The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

DEPENDABLE SENSOR NETWORKS

A Thesis in

Computer Science and Engineering

by

Guiling Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2006

The thesis of Guiling Wang has been reviewed and approved* by the following:

Piotr Berman
Associate Professor of Computer Science and Engineering
Thesis Co-Adviser
Co-Chair of Committee

Thomas F. LaPorta
Professor of Computer Science and Engineering
Thesis Co-Adviser
Co-Chair of Committee

Mary Jane Irwin
Distinguished Professor of Computer Science and Engineering

George Kesidis
Associate Professor of Electrical Engineering

Runze Li
Associate Professor of Statistics

Srimat Chakradhar
Department Head of NEC Research Labs
Special Member

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

*Signatures are on file in the Graduate School.

Abstract

Due to small size, low cost, and many other attractive features of sensor nodes, sensor networks have become adapted to a vast array of applications in both military and civil sectors, such as military surveillance, smart homes, and remote environment monitoring.

To enable the usage of sensor networks in these applications, network dependability is the basic requirement. Specifically, a sensor network must successfully detect the phenomenon of interest, and transmit the generated data to the users reliably. However, this is a challenging task due to the harsh working environment and the extremely limited capability of sensor nodes.

In this thesis, I have presented a number of protocols that are designed to improve the dependability of sensor networks. One fundamental requirement for a dependable sensor network is a suitable coverage, since a sensor network is deployed to sense a target environment, and coverage determines how well it can detect the phenomena of interest. Given such a requirement, the first step of my thesis research is to efficiently deploy sensor nodes to achieve required coverage. After the initial deployment, the dependability of the network may be degraded due to node failure, malfunction or compromise. To keep a sensor network dependable during its lifetime, I address both fault tolerance and security issues. In particular, the second step of my thesis research is two-fold:

self-healing in response to node malfunction; and security mechanisms to protect communication. Extensive simulation has shown that our schemes are effective in improving the dependability of sensor networks in an efficient way.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I am very grateful to my advisors Professor Thomas F. La Porta and Professor Piotr Berman. Professor La Porta enlightened my initial research interest and opened a door to research for me. After that, he guided me to achieve one progress after another with his constructive and insightful suggestions. I can feel my improvement after countless discussion and communication with him. He led me to deeply understand the concept of "tradeoff"; he taught me how to view problems in a big picture; ····· He guided me not only in research, but also in career; he is not only helping, but also generous. I still can remember he helped me on my paper when he got a heavy cold. Professor Berman always stimulated me with his smart brain of a Mathematician. His comments are sharp and abstract. He could quickly point out the problems of my algorithms and stimulated me to work out intelligent solutions. He helped me and guided me not only as an advisor, but also like a father.

I would like to thank Dr. Srimat Chakradhar, who was my supervisor when I was doing internship at NEC Research Labs. With his stimulating and intelligent supervision, I worked out the pairwise key establishment schemes for sensor networks.

I would like to thank Professors George Kesidis, Runze Li, Guohong Cao, and Martin Fürer for their constructive and insightful comments on this thesis and for their generous help on my research.

I would especially like to thank Professor Mary Jane Irwin. I could never express enough my gratitude to her for her support, help, encouragement and her guidance, without which, I could not work out this thesis.

I would also like to thank my co-authors, Dr. Wensheng Zhang and Dr. Haoying Fu for their discussions, cooperations and generous help.

Chapter 1

Introduction

Recent advances in micro-electro-mechanics, micro-processor and wireless communication have enabled the design of small-size, low-cost sensor nodes, in which all the sensing, computation, and communication capabilities are integrated. When a large number of such sensor nodes are deployed into some space, they can self-organize into a multihop wireless sensor network [8, 7, 72], to sense the environment, and generate, process, and transmit the data of interest. Since wireless sensor networks can greatly enhance our capability of monitoring and controlling the physical environment, they have become adapted to a vast array of applications in both military and civil sectors, such as military surveillance, smart homes, and remote environment monitoring. They are expected to be intensively utilized in the future.

