}$
sort the crucial set χ in increasing order
$R[0][0] = \frac{1}{2} \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}, P_0 = 0, k[0] = \sum_{i \in A} \overline{a_0 a_i} - \sum_{i \in B} \overline{a_0 a_i}$
/*the traversal starts at $s_0.end = 0$. A, B denote the set of neighbors whose on-period spans across 0 and $-b_0$ (equivalently $l - b_0$), respectively.*/
$j = 1$ /*the index of the crucial point*/
while $\chi \neq \emptyset$ **do**
  $R[0][j] = R[0][j-1] + k[j-1](P_j - P_{j-1})$
  if $s_0.end == s_i.start$ **then**
    $k[j] = k[j-1] + \overline{a_0 a_i}$
  end if
  if $s_0.end == s_i.start + b_0$ **then**
    $k[j] = k[j-1] - \overline{a_0 a_i}$
  end if
  if $s_0.end == s_i.end$ **then**
    $k[j] = k[j-1] - \overline{a_0 a_i}$
  end if
  if $s_0.end = s_i.end + b_0$ **then**
    $k[j] = k[j-1] + \overline{a_0 a_i}$
  end if
  $\chi = \chi - \{P_j\}$, $j = j + 1$
end while
$R[0][i] = R[0][0]$ /*the traversal ends at $s_0.end = l$ */
connect the neighboring points piecewise by lines
select the optimal $s_0.end$ with the minimum $R[0]$

between. For example in Fig. 2.3, initially at P_0, $s_0.end = 0$, $R[0] = \frac{1}{2}\overline{a_0 a_i} \times \overline{s_0 s_1}$ and $k[0] = -\overline{a_0 a_1}$. As $s_0.end$ moves right, $R[0]$ decreases linearly until it hits P_1. At P_1, $s_0.end$ enters b_2 and $s_0.start$ leaves b_2, so the slope k increases by $(\overline{a_0 a_1} + \overline{a_0 a_2})$. Then $R[0]$ begins to increase linearly with $k[1] = -\overline{a_0 a_1} + \overline{a_0 a_1} + \overline{a_0 a_2} = \overline{a_0 a_2}$ until it reaches P_2. Similarly, as $s_0.end$ continues to move right, the value of k varies at the subsequent crucial points. When $s_0.end$ arrives at P_{13}, the value of $R[0]$ over the whole range of $[0, l]$ can be obtained, after which the same cycle is repeated.

With all the values of $R[0]$ at different points, the minimum $R[0]$ and its corresponding $s_0.end$ can be identified. In Fig. 2.3, at P_1, P_2, where $s_0.end = b_0/2$, $R[0] = 0$ is the minimum. In this case, $s_0.end = b_0/2$ is the only optimal schedule. However, in other cases, it is possible that the minimum $R[0]$ is achieved

at multiple $s_0.end$. Then, we can break the tie arbitrarily and pick any optimal $s_0.end$.

The complexity of Line Traverse Algorithm is only $O(d)$, where d is the node degree. Suppose a node has d neighbors, then there are at most $4 * d + 2$ crucial points to be examined, and at each point only linear algebraic operation is performed.

2.4.2 Convergence Property

In our distributed algorithm, each node locally optimizes its own schedule as long as its schedule does not remain locally optimal. Since altering a node's schedule can affect the redundancy of its neighbors, the schedule adjustment at different nodes may conflict with each other and the adjustment process may never end. For example, if two neighboring nodes adjust their own schedules at the same time, they may not be aware that their neighbor's schedule has been changed and cannot achieve local optimality. Next, we provide guidelines to guarantee that each node can converge to its local optimality.

Theorem 4. *Given a graph G and arbitrary schedules, a distributed algorithm will terminate in a finite number of steps and after termination each node's schedule will converge to the local optimality, if*

- *no neighboring nodes optimize their schedules at the same time*
- *each node's local adjustment continues as long as its local objective can be improved for at least a predefined threshold δ.*

Proof. First, we want to show that if the local objective improves (i.e., the redundancy at a node decreases), the global pair-wise redundancy will improve as well (i.e., the total redundancy decreases). Without loss of generality, suppose a node s_0 optimizes its local schedule. Because $R[0] = \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}$, we have $\Delta R[0] = \sum_{i \in N(0)} \Delta R[i]$. Further because the sum of the local redundancy is $R_s = \sum_{i=1}^{n} R[i]$, we have $\Delta R_s = \Delta R[0] + \sum_{i \in N(0)} \Delta R[i]$. It is followed that $\Delta R_s = 2\Delta R[0]$, which shows that each time the local objective at s_0 is improved

by $\Delta R[0]$, R_s is improved by $2\Delta R[0]$. Second, it can be shown that the global pairwise redundancy is bounded, i.e., $R_s \leq \sum_{i=1}^{n} \sum_{j \in N(i)} \overline{a_i a_j} \times \min\{b_i, b_j\}$, where a_i, a_j and b_i, b_j are constant values. Therefore, the algorithm could terminate after a finite number of steps if the stated conditions are satisfied, where the threshold δ could be set arbitrarily small to approximate the local optimal point.

□

2.4.3 Distributed Protocol Design

Theorem 4 tells us that for a distributed protocol to converge, all three conditions have to be satisfied. Before presenting our distributed protocol, let's see two simple algorithms.

- *Random Algorithm*: each node generates a random schedule individually.
- *Serial Optimization Algorithm*: each node first generates a random schedule, based on which the schedule is locally optimized one by one. This serial optimization process is repeated until no improvement can be made beyond the predefined threshold δ.

Each of the above algorithms has its pros and cons. The random algorithm is simple, distributed and has no message complexity. It can serve as a baseline for comparison. The serial optimization algorithm uses the Line Traversal Algorithm as a functional module to ensure that every node can achieve its local optimality, but it is centralized. In addition, for the serial algorithm to converge, many iterations are needed until no improvement can be made. Therefore, the serial algorithm takes a long time to terminate.

To retain the merit of the serial algorithm and remedy its weakness, we propose a parallel optimization protocol (POP). The basic idea of POP is to let many nodes locally optimize their schedules (using Line Traversal Algorithm) in parallel, so that it can converge much faster than the serial algorithm. According to Theorem 4, a set of non-neighboring nodes can adjust their own schedules simultaneously without causing any conflict. From the algorithmic point of view, to search for such set of non-neighboring nodes is equivalent to find an *Independent*

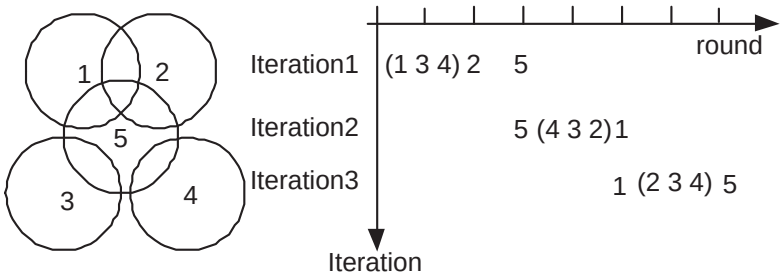


Figure 2.4. An example to illustrate the POP protocol

Set [13], which is defined as a subset of nodes among which there is no edge between any two nodes. The set is a *maximal independent set (MIS)* if no more edges can be added to generate a bigger independent set. To find the MIS, each node independently determines whether it belongs to the set by comparing its weight with its neighbors. If it has the *best* weight in the neighborhood, it elects itself as belonging to the set, and then no other neighbors can be chosen. In general the algorithm can be denoted as $MIS(weight, criteria)$, where the weight can be *id, degree, energy*, etc., and the criteria can be either *smallest* or *largest*. The criteria is used to interpret the meaning of best weight, i.e., the smallest or the largest.

Algorithm II lists the pseudo code of the POP protocol which can be implemented in a distributed manner. For clarity of presentation, we first introduce the protocol in a centralized manner, and then give guidance to its distributed operation. Initially, all nodes are unlabeled. Then, each node individually determines whether it belongs to the MIS by comparing its weight with the neighbors. The labeled nodes locally optimize their schedules, after which the MIS algorithm will continue to run among the remaining unlabeled nodes. We term the time a *round* if during this period a MIS is found and local optimization is executed in parallel at the nodes of the MIS. Several rounds comprise an *iteration* during which the coalition of the MIS elected can have all the nodes labeled. The MIS algorithm continues to run round after round and iteration after iteration until no improvements can be made to any node's schedule.

At the end of an iteration, all nodes' labels are removed and a new iteration starts with the criteria reversed, i.e., "smallest" becomes "largest" and vice versa. Therefore, the iterations alternate between the increasing and decreasing order of

weight in executing the MIS algorithm. The criteria is reversed to facilitate the distributed operation, so that the nodes belonging to the MIS in the last round of previous iteration can start a new iteration.

Algorithm 3 parallel optimization algorithm

Input: a graph $G(N, E)$
Output: the local optimal schedule of each node, i.e., $s_i.end, i = 1...n$
Procedure:

each node generates a random schedule independently
discover neighbors and exchange the schedule with each other
initialize $improve = Threshold$, $criteria = smallest$, $weight = id$
while $improve \geq Threshold$ **do**
    unlabel all the nodes /*start a new iteration*/
    while there are still nodes unlabeled **do**
        /*start a new round*/
        run distributed algorithm $MIS(weight, criteria)$
        run local optimization algorithm (i.e., line traversal algorithm) for each node of MIS, record $improve$
    end while
    if $criteria == smallest$ **then**
        $criteria = largest$
    else
        $criteria = smallest$
    end if
end while

Fig. 2.4 shows an example. In the first round, after $MIS(id, smallest)$ is executed, nodes s_1, s_3, s_4 which have the smallest id among its neighbors, are selected to form a MIS and optimize their schedules simultaneously without conflict. Then, the MIS algorithm is executed for two more rounds among the remaining nodes, during which the MIS obtained in the second and third round consists of s_2 and s_5 respectively. So far all the nodes are labeled, so the first iteration ends. After that, nodes are unlabeled again and the second iteration starts. The algorithm $MIS(id, largest)$ is executed with the criteria reversed, so that s_5 (with the largest id among its neighbors) can initiate the second iteration. Similar to the first iteration, three rounds are needed in both the second and the third iteration. Note that the last round of the previous iteration coincides with the first round of the current iteration because their respective MIS is the same and there is no need to optimize the schedule of the same MIS twice. Overall, 7 rounds are needed for nodes to adjust their schedules in three iterations. This is much faster than the serial algorithm which needs $5 \times 3 = 15$ rounds.

It is straightforward to make Algorithm II distributed. Since both the MIS algorithm and the local optimization algorithm are distributed, the issue here is to let each node know when to elect itself to the MIS, when to start a new iteration with the reversed criteria, and when to terminate. To achieve this, we define a control packet in the format of $msg(id, criteria, schedule)$. After a node elects itself as belonging to the MIS and adjusts its schedule, it broadcasts $msg(id, criteria, schedule)$ to its neighbors. If the criteria is the "smallest" (or "largest"), the neighbors with the larger (or smaller) id will have the sender labeled, and check the sender's schedule to see whether it has changed. After a MIS is elected, all the nodes in the MIS will be labeled by their neighbors. Therefore, at least one unlabeled node's id will become the smallest (or largest) among the remaining unlabeled neighbors and thus eligible to adjust its own schedule.

To start a new iteration, the criteria needs to be reversed. If a elected node finds itself to be the last node among its neighbors to be elected, it will realize that it is his responsibility to reverse the criteria, and start a new iteration by broadcasting an updated message. When other nodes receive the message with the reversed criteria for the first time, they will realize that a new iteration starts, so the labels of their neighbors are reset. A timer is set to control the termination of the algorithm at each node. If the node cannot improve its schedule beyond the predefined threshold δ after a few more iterations, it will exist and start using the calculated schedule.

The message complexity of the POP protocol is $O(n)$, which grows linearly with the number of nodes. This is because each node in each iteration broadcasts two messages: one is to exchange the id, *criteria* and *schedule* in the beginning, while the other is to announce its labeled status after being selected to the MIS. Therefore, the message complexity of each node is $O(2T)$, where T is the number of iterations required for the protocol to terminate. According to our experiments in Section 2.5, T is a small constant with the typical parameter setting, e.g., $T \leq 5$ when $\delta = 1, n \leq 500$ and battery/network lifetime ratio is $\frac{1}{5}$.

2.4.4 Discussions and Future Work

In this chapter, we assume the disk sensing model is used where the sensing range is modeled by a disk and a point is covered if and only if it falls within the sensing disk of one of the sensors. While the disk model provides valuable high-level guidelines, it may not accurately reflect the performance in reality. Recently, some researchers start to investigate the impact of link irregularity and the corresponding non-disk model on the performance of the sensor networks [14], [15], [16, 17]. For example, the work in [16] employ a empirical approach to estimate the sensing range. A probability model is used in [17] to depict the coverage property of the sensor network where the coverage probability of a point depends on the distance from the monitoring sensors.

To adapt the POP protocol to the non-disk model, we can leave the big framework intact but change the method to calculate the local coverage redundancy. The algorithm still executes in iterations, but during each iteration each node calculates the pair-wise coverage redundancy based on the specific non-disk model. Taking the probability model as an example, the local coverage redundancy of node s_0 can be calculated by $R[0] = \sum_{i \in N(0)} \int_{\theta=0}^{2\pi} \int_{\rho=0}^{w} P(\rho, \theta) \times \overline{s_0 s_i}$, as compared with $R[0] = \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}$ in the disk model. The new calculation is based on the polar coordinate system, with the middle point of the line connecting the pair-wise neighboring sensors as the pole. In particular, $P(\rho, \theta)$ is the coverage probability calculated based on the specific model, $w = \frac{\sqrt{4r^2 - d^2 \sin\theta^2} - d\cos\theta}{2}$ and $\overline{s_0 s_i}$ follows Eqn. 2.19. In general, extension of the coverage property to the non-disk model is still an open issue in many situations. We leave the complete design and evaluation to the future work.

Another issue worth of further investigation is the connectivity property of the sensor network. Although in this chapter we consider network lifetime as a constraint and connectivity is not our focus, achieving continuous connectivity is still valuable for the data delivery. It has been proved that when the communication range is at least twice the sensing range, the full coverage implies the connectivity of the sensor network [4]. However, in our work we study the scenario where the sensors may not be sufficient enough to sustain both coverage and lifetime, so sometimes coverage has to be traded for lifetime, resulting in the partial coverage.

As far as we know, the condition under which the connectivity can be achieved in the partially covered sensor network is still an open issue. Although we did not solve it in this work, we point out this is an interesting issue for the future research and have proposed a remedy solution in our previous work [18]. In [18], we design a new set of routing protocols for the data delivery over the intermittently connected network. In an intermittently connected network, the network may not be physically connected at all instants, but the data can still be delivered to the destination in a store-and-forward fashion.

2.5 Performance Evaluations

In this section, we evaluate the performance of the proposed POP protocol. In the simulation, n sensors are randomly deployed in a 10×10 square area, with n varying from 100 to 500. The sensing range is 1 unless otherwise specified. We specifically examine the scenario where the coverage and lifetime requirement cannot be satisfied at the same time. For example, when $n = 500$ and the battery/network lifetime ratio is $\frac{2}{5}$, the full coverage cannot be maintained throughout the network lifetime using the algorithm in [4]. Both homogeneous and heterogeneous battery states are considered. In the homogeneous case, every node has the same battery/network lifetime ratio ν, but in heterogeneous case ν_i is a random variable uniformly distributed in $[\nu/2, 3\nu/2]$ with ν as the average ratio. The experiments are done over a customized C++ simulator.

Three schemes are evaluated, namely, random, serial, and POP, in terms of coverage redundancy, convergence time, and event detection probability. As the global coverage/coverage redundancy is infeasible to compute, we use the sum of local coverage redundancy as an approximation. The randomized event is considered whose location of occurrence is uniformly distributed in time and space, and whose length of occurrence e is normalized as the event/cycle ratio, i.e. $\frac{e}{l}$. The event detection probability is calculated by simulating 1000 randomized events.

To compare with the existing schemes, we implement an extended version of the Coverage Configuration Protocol (CCP), which is shown to outperform other schemes in most of the scenarios [4]. While the objective of the original CCP is to select the minimum number of sensors to provide the full coverage, we extended

it to a continuously operational case where the sensor node may die of limited battery. After a sensor dies, each sleeping sensor needs to decide whether it should be activated to remedy the coverage hole based on the eligibility rule in [4]. We evaluate CCP in terms of coverage redundancy and network lifetime. The network lifetime is defined as the period during which half of the nodes fail.

2.5.1 Determine The Optimization Threshold δ

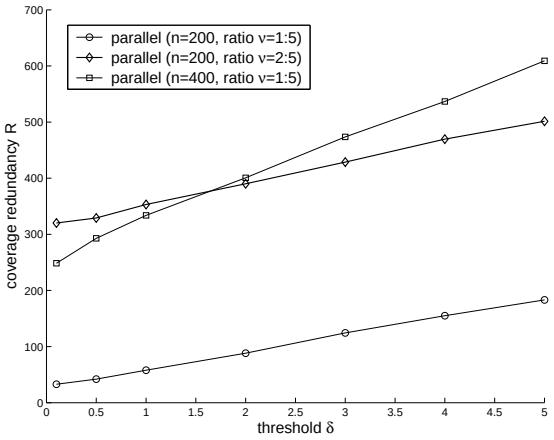


Figure 2.5. Relationship between the coverage redundancy and δ (homogeneous)

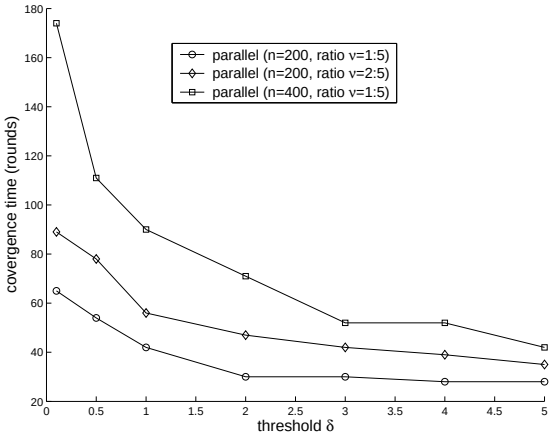


Figure 2.6. Relationship between the convergence time and δ (homogeneous)

δ is the threshold of improvement made at each step. It determines how accu-

rate the algorithm can approach the local optimality and how fast the algorithm can converge. From Figs. 2.5, 2.6, it can be seen that δ affects the coverage redundancy and the convergence time in different ways. As δ increases, the redundancy will rise but the convergence time goes down. In other words, the objectives of redundancy and convergence time conflict with each other from the perspective of δ. To make the coverage redundancy better, a smaller δ should be used, but to improve the convergence time, a larger δ should be employed. To balance coverage redundancy and convergence time, we set δ to be 1 in the following experiments.

2.5.2 Comparing POP With Other Schemes

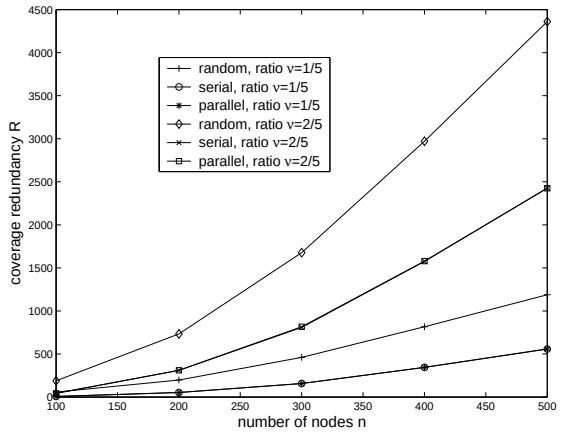


Figure 2.7. Comparison of coverage redundancy (homogeneous)

Figs. 2.7, 2.8 show that the number of nodes n and the on-period/network lifetime ratio ν affect the system performance in a similar way. Both coverage redundancy and convergence time increase as more sensors are deployed or as larger on-periods are used. In terms of redundancy, serial and POP have similar performance, and both outperform the random algorithm substantially. The improvement gradually decreases as the number of nodes increases. For instance, the performance improvement is over 100% when $n = 200$ but reduces to about 80% when $n = 400$. This is because as more sensors are deployed, it is more likely that the random algorithm can produce a relatively good schedule. In terms of convergence time, POP is much faster than the serial algorithm because parallel

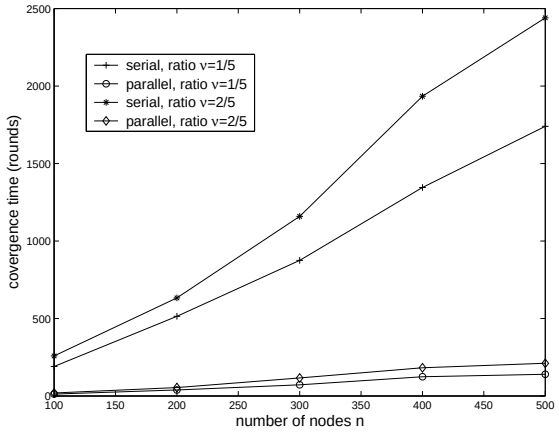


Figure 2.8. Comparison of convergence time (homogeneous, $\nu = \frac{1}{5}$)

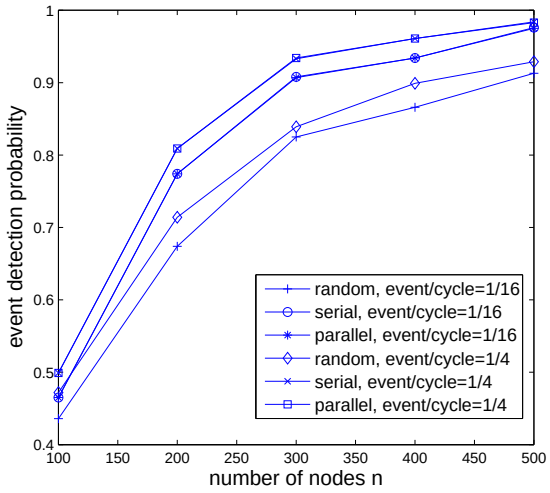


Figure 2.9. Comparison of event detection probability (homogeneous)

optimizations can take place at the same time. As shown in Fig. 2.8, irrespective of the number of nodes, the convergence time of POP is only $\frac{1}{10}$ of the serial algorithm.

Figs. 2.9, 2.10 compare the different schemes in terms of event detection probability. In Fig. 2.9 the X-axis corresponds to the number of nodes, and in Fig. 2.10 the X-axis corresponds to the event/cycle ratio. It can be observed that for all the schemes the detection probability increases as the number of nodes increases. The

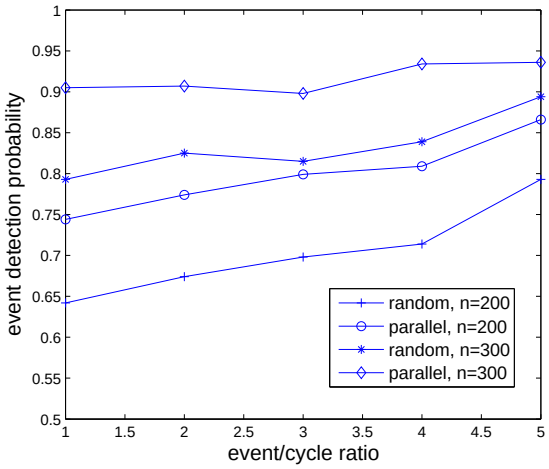


Figure 2.10. Comparison of event detection probability (homogeneous)

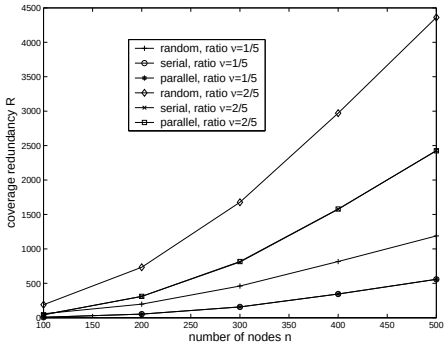


Figure 2.11. Comparison of coverage redundancy (heterogeneous)

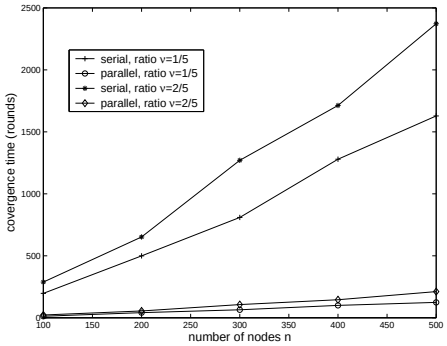


Figure 2.12. Comparison of convergence time (heterogeneous)

initial detection probability is below 50% when $n = 100$ but gradually approaches 1 as n increases to 500. Another observation is that the detection probability also increases as the length of event increases. This can be understood since the longer the event persists the easier it should be detected. Among the different schemes, the serial algorithm and POP has almost the same performance, both of which outperform the random algorithm. The improvement is about 15% in terms of event detection probability.

Figs. 2.7, 2.8 study the performance of homogeneous cases. The same trend exists in the heterogeneous case as shown in Figs. 2.11, 2.12.

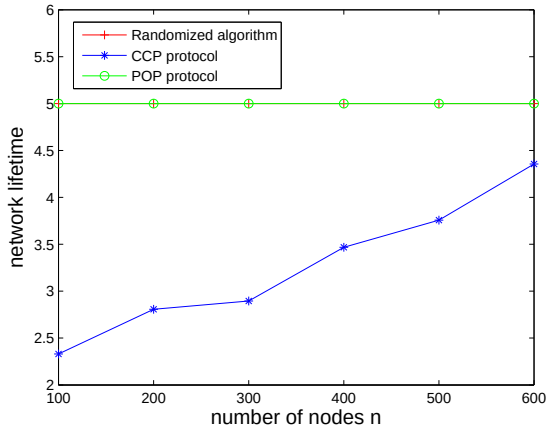


Figure 2.13. Comparison of network lifetime with CCP (heterogeneous, $\nu = \frac{2}{5}$)

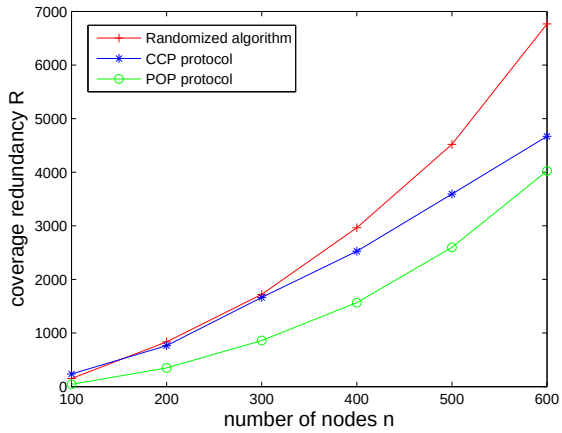


Figure 2.14. Comparison of coverage redundancy with CCP (heterogeneous, $\nu = \frac{2}{5}$)

Figs. 2.13, 2.14 compare the proposed schemes with CCP. In Fig. 2.13, it is shown that CCP is not able to meet the lifetime requirement in the scenarios simulated. However, as more sensors are deployed, it can support a longer network lifetime. For example, when $n = 600$ it maintains almost 90% of the required lifetime with the full coverage. By contrast, the randomized algorithm and POP divide network lifetime into cycles and within each cycle the locations of their "on" periods are different. Therefore, both the algorithms bear the lifetime constraint in mind and can satisfy the lifetime requirement regardless of the node density, as seen in Fig. 2.13. In Fig. 2.14, it is shown that CCP's coverage redundancy is

consistently larger than that of POP. Compared with the randomized algorithm, the performance of POP is close to the randomized algorithm when the number of sensors is small, but substantially improves as more sensors are deployed. This is because when sensor node density becomes larger, the network lifetime will increase (as shown in Fig. 2.13), then the coverage redundancy will reduce as a result.

2.6 Related Work

The sensor coverage problem has been extensively studied in the literature. Depending on the subject covered, most existing works can be classified into area coverage, point coverage, and barrier coverage [2]. In terms of area coverage, many works focus on how to select the minimum number of sensors to preserve the coverage degree (e.g., 1-degree or k-degree) [3, 4, 5, 6, 19, 20, 21], but they provide no network lifetime guarantee. In [22], a centralized scheduling algorithm is proposed to sequentially activate the sensor cover and guarantees a $O(logn)$ factor of the maximum network lifetime, where n is the total number of nodes. Further in [23], a distributed scheduling algorithm is proposed which achieves a $O(logn * lognB)$ performance factor, where B is the upper bound of the initial battery. Besides coverage requirement, the connectivity property also attracts lots of research attention. For example, when the coverage requirement can be satisfied, the conditions to achieve the communication connectivity have been derived in [4, 24]. When the coverage requirement cannot be satisfied all the time, e.g., in presence of partial coverage, there are routing protocols proposed in [18] to ensure the delivery of the data to the sink in the store-and-forward fashion over the intermittent link.

The model of point coverage is studied in [25, 1], with the objective to maximize the network lifetime when only a given set of targets needs to be covered. The model is NP-hard in general, so some greedy heuristics are proposed based on the linear programming relaxation [25]. But with the assumption that one sensor covers only one target a time, the optimal schedule can be derived based on the technique of matrix decomposition [1]. In the special case, each target to be covered is the sensor node itself, e.g., in the scenario of network monitoring. In [26, 27], distributed algorithms are proposed to construct the monitoring architecture for

sensor networks where sensor nodes can monitor each other within a predefined communication range.

The concept of barrier coverage is first proposed by [28], with the objective to minimize the probability of undetected penetration through the barrier. In [29], a global algorithm is proposed to determine whether a region is k-barrier covered. Although it has been proved that given a sensor deployment, sensors cannot locally determine whether the deployment provides global barrier coverage, a distributed algorithm is proposed based on the concept of local barrier coverage [30] assuming the intruders move along restricted crossing paths in rectangular areas. Later, the restriction on crossing paths is removed in [31, 32] where the barrier construction algorithm is proposed when the sensors are deployed according to a poison point process [31] or along a line [32].

There also exist other coverage models for specific applications. For example, trap coverage [33] is used to bound the diameter of the coverage holes; surface model [34] is proposed for the coverage of complex surface in the 3-D space; double mobility model [35] is used to cover the sea surfaces; alpha coverage model [36] is designed for the vehicular internet access.

While all of the above works treat the lifetime as objective, we consider the network lifetime as a constraint and aim to schedule each sensor's on-period to minimize the total spatial-temporal coverage redundancy. This reverse formulation is especially useful for mission-driven sensor networks, where the network lifetime may have higher priority over coverage and the pre-deployed resources may not meet the changing mission requirements all the time. Thus our work is complementary to the existing works, which can apply when the sensor node density is sufficient to provide the preferred coverage degree for a specified length of time.

There are other application-driven scheduling algorithms, e.g., for minimum latency routing [37, 38, 39], target tracking [40, 41, 42, 43], event detection [44], and throughput optimization [45]. All these works support only a single mission and do not treat network lifetime as the objective or constraint. By contrast, our objective is to maximize the spatial-temporal coverage with the network lifetime as the constraint. Our model is different from the traditional maximum coverage problem in the spatial domain, which is known to have a $(1 - 1/e)$-approximation bound [12]. This is because in the spatial-temporal domain, we not only need to select

sensors but also need to determine their corresponding schedules in a continuous cycle. Therefore, the $(1 - 1/e)$ ratio cannot be applied here. By contrast, we model the spatial-temporal coverage as a vector and propose a $\frac{1}{2}$-approximation algorithm for our problem.

2.7 Conclusions

As mission-driven sensor networks usually have stringent lifetime requirement, sometimes coverage has to be traded for network lifetime. In this chapter, we studied how to schedule sensor active time to maximize the spatial-temporal coverage while meeting the lifetime constraint. While the optimization of the global objective is NP-hard, we have proposed both centralized and distributed algorithms with low complexity. It was proved that the centralized algorithm has an approximation ratio of $\frac{1}{2}$, and the distributed parallel optimization protocol (POP) can ensure each node to converge to local optimality without conflict with each other. The computational complexity of POP is only $O(d)$ per node, where d is the maximum node degree, and its message complexity is $O(n)$, which is linear with the number of nodes. Theoretical and simulation results showed that POP substantially outperforms other schemes in terms of coverage redundancy, convergence time, network lifetime and event detection probability.

Chapter 3

Multi-Target Defense in Landmine Networks

3.1 Introduction

Traditional landmines are commonly triggered by the pressure of the moving target (e.g., tank, vehicle, personnel) that steps on it. However, there are cases where the ground surface is not suitable to bury the mine or the mine needs to be triggered by the target within a certain distance. In these cases, the pressure-triggered mine can no longer be used or bring optimal performance. In addition, the traditional landmines lack of self-destruction capability and may linger as a threat for a long time, causing the postwar disposal issue. Therefore, many ongoing efforts have focused on developing the off-route mines that could be triggered by means such as sound, magnetism and vibration, so that the mine could detonate even when it is not touched. As far as we know, the work in [46] is the only published effort to achieve this goal by integrating the latest sensor technology [47] into the landmine design (so called smart-mine). In agreement with [46], we believe that the marriage with sensor technology could bring new opportunities and even revolutionize the entire mine industry. While [46] mainly focuses on the single sensor-enabled mine design, in this chapter we take one step further and investigate the networking opportunities that the sensor technology can bring to the next generation landmine.

Research in the impact engineering [7, 8, 9, 10] shows that the destructive effect

of a mine explosion on a target depends on many factors such as the type/model of the target and mine, the distance between the target and mine, etc. In this regard, the sensor technology fills the gap between the mine industry and impact engineering by providing information crucial for decision making. With the embedded sensor, a mine is able to detect, classify the target, or even measure the distance from the target. By forming a smart-mine network, different mines can further exchange status information with each other, thus reaching agreement on a globally efficient strategy. Fig. 3.1(a) shows a small smart-mine network of 2 targets and 5 smart-mines, where the link indicates whether the target is within the explosion range of the mine. Based on the sensed distance information, each mine can estimate the blast impact on its neighboring target. Then different mines could exchange the information with one another and collaboratively decide who need to be triggered. For example, if mine 1 is close enough to disable both targets, the other mines do not have to be triggered; otherwise, more mines need to participate.

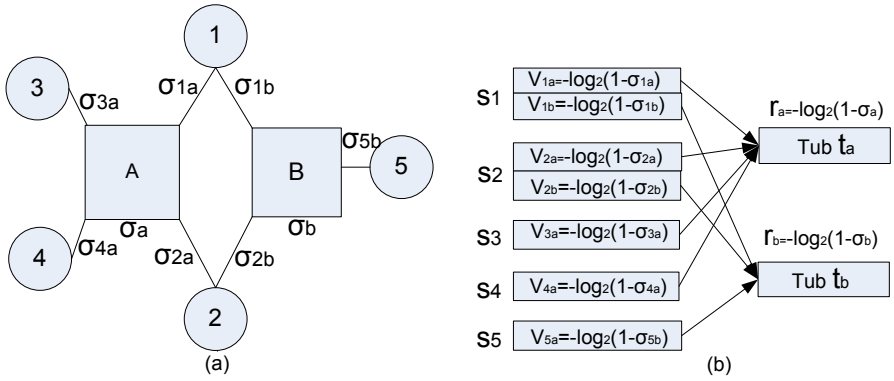


Figure 3.1. (a) A small smart-mine network, with the square denoting the targets, and the circle denoting the smart-mine. (b) the corresponding bucket-tub model, with the bucket set denoting the mine and the empty tub denoting the target.

In this chapter, we formulate the detonable mine selection problem based on two observations. First, as a single mine may not be powerful enough to destroy a single target, multiple mines may be used. Second, if several targets are within the explosion scope of a mine, one explosion could destroy multiple targets at the same time. Therefore, we need to consider the effects of multiple explosions on one target as well as the effect of single explosion on multiple targets. Our objective is

to minimize the total cost of mines (i.e., the number of mines if each mine has the same cost) subject to the constraint that all the targets should be destroyed with a predefined probability. For example in Fig. 3.1(a), the objective is to select a minimum number of mines to destroy the two targets with a given probability. To resolve this problem, there are several challenges. For example, how to quantify the destructive effect of the mine explosion on a target, how to set the criteria based on which different mines are picked, and how to find a way to let different smart-mines to negotiate with each other.

To address these challenges, we employ the results in impact engineering and calculate the collaborative effect of multiple explosions. Due to the complexity of the formulation, we transform the problem using a novel bucket-tub model and prove that it is NP-hard. Based on the new problem formulation, we propose two classes of approximation algorithms, i.e., greedy algorithm and layering algorithm. It is proved that the layering algorithm can achieve an approximation ratio of $\alpha \cdot f$, where $\alpha \geq 1$ is the tunable relaxation factor and f is the maximum number of mines that a target is associated with. In other words, the cost of the layering algorithm is bounded by $\alpha \cdot f$ times the minimum cost of the optimal solution. The greedy algorithm is shown to have an approximation ratio of $\sum_j R_j$, where R_j is the coefficient in the related integer program. To facilitate different mines to negotiate with each other in a distributed manner, we also present a local greedy algorithm, which produces the same solution set as the global greedy algorithm. By using parallel operations, the algorithm execution time can be significantly reduced.

The rest of the chapter is organized as follows. Section II overviews the related work. Section III formulates the problem and transforms it using a novel bucket-tub model. Section IV proposes the greedy algorithm and layering algorithm, and proves their approximation ratios. Section V shows how to implement the greedy algorithm in a distributed manner. Performance evaluations are done in Section VI and Section VII concludes the chapter.

3.2 Related Work

The recent advance in wireless sensor networks has brought new challenges and horizons to many fields. In [46, 48], there are some ongoing efforts to integrate the sensor into the military weapon system, with the purpose to improve the military performance and reduce the civilian casualties. In [48], a sensor network-based mobile countersniper system is introduced. Their purpose is to use soldier-wearable networked sensors to localize the shooter and classify the weapons. In [46], The authors discuss the feasibility of sensor-enabled landmine, so called smart-mine. Compared with the traditional landmine, the smart-mine has distinguished features such as the target discernment, ability to talk to the neighboring mine, and self destruction after the war ceases. Different from [46], which mainly focus on the feasible design of a single smart-mine, we investigate the advantage of forming a sensor network and study how to minimize the landmine cost.

In the field of impact engineering, numerous research has been carried out to study the effect of the buried mine explosion. Depending on the types of targets, these works fall into several categories. In [9, 10], the authors study the blast injury on human. It is shown that the distance between a person and the bomb is one of the major factors in determining the survival probability. On the other hand, predicting the blast effect on the armed vehicles or tanks has also received lots of attentions in recent years. Many mathematical and experimental results about the deformation or failure of a calmed plate subject to the explosive loading are reported [7, 8]. There are a variety of models and each model uses different criteria to describe the failure modes, mental properties, soil condition, mine characteristics, rupture strain, etc. In all the models, the standoff distance from the explosion point plays an important role in determining the blast impact on the vehicles or tanks.

In this chapter, we make the first attempt to fill the gap between the mine industry and the impact engineering by formulating the detonable mine selection problem. With the explosion scope mapped to the sensing range, our problem looks alike the sensor coverage problem, whose objective is to select the minimum number of sensors to cover the whole field [4, 5, 24, 18]. However, the existing techniques cannot be simply applied, because most work on coverage assumes a

disk-model, where a point is thought 100% covered if it is within the sensing disk of a sensor. The simplified assumption makes the problem tractable but may not be realistic. There is also some recent work based on the probability coverage model [17]. However, in their work it is assumed that the sensors are densely deployed (i.e., each point is covered by at least three sensors), and the coverage probability of each point can be pre-computed independent of the targets. Besides the economic issue of deploying a smart-mine network with high density, it is infeasible to compute the target's survival probability before the targets enter the field. Thus, decision has to be made on the fly after a intruder is detected, classified, and localized.

3.3 Problem Formulation

We consider a defense scenario where m intruders (targets) enter a field with n smart-mines. Each smart-mine relies on the embedded sensors to cooperate with each other to detect, classify, and localize the targets. By employing the results in impact engineering, each mine is assumed to be able to estimate the consequence of explosion on the intruding target. Following some literatures [9, 10], we use the metric of *failure probability* to quantitatively measure the blast impact. The failure probability reflects the chance that the target fails to function properly after the explosion due to the consequence such as personnel casualty, flat tire, or the tearing of the plate. For example, Fig. 3.2(a) shows the ESTC outdoor blast model [9], where the failure probability (P) of the intruder is a function of range and quantity of explosives. Fig. 3.2(b) further depicts the relationship between the failure probability and standoff distance, when $Q = 8$ kilograms.

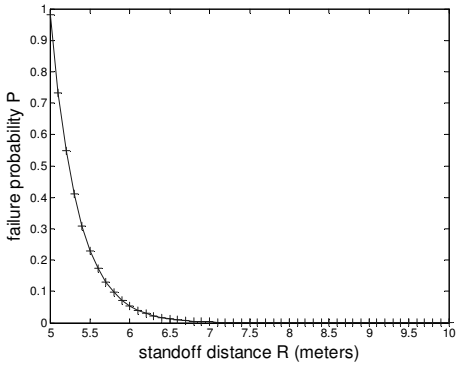
There are also other models available based on the metric of deformation factor, which describes the concrete degree of the plate deformation that the explosive force has caused to the metal-made targets such as tank and armored vehicle. We leave its study to the future work.

We assume the distance between each target and mine is known, e.g., via GPS or other localization schemes [49]. With the known distance, the failure probability of the target can be estimated, e.g., via the mathematical model [9] or experiential model [10]. With these assumptions, the *detonable mine selection*

$$P = \frac{e^{[-5.785 \times (\frac{R}{Q^{\frac{1}{3}}}) + 19.047]}}{100}$$

where R is standoff distance and Q is the quantity of explosives. The formula is valid if $2.5 < R/Q^{\frac{1}{3}} < 5.3$.

(a)



(b)

Figure 3.2. the ESTC outdoor blast model ($Q = 8$ kilogram)

problem (minMS) is formalized as follows.

Given a smart-mine network with n mines and m targets, a failure based blast model, a set of probabilities $\sigma_j, j = 1...m$, we want to select the minimum-cost subset of mines such that each target j can be destroyed with a probability beyond σ_j.

The minMS problem can be expressed in the form of integer program.

$$\text{Min } C = \sum_{i=1}^{n} c_i x_i \tag{3.1}$$

$$\text{ST: } 1 - \prod_{i=1}^{n} x_i(1 - \sigma_{ij}) \geq \sigma_j, \forall j \tag{3.2}$$

$$x_i \in \{0, 1\}, i = 1...n \tag{3.3}$$

$$c_i \geq 0, i = 1...n \tag{3.4}$$

$$0 \leq \sigma_{ij}, \sigma_j \leq 1, i, j = 1...n \tag{3.5}$$

In the above model, $c_i, \sigma_{ij}, \sigma_j$ are given variables, where c_i is the cost of mine i, σ_{ij} is the failure probability of target j due to the explosion of mine i, and σ_j is the threshold probability of target j. x_i is the indicator variable to be determined. $x_i = 1$ if mine i is selected; $x_i = 0$ otherwise. The purpose of the optimization is to determine the variables x_i to minimize the total cost C subject to the constraints 2-5. Constraint 2 states that the failure probability of target j resulted from the

aggregated blasts should exceed σ_j.

Constraint 3.2 can be transformed by applying the logarithmic operation. let $v_{ij} = -log_2(1-\sigma_{ij}), r_j = -log_2(1-\sigma_j)$, then Constraint 3.2 becomes

$$\sum_{\{i|x_i=1\}} v_{ij} = \sum_{i=1}^{n} x_i v_{ij} \geq r_j, \forall j \tag{3.6}$$

Based on this transformation, the following theorem gives the hardness result.

Theorem 5. *The problem of minMS is NP-hard.*

Proof. The problem of minMS can be proved to be NP-hard via a reduction from the set-cover problem [11], which can be stated as follows. Given a universe $U = \{e_1, ..., e_n\}$ of n elements, a collection of subsets of U, $S = \{s_1, ..., s_k\}$, where the cost of subset s_i is $c_i, i = 1...k$, find a minimum cost subcollection of S that coves all elements of U.

To see the reduction, let $\sigma_{ij} = \frac{1}{2}$ if mine i covers target j, and $\sigma_{ij} = 0$ otherwise. Then $v_{ij} = 1$ if mine i covers target j, and $v_{ij} = 0$ otherwise. Further let $\sigma_j = \frac{1}{2}$, then $r_j = 1$. Then Constraint 3.6 becomes $\sum_{\{i|e_j \in s_i\}} x_i \geq 1, \forall e_j$. With mine mapped to the set and target mapped to the element, this is equivalent to say that for any element e_j, at least one set that covers e_j is chosen, which is exactly the constraint of the set cover problem. Therefore, the set-cover problem can be reduced to a special case of minMS problem. Since the set cover problem is known NP-hard, the minMS problem is also NP-hard. □

The transformation not only helps establish the hardness result, but also provides us with a new perspective to view the minMS problem. While the original Constraint 3.2 involves the multiplicative operation, which is complicated to solve, we transform it to Constraint 3.6 after applying the logarithm operation, which only needs additive operation. This motivates us to design a *bucket-tub* model to solve the minMS problem based on the transformed constraint.

In the bucket-tub model, the whole network is represented by a bipartite graph, with the mines on the left and the targets on the right. Each mine i is modeled as a set of buckets each of which can supply $v_{ij} = -log_2(1-\sigma_{ij})$ volume of water to target j, and each target j is modeled as an empty bucket which can hold up to

$r_j = -log_2(1-\sigma_j)$ volume of water. The link between mine i and target j can be seen as the channel through which the water could flow from the buckets to the tubs. Then the problem becomes how to find the minimum-cost subset of bucket sets on the mine side to fill up the empty tubs on the target side.

The following notations are defined for the bucket-tub model, and used throughout the chapter.

- s_i: bucket set i.
- S: the collection of bucket sets, i.e., $S = \{s_i | i = 1...n\}$.
- t_j: tub j.
- T: the collection of tubs, i.e., $T = \{t_j | j = 1...m\}$.
- v_{ij}: the volume of water in s_i available for tub t_j.
- v_i: the total volume of water in s_i, i.e., $v_i = \sum_{j=1}^{m} v_{ij}$.
- r_j: the residue space of tub t_j.
- c_i: the cost of bucket set s_i.

In the bucket-tub model, $v_{ij} = -log_2(1-\sigma_{ij}), r_j = -log_2(1-\sigma_j)$, where σ_{ij} and σ_j are the parameters of the original problem. Without loss of generality, it is assumed that $v_{ij} \leq r_j$ (otherwise, we can reduce v_{ij} to r_j without affecting the result), and $\sum_{i=1}^{n} v_{ij} \geq r_j$ (otherwise, the requirement of the tub t_j can never be satisfied).

Fig. 3.1(b) shows the bucket-tub graph corresponding to Fig. 3.1(a). Each mine is modeled by a set of buckets and each target is modeled by an empty tub. Each bucket can provide v_{ia}, v_{ib} volume of water to the tub t_a and t_b, whose residue space is r_a, r_b, respectively. The purpose is to select the minimum-cost bucket sets to fill the tubs t_a, t_b with water. In general, the problem can be reformulated using the bucket-tub model as follows:

Given a bipartite graph $G(S,T)$, where $S = \{s_i | i = 1...n\}, T = \{t_j | j = 1...m\}$, together with the parameters $v_{ij}, r_j, c_i, i = 1...n, j = 1...m$, we want to find the minimum-cost subcollection of bucket sets to fill all the tubs, i.e., $\sum_{i=1}^{n} x_i v_{ij} \geq r_j, \forall j$.

The new problem formulation could help us avoid the complex form of the original formulation and motivates our design of the algorithms based on the bucket-tub model in the next section.

3.4 Algorithms and Their Performance Bound

In this section, we present two approximation algorithms based on the bucket-tub model, i.e., greedy algorithm, layering algorithm, and prove their performance bound. We use S_{gre}, S_{lay} to denote the solution set produced by each algorithm, and use GRE, LAY to denote their corresponding cost, respectively. For example, for the greedy algorithm $GRE = \sum_{s_i \in S_{gre}} c_i$. Here, the subscript takes the form of $s_i \in S_{gre}$ as an abbreviation of $\{i | s_i \in S_{gre}\}$, and the same form of abbreviation will be applied in the remainder of the chapter.

3.4.1 Greedy Algorithm

Algorithm 4 Greedy Algorithm

Input: a bucket-tub graph $G(S,T)$
Output: a subcollection of bucket sets and its cost, i.e., S_{gre} and GRE
Procedure:

$S_{gre} = \emptyset$
while $T \neq \emptyset$ **do**
  calculate $e_i = \frac{c_i}{v_i}$ for each $s_i, i = 1...n$.
  pick the most effective set s_k with $e_k = \min e_i$, i.e., $S_{gre} = S_{gre} + \{s_k\}$.
  /*update r_j and v_{ij} of the remaining bucket sets and tubs*/
  for $(j \in \{j | v_{kj} > 0\})$
    $r_j = \max(r_j - v_{kj}, 0)$
    for $(i \in \{i | v_{ij} > 0\})$
      $v_{ij} = \min(v_{ij}, r_j)$
    end for
  end for
  remove any tub whose residue space is 0, i.e., $T = T - \{t_j | r_j = 0\}$.
  remove any bucket set whose connected tubs are all full, i.e., $S = S - \{s_i | v_i = 0\}$.
end while
Output S_{gre} and GRE.

The greedy algorithm is executed in iterations. During each iteration, it picks the most cost-effective bucket set, and fills its water to the connected tubs. We use e_i to denote the *average cost* of s_i, which is defined as $e_i = c_i / \sum_{j=1}^{m} v_{ij} = c_i / v_i$.

Then the most cost-effective set is the set with the minimum average cost. Each time a set s_k is chosen, r_j and v_{ij} need to be updated (as in line 6-11 of Algorithm 1). The whole process is then repeated iteration after iteration, until all the tubs become full. The pseudo code is summarized in Algorithm 1 .