Dependability is the basic requirement in designing a sensor network and enabling the wide usage of sensor networks. A sensor network must successfully detect the phenomenon of interest, and transmit the generated data to the users reliably. For example, a sensor network monitoring a battle field must detect every enemy tank intruding the field and report the event to the command center in a timely manner.

However, it is a challenging task to design a dependable sensor network because of the harsh working environment and the extremely limited capability of sensor nodes.

In many applications, sensor networks are deployed in unattended fields, in which, a desirable sensor distribution is difficult to achieve since manual deployment is nearly impossible and the deployment may be affected by uncontrollable factors such as wind and obstacles. This non-optimal deployment may result in mis-detection of events of interest. Such environments also make it infeasible to recharge the batteries of sensors. Consequently, some sensor nodes may run out of power and die. Moreover, in such environments, sensor nodes are vulnerable to physical damage or node capture, resulting in node failure or node compromise. For dependability, the problem of node compromise must be addressed. However, the limited capability of sensor nodes makes it difficult to employ the conventional security mechanisms such as public key crypto systems, which are costly in terms of memory and CPU consumption. Therefore, efficient and cheap security mechanisms must be developed to protect sensor networks.

My thesis research has focused on improving the dependability of sensor networks. One fundamental requirement for a dependable sensor network is a suitable coverage since a sensor network is deployed to sense a target environment, and coverage determines how well a sensor network can detect the phenomena of interest. Given such a requirement, the first step of my thesis research is how to efficiently deploy sensor nodes to achieve required coverage. After the initial deployment, the dependability of the network may be degraded due to node failure, malfunction or compromise. To keep a sensor network dependable during its lifetime, I address issues of fault tolerance and security. In particular, the second step of my thesis research is two-fold: self-healing in response to node malfunction; and security mechanisms to protect communication.

1.1 Sensor Deployment

In many applications such as remote surveillance, manual deployment of sensor nodes is infeasible. Nodes may be deployed from aircrafts, for example. As a result, the sensing and communication coverage cannot be controlled. In early works, researchers assume that sensor nodes are static and that a large number of redundant nodes are deployed in order to achieve a desired level of coverage. This however may introduce high cost and still cannot guarantee coverage, especially in the presence of obstacles. Recently, mobile sensors [78] have been developed and are expected to be applied in practice shortly. This motivates us to adopt mobile sensor nodes to address the coverage problem. The introduction of mobile sensor nodes, however, brings forth new issues such as how mobile nodes interact with each other and with static nodes, and brings forth new cost, performance, and communication tradeoffs. I have investigated these topics in a stepwise manner. First, I assume a system with all mobile sensor nodes, and study how the nodes can effectively collaborate to improve coverage in a distributed fashion. Then, I consider a more realistic system including a mixed set of mobile and static sensor nodes. In the third step, I focus on the movement cost of mobile sensor nodes, and propose efficient motion planing algorithms.

1.1.1 Deploying Pure Mobile Sensor Networks

To deploy mobile sensors from where they are initially distributed, I proposed three algorithms, VOR, VEC, and Minimax [41, 39], for individual nodes to independently calculate their required movement to increase coverage, and two sets of distributed

protocols to coordinate their movements and exchange information. The different combinations of the calculation algorithms and protocols, tuned by multiple parameters, allow users to choose different levels of coverage, energy consumption and deployment time depending on system requirements.

1.1.2 Deploying Mixed-type Sensor Networks

With a mix of mobile and static sensors, it becomes a NP-hard problem to place mobile sensors to maximize sensing coverage. To tackle this NP-hard problem in a distributed fashion, I have proposed two distributed *bidding* protocols [40, 42, 43], the basic bidding protocol and the proxy-based bidding protocol, to move sensors. In the protocols, static sensors detect coverage holes and act as bidders to bid mobile sensors based on the size of the detected holes. Mobile sensors act as servers, accept the highest bids and heal the largest holes. The protocols can achieve approximately optimal coverage. Also, we give insight on how to construct such networks most economically by varying the ratio of fixed and mobile sensors.