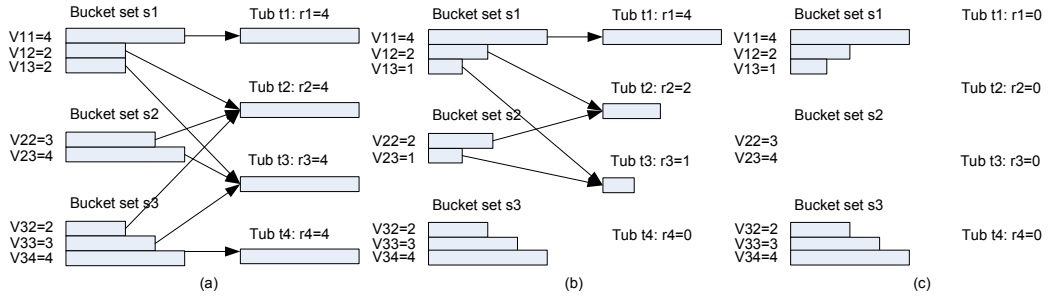


Figure 3.3. An example of the greedy algorithm

Fig. 3.3 uses an example to illustrate the greedy algorithm. There are three bucket sets and four empty tubs. Each bucket set has cost 1, with the volume size and residue space as given in Fig. 3.3(a). According to Algorithm 1, in the first iteration (Fig. 3.3b), s_3 is chosen because it is the most effective set, i.e., $e_3 = \frac{1}{4+3+2} = \frac{1}{9}$. Following that, r_i, v_{ij} will be updated accordingly. For example, $r_3 = 4 - 3 = 1, v_{23} = \min(4, 1) = 1$. In the second iteration, s_1 is chosen because $e_1 = \frac{1}{7} < e_2 = \frac{1}{3}$. Since s_1, s_3 together can fill the space of all the tubs (Fig. 3.3c), the algorithm terminates with the total cost of 2.

3.4.2 Layering Algorithm

As shown in Fig. 3.4(a), the layering algorithm decomposes the given graph into layers, and each layer is a bucket-tub graph by itself. At each layer, the cost of any set is no greater than $\alpha \cdot e_{min}(k)$, where $\alpha \geq 1$ is the *relaxation factor* and $e_{min}(k)$ is the minimum average residue cost at layer k. Denote the graph of layer k by $G(k)$. Similarly, the same method is used to extend the definition of other variables. For example, the variables $S(k), T(k), c_i(k), v_{ij}(k), v_i(k), r_j(k)$ all preserve the original definition, but specifically refer to layer k.

Algorithm 5 Layering Algorithm

Input: a bucket-tub graph $G(S,T)$
Output:a subcollection of bucket set and its cost, i.e., S_{lay} and LAY
Procedure:

$k=0, G(0)=G, S(0)=S, T(0)=T, S_{lay}=\emptyset$
$RC_i(0)=c_i, \forall i \in \{i|s_i \in S\}$
/*decompose graph into layers*/
while $T(k) \neq \emptyset$ **do**
  identify the zero bucket set $Z_b(k)=\{s_i(k)|v_i(k)=0\}$.
  $S(k+1)=S(k)-Z_b(k)$.
  calculate $e_{min}(k)=\min e_i(k)=\min\frac{RC_i(k)}{v_i(k)}$.
  pick $P(k)=\{s_i|e_i(k)\in[e_{min}(k),\alpha\cdot e_{min}(k)],\alpha\geq 1\}$.
  $S(k+1)=S(k)-P(k)$.
  /*update c_i, v_{ij}, r_j of the remaining bucket sets and tubs*/
  for $(i \in \{i|s_i \in S(k)\})$
    $c_i(k)=\min(c_i(k), \alpha\cdot e_{min}(k)\times v_i(k))$
    $RC_i(k+1)=RC_i(k)-c_i(k)$
  end for
  for $(j \in \{j|v_{lj}(k)>0, s_l \in P(k)\})$
    $r_j(k+1)=\max(r_j(k)-v_{lj}(k),0)$
    for $(i \in \{i|v_{ij}(k)>0\})$
      $v_{ij}(k+1)=\min(v_{ij}(k), r_j(k))$
    end for
  end for
  identify the zero tub set $Z_t(k)=\{t_j|r_j(k)=0\}$.
  $T(k+1)=T(k)-Z_t(k)$.
  $k=k+1$.
end while
Output $S_{lay}=\sum_{k=0}^{L-1} P(k)$ and LAY.

Algorithm 5 lists the pseudo code of the layering algorithm and Fig. 3.4(a) shows its illustrative diagram. In the beginning, $G(0)=G, S(0)=S, T(0)=T$. Following that, the layers are constructed one after another. For example, $G(k+1)$ is built upon $G(k)$ as follows. First, remove from $S(k)$ the bucket sets which have zero volume of water, and use $Z_s(k)$ to denote the collection of such sets, i.e., $Z_s(k)=\{s_i|v_i(k)=0, s_i\in S(k)\}$. Second, define $RC_i(k)=c_i-\sum_{l=0}^{k-1} c_i(l)$ to be the *residue cost* of s_i at layer k, and $e_i(k)=\frac{RC_i(k)}{v_i(k)}$ to be the *average residue cost* of s_i at layer k. Then calculate the minimum average residue cost by $e_{min}(k)=\min_i\frac{RC_i(k)}{v_i(k)}$. Further identify and pick the collection $P(k)=\{s_i|e_i(k)\in[e_{min}(k),\alpha\cdot e_{min}(k)],\alpha\geq 1\}$, which are removed from $S(k)$. Then we have $S(k+1)=S(k)-P(k)-Z_s(k)$. Third, calculate the cost of each bucket set s_i at layer k, i.e., $c_i(k)=\min(c_i(k),\alpha\cdot$

$e_{min}(k) \times v_i(k)$). It is followed that the average residue cost of each set at layer k is at most $\alpha \cdot e_{min}(k)$. $r_j(k)$ and $v_{ij}(k)$ are updated in a manner similar to the greedy algorithm. Finally, remove from $T(k)$ the collection of tubs with zero residue space, denoted by $Z_t(k)$. Then we have $T(k+1) = T(k) - Z_t(k)$. The entire process will repeat layer after layer until the residue space of all the tubs is filled. Suppose the graph is decomposed into total $L+1$ layers, as shown in Fig. 3.4(a), then the collection $S_{lay} = \sum_{k=0}^{L-1} P(k)$ will be the output.

From the above description, it can be seen that the cost of each bucket set is decomposed into layers. Suppose a bucket set s_i stays in the bucket-tub graph until layer k, then we have $c_i = \sum_{l=0}^{k} c_i(l)$. At each layer l, we pick the sets whose average residue cost falls below $\alpha \cdot e_{min}(l)$. Therefore, the relaxation factor α determines how many bucket sets can be chosen per layer. When a larger α is used, more bucket sets can be chosen at each layer, so the graph is decomposed into fewer layers and involves less computation overhead. On the other hand, when α increases, the approximation ratio of the algorithm will increase, as shown in subsection 3.4.3. Therefore, an appropriate relaxation factor is required to strike a balance between the computational burden and performance bound.

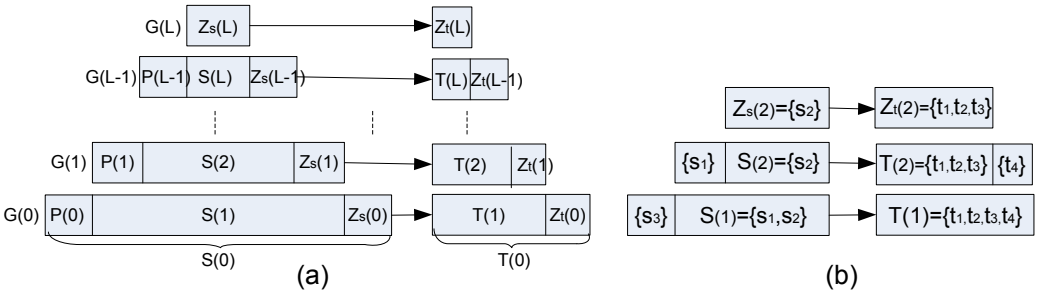


Figure 3.4. Illustration of the layering algorithm (a) the general case, where a given graph G is decomposed into $L+1$ layers. (b) a special case, taking Fig. 3.3 as an example

Fig. 3.4(b) uses the same example of Fig. 3.3 to illustrate the layering algorithm, assuming $\alpha = 1$. In the beginning, $G(0) = G$. $P(0) = \{s_3\}$ because $s_3(0)$ has the smallest average residue cost, i.e., $e_{min}(0) = e_3(0) = \frac{1}{9}$. $Z_s(0) = \emptyset$. Then $c_1(0) = e_{min}(0) \times v_1(0) = \frac{8}{9}, c_2(0) = \frac{7}{9}$, and r_j, v_{ij} is updated in the same way as

in Fig. 3.3(b). At layer 1, $P(1) = \{s_1\}$ because $e_{min}(1) = e_1(1) = \frac{(1-\frac{8}{9})}{7} = \frac{1}{63}$. As seen from Fig. 3.3(b), $r_4(1) = 0$, thus $Z_t(1) = \{t_4\}$. Since $T(3) = \emptyset$, the graph is decomposed into three layers, and the algorithm outputs $S_{lay} = P(0) + P(1) = \{s_1, s_3\}$. On the other hand, if we set $\alpha = 2$, a similar calculation will show that $S_{lay} = \{s_1, s_2, s_3\}$ is the output, with only one layer constructed.

Although the layering algorithm looks more complicated than the greedy algorithm, it has the same worst-case computational complexity $O(n^2 + mn)$. In addition, it has a provable approximation ratio that relies only on the relaxation factor and the maximum number of mines connected with a tub, as shown in the next subsection.

3.4.3 Performance Bound

In this section, we will establish the performance bound for the layering algorithm and greedy algorithm. We will first prove that the layering algorithm has an approximation ratio of $\alpha \cdot f$, where f is the maximum number of sets/mines that a tub/target is associated with, and α is the tunable relaxation factor. After that, we will show that the greedy algorithm has an approximation ratio of $\sum_j R_j$, where R_j is the coefficient in the related integer program.

Lemma 1. *In the layering algorithm, given a graph $G(k)$ at layer k, where the average residue cost of all the bucket sets falls into $[e_{min}(k), \alpha e_{min}(k)], \alpha \geq 1$, there is $\sum_{s_i \in S(k)} c_i(k) \leq \alpha \cdot f \cdot OPT(G(k))$, where $OPT(G(k))$ is the cost of the optimal solution to $G(k)$.*

Proof. At layer k, because $e_i(k) \in [e_{min}(k), \alpha e_{min}(k)]$, there is $c_i(k) = e_i(k) \times v_i(k) \leq \alpha \cdot e_{min}(k) v_i(k)$. Further because $v_{ij} \leq r_j$, and each tub is connected with at most f buckets, there is $\sum_{s_i \in S(k)} v_i(k) \leq f \cdot \sum_{t_j \in T(k)} r_j(k)$. In other words, the total volume of water that the buckets can provide is no greater than f times the total space volume of the tubs. Then we have

$$\sum_{s_i \in S(k)} c_i(k) \leq \alpha \cdot f \cdot e_{min}(k) \sum_{t_j \in T(k)} r_j(k) \tag{3.7}$$

Use $S_{opt}(k)$ to denote the collection of bucket sets chosen in the optimal solution

for $G(k)$. It is followed that the total water provided by the buckets in $S_{opt}(k)$ can fill the tubs in $T(k)$, i.e., $\sum_{s_i \in S_{opt}(k)} v_i(k) \geq \sum_{t_j \in T(k)} r_j(k)$. Then we have

$$\mathrm{OPT}(G(k)) = \sum_{s_i \in S_{opt}(k)} c_i(k) \geq e_{min}(k) \times \sum_{t_j \in T(k)} r_j(k) \tag{3.8}$$

Combining Eqn. 3.7 & 3.8 gives the lemma.

□

Lemma 2. *Suppose S_0 denotes an arbitrary solution to G, i.e., the bucket sets in S_0 can collaboratively fill the tubs in G. Then $S_0 \cap G(k)$ is a solution to $G(k)$ in the layering algorithm.*

Proof. Since S is a solution to G, for any tub t_j in G, $\sum_{s_i \in S_0} v_{ij} \geq r_j$. We divide the solution S_0 in two parts, i.e., $S_0 = (S_0 \cap G(k)) + (S_0 - S_0 \cap G(k))$. In the following, we will prove that the collection $(\sum_{l<k} P(l) + S_0 \cap G(k))$ is a solution to G. Define $Q(j) = \{s_i | s_i \in (S_0 - S_0 \cap G(k)), v_{ij} > 0\}$ for each t_j. We want to show that any tub t_j associated with $Q(j)$ can be filled by the buckets in $(\sum_{l<k} P(l) + S_0 \cap G(k))$.

Given a tub t_j, for any set $s_i \in Q(j)$, there are only two possible cases, i.e., s_i belongs to some removed set ($s_i \in Z_s(l), l < k$) or s_i belongs to some chosen set ($s_i \in P(l), l < k$). If it is the former case, it implies that tub t_j has already been filled till layer l, i.e., $r_j(l) = 0$. Thus without loss of generality, assume all sets $s_i \in Q(j)$ belong to the latter case, i.e., $s_i \in \sum_{l<k} P(l)$. It is followed that $Q(j) \subseteq \sum_{l<k} P(l)$ for each j. Then we have

$$\begin{aligned} \sum_{s_i \in \sum_{l<k} P(l)} v_{ij} \; + \sum_{s_i \in S_0 \cap G_k} v_{ij} &\geq \sum_{s_i \in (S_0 - S_0 \cap G_k)} v_{ij} + \sum_{s_i \in S_0 \cap G(k)} v_{ij} \\ = \sum_{s_i \in S_0} v_{ij} \;\; &\geq \;\; r_j, \text{for each } j \end{aligned} \tag{3.9}$$

Eqn. 3.9 shows that the set $(\sum_{l<k} P(l) + S_0 \cap G(k))$ is a solution to G. Therefore, $S_0 \cap G(k)$ must be a solution to $G(k)$. □

Theorem 6. *The layering algorithm has an approximation ratio $\alpha \cdot f$, i.e., $LAY \leq \alpha \cdot f \cdot OPT$, where OPT is the cost of the optimal solution to G.*

Proof. Use S_{opt} to denote the collection of bucket sets chosen in the optimal solution to G. Because in the layering algorithm, each set decomposes its cost into different layers, for any set s_i chosen at layer k we have

$$c_i = \sum_{l \le k} c_i(l) = \sum_{l \le k} e_{min}(l) v_i(l), \text{ for } \forall s_i \in S(k) \tag{3.10}$$

Suppose G is decomposed into $L+1$ layers, as shown in Fig. 3.4(a). Then, the total cost of the sets chosen by the layering algorithm is

$$LAY = \sum_{s_i \in \sum_{k<L} P(k)} c_i = \sum_{s_i \in \sum_{k<L} P(k)} \sum_{l \le k} e_{min}(l) v_i(l) \tag{3.11}$$

In Eqn. 3.11, the total cost is first summed vertically (over different layers), and then horizontally (over different sets) in Fig. 3.4(a). Alternatively, we can calculate by summing first horizontally (over different sets within a layer) and then vertically (over different layers). Combining the fact that $P(k) = S_{lay} \cap G(k)$, we have

$$LAY = \sum_{k<L} \sum_{s_i \in P(k)} c_i(k) = \sum_{k<L} \sum_{s_i \in (S_{lay} \cap G(k))} c_i(k) \tag{3.12}$$

Following the similar steps of Eqn. 3.10- 3.12, we now calculate the total cost of the optimal solution. For any set $s_i \in Z_s(k)$ removed at layer k, because $c_i(k) \ge e_{min}(k) v_i(k) = 0$, we have

$$c_i = \sum_{l<k} c_i(l) + c_i(k) \ge \sum_{l<k} e_{min}(l) v_i(l), \text{ for } s_i \in Z_s(k) \tag{3.13}$$

Therefore, the total cost of the optimal solution is

$$OPT = \sum_{k \le L} \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \ge \sum_{k<L} \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \tag{3.14}$$

On the other hand, at layer k for any solution $S_0(k)$ to $G(k)$, we have

$$\sum_{s_i \in S_{opt}} c_i(k) \le \sum_{s_i \in S_0(k)} c_i(k) \le \sum_{s_i \in S(k)} c_i(k) \tag{3.15}$$

According to Lemma 2, both $(S_{lay} \cap G(k))$ and $(S_{opt} \cap G(k))$ are solutions to $G(k)$. Thus, by Eqn. 3.15 and Lemma. 1, we get

$$\sum_{s_i \in (S_{lay} \cap G(k))} c_i(k) \leq \alpha \cdot f \cdot \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \tag{3.16}$$

Finally, following Eqn. 3.12, 3.14 , 3.16, we have

$$LAY \leq \alpha \cdot f \cdot OPT \tag{3.17}$$

□

To establish the performance bound of the greedy algorithm, we can exploit some approximation result on integer programming [50, 51]. However, to do this, the coefficients in Constraint 3.6, i.e.,v_{ij}, r_j have to be integers. Observe that when v_{ij} and r_j are multiplied by the same constant, the integer programming model in Section 3.3 remains unchanged. Thus, we can safely scale the problem to convert the coefficients to integers via some existing techniques such as [52]. Then we get the following theorem.

Theorem 7. *The greedy algorithm has an approximation ratio of $\sum_j R_j$, where R_j is the integer coefficient corresponding to r_j after scaling the Constraint 3.6.*

The proof of Theorem 7 directly follows the result of [51] which states that applying the greedy heuristic to any integer program with the form of MIN(cx) subject to $Ax \geq b$ and $x \in \{0, 1\}$ has an approximation ratio of $H(k)$, where $k = \sum_j b_j$.

3.5 Distributed Implementation

In this section, we will present a distributed implementation of the greedy algorithm. The aforementioned greedy algorithm iteratively picks up the most cost-effective set and updates the residue space of the tubs and the available water in the buckets. We call it the *global greedy algorithm*, which requires centralized knowledge. Nevertheless, based on a newly defined *bucket graph*, we could design a *local greedy algorithm* which involves only the local message exchange and achieves the same performance as the global version of the algorithm.

In a bucket graph, each node represents a bucket set. Two nodes have a link with each other if the two bucket sets share a common neighboring tub in the corresponding bucket-tub graph. For example, Fig. 3.5(a) shows the bucket graph corresponding to the smart-mine network in Fig. 3.1(a). Although a bucket graph should be composed only of the bucket sets and the links between them, in Fig. 3.5 we explicitly draw the targets (with residue space r_a and r_b) and use dashed lines to depict the relationship between the bucket sets and tubs for a better illustration. The weight over each dashed line, i.e., v_{ij}, denotes the water available in the bucket set for the specific tub. In reality, all these information needs to be cached at each node. The observation in the following lemma can help us design a *local greedy algorithm* based on the bucket graph, which can be proved to produce the same solution set as the global greedy algorithm.

Lemma 3. *Given a bucket graph G_b, if a bucket set represented by a node is the most cost-effective set among its neighbors, it will be part of the solution set chosen in the global greedy algorithm.*

Proof. Suppose a node s has the minimum average cost among its neighbors, denoted by $N(s)$. Then no node among $N(s)$ shall be picked before s is picked in the global greedy algorithm. This is because the average cost $e_i = \frac{c_i}{v_i}$ at each node i is a non-decreasing function as other nodes are picked. Therefore, s remains to be the most cost-effective among its neighbors. In addition, to fill the tubs associated with s, at least one node among s and $N(s)$ has to be picked. Thus, eventually s will be chosen by the global greedy algorithm. □

We assume the communication range of the sensor is at least twice the range of the explosion[1]. Thus, two neighboring nodes in the bucket graph can communicate directly. The local greedy algorithm is executed in iterations. During each iteration, the nodes which have the smallest average cost among their neighbors will elect themselves. Then, information about v_{ij}, r_j is updated in a distributed manner, after which another iteration will begin. From the algorithmic point of view, the nodes elected per iteration constitute an Independent Set, which is defined as a subset of nodes among which there is no link between any two nodes. The set

[1]The normal communication range of a sensor is around tens of meters; the explosion range is less than ten meters for certain types of mine.

is a maximal independent set (MIS) if no more nodes can be added to generate a bigger independent set [13]. Therefore, our algorithm is equivalent to find a MIS by electing the most effective nodes among their neighbors in each iteration.

Algorithm 6 Distributed Greedy Algorithm

Input: node s, its neighbors $N(s)$, and its connected tub set T_s
Output: election or non-election status
For Each Node s:
1: **while** $T_s \neq \emptyset$ **do**
2: **if** s has the smallest average cost among its neighbors **then**
3: s elects itself, broadcasts $msg(s, ELE, c_s)$, and exits.
4: **end if**
5: **if** $msg(id, ELE, cost)$ is received **then**
6: update the residue space in $T_s \cap T_{id}$, and remove the tubs with zero residue space from T_s.
7: recalculate c_s, and broadcast $msg(s, UPD, c_s)$. Go to 2
8: **end if**
9: **if** $msg(id, UPD, cost)$ is received **then**
10: update the average cost of neighbor s_{id}. Go to 2.
11: **end if**
12: **end while**
13: output the non-election status.

The pseudo code of the local greedy algorithm is listed in Algorithm 6. The message format is defined as $msg(id, type, cost)$, where id and $cost$ are the fields related to the sender, and $type$ can be set to be ELE or UPD, which corresponds to the election notice or the update report. An election notice is sent out when a node finds itself to be the most cost-effective set among its neighbors (line 2-4). When an election notice is received, the average cost of the receiver is updated in a manner similar to the global greedy algorithm, after which an update report is sent out (line 5-8). When an update report is received, the average cost of the sender is simply updated to the $cost$ value contained in the message (line 9-11).

Theorem 8. *The local greedy algorithm is guaranteed to terminate, and upon termination, the same solution set is produced as in the global version of the algorithm.*

Proof. At any instant, there is at least one node which has the smallest average cost among its neighbors and thus is qualified to elect itself. Therefore, the algorithm will not stop until the requirements of all the tubs are satisfied. In addition, Lemma 3 assures that the same solution set will be produced as in the global

greedy algorithm upon termination, although the bucket sets may be picked in a different order. □

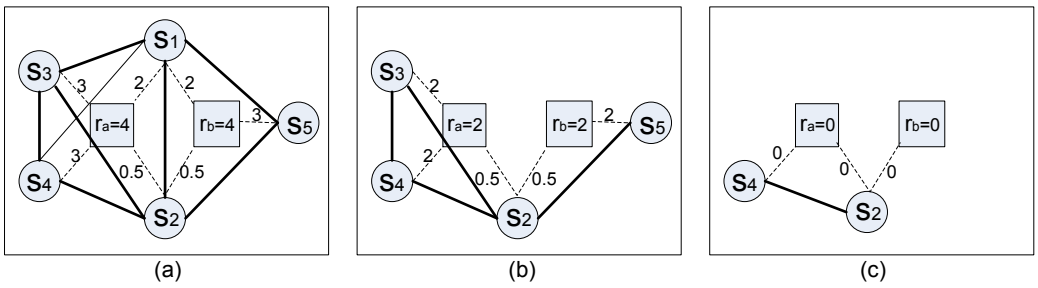


Figure 3.5. An example to illustrate the local greedy algorithm. (a) the bucket graph corresponding to Fig. 3.1. (b) the bucket graph after node s_1 is elected in the first iteration. (c) the bucket graph after nodes s_3, s_5 are elected in the second iteration.

Fig. 3.5 uses a small example to illustrate the local greedy algorithm. In Fig. 3.5(a), there are five bucket sets (circle node), which are connected with two targets (square node). Each bucket set has cost 1 and can provide v_{ij} volume of water for each tub, as shown over the dashed arrows. Each tub has the empty space of 4. In the first iteration, s_1 elects itself since it has the smallest average cost among its neighbors (tie is broken by id). Then v_{ij}, r_a, r_b are updated and the algorithm continues to run among the remaining nodes (Fig. 3.5b). Similarly, in the second iteration, s_3, s_5 elect themselves, after which the tubs become full (Fig. 3.5c).

Like the layering algorithm, the election criteria in the local greedy algorithm can also be relaxed to reduce the number of iterations required. We call it the *fast local greedy algorithm*. Instead of electing only the nodes with the smallest average cost among its neighbors, any node whose average cost falls in the range of $[e_{min}, \alpha e_{min}]$ can be elected, where e_{min} is the minimum average cost among the neighbors and $\alpha \geq 1$ is the relaxation factor. To implement this, we need to change the message format to $msg(id, type, cost, e_{min})$ by adding a field on the minimum average cost. Any node whose average cost is the local minimum will set up the field e_{min} accordingly. For other nodes that receive the election notice with e_{min} properly set, they will check whether their average cost belongs to $[e_{min}, \alpha e_{min}]$ and decide whether to elect themselves. This way, more nodes will be elected per

iteration and less total iterations will be needed.