1.1.3 Motion Planning

Most applications utilizing mobile sensors reduce energy consumption by reducing the number of movements and the moving distance of each sensor. I have worked on the problem of minimizing the energy consumption given a moving task from the applications. I have proposed algorithms [45] to optimize sensor movement for energy efficiency, which can be implemented as middleware and can be called transparently by the applications utilizing the motion capability of sensors. Unlike previous work, I adopt

a complete energy model to characterize the entire energy consumption in movement. Based on the model, I proposed an optimal velocity schedule for minimizing energy consumption when the road condition is uniform and a near optimal velocity schedule for the variable road condition by using continuous-state dynamic programming.

1.2 Dependable Sensor Networks

Starting from a good point with satisfactory sensor distribution after the initial deployment, the performance of a sensor network may be degraded or even become non-trustworthy afterwards due to node failure, node compromise, etc. This is because sensor nodes are resource-constrained, have limited or no physical protection, and are subject to eavesdropping and unauthorized access. All these factors make it a challenging task to maintain the dependability of a sensor network during its lifetime. The second step of my thesis research focuses on constructing a dependable and self-healing sensornet. My research is conducted from two aspects: First, I address the ability of a sensor network to self-heal when node or network malfunction is detected and reported. My objective is to construct a sensornet which is resilient to node failures and reactive to events. Second, I propose security mechanisms which can protect the communication of sensornets, quickly exclude the compromised sensor nodes from communication, and greatly reduce the possibility of unauthorized access.

1.2.1 Sensor Relocation

When sensors fail or are excluded from the network for some reason, resulting in coverage or communication holes, the defective nodes should be replaced with well-functioning nodes. The replacement should be done in a timely manner to reduce the impact of the malfunctioned nodes. It should be done in an efficient and balanced manner, considering energy efficiency. Also, sensor replacement should affect the applications currently running in the network as little as possible. Considering all these challenging requirements, I have proposed a sensor relocation framework [44], in which a *Grid-Quorum* solution can quickly locate the closest redundant sensor with low message complexity, and a *cascaded movement* can relocate the redundant sensor in a timely, efficient and balanced way.

1.2.2 Pair-wise Key Establishment for Sensor Networks

Wireless sensor networks are subject to eavesdropping and node capture and compromise in an unattended working environment. To provide many security features, it is necessary for the neighboring nodes to share a pair-wise key. I have designed a pair-wise key establishment scheme based on a novel idea of key migration [46]. The scheme can provide a high degree of secure connectivity and tolerate node compromise.

1.3 Thesis Organization

The rest of my thesis is organized as follows. I present the distributed algorithms to deploy pure mobile sensor networks in Chapter 2 and the algorithms to deploy a mixed of mobile and static sensors in Chapter 3. Algorithms to optimize sensor movement for

energy efficiency is presented in Chapter 4. In Chapter 5, I present the sensor relocation schemes, and in Chapter 6, I present the pair-wise key establishment schemes for sensor networks. I survey related work in Chapter 7 and finally conclude the thesis in Chapter 8.

Chapter 2

Deploying Pure Mobile Sensor Networks

2.1 Introduction

In this chapter, we study the problem of placing mobile sensors to get high coverage. Most previous research efforts on deploying mobile sensors are based on centralized approaches. For example, the work in [91] assumes that a powerful cluster head is available to collect the sensor locations and determine the target locations of the mobile sensors. However, in many sensor deployment environments such as disaster areas and battle fields, a central server may not be available. It may also be hard to organize sensors into clusters due to network partitions. Further, centralized approaches introduce a single point of failure. Sensor deployment has also been addressed in the field of robotics [49], where sensors are deployed iteratively one by one, utilizing the location information obtained from the previous deployment. Since sensors are deployed one by one, the deployment time is very long which can significantly increase the network initialization time.

In this chapter, we propose two sets of distributed protocols for controlling the movement of sensors to achieve target coverage: *basic protocols* and *virtual movement protocols*. In the basic protocols, sensors move iteratively, eventually reaching the final destination. In each iteration, sensors detect coverage holes using a Voronoi diagram. If holes exist, they calculate the target locations to heal the holes and move. In the virtual

movement protocols, sensors do not perform iterative physical movement. Instead, after calculating the target locations, sensors move virtually, and exchange these new virtual locations with the sensors which would be their neighbors if they had actually moved. The real movement only occurs when the communication cost to reach their logical neighbors is too high or when they determine their final destinations.