Algorithm 6 is a distributed implementation of the global greedy algorithm. According to Theorem 8, the local greedy algorithm picks the same collection of bucket sets as in the global greedy algorithm, without degrading the performance. On the other hand, the fast greedy algorithm uses the relaxation factor to trade off the performance for the lower message overhead and computational burden. Both the algorithms have the worst case message complexity $O(d)$ per node, where d is the maximum number of neighbors among all the nodes in the bucket graph.

3.6 Performance Evaluation

In this section, we verify the effectiveness of the proposed algorithms through simulations. In the simulation, n mines and m targets are randomly deployed in a 10×10 square area, with n varying from 100 to 1000, and m varying from 10 to 100. The blast range and the sensing range are normalized to be 1. The communication range is set to be 2, which is twice of the blast range and sensing range. The cost of each mine is uniformly distributed between 1 and 4 units. To emulate the destructive impact on the target, the blast model in Fig. 3.2 is used. The threshold probability for each target is unified to be 0.9.

Besides the greedy algorithm and layering algorithm, two other simple heuristics are evaluated, i.e., random algorithm and max-weight first algorithm. The random algorithm arbitrarily picks the bucket set, while the max-weight first algorithm picks the bucket set in which a single bucket has the largest volume of water available for the connected tub. This is equivalent to select the mine which has the shortest distance to the target. The experiments are done over a customized C++ simulator and each point is a statistical average of 50 experiments.

3.6.1 Comparison of Different Algorithms

Figs. 3.6, 3.7 compare the cost of different algorithms with varying number of mines, when the number of targets is 50 and 100, respectively. It is shown that the greedy algorithm consistently performs best. The layering algorithm stays close to the greedy algorithm when the relaxation factor R is 1. As the relaxation factor

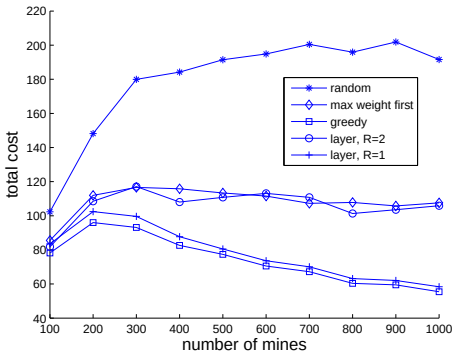


Figure 3.6. effect of number of mines n ($m = 50$)

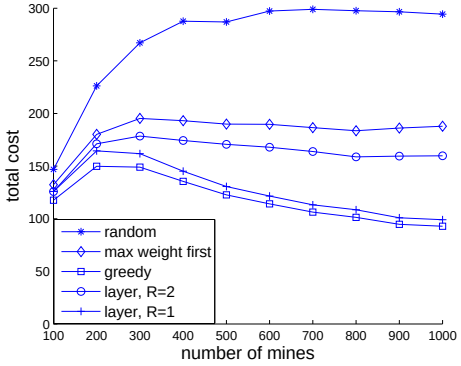


Figure 3.7. effect of number of mines n ($m = 100$)

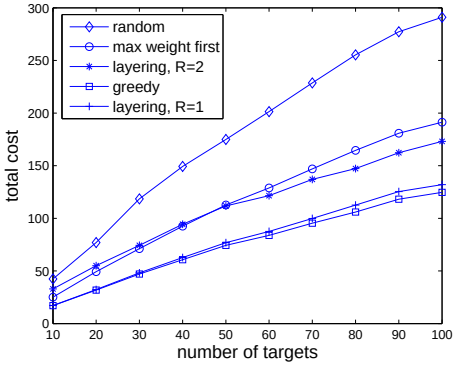


Figure 3.8. effect of number of targets m ($n = 500$)

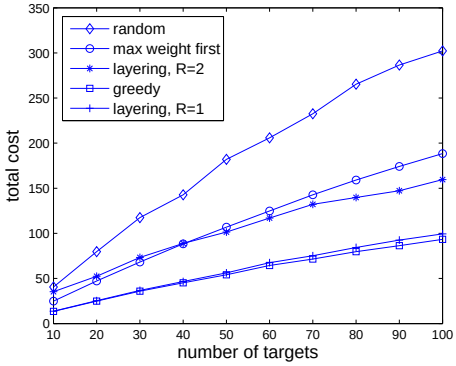


Figure 3.9. effect of the number of targets m ($n = 1000$)

increases from 1 to 2, the performance of the layering algorithm becomes worse, but still better than the max-weight first algorithm. Regarding the performance of the random algorithm, a big gap can be observed compared with the other algorithms.

Another observation is that increasing the number of mines may affect the trend of different algorithms in different ways. As the number of mines increases, the cost of the random algorithm tends to increase, due to its pure randomness. However, increasing the number of mines beyond 200 (which is sufficient to destroy all the targets) can cause the other algorithms to gradually decrease. This is because all other algorithms exploit certain level of intelligence when selecting the mines.

Thus the more candidate mines available, the more likely a better solution can be produced.

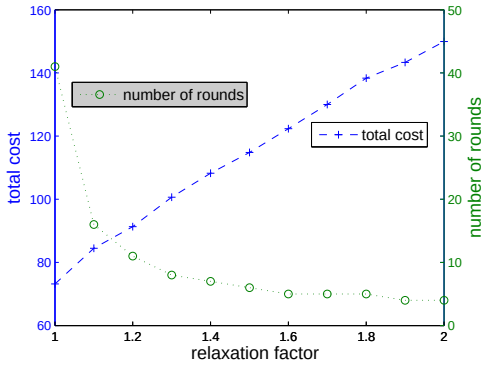


Figure 3.10. effect of the relaxation factor R in the fast greedy algorithm ($n = 500, m = 50$)

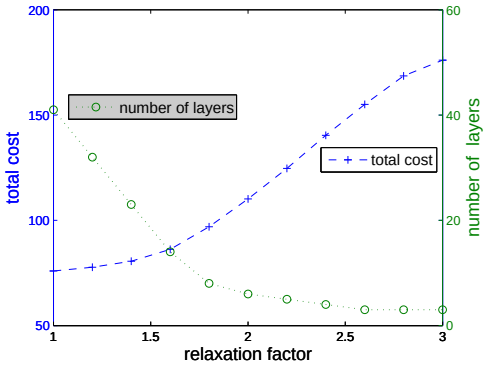


Figure 3.11. effect of the relaxation factor R in the layering algorithm ($n = 500, m = 50$)

Figs. 3.8, 3.9 further compare the cost of different algorithms with varying number of targets, when the number of mines is 500 and 1000, respectively. Regarding the relative performance among different algorithms, the same conclusion can be drawn as in Figs. 3.6, 3.7, with the random algorithm to be the worst and the greedy algorithm to be the best. It can be noticed that the cost of all the algorithms increases as the number of targets increases. However, the increase for the random algorithm is roughly linear, but for the other algorithms it is sublinear. This implies that a smarter algorithm such as the greedy algorithm is less affected by the increasing number of targets, because it takes into account both the effect of single explosion on multiple targets and collaborative explosions on a single target.

3.6.2 Effect of The Relaxation Factor

Relaxation factor R is an important parameter for the fast greedy and layering algorithm. Figs. 3.10, 3.11, 3.12, 3.13 use double y-axes to study the effect of relaxation factor R, with the left y-axis to denote total cost and the right y-axis to denote number of rounds/layers. In particular, Fig. 3.10 shows the effect of R in the fast greedy algorithm, when $m = 50, n = 500$. As R increases, the cost of the fast greedy algorithm increases, but the number of its execution rounds reduces.

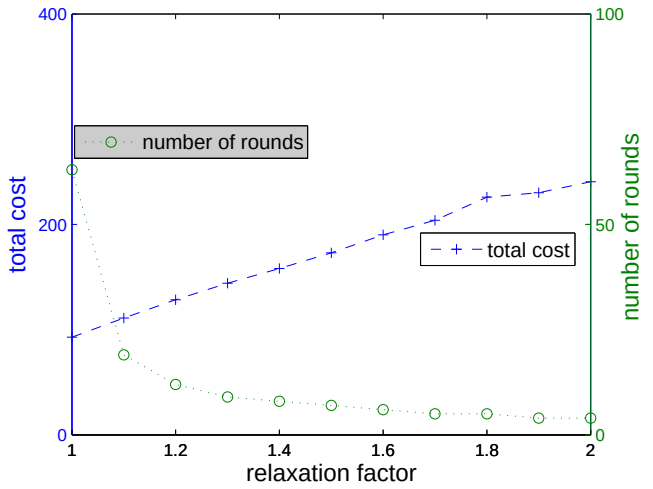


Figure 3.12. effect of the relaxation factor R in the fast greedy algorithm($n = 1000, m = 100$)

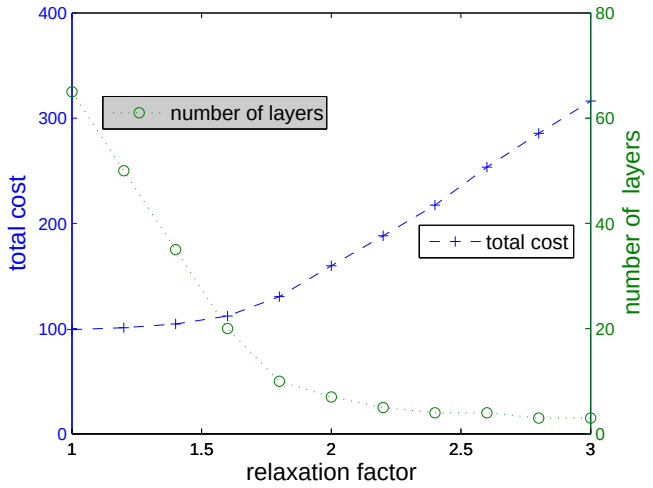


Figure 3.13. effect of the relaxation factor R in the layering algorithm ($n = 1000, m = 100$)

For example, when R increases from 1.0 to 1.1, its cost rises from 73 to 84 while its computation rounds reduce from 41 to 16. This is expected since the value of the relaxation factor R reflects the strictness of the selection criteria. When the criteria is relaxed, i.e., with larger R, more mines can be picked per round thus less number of total rounds are required.

Fig. 3.11 shows the effect of R in the layering algorithm, when $m = 50, n = 500$. Similar to Fig. 3.10, as R increases, the cost of the layering algorithm increases, but the number of layers reduces.

Figs. 3.12, 3.13, where $m = 100, n = 1000$, lead to the same conclusion as Figs. 3.10, 3.11. Since in both algorithms, the message overhead or the computational burden is proportional to the number of rounds/layers, R needs to be appropriately set to strike a balance between different performance metrics.

3.7 Conclusions

This chapter explores the potential of networking the landmines via sensor technology. Specifically, we formulate a detonable mine selection problem, with the objective to destroy the intruding target using the minimum cost. Due to its NP complexity, we focus on the design of approximation solutions based on a novel bucket-tub model, where mine is mapped to bucket set and target is mapped to tub. Two classes of approximation algorithms are proposed, whose performance bound away from the optimal result is given. Among them, it is shown that the layering algorithm can achieve an approximation ratio that relies only on the maximum number of mines/buckets that a target/tub is associated with, and that the greedy algorithm can be implemented in a pure distributed manner without degrading the performance. Theoretical analysis and extensive simulations verified the effectiveness of the proposed algorithms.

Our work is among the first efforts towards filling the gap between the mine industry and impact engineering. More interesting dimensions such as trying different blast models, dealing with the mobile case by taking advantage of the target's mobility pattern can be investigated. As the landmine may linger in the field for a long time, self monitoring and postwar disposal issues are also worth further study.

Chapter 4

Network Monitoring For Mission Driven Sensor Network

4.1 Introduction

As sensor nodes usually operate in an unattended, harsh environment, they are prone to failure and may run out of battery [47]. To make sensor network reliable as well as adaptable, sensor status (such as liveness, density estimation, residue energy, etc.) has to be closely monitored and made known to the sink, which can promptly react to sensor status changes. For instance, to activate other sleeping sensors if they exist, to deploy additional sensors or to dispatch mobile sensors to a region where a significant percentage of sensors have failed.

In distributed systems, the only way to learn the status of a node is through receiving messages from the node. For example, in IP networks, the poller-pollee structure [53] has been widely used for network management, where some specialized nodes are called *pollers* and the other nodes are called *pollees*. Each poller monitors its pollees by actively sending a "ping" message and then waiting for the reply or by passively waiting for the pollees to send messages periodically.

Compared with wired networks, designing monitoring mechanisms for sensor networks has several challenges. The first major issue is the false alarm, which results from the failure characteristics of the wireless links. When a status report is lost due to transient link interference/failure, the system may consider it as the

result of a node failure, thus a false alarm would be triggered. Second, instead of over a fixed topology, sensors need to self-organize themselves into a monitoring architecture in a distributed manner. Finally, since sensors are usually untethered and powered by battery, energy efficiency is an important issue.

In this chapter, we propose solutions to address the challenge specific to sensor networks, to design a fault tolerant, energy efficient monitoring system in a distributed manner. The whole architecture is build upon the poller-pollee structure, where sensors self-organize themselves into two tiers, with pollees in the lower tier and pollers in the upper tier. The pollees send status reports to the pollers along multihop paths, during which the intermediate nodes do the aggregation to reduce the message overhead. Each poller makes local decisions based on the received aggregated packets, and forwards its decision towards the sink.

One weakness of the poller-pollee structure is that the poller or the communication link may fail, which could cause false alarms. To address this issue, each pollee is associated with multiple pollers. Although using multiple pollers can be fault tolerant, sending status reports to multiple pollers at the same time can significantly increase the power and bandwidth consumption. We thus propose a novel solution where the monitored sensor sends the status reports to different pollers in a round robin manner. Status reports from different pollers can thus be combined to reduce false alarm. For example, although the report through one poller is lost due to transient communication link interference/failure, the sink may still receive the pollee's report from another poller in the next time interval. Through analysis, we show that the round robin based multi-poller solution can significantly reduce the false alarm rate compared to the single poller scheme.

When building the monitoring architecture, we focus on the fundamental tradeoff between the number of monitoring nodes (i.e., pollers) and the false alarm rate. Most of the previous work targets at minimizing the number of pollers only, because selecting more pollers will enhance the difficulty of tracking the status of each poller and thus increase the network management cost [53]. However, in a lossy environment, the false alarm rate can be adversely affected by a smaller number of pollers. For example, if the number of pollers is too small, some pollees will be too far away from the poller, and then the chance of link transient failure will be higher and the false alarm rate will be larger. To balance the tradeoff between

the number of pollers and false alarm rate, we propose a distributed deterministic algorithm, which uses two parameters k_1, k_2 to guide a better distribution of poller and pollee; i.e., no two pollers are less than k_1 hops away from each other, and no pollee is more than k_2 hops away from its poller. This property enables us to minimize the number of pollers while bounding the maximum false alarm rate. We discuss how to set up these parameters and further reduce the message overhead based on a randomized technique.

To increase the energy efficiency and reduce the monitoring overhead, we take the hop-by-hop aggregation opportunities in sensor networks. When pollees, which can be multiple hops away from the poller, send status reports to their pollers, the status reports can be aggregated to reduce the number of packets needed. It is nontrivial to determine which aggregation path should be used in order to achieve better aggregation. In this chapter, we formulate the selection of the optimal aggregation path to minimize the transmission energy as a NP-hard problem, and prove that an opportunistic greedy forwarding scheme has an approximation ratio of $\frac{5}{4}$. We also prove that if the pollee is within 2 hops of the poller, i.e., $k_2 = 2$, the bounded ratio is $1 + \frac{s-1}{s^2}$.

The rest of the chapter is organized as follows. Section II presents the poller-pollee based monitoring architecture. Section III proposed and analyzed the round robin based multi-poller scheme. Section IV formulates the minimum poller selection problem and proposes the distributed algorithms. Section V focuses on the optimal aggregation path problem, and studies the greedy algorithm with constant approximation ratio. Performance evaluations are done in Section VI. Section VII overviews the related work and Section VIII concludes the chapter.

4.2 Architecture and Problem Formulation

In this section, we present the multi-poller structure, based on which we build the sensor monitoring system.

4.2.1 The Multi-Poller Structure

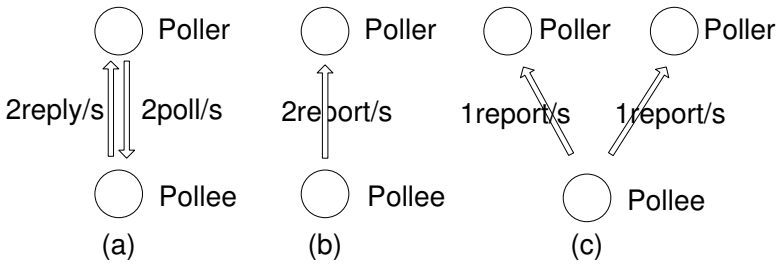


Figure 4.1. Basic and extended poller-pollee structure

In distributed systems, the only way to learn the status of a node is through receiving message from the node. We can have the monitoring node (i.e., poller) send a "ping" message and then wait for the reply (Fig. 4.1a), or wait for the node being monitored (i.e., pollee) to proactively send a message (Fig. 4.1b). As a sensor network is a large distributed system, the poller and pollee structure can also be used for monitoring the sensor network status. Since two messages are used in Fig. 1(a), whereas only one is used in Fig. 1(b), we prefer to having pollees send periodic reporting messages to the poller to reduce energy consumption, and only resort to ping-response messages when necessary. However, traditional poller-pollee structure like Fig. 4.1(a) and Fig. 4.1(b) is not resilient to node or communication failure because each pollee is associated with only one poller. To tolerate these failures, we extend the poller-pollee structure by associating each pollee with multiple pollers, as shown in Fig. 4.1(c).

4.2.2 Architecture

Different monitoring architectures can be built on the poller-pollee structure (Fig. 4.1). Given the scale of sensor networks, a centralized polling scheme, as depicted in Fig. 4.2(a), is not scalable. Instead, Fig. 4.2(b)(c) has a two-tier distributed polling structure where pollees send status reports to the pollers, which in turn generate aggregated status reports and forward them to the sink.

As each pollee in Fig. 4.2(b) is monitored by only one poller, it is not resilient to the poller failure or communication failure. To reduce the false alarm rate, we apply the multi-poller concept to Fig. 4.2(c). For simplicity, each pollee only has two pollers, referred to as the primary poller and the secondary poller. As shown in Fig. 4.2(c), pollee 1 is monitored by poller 1 which is the primary, and poller 2

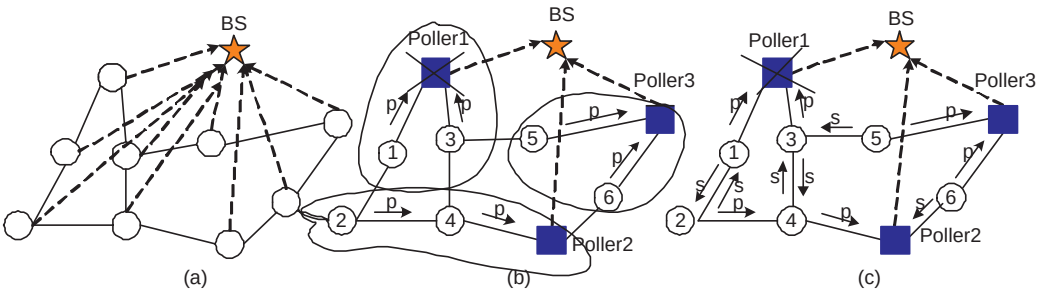


Figure 4.2. Different monitoring infrastructures: (a) centralized (b) distributed (c) distributed and fault tolerant, where circles denote pollees, squares denote pollers, arrows denote the paths followed by packets, p-arrow represents the path to the primary poller and s-arrow represents the path to the secondary poller. Each solid line denotes a physical link, and each dashed line represents a logical path that may consist of multiple physical links.

which is the secondary.

The communication path between the pollee and the sink consists of two parts: pollee-to-poller and poller-to-sink. In this chapter, we focus on the reliability issue of the pollee-to-poller part and leave the other part as future work.

4.2.3 Problem Formulation

We consider a network of n sensors, where all sensors are capable of being either pollers or pollees. At first hand, we want to select the minimum number of nodes as pollers so that the management cost of pollers can be minimized. On the other hand, if the number of pollers is too small, some pollees will be many hops away from a poller, thus increasing the false alarm rate. Therefore, our goal is to strike a balance between the number of pollers and false alarm rate. Since the pollers may also fail, we associate each pollee with $\omega \geq 1$ pollers. Each pollee maintains pointers to the different pollers but sends status report to only one poller at a time. When the poller fails, the associated pointers should be outdated and the next poller on the list will be used.

We formulate the poller-pollee assignment problem *minPL*, with the objective to minimize the number of pollers while limiting the maximum false alarm rate as follows.

Given a network graph, the error rate of each link, determine (1) which nodes

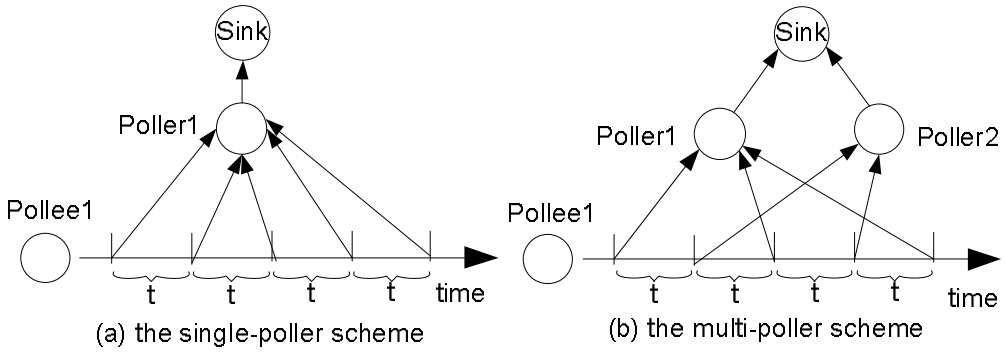


Figure 4.3. Asynchronous operation at poller and pollee, with the arrow denoting the status reporting

are pollers and which nodes are pollees, (2) a many-to-many mapping where each node is associated with ω pollers, to minimize the total number of pollers while limiting the maximum false alarm rate of pollees.

Theorem 9. *The problem* minPL *is NP-Hard.*

Proof. The problem minPL can be proved to be NP-hard via a reduction from the minimum k-hop dominating set problem [54], which can be seen as a special case of minPL when $\omega = 1$ and the link failure rate is constant. This is because according to the result in the next section, different hop numbers between pollee and poller correspond to different false alarm rates. Therefore, satisfying the constraint of maximum false alarm rate is equivalent to limiting the maximum number of hops from pollee to poller. □

4.3 The Round Robin Based Multi-Poller Scheme and Its Performance analysis

In this section, we present the round robin based multi-poller scheme, and give the false alarm rate through analysis.

4.3.1 The Round Robin Based Multi-Poller Scheme

Recall that the multi-poller structure is proposed to deal with sensor and communication link failures. To reduce the communication overhead, we propose a round

robin based multi-poller scheme, where the monitored sensor sends the status report to different pollers in a round robin manner. Status reports from different pollers can thus be combined to reduce false alarm rate. For example, even the report towards one poller is lost due to communication link failure during the current polling interval, the report may still reach another poller through a different link in the next polling interval. Based on the combined reports through different pollers, the sink can thus make a better decision about the pollee status every detection period.

Further let t denote the polling time interval at the pollee, and let ω denote the number of pollers for each pollee. Suppose the sink needs time T_d, referred to as the detection time, to detect a pollee failure. The single-poller and multi-poller scheme can be stated as follows:

- *Single-poller scheme*: each pollee sends a report to the same poller every t; each poller collects the reports from its pollees every t and forwards the compressed report to the sink every t; the sink evaluates the failure condition every T_d. It is required that $t \leq T_d$.

- *Multi-poller scheme*: each pollee schedules a report to different pollers every t; each poller collects reports from each of its pollees every ωt and forwards the compressed report to the sink every ωt; the sink evaluates the failure condition every T_d. It is required that $\omega t \leq T_d$.

Status reports can be aggregated during the transmission from the pollee to the poller. The poller, pollees, and the physical link between them form a tree. If a node is at the edge of the tree, it is called an *edge node*; otherwise it is called a *non-edge node*. Then in the above polling schemes, each non-edge pollee can collect reports from each of its children and sends the aggregated report (including its own) to the same poller every t;

Fig. 4.3 illustrates the single poller scheme and multi-poller scheme using an example. In Fig. 4.3(a), pollee 1 sends reports to poller 1 every t; in Fig. 4.3(b), pollee 1 sends reports to poller 1 or poller 2 every $2t$ in a round robin manner. The sink receives two reports every $2t$ if it chooses detection time $T_d = 2t$. With the single poller scheme, if some communication link between pollee 1 and the sink has

some problem, the sink will have a false alarm on the status of pollee 1. However, the false alarm may be eliminated by the use of an alternative poller in the multi-poller scheme. In general, with the same detection time delay and the same amount of traffic, the multi-poller scheme will have a much smaller false alarm rate on the status of the pollee. It is possible that T_d can be set to t in the single poller scheme in the above example. Although this can reduce the failure detection time, based on the results of [55] and our analytical results in the next section, the false alarm rate with this change can be very high compared to increasing T_d to $2t$. Next, we give formal analysis on the false alarm rate of these two schemes.

4.3.2 The False Alarm Rate

In this section we assume the detection time period T_d is the same for both single-poller and multi-poller scheme, and evaluate the false alarm rate in the presence of link failures.

In wireless medium, the characteristics of node failure and link failure are quite different. While the node failure may be permanent, the wireless link failure happens from time to time, and once occurs it will persist for a period of time. During the transmission of messages, if an intermediate node is detected to fail, its downstream nodes can simply bypass it by choosing an alternative next hop. But the same technique cannot be applied to the case of link failure, which is transient in nature. Therefore we study the link failure and model it as a continuous-time Markov chain, as adopted by some existing works [56, 57]. The following notations are used in the analysis:

f_l: link failure rate

h: number of hops from pollee to poller

$F_1(h, T_d), F_w(h, T_d)$: the false alarm rate for the single poller scheme, and the multi-poller scheme, where the subscript w represents the number of pollers for each pollee.

Each link is modeled as a two state markov chain (Fig. 4.4(a)). The transition rate from state 0 ("on") to state 1 ("off") and from state 1 to state 0 is λ and μ respectively. If the path between the poller and pollee is multihop, it can be modeled by the higher dimensional markov chain. For example in Fig. 4.4(b), the

two-hop path (i.e., $h = 2$) is represented by four states with "11" denoting the state at which both links fail. Further assume each pollee has ω pollers ($\omega \geq 1$), and a packet is sent to each poller every ωt period. We are interested in the false alarm rate $F_w(h, T_d)$, given h and the detection period T_d.

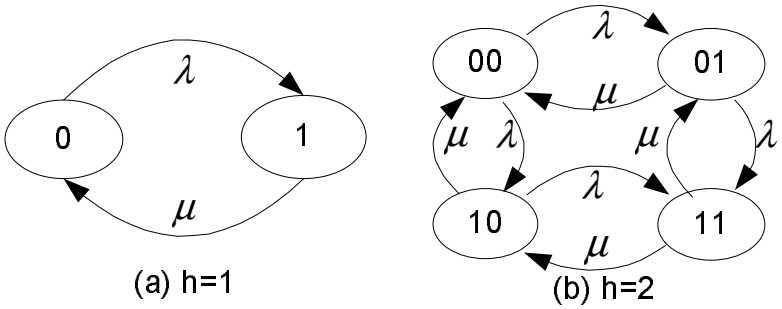


Figure 4.4. Continuous-time markov chain link model

- The continuous-time markov chain of a link (Fig. 4.4a): To solve the markov chain in Fig. 4.4(a), its limiting probability π_0, π_1, i.e., the long-run proportion of time that the process is in state 0 and state 1, can be calculated by $\pi_0 = \frac{\mu}{\lambda+\mu}, \pi_1 = \frac{\lambda}{\lambda+\mu}$. The transition probability $p_{ij}(t)$, the probability that the state is in j at time t provided that the current time is i, is given by:

$$p_{00}(t) = \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)t} + \frac{\mu}{\lambda+\mu} \tag{4.1}$$

$$p_{10}(t) = \frac{\mu}{\lambda+\mu} - \frac{\mu}{\lambda+\mu} e^{-(\lambda+\mu)t} \tag{4.2}$$

- The multihop path with single poller (Fig. 4.4b): Without loss of generality, we assume T_d/t is an integer and discretize the time using interval t. Then $F_1(h, T_d)$ can be calculated as follows.

  1. When $T_d = t$,

$$F_1(h, t) = 1 - \pi_0(t)^h, \text{ for } \forall h \tag{4.3}$$

  2. When $T_d = 2t, h = 2$,

$$\begin{aligned} F_1(2, 2t) = {} & \pi_1\pi_0(1 - p_{10}(t)p_{00}(t)) \\ & + \pi_0\pi_1(1 - p_{00}(t)p_{10}(t)) \end{aligned}$$

$$+\pi_1\pi_1(1-p_{10}(t)p_{10}(t)) \tag{4.4}$$

Above calculation numerates the three cases of false alarm in which the two-hop path fails in both t intervals.

3. When $T_d = 2t, h = 3$,

$$F_1(3,2t) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1}\pi_i\pi_j\pi_k(1-p_{i0}(t)p_{j0}(t) \\ p_{k0}(t)) - \pi_0^h(1-p_{10}(t)p_{10}(t)p_{10}(t)) \tag{4.5}$$

4. Eqn. 4.5 can be generalized to calculate $F_1(h,2t)$

5. In general, $F_1(h,T_d)$ can be calculated by

$$\begin{aligned} & F_1(h,t) \quad (P[s(t+t_0)=1|s(t_0)=1])^{\frac{T_d}{t}-1} \\ = \; & F_1(h,t) \quad \left[\frac{(P[s(t+t_0)=1,s(t_0)=1])}{P[s(t_0)=1]}\right]^{\frac{T_d}{t}-1} \\ = \; & F_1(h,t) \quad \left[\frac{F_1(h,2t)}{F_1(h,t)}\right]^{\frac{T_d}{t}-1} \end{aligned} \tag{4.6}$$

where $s(t_0)$ denotes the path state at t_0, $P[s(t+t_0)=1|s(t_0)=1]$ is the transition probability, and $F_1(h,t)$, $F_1(h,2t)$ are given by 1), 4). Above follows because the markov chain is homogeneous, i.e., its transition probability does not depend on the starting time.

- Multihop path with multi-poller: Unlike the single poller scheme, the failure of the multihop paths towards different pollers are independent. Therefore $F_\omega(h,T_d)$ is given by:

$$F_\omega(h,T_d) = (1-\pi_0(t)^h)^{T_d/t} \tag{4.7}$$

Because of the bursty nature of the wireless link failures, the monitoring of sensor network could benefit from the multi-poller based scheme. The amount of the benefits depends on the time correlation characteristics of the link failures. On one extreme, if the link failures are not time-correlated, the performance of

the multi-poller based scheme is simply reduced to that of the single-poller based scheme. The correlation characteristics of the transmission failures is studied in [58] which uses the statistical tool to analyze the collected packet traces over the real sensor testbed. In [58], wireless links are classified into three categories based on the correlation characteristics, i.e., good link, intermediate link, and bad link, where the bad links denotes the most correlated link. For our evaluation, we consider the link quality as the intermediate level and set $f_l = 0.1, \lambda = \mu = \frac{1}{4}$ in the markov chain model. For the bad link, the performance benefits should be higher.

Fig. 4.5 shows the false alarm rate as a function of the number of hops from poller to pollee when $f_l = 0.1$. As can be seen, the false alarm rate increases as h increases, since longer path is more vulnerable to failure. From the figure, we can also see the importance of T_d. As T_d increases, the false alarm rate decreases. For example, even for the single poller scheme, when $h = 1$, $F_1(h, t) = 0.1$, but $F_1(h, 2t) = 0.025$; that is, doubling the detection time can reduce the false alarm rate by as much as 75%. With a longer detection time, the sink can still receive the next status report even if the previous status report has been lost. As a result, the sink will have much less probability to generate a false alarm due to link failure. This is consistent with the result of [55].

From the figure, we can see that the multi-poller scheme consistently outperforms the single poller scheme in terms of false alarm rate. For example, with $h = 1$, $F_1(h, 2t) = 0.025$, but $F_2(h, 2t) = 0.01$, which represents a 60% false alarm reduction. This is because the status report of the multi-poller scheme is transmitted to the sink through different paths, and hence can tolerate longer period of link failure at any path. Note that, due to the use of round robin scheme, the multi-poller scheme still keep similar bandwidth consumption as the single poller scheme.

Fig. 4.6 shows the false alarm rate when $f_l = 0.2$. We can easily see the same trend as that of Fig. 4.5. The only difference is that the false alarm rate increases as the f_l increases.

Discussion: To simplify the analysis, we assume the average link failure rate is homogeneous. In some cases, different links could have large variation of quality. Thus, there may be a need to introduce adaptive elements to the polling strategy

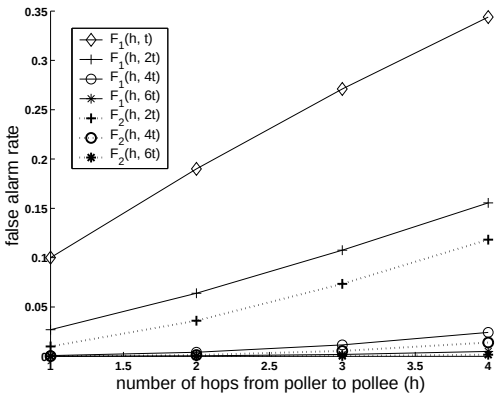


Figure 4.5. False alarm rate ($f_l = 0.1$)

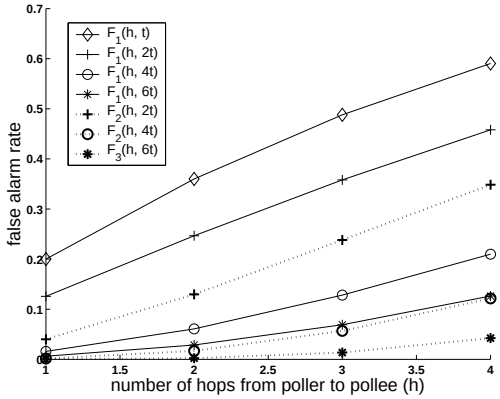


Figure 4.6. False alarm rate ($f_l = 0.2$)

and design weighted round robin based polling schemes. Note that our analysis is the first step to fully understand the round robin based multi-poller scheme. As the initial step, our analysis is based on a 2-state Markov model, but more complex and realistic models such as Weibull model [59, 60] are also worth of further investigation.

4.4 Distributed Poller-Pollee Assignment Algorithms

In this section, we propose the distributed algorithms to solve the poller-pollee assignment problem as formalized in subsection 4.2.3.

The construction of poller-pollee structure shares some similarity with the traditional clustering scheme, where a poller is similar to a cluster head. However, there is fundamental difference between them. First, the traditional clustering schemes are single-hopped, but the pollee should be within some bounded hops of its poller. Second, with multihops between the poller and pollees, aggregation is used to reduce the monitoring traffic, which is not considered in clustering schemes. Third, each pollee may be associated with $\omega \geq 1$ pollers to be fault tolerant, whereas each cluster member only has one cluster head. Thus, the traditional clustering scheme is only a special case of the single-hop poller-pollee structure with $\omega = 1$. Below, we first propose a distributed deterministic poller selection algorithm, and

then present a hybrid algorithm to further reduce the message overhead.

4.4.1 The Randomized Algorithm

The randomized algorithm is presented as a baseline for comparison. Each node elects itself as a poller with probability ρ. Pollers then announce their poller status within k hops. Sensor nodes that did not elect themselves as pollers will be pollees. The randomized algorithm is very simple, yet it may produce some pathological scenario where multiple pollers may cluster together in some area and no poller exists in some other area. To address this problem, we propose a deterministic algorithm.

4.4.2 The Deterministic Algorithm

Algorithm 7 Deterministic Poller Selection Algorithm

Input: a graph $G(N, E), k_1, k_2$
Output: a Poller Set S_{er}, a Pollee Set S_{ee}
Procedure: $Determine(k_1, k_2)$

```
1: Initialize the status and the timer
2: Broadcast locally to get k1-hop neighborhood information
3: if timer not expired then
4:     if id is the smallest among k1 hop unlabeled neighbors then
5:         broadcast pollerID = id within k2 hops, exit
6:         /*S_er = S_er ∪ {id}*/
7:     end if
8:     wait until a packet is received or the timer is expired
9:     if pollerID received then
10:        broadcast polleeID within k1 hops, exit
11:        /*S_ee = S_ee ∪ {id}*/
12:    end if
13:    if polleeID received then
14:        update the unlabeled List within k1 hops, reset timer and go to 4
15:    end if
16: else
17:    go to 5
18: end if
```

The proposed deterministic algorithm is based on the distributed *maximal independent set (MIS)* algorithm [13]. An *Independent Set* is a subset of nodes among which there is no edge between any two nodes. The set is a MIS if no more edges can be added to generate a bigger independent set. In the deterministic algorithm,

the concept of MIS is extended to the multihop environment. Two parameters k_1, k_2 are used to govern the distribution of the pollers and pollees, to ensure that no two pollers are less than k_1 hops from each other and no pollee is more than k_2 hops from its poller. That is, the poller set S_{er} is a k_1-hop MIS, in which no two nodes are less than k_1 hops away from each other.

Since parameters k_1, k_2 control the geometrical properties of the poller and pollee distribution, they should be determined beforehand according to user's demand. First, given the constraint of false alarm rate, k_2 can be determined, based on the relationship between the false alarm and the hop distance such as in Fig. 4.5. Thus, bounding the maximum false alarm rate becomes equivalent to bounding the maximum distance from pollee to the poller. Second, after k_2 is determined, k_1 can be selected. Although a large k_1 could reduce the number of pollers selected but some pollee may not find ω pollers within k_2 hops. Therefore, an appropriate k_1 should be chosen to strike a balance between the number of pollers and the number of unlabeled nodes. This can be achieved by the experiments, as in Section 4.6.

Given k_1, k_2, Algorithm I lists the pseudo code of the deterministic poller selection algorithm. At the outset, each node needs to obtain the k_1-hop neighborhood information by a localized broadcast. Then the algorithm proceeds in rounds. In each round, if a node has the smallest id among the k_1-hop unlabeled neighbors, it elects itself as belonging to the poller set S_{er}, and broadcasts $pollerID = id$ within k_2 hops. Nodes that are not yet labeled but received the poller declaration will label themselves as pollees, and broadcast $polleeID = id$ within k_1 hops. After that, a new round will start, during which the algorithm is executed among the remaining unlabeled nodes. This process repeats until all nodes are labeled either as poller or pollee.

The upper diagram of Fig. 4.7 uses a small network of 11 nodes to explain the deterministic algorithm, assuming $k_1 = k_2 = 1$. In the first round, node 1 and node 3 elect themselves as poller since their id is the smallest among the 1-hop neighborhood, and their 1-hop neighboring nodes 2,4,6,7,8,10 are recruited as pollees. Then the labeling process will repeat among the remaining nodes until everyone is labeled. Thus, in the second round, node 5 is labeled as poller and node 9 is labeled as pollee. In the third round, node 11 is labeled as poller.

The message complexity of Algorithm I can be computed as follows. Assume

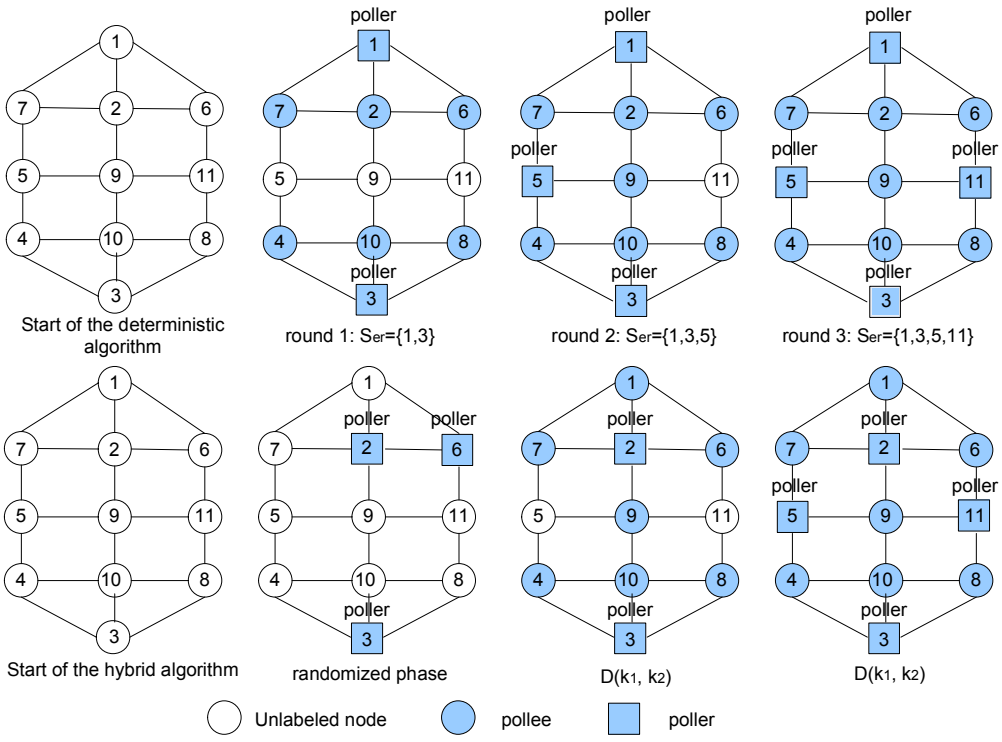


Figure 4.7. A numerical example. The deterministic algorithm (above) runs in three rounds, exchanging 22 messages. The hybrid algorithm (below) has a randomized phase and two deterministic phases, exchanging 10 messages.

there are n nodes. At first, each node needs to broadcast within k_1 hops to get the neighborhood information (line 2). After that, each poller needs to broadcast within k_2 hops (line 5) and each pollee needs to broadcast within k_1 hops (line 10). Suppose $k = k_1 = k_2$, each node broadcasts within k hops exactly twice during the execution of the algorithm. Further suppose d is the node density, and r is the node communication range. The message complexity can be estimated by $O(2 * n * d * (kr)^2) = O(k^2nd)$.

It is possible to set k_1 as a fractional number. For example, when $k_1 = 1.2$, it means that each node chooses $\lfloor k_1 \rfloor = 1$ with probability 0.8 and chooses $\lceil k_1 \rceil = 2$ with probability 0.2. As k_1 controls the distance between the neighboring pollers, allowing k_1 to be a fractional number will provide more flexibility to control the number of pollers, which can be seen from Section 4.6. Then, Algorithm 1 needs to be extended as follows. First, set ${k_1}' = \lfloor k_1 \rfloor$ with probability $\lceil k_1 \rceil - k_1$ and set ${k_1}' = \lceil k_1 \rceil$ with probability $k_1 - \lfloor k_1 \rfloor$. Second, each node elects itself as poller

among the $k_1{}'$-hop unlabeled neighbors (line 4). Third, each node broadcasts id within $\lceil k_1 \rceil$ hops (line 5,10), but updates the unlabeled list within $k_1{}'$ hops (line 14).

4.4.3 The Hybrid Algorithm

Algorithm 8 Hybrid Poller Selection Algorithm

Input: a graph $G(N, E)$, k_1, k_2, ρ
Output: a Poller Set S_{er}, a Pollee Set S_{ee}
Procedure: $Hybrid(k_1, k_2)$
1: Initialize the node label, timer, $S_{er} = \phi, S_{ee} = \phi$
2: generate a random number $\sigma \in (0, 1)$
3: **if** $\sigma < \rho$ **then**
4: $S_{er} = S_{er} \cup \{id\}$
5: **end if**
6: **if** $id \in S_{er}$ /*node id is temporarily labeled as poller*/ **then**
7: execute the deterministic algorithm $Determine(k_1, k_2)$
8: /*S_{er} and S_{ee} are updated*/
9: **end if**
10: **if** node id is unlabeled **then**
11: wait until a packet is received or the timer is expired
12: **if** $pollerID$ received **then**
13: label itself as pollee /*$S_{ee} = S_{ee} \cup \{id\}$*/
14: **end if**
15: **end if**
16: **if** $id \in \{N - S_{er} - S_{ee}\}$ /*node id is not labeled*/ **then**
17: execute the deterministic algorithm $Determine(k_1, k_2)$
18: **end if**

In the deterministic algorithm, each node needs to broadcast within k hops twice, assuming $k = k_1 = k_2$. As k becomes larger, the message complexity increases dramatically. To reduce this message overhead, we propose a hybrid algorithm which combines the randomized algorithm and the deterministic algorithm. As shown in Algorithm 8, the algorithm starts with a randomized phase, during which each node labels itself as a temporary poller with a probability ρ. After that, the deterministic algorithm is executed among the temporary pollers. If two pollers are less than k_1 hops away from each other, one of them will change its role from poller to pollee and the other one will confirm itself as poller. After receiving the confirmed $pollerID$, the unlabeled nodes within k_2-hop range of the confirmed poller will be recruited as pollees. To implement this, a field in the packet header needs to be reserved to differentiate the confirmed $pollerID$ from the temporary

$pollerID$. After receiving the confirmed $pollerID$, the unlabeled nodes change the status to pollee. Finally, the deterministic algorithm is executed among the set of unlabeled nodes, i.e., $\{N - S_{er} - S_{ee}\}$, to have all the nodes labeled.

Compared with the deterministic algorithm, the hybrid algorithm has much less message overhead. First, due to the use of the randomized phase, each node does not have to make the initial $k_1 - hop$ broadcast. Second, when the deterministic algorithm is executed for the first time (line 7), only the temporary pollers are involved. Third, after receiving the confirmed $pollerID$, many unlabeled nodes become pollees who are refrained from broadcast. Thus, only a small portion of the nodes still remain unlabeled and execute the deterministic algorithm for the second time (line 17). In the following, we give a detailed analysis of the message overhead, based on which the optimal value of ρ can be derived to minimize the message overhead.

For ease of illustration, we first assume $k_1 = k_2 = k$, and then extend it to the general case of $k_1 \neq k_2$. First, the randomized phase introduces zero message overhead. As a result, on average a fraction ρ of the nodes are labeled as pollers and execute the deterministic algorithm for the first round (line 7). Among the $n\rho$ temporary pollers, a fraction of nodes, say, α, are confirmed as pollers, and a fraction $1 - \alpha$ of the nodes become pollees. During the execution of the first-round deterministic algorithm, the $n\rho\alpha$ confirmed pollers will recruit unlabeled nodes within k hops as pollee. After that, only a fraction, say, β, of the $n(1 - \rho)$ unlabeled nodes still remain as unlabeled and execute the second round (line 17). Therefore, the total number of nodes involved in the two rounds of the deterministic algorithm is $n\rho + n(1-\rho)\beta$, with all the other nodes recruited as pollee by the $n\rho\alpha$ confirmed pollers. Further use $M(k)$ to denote the message complexity of k-hop broadcast. Since each participating node needs to broadcast twice within k hops, the overall message complexity is:

$$2[n\rho + n(1 - \rho)\beta]M(k) \tag{4.8}$$

β is the fraction of $n(1-\rho)$ unlabeled nodes that do not fall in the k-hop range of any of the $n\rho\alpha$ confirmed pollers. For a node n_1, the probability that it falls within the k-hop range of node n_2 is $\frac{\pi k^2 r^2}{a^2}$, where a^2 denotes the area size. Thus,

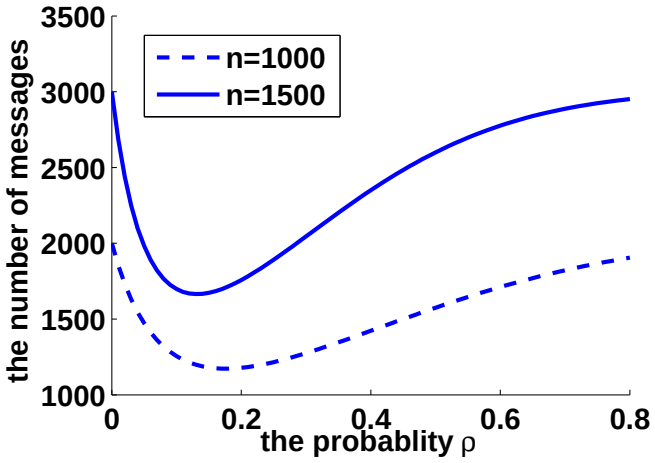


Figure 4.8. The relationship between probability ρ and the message overhead

$$\beta = (1 - \frac{\pi k^2 r^2}{a^2})^{n\rho\alpha} \tag{4.9}$$

α denotes the fraction of nodes that are to be labeled as poller if $n\rho$ uniformly distributed nodes run the deterministic algorithm. When two nodes are within k hops, one of them will refrain from being a poller with half chance. Since there are total $n\rho$ nodes, the probability for each node to become a poller is

$$\alpha = (1 - \frac{\pi k^2 r^2}{2a^2})^{n\rho - 1} \tag{4.10}$$

Substituting Eqn. 4.10 and Eqn. 4.9 into Eqn. 4.8 will give us the message complexity when $k_1 = k_2$.

Fig. 4.8 numerically evaluates the effect of varying the number of nodes n and the probability ρ on the amount of message overhead, when $a = 20, r = 1, k_1 = k_2 = 1$. It is shown that each curve has an optimal ρ corresponding to the minimum message overhead. In addition, the optimal value of ρ reduces as the number of nodes n increases. For example, ρ is about 0.2, 0.15, when $n = 1000, 1500$, respectively.