In both the basic and virtual movement protocols, three algorithms, VEC, VOR, and Minimax, are proposed to calculate the target locations if coverage holes exist. In VEC, sensors move away from a dense area; in VOR, sensors migrate towards holes; in Minimax, sensors also move towards holes, but more conservatively with the consideration of not generating new holes. Simulation results show that our distributed protocols are effective in terms of coverage, deployment time and movement.

The rest of the chapter is organized as follows. Section 2.2 introduces some preliminaries. In section 2.3, we present the basic self-deployment protocols and in section 2.4, we present the virtual movement protocols. Section 2.5 evaluates the performance of the proposed protocols. Based on the simulation results, we justify our design and discuss future work in Section 2.6.

2.2 Preliminaries

In this section, we present the necessary background on localization techniques, path planning, the sensing model and Voronoi diagrams.

2.2.1 Localization Techniques

Location awareness is important for wireless sensor networks since many applications such as environment monitoring and target tracking depend on knowing the locations of sensor nodes. Due to the ad hoc nature of such networks, each node must determine its location. For outdoor systems, Global Positioning System (GPS) [1] is one method for this purpose. Because GPS may not be cost effective or work well indoors, other techniques have been proposed to enable each node to determine its location indoors with only limited communication with nearby nodes. Most of these methods exploit received signal strength [66], time difference of arrival of two different signals [76], and angle of arrival [22]. Hu et al. [51] have provided detailed discussion of these techniques. In the subsequent discussions of this paper, we assume that sensor nodes know their locations.

2.2.2 Path Planning

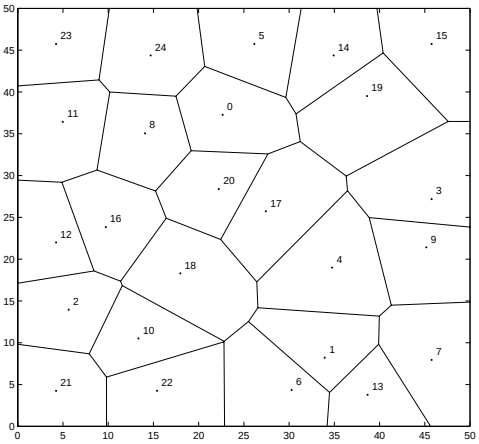
In systems that exploit mobile sensors, finding paths on which these mobile sensors can move to desired destinations, especially when there exist obstacles in the field, is an important problem. The problem has been studied in the area of robotics [21, 57]. Recently, Li et al. [58] studied the problem in sensor networks. They combined the above methods to find the best motion path and to exploit the distributed nature of sensor networks. In this paper, we do not study this problem further; we assume that mobile sensors can move to any location where they are asked to move based on the existing techniques. We comment more on the impact of this assumption in Section 2.5.

2.2.3 Sensing Model

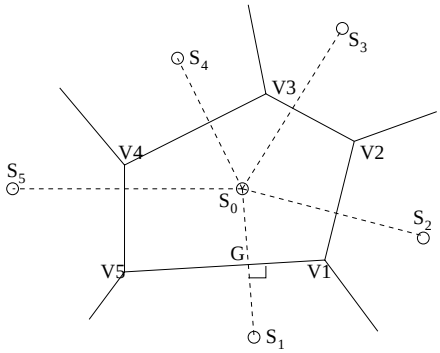
Each type of sensor has a unique sensing model characterized by its sensing area, resolution and accuracy. The sensing area depends on multiple factors such as the strength of the signals generated at the source, the distance between the source and the sensor, the attenuation rate in propagation, and the desired confidence level of sensing. Let us consider an application [31] in which a network of acoustic sensors is deployed for detecting mobile vehicles. Due to signal attenuation, sensors closer to a vehicle can detect higher strength of acoustic signals than sensors farther away from the vehicle, and thus have higher confidence for detecting the vehicle. Therefore, given a confidence level, we can derive a sensing range surrounding each sensor. In this paper, we only consider isotropic sensing models. Each sensor node is associated with a sensing area which is represented by a circle with the same radius. This is a common assumption when comparing algorithms for sensing coverage [62, 63].