Fig. 4.7 uses a simple network to compare the deterministic algorithm and hybrid algorithm. In the hybrid algorithm (lower diagram), the randomized phase selects three nodes, i.e., nodes 2,3,6, as temporary pollers, among which the deterministic algorithm is executed. Node 6 resigns from the role of poller and becomes

pollee because its *id* is larger than its neighbor node 2, which is also a poller. As a result, nodes 2,3 are confirmed as pollers and broadcast locally to recruit nodes 1,7,9,4,8,10 as pollees. After that, only nodes 5,11 still remain unlabeled, among which the second round of the deterministic algorithm is executed. In comparison of the message overhead, the deterministic algorithm has $2 \times 11 = 22$ message exchanges, but the hybrid algorithm only has $2 \times (3+2) = 10$ message exchanges. The message complexity is reduced by over 50% in this example. As k_1, k_2 become larger and the network grows to a larger scale, the message saving will be more significant.

After pollers are determined, each pollee needs to select up to ω pollers based on the received announcements. The selection could be simply based on criteria such as the distance to the poller. Each pollee maintains a list of pointers to the ω pollers, and each pointer links to the routing entry (e.g., distance, next hop) of a distinct poller. After a poller fails, each pollee will update its active poller list, remove the stale entry and choose the next poller on the list. As more pollers fail, new pollers can be elected using the algorithm in the next subsection.

4.4.4 Poller Re-election In Case Of Failure

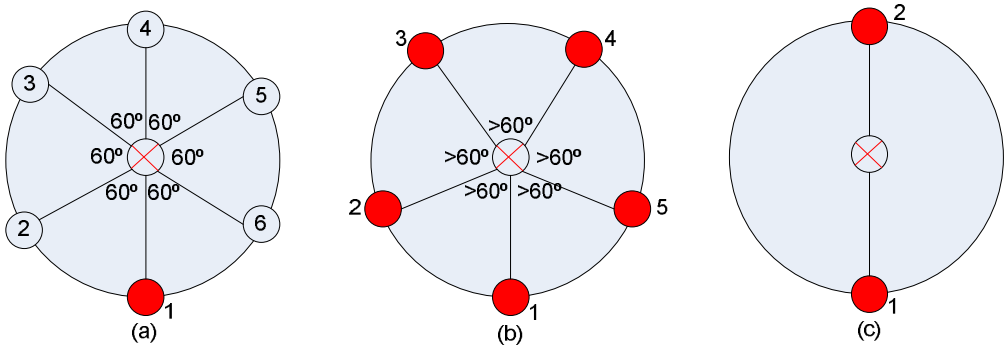


Figure 4.9. The candidate set of new pollers can be reduced to a few temporary pollers. In the diagram, the centered node is the failed poller, the other nodes are its neighboring pollees, and the circle delineates its neighborhood. (a) the best case where only one node (with the smallest id) is elected as the temporary poller. (b) the worst case where up to five temporary pollers can be elected. (c) two temporary pollers may not be able to reach one another in two hops due to the lack of an intermediate node.

Algorithm 9 Poller Re-election Algorithm

```
    /*For each neighboring pollee of the failed poller with id_fail*/
 2: if a pollee has the smallest id among its neighbors then
        label itself as temporary poller
 4:     broadcast Msg(id_fail, id_temp = id, null, hop = 2)
        while timer not expired do
 6:         receive Msg(id_fail, id_temp, id_list,0)
            if id_temp < id then
 8:             resign from poller and label itself as pollee
            end if
10:     end while
        if confirmed as a poller when the timer expires then
12:         announce the poller status within the k hops of the failed poller
        end if
14: else
        /*keep its status as pollee*/
16:     while timer not expired do
            receive Msg(id_fail, id_temp, id_list, hop)
18:         if some temporary poller among its neighbors is not listed in id_list and hop > 0 then
                append the id of the unlisted temporary poller to id_list
20:             broadcast Msg(id_fail, id_temp, id_list, hop − 1)
            end if
22:     end while
    end if
24: /*For each neighboring poller of the failed poller with id_fail*/
    broadcast Msg(id_fail, id_temp = −1, null, hop = 2)
```

As more pollers fail, some pollee may not be able to find ω pollers within k hops, so pollers need to be reelected. The reelection can be initiated by the sink, because it knows the status of each node. Alternatively, it can be initiated by the neighbor of the failed poller, to localize the reelection process. The pollee can monitor the link status of its neighboring poller through overhearing at the MAC layer, without adding extra overhead. For example, if the pollee does not hear any transmission from the neighboring poller within a time period (e.g., T_d), it infers that the poller fails. Then, a new poller will be elected among the 1-hop neighbors of the failed poller based on Algorithm 2.

Algorithm 2 is executed among the neighbors of the failed poller - nodes outside of this range are not involved. Among the neighboring pollees of the failed node, the pollees with the smallest *id* among their neighbors will first label themselves as temporary pollers (line 2-3), from which a new poller will be selected. In this way, the candidate set can be reduced from many possible neighbors to a few temporary pollers. Figs. 4.9(a)(b) show that there are at least one and at most five

temporary pollers regardless of the total number of nodes involved. In Fig. 4.9(a), each pollee has two neighboring pollees, but only node 1 has the smallest id among its neighbors, thus being elected as the temporary poller. However, in Fig. 4.9(b), where no two pollees are neighbors to each other, five temporary pollers are elected. These temporary pollers can further negotiate with each other to decide the final poller.

Each temporary poller starts a 2-hop constrained broadcast among the neighbors of the failed node (line 4). The format of the broadcast message is defined as Msg(id_{fail}, id_{temp}, id_{list}, hop), where id_{fail} is the id of the failed poller, id_{temp} is the id of the temporary poller that generates the message, id_{list} records the id of the other temporary pollers that have received this message, and hop defines the broadcast range. When a temporary poller receives the message from another poller, it examines the field id_{temp}. If id_{temp} is smaller than its own id, it will resign from the role of temporary poller and label itself as pollee (line 7-9). When a pollee receives the message, it checks the field id_{list}. If there is some neighboring temporary poller that is not listed in id_{list}, the pollee will continue to forward the message (line 18-21). Otherwise, it drops the message. Since the message is circulated among the neighbors of the failed node, we set $hop = 2$. In most cases, this can ensure the message to be delivered to the other temporary pollers, so that there is only one temporary poller (i.e., with the smallest id) to be confirmed. However, in some sparse networks such as Fig. 4.9(c), the message may not be able to reach other temporary pollers due to the lack of intermediate node, so there may be multiple pollers to be confirmed.

After the new poller is confirmed, it initiates a constrained broadcast within k hops of the failed poller (line 11-13). Based on the received message, each pollee can update the list of the active pollers and renew the pointers to them.

On the other hand, if the failed poller has some other alive poller as a neighbor, there is no need to reselect a new poller among the pollees. The alive poller can simply broadcast a message with $id_{temp} = -1$ to invalidate the status of the temporary pollers (line 25).

4.5 The Optimal Aggregation Path Problem

4.5.1 The Problem Formulation

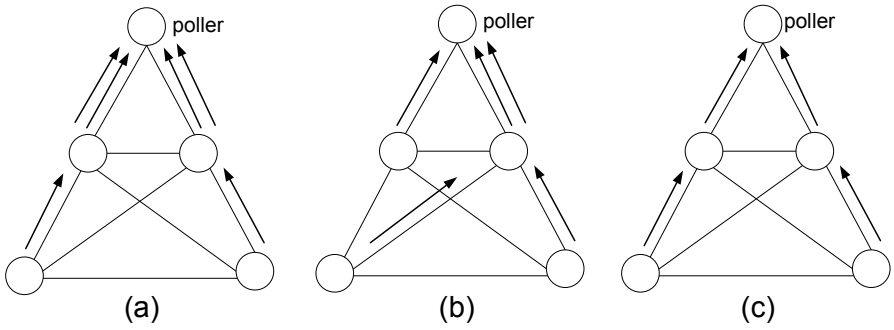


Figure 4.10. An example to illustrate the benefit of aggregation: (a) without aggregation (b) aggregation path I (c) aggregation path II, where each line denotes a physical link, and each arrow denotes a packet transmission over one hop (packet-hop)

After the poller-pollee relationship is established, each pollee starts to periodically send a report to its poller. Along the multihop path from pollee to poller, each non-edge pollee collects reports from its downstream children and sends an aggregated report (including its own) to the poller. For example in Fig. 4.10, a pollee may have multiple paths to the poller, which results in different energy efficiency. Suppose the *aggregation ratio* (s) of the nodes in Fig. 4.10 is 2, which is defined as the number of status reports that can be aggregated into one packet. Each of the four pollees sends a report to the same poller periodically. Fig. 4.10(a) shows that without aggregation, 4 packets will be needed for each period, with a cost of 6 packet-hops. If the reports are aggregated along the path shown in Fig. 4.10(b), all 4 reports can be packed into 3 packets, with a cost of 5 packet-hops. If the aggregation path of Fig. 4.10(c) is used, only 2 packets are needed, with a total of 4 packet-hops.

Assume transmitting one packet over one hop consumes one unit of energy. The problem of finding the optimal aggregation path ($optPH$) can be formulated as follows.

In a network graph, given the poller-pollee relationship, a constant aggregation ratio, each node needs to send a report to its poller periodically. For each poller,

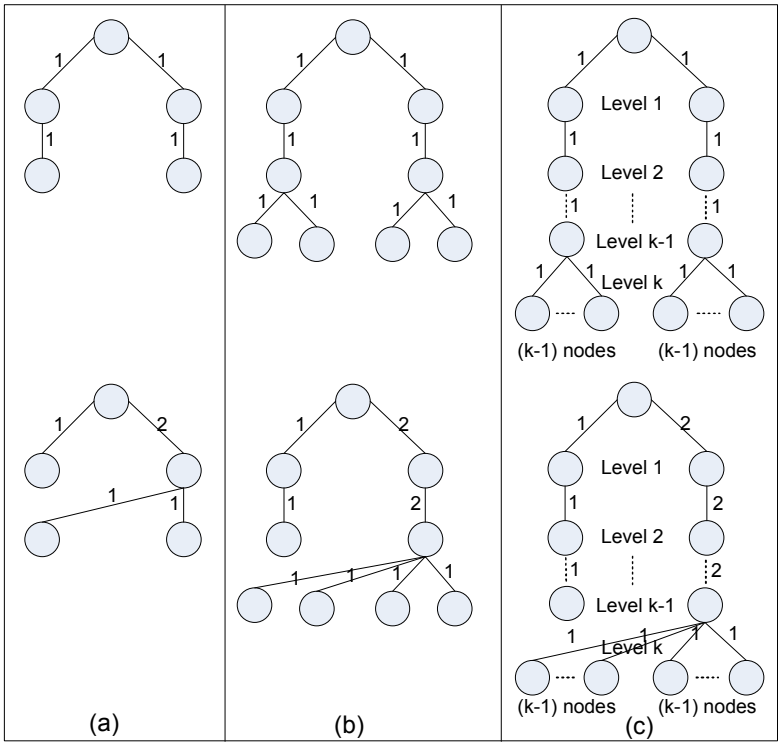


Figure 4.11. Worst-case topology with the best strategy (above) and worst strategy (below). (a) 2-level connected tree, $s = 2$. (b) 3-level connected tree, $s = 4$. (c) k-level connected tree, $s = 2(k - 1)$.

find the optimal aggregation paths from all its pollees, such that the total energy consumption is minimized.

When the aggregation ratio s is infinite, it is called perfect aggregation. That is, infinite number of reports can be aggregated into one packet. It is not hard to see that the optimal solution of this special case is the minimum spanning tree. For the general case where the aggregation ratio is finite, it can be proved that the problem of optPH is NP-hard, which is consistent with the findings in [61, 62]. The proof is given in Appendix B.

Theorem 10. *The problem of optPH is NP-hard.*

4.5.2 A Greedy Algorithm And Its Approximation Ratio

Although the problem of selecting the optimal aggregation path is NP-hard, we can design some heuristics. A simple greedy algorithm is to arbitrarily select the

next hop to forward, as long as it is on the shortest path to the poller, and opportunistically aggregate the status reports at the intermediate nodes. This seemingly simple algorithm has surprisingly good performance, with a tight bound of $\frac{5}{4}$. That is, the energy consumption resulted from the greedy algorithm will not be more than $\frac{5}{4}$ times the optimum solution. Before proving the approximation bound of the greedy algorithm for the optPH problem, we first define some terminologies.

Within a polling domain, the shortest paths from all the pollees to the poller of the domain form a tree, which may be partitioned into different levels. A node is said to be at *level* i ($i \geq 0$) if it is i hops away from the sink (i.e., the poller). Therefore, the sink is the only node at level 0. A rooted tree is called a *level-connected tree* if the nodes belonging to the neighboring levels are connected with each other. Specifically, a k-level-connected tree is composed of the nodes at level-i, $i = 0, 1...k$. Given a tree topology, the set of aggregation paths that consume the least/most energy is called the *best/worst paths*, denoted as H_b and H_w, respectively. Correspondingly, the aggregation strategy that can produce the best/worst paths is called the *best/worst strategy*. Given a topology, the ratio of energy consumption between the worst and best paths determines the *performance bound/ratio* of the topology, denoted as δ. Denote the energy consumption of the best path H_b and worst path H_w as E_b, E_w, then we have $\delta = \frac{E_w}{E_b}$. Given an algorithm, the maximum performance bound/ratio among all possible topologies, denoted as δ_{max}, is called the *approximation ratio* of the algorithm. The *worst-case topology* is the topology whose performance bound equates the approximation ratio of the algorithm.

To have an intuitive idea about how to calculate the performance bound, we first show some topologies in Fig. 4.11, whose performance bound is $\frac{5}{4}$. We can later prove that $\delta = \frac{5}{4}$ is the maximum performance bound among all topologies, so these classes of k-level-connected tree are actually the worst-case topology. In Fig. 4.11, the worst-case k-level connected tree is constructed by having each level-$i, i = 1...k-1$ with 2 nodes and level-k with $2(k-1)$ nodes, where Figs. 4.11(a)(b) are the special cases when $k = 2, 3$ and Fig. 4.11(c) is the general case. Note that in each level-connected tree, nodes at the neighboring levels are all connected with each other. But for the clarity purpose not all the links are shown in Fig. 4.11, with only the aggregation paths depicted. The number shown by each link indicates

the number of packet transmissions over that link. In Fig. 4.11, the upper diagram represents the best aggregation paths, and the lower diagram represents the worst aggregation paths. A further calculation shows that the energy consumptions of the best and worst paths are $4(k-1)$ and $5(k-1)$ in Fig. 4.11(c). With $s = 2(k-1)$, we have $\delta = \frac{5}{4}$.

In the rest of this section, we will prove that $\delta = \frac{5}{4}$ is the maximum performance bound among all topologies, and thus the approximation ratio of the greedy algorithm. As all the shortest paths constitute a tree and any tree topology is a subset of some level-connected tree, we can restrict our attention to the level-connected tree. The following theorem gives the approximation ratio for the 2-level-connected tree, which will be used to derive the approximation ratio for the arbitrary level-connected tree.

Theorem 11. *For a 2-level-connected tree, the approximation ratio of the opportunistic greedy algorithm is no greater than* $1 + \frac{s-1}{s^2}$*, where* s *is the aggregation ratio.*

Proof. Given a 2-level-connected tree, the energy spent by sending the reports from the level-2 nodes to the level-1 nodes is fixed, i.e., equal to the number of level-2 nodes. The problem optPH becomes how to minimize the energy spent between the level-1 nodes and the root by packing the level-2 reports into a minimum number of level-1 packets. We give the *best* and *worst* aggregation strategy as follows.

Best strategy: packing/sending every $s-1$ level-2 reports into a single level-1 node until all level-1 nodes have a packet of s (including its own) reports constructed without residue room left. All the remaining level-2 reports are then packed into a single level-1 node.

Worst strategy: packing/sending all level-2 reports into a single level-1 node.

Excluding its own report, each level-1 node has $s-1$ vacancies in the first packet. The best strategy tries to fully utilize all the vacancies, while the worst strategy tries to use as many packets as possible by packing all status reports aggressively into a single node. There may be other alternative best or worst strategies, but it is sufficient to find one of them for our proof. Taking Fig. 4.12(b) as an example, where $s = 3$, the best strategy packs every 2 level-2 reports into a level-1 node, and the worst strategy packs all 6 level-2 reports into a single level-1 node. A simple calculation shows that $\delta = \frac{11}{9}$.

Now assume level 1 has l_1 nodes and level 2 has l_2 nodes. Then the worst-case topology must satisfy: $(s-1)(l_1-1) < l_2 \leq (s-1)l_1$. That is, all but one of the level-1 nodes need to be fully packed. Otherwise, if $l_2 \leq (s-1)(l_1-1)$, some level-1 nodes become edge nodes without children in both the best and worst strategy, and could be removed to increase the performance ratio. In other words, the original topology is not the worst case topology when $l_2 \leq (s-1)(l_1-1)$. On the other hand, if $l_2 > (s-1)l_1$, $l_2-(s-1)l_1$ level-2 edge nodes can be removed to increase the performance ratio. To see this, removing level-2 nodes deceases the number of packets packed at level 1 by *exactly* $\lceil \frac{l_2-(s-1)l_1}{s} \rceil$ in the best strategy, but decreases by *at most* $\lceil \frac{l_2-(s-1)l_1}{s} \rceil$ in the worst strategy. Thus, the original topology is not the worst case topology when $l_2 > (s-1)l_1$.

Therefore, we set $l_2 = (s-1)l_1 - i, i = 0...s-2$. It is followed that $E_b = l_1 + l_2, E_w = \lceil \frac{l_2+1}{s} \rceil + (l_1-1) + l_2$. Then,

$$\begin{aligned} \delta - 1 = \frac{E_w - E_b}{E_b} &= \frac{\lceil \frac{l_2+1}{s} \rceil - 1}{l_1 + l_2} \leq \frac{\frac{l_2+1+(s-1)}{s} - 1}{l_1 + l_2} \\ &= \frac{1}{s}\frac{(s-1)l_1 - i}{sl_1 - i} \leq \frac{1}{s}\frac{(s-1)l_1}{sl_1} = \frac{s-1}{s^2} \end{aligned} \tag{4.11}$$

The performance ratio $1 + \frac{s-1}{s^2}$ is tight, which can be seen from Fig. 4.12. Specifically, Figs. 4.12(a)&(b) show the worst-case topology when $s = 2$ and $s = 3$. For arbitrary s, the worst-case topology can be constructed as in Fig. 4.12(c), where level 2 has s nodes and level 3 has $s(s-1)$ nodes, then we have

$$\delta = 1 + \frac{\lceil \frac{l_2+1}{s} \rceil - 1}{l_1 + l_2} = 1 + \frac{\lceil \frac{s(s-1)+1}{s} - 1 \rceil}{s(s-1) + s} = 1 + \frac{s-1}{s^2} \tag{4.12}$$

□

Theorem 12. *The approximation ratio for the arbitrary level-connected tree is no greater than* $\frac{5}{4}$*, i.e.,* $\delta_{max} = \frac{5}{4}$.

Proof. The theorem can be proved by the induction of L, the depth of the tree. First, Theorem 11 shows that the base case for 2-level-connected tree is true, i.e., $\delta_{max} = max(1 + \frac{s-1}{s^2}) = \frac{5}{4}$. Assuming the claim is true for k-level-connected tree, $\forall k \leq L$, we want to prove that $\delta_{max} = \frac{5}{4}$ for (L+1)-level-connected tree.

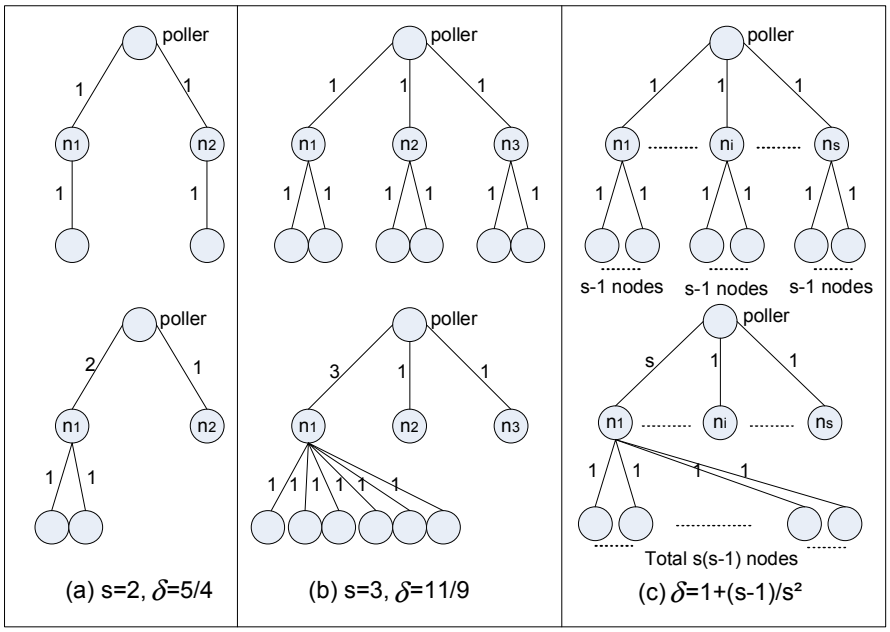


Figure 4.12. The derived bound is tight for 2-level-trees: the best strategy (above) and worst strategy (below). The number by the link indicates the number of packets transmitted over this link.

As shown in Fig. 4.13(a), we divide the total energy consumption of the (L+1)-level-connected tree into two parts, namely, the upper part consisting of the links from level-(L-1) to level-0 (Fig. 4.13(b)) and the lower part consisting of the links from level-(L+1) to level-(L-1) (Fig. 4.13(c)). Because the nodes below level-(L-1) also contribute to the energy consumption of the upper part, we need to expand the nodes below level-(L-1) onto level-(L-1). Then the level-(L-1) nodes in Fig. 4.13(b) are the coalescence of level-(L-1), level-L and level-(L+1) nodes in Fig. 4.13(a). On the other hand, The nodes at level-(L-1) and above have no contribution to the energy consumption of the lower part, so they can be safely shrunk into one virtual node. As shown in Fig. 4.13(c), the virtual node is the root of the 2-level tree. Use E_w^{up}, E_b^{up} to denote the energy consumption of the worst path and the best path in the upper part (Fig. 4.13(b)), and use $E_w^{low}, E_b^{low}$to denote the energy consumption of the worst path and the best path in the lower part (Fig. 4.13(c)). It can be seen that the above expanding and shrinking operation shall not reduce the gap between the energy consumption in the best case and worst case.

Fig. 4.13(b) is a (L-1)-level-connected tree. By the inductive hypothesis, $\delta_{max} = \frac{5}{4}$ for Fig. 4.13(b). Therefore

$$E_w^{up} - E_b^{up} \leq (\frac{5}{4} - 1)E_b^{up} = \frac{1}{4}E_b^{up} \tag{4.13}$$

Fig. 4.13(c) is a 2-level-connected tree. According to Theorem 11, $\delta_{max} = \frac{5}{4}$ for Fig. 4.13(c). Then we have

$$E_w^{low} - E_b^{low} \leq (\frac{5}{4} - 1)E_b^{low} = \frac{1}{4}E_b^{low} \tag{4.14}$$

Combining Eqn. 4.13 and Eqn. 4.14, we finally have

$$\delta = \frac{E_w^{up} + E_w^{low}}{E_b^{up} + E_b^{low}} \leq \frac{5}{4} \tag{4.15}$$

The bound of $\frac{5}{4}$ is tight as seen from Fig. 4.11, where the worst-case topologies are shown.

□

4.5.3 Estimate The Lower And Upper Bound In Practice

Although it is computationally infeasible to find the minimum value and the optimal aggregation path in practice, the upper bound E_w and the lower bound E_b can be estimated based on experiments.

Suppose m experiments are performed, and among them min and max are the minimum and maximum energy value, respectively. Then we have

$$E_b \leq min \leq max \leq E_w \tag{4.16}$$

Further because $\frac{E_w}{E_b} \leq \delta_{max}$, there is

$$max \leq E_w \leq min * \delta_{max} \text{ and } \frac{max}{\delta_{max}} \leq E_b \leq min \tag{4.17}$$

This relationship tells us that the upper bound and lower bound fall within the range of $[max, min * \delta_{max}]$ and $[\frac{max}{\delta_{max}}, min]$. The more experiments to be performed, the better chance that max and min can reflect the real maximum and minimum point and the more likely that the range of the upper and lower bound could be further narrowed down .

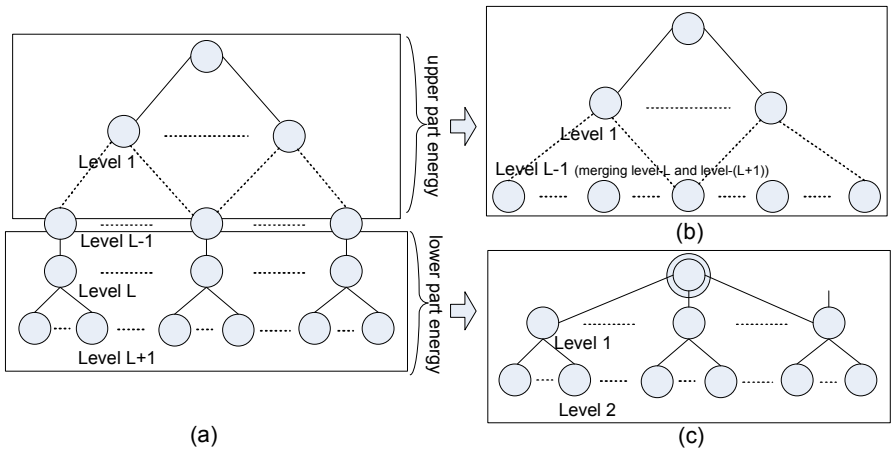


Figure 4.13. Proof of the performance bound by dividing the energy consumption into two parts. (a) the original connected tree (b) the upper part consists of the links above level-(L-1), with the nodes at level-L and level-(L+1) expanded onto level-(L-1). (c) the lower part consists of the links below level-(L-1), with the nodes at level-(L-1) and above shrunk into a single virtual node.

4.6 Performance Evaluations

In this section, we use simulation to test the parameters and evaluate the proposed algorithms. In the simulation, n sensors are randomly deployed in a 20×20 square area. Each pollee is associated with up to ω pollers, which are used for fault tolerance. Each sensor sends a status report to its poller every t, with the reports aggregated at the intermediate nodes. The sensor transmission range is 1, and transmitting one packet over one hop consumes one unit of energy. The link failure is modeled as a continuous-time markov chain with an average failure rate f_l and a detection period T_d. We set $f_l = 0.05$ and $T_d = 2t$ without otherwise specified. The experiments are done over a customized C++ simulator.

4.6.1 Parameter Setting

Three parameters need to be determined, k_1, k_2 for the deterministic algorithm and ρ for the hybrid algorithm. In Section 4.3, Fig. 4.5& 4.6 gives the relationship between the false alarm rate and the distance from the pollee to the poller when the average link failure rate is known. As a result, given the constraint of the maximum false alarm rate, k_2 can be determined to limit the number of hops from

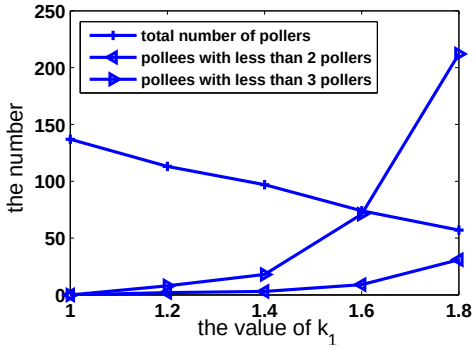


Figure 4.14. The effect of k_1 on the number of pollers and the number of pollees that cannot find ω pollers (n=1000)

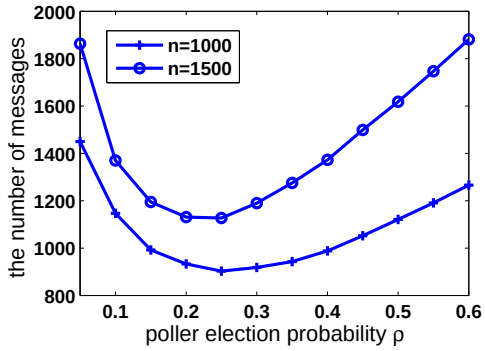


Figure 4.15. The effect of probability ρ on the message overhead in the hybrid algorithm ($k_1 = 1, k_2 = 1$)

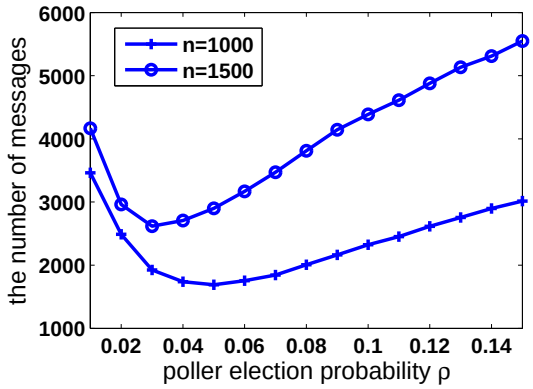


Figure 4.16. The effect of probability ρ on the message overhead in the hybrid algorithm ($k_1 = 1, k_2 = 3$)

the pollee to the poller. Suppose the false alarm rate is required to be less than 4%, then $k_2 \leq 3$ based on Fig. 4.5. We will set $k_2 = 3$ in the following experiments unless otherwise specified, based on which k_1 and ρ are chosen.

Fig. 4.14 shows the effect of k_1. As k_1 increases, there will be less number of pollers because k_1 controls the distance between the neighboring pollers. However, the number of pollees that cannot find ω pollers increases as k_1 increases. For example, as k_1 increases from 1 to 1.8, the number of pollees that cannot find 3 pollers ($\omega = 3$) increases from 0 to 210. This is because with less number of pollers, it is less likely for a pollee to find ω pollers within k_2 hops. As can be

seen, if $k_1 = 1$, all pollees can find three pollers. However, if $k_1 = 1.8$, over 20% pollees cannot find three pollers. We thus set $k_1 = 1$ in the following experiments, without otherwise specified. Note that k_1 may be a fractional number.

In the hybrid algorithm, each node first elects itself as poller with probability ρ. Fig. 4.15 shows how the value of ρ affects the message overhead when $k_1 = k_2 = 1$. Compared with the analytical result in Fig. 4.8, the same trend is observed in Fig. 4.15, where the message overhead first drops then rises as ρ increases. The point with the least number of messages corresponds to the optimal ρ^*. For example, when $n = 1000, 1500$, ρ^* is around 0.25, 0.2 in the simulation, and around 0.2, 0.15 in the analysis. The little mismatch of the theoretical result may be explained by the boundary effect of the finite field in reality. That is, in the analysis it is assumed that the sensing range of a node falls within the field of the deployment, but this is not true for the nodes on the boundary, which will affect the accuracy of the analytical result.

Similarly, Fig. 4.16 shows the relationship between the message overhead and ρ when $k_1 = 1, k_2 = 3$. It can be observed that at the optimal point, ρ^* is around 0.05, 0.03 when $n = 1000, 1500$, which is much smaller than the case when $k_1 = k_2 = 1$. This is because with larger k_2, a smaller number of pollers are able to cover most of the unlabeled nodes. In the following, we will use the optimal ρ^* corresponding to the different n, k_1, k_2.

4.6.2 Comparison of Single Poller and Multi-Poller Schemes

Fig. 4.17 compares the single poller scheme with the multi-poller scheme in presence of link failure. For fair comparison, the same detection period T_d is used. For example, $T_d = 4t$ implies that four reports are sent out within the detection period T_d. From Fig. 4.17, we have several observations. First, consistent with the numerical result, the false alarm rate dramatically drops as the detection period T_d increases. This verifies the tradeoff between the detection delay and the false alarm rate. Second, the reduction of the false alarm rate in the multi-poller scheme is over 50% compared to that of the single-poller scheme, irrespective of the number of nodes. As discussed below, the substantial performance improvement is at the cost of a marginal increase of the bandwidth consumption (shown in Fig. 4.18).

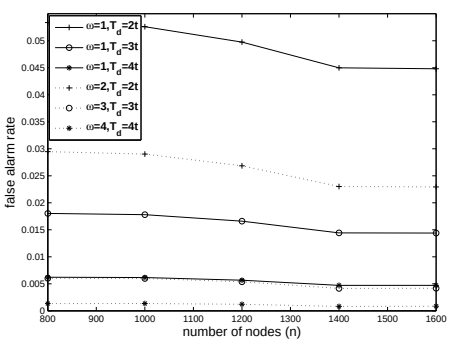


Figure 4.17. Comparison of single and multi-poller scheme (with link failure $f_l = 0.1$)

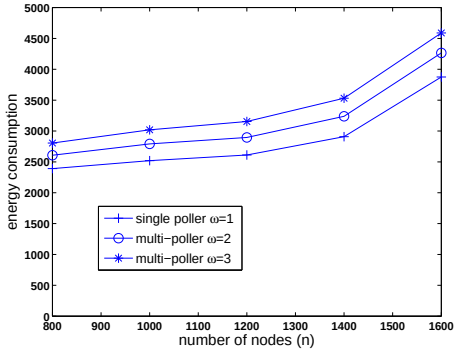


Figure 4.18. Energy comparison of single and multi-poller scheme (without failure)

Fig. 4.18 studies the energy cost associated with the multi-poller scheme. It costs 7% more energy when the number of pollers for each pollee increases from 1 to 2 or from 2 to 3. Based on our randomized algorithm, the primary path is always the shortest path, but the secondary path may be a little bit longer. As a result, packets may go through more hops to reach the secondary poller compared to that of the primary poller, resulting in more energy consumption. However, compared with the improvement on false alarm rate as shown in Figs. 4.17, the energy increase is pretty small.

4.6.3 Comparison of Different Distributed Algorithms

In this section, we compare the geometrical property of the hybrid algorithm and the randomized algorithm. Fig. 4.19 shows snapshots of the poller-pollee distribution after running the randomized and hybrid algorithm among 100 nodes in a 5×5 field. The whole network is connected but for clarity purpose we only show the links within each polling domain. Fig. 4.19(a) shows that the randomized algorithm produces a scenario where the poller may be isolated (e.g., node 79 in the up-left corner), clustered together (e.g., node 86,39,45 on top), or the pollee (e.g., node 92 in the bottom-left corner) is too far away from its poller. However, after running the deterministic algorithm in Fig. 4.19(b), the hybrid algorithm fixes these problems. For instance, in Fig. 4.19(b) there is on isolated poller and no

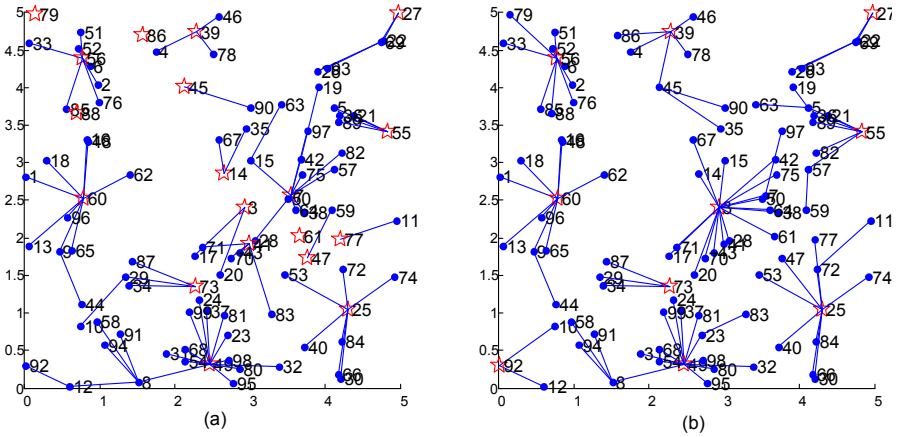


Figure 4.19. Snapshots of poller-pollee distribution, where the star denotes the poller and the dot denotes the pollee: (a) randomized algorithm, $\rho = 0.2$, (b) hybrid (randomized + deterministic) algorithm, $\rho = 0.2, k_1 = 1, k_2 = 2$

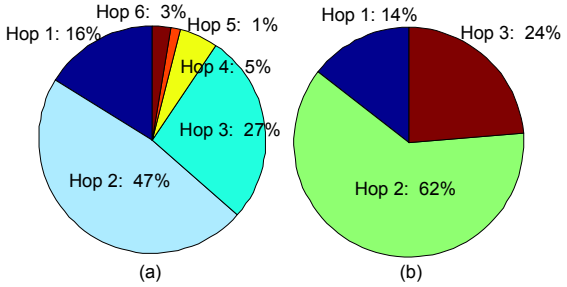


Figure 4.20. Distribution of the distance between pollers and pollees: (a)randomized algorithm (b) hybrid algorithm

poller is within one hop of each other and no pollee is more than 2 hops away from its poller.

Figs. 4.20, 4.21 further compare the hybrid algorithm with the randomized algorithm in terms of statistical geometrical property and false alarm rate. For fair comparison, we set $\rho = 0.75, 0.065$, $k = 6$, when $n = 1000, 1500$, in the randomized algorithm, to select the same number of pollers as in the hybrid algorithm. As seen in Fig. 4.20, the distance between pollers and pollees is bounded by 3 hops in the hybrid algorithm, but spans up to 6 hops in the randomized algorithm. About 8% of the pollees in Fig. 4.20(a) are more than 3 hops away from their pollers, so the constraint of the false alarm rate cannot be met in the randomized algorithm.

Fig. 4.21 shows that the hybrid algorithm outperforms the randomized algo-

rithm in terms of the average false alarm rate, which is calculated by averaging the sum of the false alarm rate over all the nodes. It can be seen that the average false alarm rate of the hybrid algorithm is about 20% or 30% smaller than that of the randomized algorithm, when $n = 1000$ or 1500, respectively.

Both the deterministic algorithm and the hybrid algorithm have provable distribution property – no poller is less than k_1 hops away from each other, and no pollee is more than k_2 hops away from its poller. However, Fig. 4.22 shows that the hybrid algorithm substantially reduces the number of messages. The reduction is about 60% and 80%, when $k_1 = 1$ and $k_1 = 2$, respectively. The benefit is thanks to the randomized phase adopted by the hybrid algorithm.

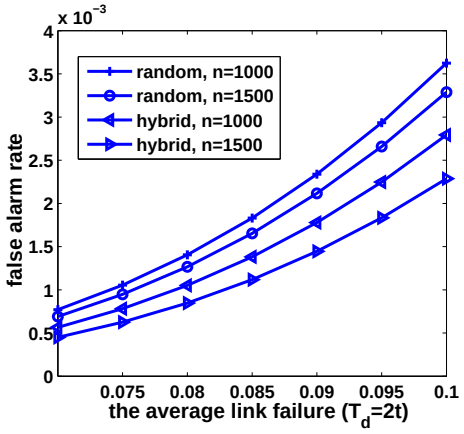


Figure 4.21. Comparison of random algorithm with hybrid algorithm

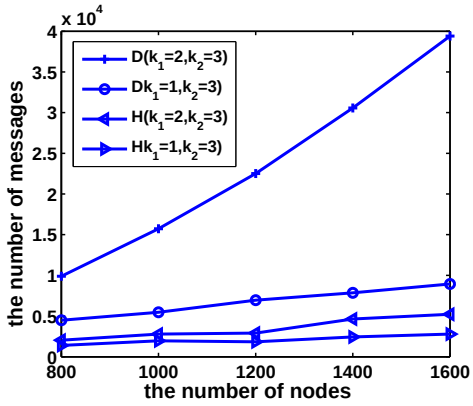


Figure 4.22. Comparison of deterministic algorithm with hybrid algorithm

4.6.4 Effect of Poller Reelection

Fig. 4.24(a) shows a snapshot of poller-pollee distribution after running the randomized algorithm among 150 nodes in a 5×5 field. The whole network is connected but for clarity purpose we only show the links within each polling domain. In Fig. 4.24(b), poller 25 and 108 fail, so their pollees have to find alternative pollers and the distance between the pollee and poller increases as a consequence. It can be predicated that as more pollers fail, some pollees may not be able to find ω pollers within k hops. Fig. 4.24(c) further shows the poller-pollee distribution after pollers are reelected based on Algorithm II in Subsection 4.4.4. It can be

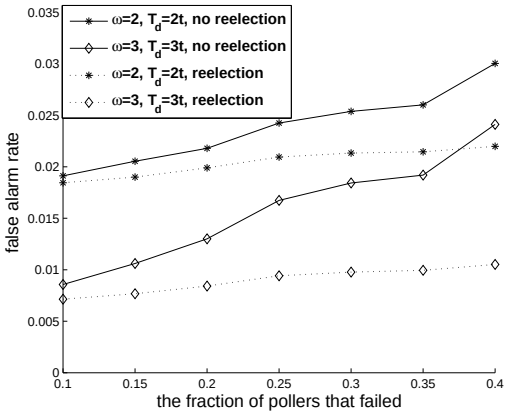


Figure 4.23. Comparison of false alarm rate in case of poller failure

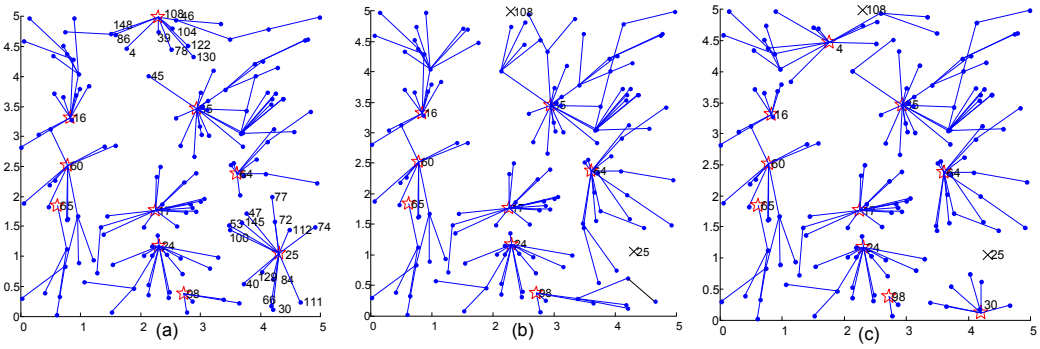


Figure 4.24. Snapshots of poller-pollee distribution: (a) after running the randomized algorithm (n=150, p=0.1, k=4, $\omega = 1$). (b) after some pollers fail. (c) after the failed pollers are replaced.

seen that poller 30, 4, which has the smallest id among the neighbors of the failed poller, are reelected to replace the failed poller 25, 108, respectively. This way, the density of the poller can remain roughly constant irrespective of the failure of the poller.

Fig. 4.23 compares the false alarm rate with and without poller reelection in the presence of poller failure. It can be seen that the false alarm rate increases as more pollers fail, and the speed of increase is much faster if new pollers are not reelected. For example, when 40% of the pollers fail, the reduction of the false alarm rate will be about 30% when $\omega = 2, T_d = 2t$, and 60% when $\omega = 3, T_d = 3t$, with poller reelection. This is because if new pollers are not reelected, some pollee

has to be associated with less number of pollers and the multi-poller scheme may reduce to the single poller scheme.

4.7 Related work

While a lot of research in sensor network focuses on the field or target monitoring [42, 15, 63], little attention has been given to the monitoring of sensor network itself. In [55, 64, 65], distributed failure detectors were proposed independently for wireless sensor networks, where each node is collaboratively monitored by its one-hop neighbors. These schemes can only detect node failure, but more general status such as link status, residue energy and coverage cannot be monitored. In [66, 67], data aggregation had been used to obtain a global abstraction of the sensors' residue energy, but only when specific continuous energy dissipation models are assumed. More recently, a local monitoring infrastructure is proposed in [68]. But their goal is to monitor the transmission over the wireless link for security purpose, instead of from the perspective of fault tolerance. By contrast, our poller-pollee based monitoring architecture can respond to queries about a variety of sensor status tailored to the application demand.

The problem of selecting a subset of nodes to form a backbone has been extensively studied in different contexts, e.g., clustering [69, 70, 71, 72], connected dominating set (CDS) [73, 54], backbone routing [74, 75], relay node placement [76], etc. The objective of these works is to minimize the cardinality of the selected subset of nodes, but ignore the constraint of false alarm rate, which is crucial in wireless sensor networks. In addition, most of these works are limited in the selection of single-hop, single vantage point (e.g., cluster head, dominator). However, our work focuses on multi-hop multi-poller monitoring architecture construction, where some geometrical properties of the poller-pollee distribution can be guaranteed.

Aggregation path selection problem has been proposed in [61, 62, 77], wherein some heuristics are developed but without performance guarantee. There are also some other aggregation schemes that target for the different application scenarios [78, 79, 80], but none of them guarantees constant approximation ratio. To the best of our knowledge, our result is the first proved constant approximation ratio

applied to the aggregation path selection schemes for the wireless sensor networks.

Our work is also orthogonal to some related work in other contexts. For example, energy consumption can be reduced by data compression [81, 82], opportunistic forwarding [83]; communication error/failure can be reduced by robust aggregation [65], reliable routing [84], sensor activity scheduling [85]. When necessary, our work can combine these schemes to further improve the performance.

4.8 Conclusions

In this chapter, we focused on the distributed design of monitoring for the mission driven sensor network. We first proposed a multi-poller based monitoring architecture. Coupled with a round robin based scheduling scheme, the false alarm rate can be reduced significantly while keeping the similar bandwidth as the single poller scheme. Based on the multi-poller structure, we proposed fully distributed algorithms to select the minimum number of pollers while bounding the false alarm rate. Then a greedy aggregation scheme was proposed to reduce the messages overhead due to monitoring. Theoretical analyses and extensive simulations showed that the deterministic algorithm can flexibly control the poller-pollee distribution property to bound the false alarm rate, the hybrid algorithm can reduce the message overhead significantly, and the greedy aggregation scheme decreases the monitoring traffic with a constant approximation ratio of $\frac{5}{4}$.

Chapter 5

Conclusions and Future Work

5.1 Summary

Unlike most existing work on sensor network, which is only targeted for a single mission, this dissertation develops an integrated framework for the multi-mission sensor network. Several separate yet related aspects have been thoroughly explored in the respective chapters, and summarized below.

In Chapter 2, we study the mission switch in the context of area surveillance. In such application, the mission usually has a stringent lifetime requirement, which could change over time. Due to the limited deployed sensors, the coverage requirement and lifetime constraint may not be able to be satisfied at the same time, and thus the coverage has to be traded for network lifetime. To deal with this tradeoff, we study how to schedule sensors to maximize their coverage during a specified network lifetime. Unlike sensor deployment, where the goal is to maximize the *spatial* coverage, our objective is to maximize the *spatial-temporal* coverage by scheduling sensors' activity after they have been deployed. Since the optimization problem is NP-hard, we first present a centralized heuristics whose approximation factor is proved to be $\frac{1}{2}$, and then propose a distributed parallel optimization protocol (POP). In POP, nodes optimize their schedules on their own but converge to local optimality without conflict with one another.

In Chapter 3, we explore the resource allocation issue in the landmine networks. Our study is based on a multi-target defense scenario, where our mission is to destroy the multiple intruding targets using the minimum cost pre-deployed

landmine. Resource allocation in this case is especially complex since the explosion of a single landmine can affect multiple targets and the destruction of a single target may require the involvement of multiple landmines. To deal with the problem complexity, we focus on the design of approximation solutions based on a novel bucket-tub model, where mine is mapped to bucket set and target is mapped to tub. Two classes of approximation algorithms are proposed, whose approximation ratio is derived. Among them, it is shown that the layering algorithm can achieve an approximation ratio that relies only on the maximum number of mines/buckets that a target/tub is associated with, and that the greedy algorithm can be implemented in a pure distributed manner without degrading the performance.

In Chapter 4, we focus on the monitoring architecture design for the mission driven sensor networks. Due to the dynamic nature of sensor status, a sound monitoring architecture plays a major role in offering timely evaluation of the mission status and is fundamental to the operation of multi-mission sensor network. This motivates us to design the multi-poller based monitoring architecture. Coupled with a round robin based scheduling scheme, the multi-poller based monitoring can reduce the false alarm rate significantly while keeping roughly the same bandwidth as the single poller scheme. To construct the monitoring architecture, we formulate a many-to-many poller-pollee assignment problem and present three distributed algorithms (i.e., random, deterministic, and hybrid). We have also explored the hop-by-hop aggregation opportunity between the poller and pollee, and formulate the optimal aggregation path problem. We solve this NP-hard problem by designing an opportunistic greedy algorithm, which achieves an approximation ratio of $\frac{5}{4}$. As far as we know, this is the first proved constant approximation ratio applied to the aggregation path selection schemes over the wireless sensor networks.

5.2 Future Directions

The multi-mission sensor network is an emerging area, full of opportunity for research and development. This dissertation addresses several important aspects associated with the multi-mission sensor network, i.e., mission switch, resource allocation, network monitoring, but there are still many other issues worthy of

in-depth investigation. In the following, I'll outline several interesting directions that could be further explored.

1. **Network Reprogramming and Code Dissemination**: as the mission switch or some event occurs, the network may need to be reprogrammed via different parameters or replace the old code with the new one [86, 87, 88, 89]. This motivates the design of code dissemination protocol for multi-mission sensor network [90, 91, 92, 93]. While most existing code dissemination methods solely rely on the network-wide broadcast, they are not suitable when the dissemination area is not geometrically continuous. To resolve this challenge, new dissemination techniques and routing modules are required, which could minimize the redundant retransmissions irrespective of the distribution properties of the affected area.

2. **Mission Oriented Congestion Control**: in multimission sensor networks, different missions not only compete for the sensor resource but also rival for the bandwidth resource. This brings the necessity of congestion control. Without a proper congestion control mechanism, the whole network performance will be brought down and the individual mission goal will be endangered. However, most existing work on congestion control is carried out at the packet-level or flow level [94, 95, 96, 97]. There still lacks of literature of congestion control at the mission level. This begets the challenges such as how to clearly define mission oriented congestion control, how to identify the mission to which the packet/flow belongs, how to release transilient congestions and avoid longer-term congestions in the wake of resource constraint and mission switch.

3. **Multi-Missions in Intermittently Connected Networks**: in the intermittently connected networks, such as delay tolerant network [98, 99], mobile phone network [100], the network is not connected all the time due to the mobility and low density of active nodes. Multi-mission control in such environment is particular challenging. The algorithmic design on various aspects, e.g., resource allocation, mission switch, network monitoring, needs to take into account the inherent nature of intermittent connectivity in such network, which usually demands a fundamentally different design from the traditional

approach. For example, in our previous work [18], we have proposed a new routing protocol when a large fraction of sensor nodes are in the sleeping state, because the traditional approach such as the shortest path algorithm cannot be applied here.

Appendix A

The NP Proof of Problem maxCov

Theorem. *The problem of maxCov is NP-hard.*

Proof. Theorem 1 tells that the problem *maxCov* can be transformed to the problem *minRed*. Therefore, we only need to prove problem *minRed* is NP-hard.

The *minRed* problem can be proved to be NP-hard via a reduction from the graph k-coloring problem, which asks whether a given graph G can be colored using k colors such that no two neighboring vertexes have the same colors [11]. Given an instance I of graph k-coloring problem, we can construct an equivalent instance I' of *minRed decision* problem in polynomial time, such that instance I has a solution if and only if instance I' has a solution. The decision problem can be stated as follows : let the node battery life B and network life L satisfy $\frac{L}{B} = k$. In the same graph G of k-coloring problem, is there a node scheduling scheme such that the total coverage redundancy is 0 in the *minRed* decision problem?

To see the equivalence of the two problems, we divide each cycle in instance I' into $k = \frac{L}{B}$ time slots, with each time slot mapped to a distinct color in instance I. In essence, the instance I concerns about the assignment of one of the k colors to each node while the instance I' concerns about the allocation of one of the k time slots to each node. Then it can be observed that if there exists a k-coloring scheme where each node is assigned a color different from that of its neighbors', there also exists a corresponding scheduling scheme where each node is allocated a different time slot from that of its neighbors, with the total coverage to be 0.

On the other hand, suppose there is a scheduling scheme for instance I' where

the total coverage is 0, we can always find a solution to the instance I of graph k-coloring problem. Since the "on" period may not exactly occupy a time slot, we need first preprocess the schedule by aligning the on-period of each node with its left time slot. By doing this, the total coverage remains 0 and the aligned schedule is still the solution for instance I. After that, to construct a solution to the instance I' simply becomes equivalent to "color" each vertex using one of the k time slots, such that no neighboring vertexes have the same colors. In this regard, the solution of instance I' readily produces a corresponding solution for instance I.

To sum up, G can be k-colored if and only if there is a zero coverage redundancy scheduling scheme for G, which means that the graph coloring problem can be reduced to *minRed* decision problem in polynomial time. As the graph k-coloring problem is NP-hard, the *minRed* decision problem is NP-hard. On the other hand, the *minRed* optimization problem is at least as hard as its decision problem, and thus is also NP-hard. □

Appendix B

The NP Proof of Problem OptPH

Theorem 10. *The problem of optPH is NP-hard.*

Proof. For each poller, the paths from all its pollees form a tree. Since the overall energy consumption is the sum of the energy consumption with respect to each poller, we only need to focus on one such tree with the poller as root and all its pollees as members.

The problem of optPH can be proved to be NP-hard via a reduction from the subset sum problem. The subset sum problem over the finite field can be stated as: given an integer s and a set of l integers, i.e., $k_1, k_2...k_l \in [0, s-1]$, does any

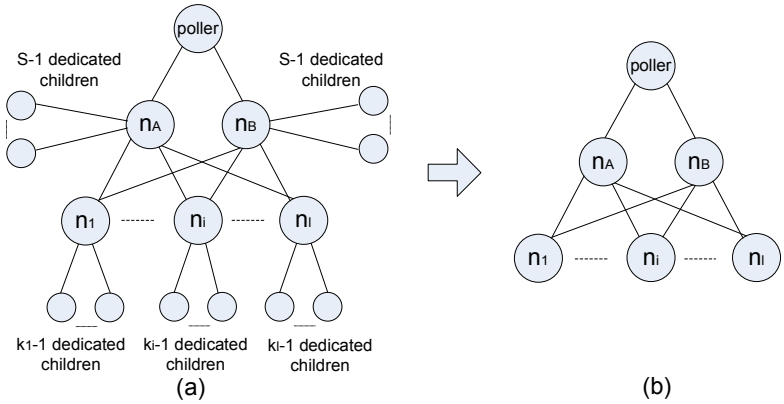


Figure B.1. A tree topology to show the problem optPH is NP-hard. (a) initial topology: each node sends a report to the poller (b) transformed topology: each $n_i, i = 1...l$ sends k_i reports to the poller, with n_A, n_B sending nothing.

subset sum to zero modulo s? Given an instance I of the subset sum problem, we can construct an equivalent instance II of the optPH problem, whose topology is shown in Fig. B.1(a). In Fig. B.1(a), level 0 consists only of the root, i.e. the poller. Level 1 has two nodes, n_A, n_B, each of which has $s-1$ dedicated children (i.e., edge node). n_A, n_B also share l children in common, namely, $n_i, i = 1...l$ and each n_i has $k_i - 1$ dedicated children. Every polling period, each node sends a report towards the poller/root. The equivalence of instances I & II can be proved by reducing Fig. B.1(a) to Fig. B.1(b). First, since all dedicated children have only one parent, they can be summarized by their parent. Therefore, Fig. B.1(b) has three levels, where at level 2, n_i has $(k_i - 1) + 1 = k_i$ reports to send, and at level 1, n_A and n_B have $(s-1)+1 \ mod \ s = 0$ reports to send, with s as the aggregation ratio. Note that when n_A and n_B has total s reports, they could be packed into one packet without residue room left. Therefore, changing the number of reports at n_A and n_B from s to 0 does not affect the optimal aggregation path.

It can also be proved that instances I of the subset sum problem and the instance denoted by Fig. B.1(b) are equivalent. This is because the objective in Fig. B.1(b) is to select the next hop for $n_i, i = 1...l$ between n_A and n_B, such that the minimum number of packets can be packed at level 1. Then, if a set of aggregation paths satisfy that an integer number of whole packets can be packed at either n_A or n_B, it must be an optimal solution. Therefore, there exists a subset that sums to zero modulo s for instance I if and only if in the optimal solution of Fig. B.1(b), n_A or n_B packs the reports into an integer number of whole packets, which means that the subset problem can be reduced to the optPH problem. Since the subset sum problem over the finite field is known to be NP-complete, it follows that the optimization problem optPH is NP-hard.

□

Bibliography

[1] Liu, H., P. Wan, C.-W. Yi, X. Jia, S. Makki, and P. Niki (2005) "Maximal lifetime scheduling in sensor surveillance networks," in *IEEE INFOCOM*.

[2] Cardei, M. and J. Wu (2005) "Energy-Efficient Coverage Problems in Wireless Ad Hoc Sensor Networks," *Journal of Computer Communications on Sensor Networks*.

[3] Tian, D. and N. Georganas (2002) "A coverage-preserving node scheduling scheme for large wireless sensor networks," in *ACM international workshop on Wireless sensor networks and applications*.

[4] Wang, X., G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill (2003) "Integrated coverage and connectivity configuration in wireless sensor networks," in *ACM SenSys*.

[5] Yan, T., T. He, and J. A. Stankovic (2003) "Differentiated surveillance for sensor networks," in *ACM SenSys*.

[6] Zhang, H. and J. C. Hou (2005) "Maintaining sensing coverage and connectivity in large sensor networks," *Wireless Ad Hoc and Sensor Networks*.

[7] Neubergera, A., S. Pelesc, and D. Rittela (2007) "Scaling the response of circular plates subjected to large and close-range spherical explosions. Part II: Buried charges," *International Journal of Impact Engineering*.

[8] Jacoba, N., G. Nuricka, and G. Langdon (2007) "The effect of standoff distance on the failure of fully clamped circular mild steel plates subjected to blast loads," *International Journal of Impact Engineering*.

[9] Edmondson, N. (1992) *Fatality probabilities for people in the Open when exposed to blast*, *Tech. Rep. RANN/2/49/00082/90*, SRD.

[10] (2002) *Controlling risks around explosive stores: review of the requirements on separation distances*, *Tech. rep.*, MBTB Limited, Health and Safety Executive, United Kingdom.

[11] Alsuwaiyel, M. H. (1999) *Algorithms Design Techniques and Analysis*, World Scientific Publishing Company.

[12] Khuller, S., A. Moss, and J. Naor (1999) "The budgeted maximum coverage problem," *Information Processing Letter*, **70**(1), pp. 39–45.

[13] Basagni, S. (1999) "A Distributed Algorithm for finding a Maximal Weighted Independent Set in Wireless Networks," in 11^{th} *International Conference on Parallel and Distributed Computing and Systems (PDCS).*

[14] Xing, G., R. Tan, B. Liu, J. Wang, X. Jia, and C.-W. Yi (2009) "Data fusion improves the coverage of wireless sensor networks," in *ACM Mobicom.*

[15] Liu, C. and G. Cao (2009) "Minimizing the cost of mine selection via sensor networks," in *IEEE INFOCOM.*

[16] Hwang, J., T. He, and Y. Kim (2009) "Exploring In-Situ Sensing Irregularity in Wireless Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems.*

[17] Hefeeda, M. and H. Ahmadi (2009) "An Integrated Protocol for Maintaining Connectivity and Coverage under Probabilistic Models for Wireless Sensor Networks," *Ad Hoc & Sensor Wireless Networks.*

[18] Su, L., C. Liu, H. Song, and G. Cao (2008) "Routing in Intermittently Connected Sensor Networks," in *ICNP.*

[19] Zou, Y. and K. Chakrabarty (2005) "A Distributed Coverage and Connectivity Centric Technique for selecting Active Nodes in Wireless Sensor Networks," *IEEE Tran. Computer.*

[20] Kumar, S., T. H. Lai, and J. Balogh (2004) "On k-Coverage in a Mostly Sleeping Sensor Network," in *ACM MOBICOM.*

[21] Funkey, S., A. Kesselman, F. Kuhn, and Z. Lotker (2007) "Improved Approximation Algorithms for Connected Sensor Cover," *Wireless Networks.*

[22] Berman, P., G. Calinescu, C. Shah, and A. Zelikovsly (2005) "Efficient energy management in sensor networks," *Ad hoc and sensor networks.*

[23] Kasbekar, G. S., Y. Bejerano, and S. Sarkar (2009) "Lifetime and coverage guarantees through distributed coordinate-free sensor activation," in *ACM Mobicom.*

[24] Bai, X., S. Kumar, D. Xuan, Z. Yun, and T. H. Lai (2006) "Deploying Wireless Sensors to Achieve Both Coverage and Connectivity," in *ACM MOBIHOC*.

[25] Cardei, M., M. Thai, Y. Li, and J. Wu (2005) "Energy-Efficient Target Coverage in Wireless Sensor Networks," in *IEEE INFOCOM*.

[26] Liu, C. and G. Cao (2009) "An Multi-Poller based Energy-Efficient Monitoring Scheme for Wireless Sensor Networks," in *IEEE INFOCOM mini-conference*.

[27] ——— (2010) "Distributed Monitoring and Aggregation in Wireless Sensor Networks," in *IEEE INFOCOM*.

[28] Meguerdichian, S., F. Koushanfar, M. Potkonjak, and M. B. Srivastava (2001) "Coverage Problems in Wireless Ad-hoc Sensor Networks," in *IEEE INFOCOM*.

[29] Kumar, S., T. H. Lai, and A. Arora (2005) "Barrier Coverage With Wireless Sensors," in *ACM Mobicom*.

[30] Chen, A., S. Kumar, and T. H. Lai (2007) "Designing localized algorithms for barrier coverage," in *MOBICOM*.

[31] Liu, B., O. Dousse, J. Wang, and A. Saipulla (2008) "Strong barrier coverage of wireless sensor networks," in *ACM Mobihoc*.

[32] Saipulla, A., C. Westphal, B. Liu, and J. Wang (2009) "Barrier Coverage of Line-Based Deployed Wireless Sensor Networks," in *IEEE INFOCOM*.

[33] Balister, P., Z. Zheng, S. Kumar, and P. Sinha (2009) "Trap Coverage: Allowing Coverage Holes of Bounded Diameter in Wireless Sensor Networks," in *INFOCOM 2009, IEEE*, pp. 136 –144.

[34] Zhao, M.-C., J. Lei, M.-Y. Wu, Y. Liu, and W. Shu (2009) "Surface Coverage in Wireless Sensor Networks," in *INFOCOM 2009, IEEE*, pp. 109 –117.

[35] Luo, J., D. Wang, and Q. Zhang (2009) "Double Mobility: Coverage of the Sea Surface with Mobile Sensor Networks," in *INFOCOM 2009, IEEE*, pp. 118 –126.

[36] Zheng, Z., P. Sinha, and S. Kumar (2009) "Alpha Coverage: Bounding the Interconnection Gap for Vehicular Internet Access," in *INFOCOM 2009, IEEE*, pp. 2831 –2835.

[37] Lu, G., N. Sadagopan, B. Krishnamachari, and A. Goel (2005) "Delay Efficient Sleep Scheduling in Wireless Sensor Networks," in *IEEE INFOCOM*.

[38] Cohen, R. and B. Kapchits (2007) "An Optimal Algorithm for Minimizing Energy Consumption while Limiting Maximum Delay in a Mesh Sensor Network," in *IEEE INFOCOM*.

[39] Keshavarzian, A., H. Lee, and L. Venkatraman (2006) "Wakeup scheduling in wireless sensor networks," in *ACM Mobihoc*.

[40] Ren, S., Q. Li, H. Wang, X. Chen, and X. Zhang (2007) "Design and Analysis of Sensing Scheduling Algorithms under Partial Coverage for Object Detection in Sensor Networks," *IEEE Transactions on Parallel and Distributed Systems*.

[41] Gui, C. and P. Mohapatra (2004) "Power conservation and Quality of Surveillance in Target Tracking Sensor Networks," in *ACM MOBICOM*.

[42] Zhang, W. and G. Cao (2004) "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communication*, **3**(5), pp. 1689–1701.

[43] ——— (2004) "Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks," *IEEE INFOCOM*.

[44] Cao, Q., T. Abdelzaher, T. He, and J. Stankovic (2005) "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *ACM/IEEE IPSN*.

[45] Joo, C. (2008) "A local greedy scheduling scheme with provable performance guarantee," in *ACM Mobihoc*.

[46] Merrill, W., L. Girod, B. Schiffer, D. McIntire, G. Rava, K. Sohrabi, F. Newberg, J. Elson, and W. Kaiser (2004) "Dynamic Networking and Smart Sensing Enable Next-Generation Landmines," *IEEE Pervasive Computing*.

[47] Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, I. (2002) "Wireless Sensor Networks: A Survey," *Computer Networks*, **38**(4).

[48] Volgyesi, P., G. Balogh, A. Nadas, C. Nash, and A. Ledeczi (2007) "Shooter localization and weapon classification with soldier-wearable networked sensors," in *MobiSys*.

[49] Li, M. and Y. Liu (2007) "Rendered path: range-free localization in anisotropic sensor networks with holes," in *MobiCom*.

[50] Rajagopalan, S. and V. V. Vazirani (1999) "Primal-Dual RNC Approximation Algorithms for Set Cover and Covering Integer Programs," *SIAM J. Comput.*

[51] Dobson, G. (1982) "Worst-Case Analysis of Greedy Heuristics for Integer Programming with Nonnegative Data," *Mathematics of Operations Research.*

[52] Cole, T. J. (1993) "Scaling and Rounding Regression Coefficients to Integers," *Applied Statistics.*

[53] Li, L. E., M. Thottan, B. Yao, and S. Paul (2003) "Distributed Network Monitoring with Bounded Link Utilization in IP Networks," in *INFOCOM*, San Francisco.

[54] Dai, F. and J. Wu (2006) "On Constructing k-Connected k-Dominating Set in Wireless Networks," *JPDC.*

[55] fan Hsin, C. and M. Liu (2002) "A distributed monitoring mechanism for wireless sensor networks," in *WISE.*

[56] Konrad, A., B. Y. Zhao, A. D. Joseph, and R. Ludwig (2003) "A Markov-based channel model algorithm for wireless networks," *Wireless Networks*, **9**, pp. 189–199.

[57] Babich, F., O. E. Kelly, and G. Lombardi (2000) "Generalized Markov modeling for flat fading," *IEEE Trans. Commun.*, **48**, pp. 547–551.

[58] Lee, H., A. Cerpa, and P. Levis (2007) "Improving wireless simulation through noise modeling," in *IPSN.*

[59] Murthy, D. N. P., M. Xie, and R. Jiang (2004) *Weibull Models*, John Wiley & Sons.

[60] Karner, W., O. Nemethova, and M. Rupp (2007) "Link Error Prediction in Wireless Communication Systems with Quality Based Power Control," in *ICC.*

[61] J.Park, S. and R. Sivakumar (2008) "Energy Efficient Correlated Data Aggregation for Wireless Sensor Networks," *International Journal of Distributed Sensor Networks.*

[62] Cristescu, R., B. Beferull-Lozano, and M. Vetterli (2005) "On network correlated data gathering," in *IEEE INFOCOM.*

[63] Ding, M. and X. Cheng (2009) "Fault tolerant target tracking in sensor networks," in *MobiHoc*, ACM, pp. 125–134.

[64] Rost, S. and H. Balakrishnan (2006) "Memento: A Health Monitoring System for Wireless Sensor Networks," in *SECON*.

[65] Gobriel, S., S. Khattab, D. Moss, J. Brustoloni, and R. Melhem (2006) "RideSharing: Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions," in *SECON*.

[66] Zhao, J., R. Govindan, and D. Estrin (2002) "Residual Energy Scans for Monitoring Wireless Sensor Networks," in *WCNC*.

[67] ——— (2003) "Computer Aggregates For Monitoring Wireless Sensor Networks," in *SNPA*.

[68] Dong, D., Y. Liu, and X. Liao (2008) "Self-monitoring for sensor networks," in *MobiHoc*.

[69] Younis, O. and S. Fahmy (2004) "Distributed Clustering in Ad-Hoc Sensor Networks: A Hybrid, Energy-Efficient Approach," in *INFOCOM*.

[70] Mhatre, V. and C. Rosenberg (2004) "Design Guidelines for Wireless Sensor Networks Communication: Clustering and Aggregation," in *Ad Hoc Networks Journal*, vol. 2 of *1*, pp. 45–63.

[71] Bandyopadhyay, S. and E. Coyle (2003) "An Energy-Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks," in *INFOCOM*.

[72] Chatterjee, M., S. Das, and D. Turgut (2002) "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," in *Cluster Computing*.

[73] Wan, P.-J., K. M. Alzoubi, and O. Frieder (2004) "Distributed construction of connected dominating set in wireless ad hoc networks," *Mobile Networks and Applications*, **9**(2).

[74] Srinivas, A. and E. Modiano (2006) "Mobile backbone networks – construction and maintenance," in *MOBIHOC*.

[75] Wang, Y., W. Wang, and X.-Y. Li (2005) "Distributed low-cost backbone formation for wireless ad hoc networks," in *MOBIHOC*.

[76] Misra, S., S. Hong, G. Xue, and J. Tang (2008) "Constrained relay node placement in wireless sensor networks to meet connectivity and survivability requirements," in *INFOCOM*.

[77] Chatterjea, S., T. Nieberg, N. Meratnia, and P. Havinga (2008) "A distributed and self-organizing scheduling algorithm for energy-efficient data aggregation in wireless sensor networks," *ACM Trans. On Sensor Network*, **4**(4), pp. 1–41.

[78] Fan, K.-W., S. Liu, and P. Sinha (2006) "On the Potential of Structure-free Data Aggregation in Sensor Networks," in *INFOCOM*.

[79] Zheng, R. and R. Barton (2007) "Toward Optimal Data Aggregation in Random Wireless Sensor Networks," in *INFOCOM*.

[80] Yu, B., J. Li, and Y. Li (2009) "Distributed Data Aggregation Scheduling in Wireless Sensor Networks," in *INFOCOM*.

[81] Pattem, S., B. Krishnamachari, and R. Govindan (2004) "The impact of spatial correlation on routing with compression in wireless sensor networks," in *IPSN*.

[82] Ciancio, A., S. Pattem, A. Ortega, and B. Krishnamachari (2006) "Energy-efficient data representation and routing for wireless sensor networks based on a distributed wavelet compression algorithm," in *IPSN*.

[83] Biswas, S. and R. Morris (2005) "ExOR: opportunistic multi-hop routing for wireless networks," in *MOBICOM*.

[84] Dong, Q., S. Banerjee, M. Adler, and A. Misra (2005) "Minimum energy reliable paths using unreliable wireless links," in *MOBIHOC*.

[85] Liu, C. and G. Cao (to appear) "Spatial-Temporal Coverage Optimization in Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*.

[86] Hui, J. W. and D. Culler (2004) "The dynamic behavior of a data dissemination protocol for network programming at scale," in *ACM SenSys*, pp. 81–94.

[87] Hagedorn, A., D. Starobinski, and A. Trachtenberg (2008) "Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks Using Random Linear Codes," in *IPSN*, pp. 457 –466.

[88] Levis, P., N. Patel, D. Culler, and S. Shenker (2004) "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *USENIX NSDI*, pp. 2–2.

[89] Kulkarni, S. and L. Wang (2005) "MNP: Multihop Network Reprogramming Service for Sensor Networks," in *ICDCS*, pp. 7 –16.

[90] Tsiftes, N., A. Dunkels, and T. Voigt (2008) "Efficient Sensor Network Reprogramming through Compression of Executable Modules," in *IEEE SECON*, pp. 359 –367.

[91] Panta, R., I. Khalil, and S. Bagchi (2007) "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," in *IEEE INFOCOM*, pp. 928 –936.

[92] Hyun, S., P. Ning, A. Liu, and W. Du (2008) "Seluge: Secure and DoS-Resistant Code Dissemination in Wireless Sensor Networks," in *IPSN*, IEEE, pp. 445–456.

[93] Yu, Y., L. J. Rittle, V. Bhandari, and J. B. LeBrun (2006) "Supporting concurrent applications in wireless sensor networks," in *SenSys*, ACM.

[94] Rangwala, S., R. Gummadi, R. Govindan, and K. Psounis (2006) "Interference-aware fair rate control in wireless sensor networks," in *SIGCOMM*, ACM, pp. 63–74.

[95] Karenos, K., V. Kalogeraki, and S. V. Krishnamurthy (2008) "Cluster-based congestion control for sensor networks," *ACM Trans. On Sensor Network*, **4**(1), pp. 1–39.

[96] Hull, B., K. Jamieson, and H. Balakrishnan (2004) "Mitigating congestion in wireless sensor networks," in *SenSys*, ACM, pp. 134–147.

[97] Ee, C. T. and R. Bajcsy (2004) "Congestion control and fairness for many-to-one routing in sensor networks," in *SenSys*, ACM, pp. 148–161.

[98] Gao, W., Q. Li, B. Zhao, and G. Cao (2009) "Multicasting in delay tolerant networks: a social network perspective," in *MobiHoc*, ACM.

[99] Jones, E. P. C., L. Li, J. K. Schmidtke, and P. A. S. Ward (2007) "Practical Routing in Delay-Tolerant Networks," *IEEE Transactions on Mobile Computing*, **6**(8), pp. 943–959.

[100] Sarigöl, E., O. Riva, P. Stuedi, and G. Alonso (2009) "Enabling social networking in ad hoc networks of mobile phones," *Proc. VLDB Endowment*, **2**(2), pp. 1634–1637.

Vita

Changlei Liu

Changlei Liu is a Ph.D Student at the Department of Computer Science and Engineering, Pennsylvania State University. He received his M.Phil degree from the University of Hong Kong and B.E degree from the University of Science and Technology of China, both in Electronic Engineering. His research interest is in the area of wireless sensor networks, mobile computing and distributed system.

Publications During The PhD Study:

- Changlei Liu, Guohong Cao, Spatial-Temporal Coverage Optimization for Wireless Sensor Networks, Accepted by *IEEE Transactions on Mobile Computing (TMC)*, 2010.
- Changlei Liu, Guohong Cao, Distributed Monitoring and Aggregation in Wireless Sensor Networks, *29th Annual IEEE Conference on Computer Communications* (*INFOCOM'10*), March 15-19, 2010, San Diego, USA.
- Changlei Liu, Guohong Cao, Minimizing the cost of mine selection via sensor networks, *28th Annual IEEE Conference on Computer Communications* (*INFOCOM'09*), April 19–25, 2009, Rio de Janeiro, Brazil.
- Changlei Liu, Guohong Cao, An Multi-Poller based Monitoring Scheme for Wireless Sensor Networks, *28th Annual IEEE Conference on Computer Communications* (*INFOCOM'09*) mini-conference, 2009, Rio de Janeiro, Brazil.
- Lu Su, Changlei Liu, Hui Song, Guohong Cao, Routing in Intermittently Connected Sensor Networks, *16th Annual IEEE International Conference on Network Protocols* (*ICNP'08*), Oct 19–22, 2008, Orlando, Florida, USA.
- Changlei Liu, Guohong Cao, An Energy Efficient Fault Tolerant Monitoring Architecture for Wireless Sensor Networks, *ACM Mobile Computing Review 2008*.