2.2.4 Voronoi Diagram

The Voronoi diagram [10, 32] is an important data structure in computational geometry. It represents the proximity information about a set of geometric nodes. The Voronoi diagram of a collection of nodes partitions the space into polygons. Every point in a given polygon is closer to the node in this polygon than to any other node. Figure 2.1(a) is an example of the Voronoi diagram, and Figure 2.1(b) is an example of a Voronoi polygon. We define the Voronoi polygon of s_0 as $G_0 = \langle \mathcal{V}_0, \mathcal{E}_0 \rangle$, where $\mathcal{V}_0$ is the set of Voronoi vertices of s_0, and $\mathcal{E}_0$ is the set of Voronoi edges. As shown in Figure 2.1(b), $\mathcal{V}_0 = \{V_1, V_2, V_3, V_4, V_5\}$, and $\mathcal{E}_0 = \{V_1V_2, V_2V_3, V_3V_4, V_4V_5, V_5V_1\}$. We use $\mathcal{N}_0$



(a) Voronoi diagram



(b) Voronoi polygon G_0 of s_0

Fig. 2.1. Voronoi diagram

to denote the set of Voronoi neighbors of s_0. In Figure 2.1(b), $\mathcal{N}_0 = \{s_1, s_2, s_3, s_4, s_5\}$. The Voronoi edges of s_0 are the vertical bisectors of the line passing s_0 and its Voronoi neighbors, e.g., V_1V_5 is s_0s_1's bisector.

Our sensor deployment protocols are based on Voronoi diagrams. As shown in Figure 2.1, each sensor, represented by a number, is enclosed by a Voronoi polygon. These polygons together cover the target field. The points inside one polygon are closer to the sensor inside this polygon than the sensors positioned elsewhere. If this sensor cannot detect the expected phenomenon in its Voronoi polygon, no other sensor can detect it. Therefore, to examine coverage holes, each sensor only needs to check its own Voronoi polygon. If its sensing area cannot cover the polygon, there are some coverage holes.

To construct the Voronoi polygon, sensors first calculate the bisectors of their neighbors and themselves. These bisectors (and possibly the boundary of the target field) form several polygons. The smallest polygon encircling the sensor is the Voronoi polygon of this sensor.

2.2.5 Sensing Range versus Communication Range

In a distributed case, sensors can exchange the location information by broadcasting. It is possible that some Voronoi neighbors of a sensor are out of its communication range, and consequently, the calculated polygon of this sensor is not accurate. If the sensing range is much shorter than the communication range, then the inaccurate construction of Voronoi cell will not affect the detection of coverage holes. This is because, if Voronoi neighbors cannot reach each other by direct communication, their distance

is large enough that there is a coverage hole. If communication range is similar to the sensing range, sensors may mis-detect coverage holes. We describe our heuristics to deal with the inaccurate construction of the Voronoi polygons in Section 2.3.

2.3 The Basic Deployment Protocols

Our deployment protocol runs iteratively. In each round, sensors first broadcast their locations and construct their Voronoi polygons based on the received neighbor information. Sensors then determine the existence of coverage holes by examining their Voronoi polygons. If any hole exists, sensors calculate where to move to eliminate or reduce the size of the coverage hole. Three algorithms are proposed to calculate the target locations: VEC *pushes* sensors away from a densely covered area; VOR *pulls* sensors to the sparsely covered area; and Minimax moves sensors to the center of their Voronoi polygon. Termination conditions are defined for each algorithm.

We first present distributed calculation of the Voronoi Cells, and then present the VEC, VOR and Minimax algorithms.

2.3.1 Distributed Calculation of the Voronoi Cell

It is difficult to compute Voronoi diagrams [10]. However, to detect and calculate the coverage hole, each sensor only needs to know its own Voronoi cell, whose calculation can be simplified as follows. We take sensor s_0 as an example. Initially, as shown in Figure 2.2 (a), the Voronoi cell of s_0 is set to be a large rectangle. After receiving the *hello* message from sensor s_1, s_0 knows the location of s_1 and computes the bisector line of s_1 and itself. This line is added to the original graph and two polygons are

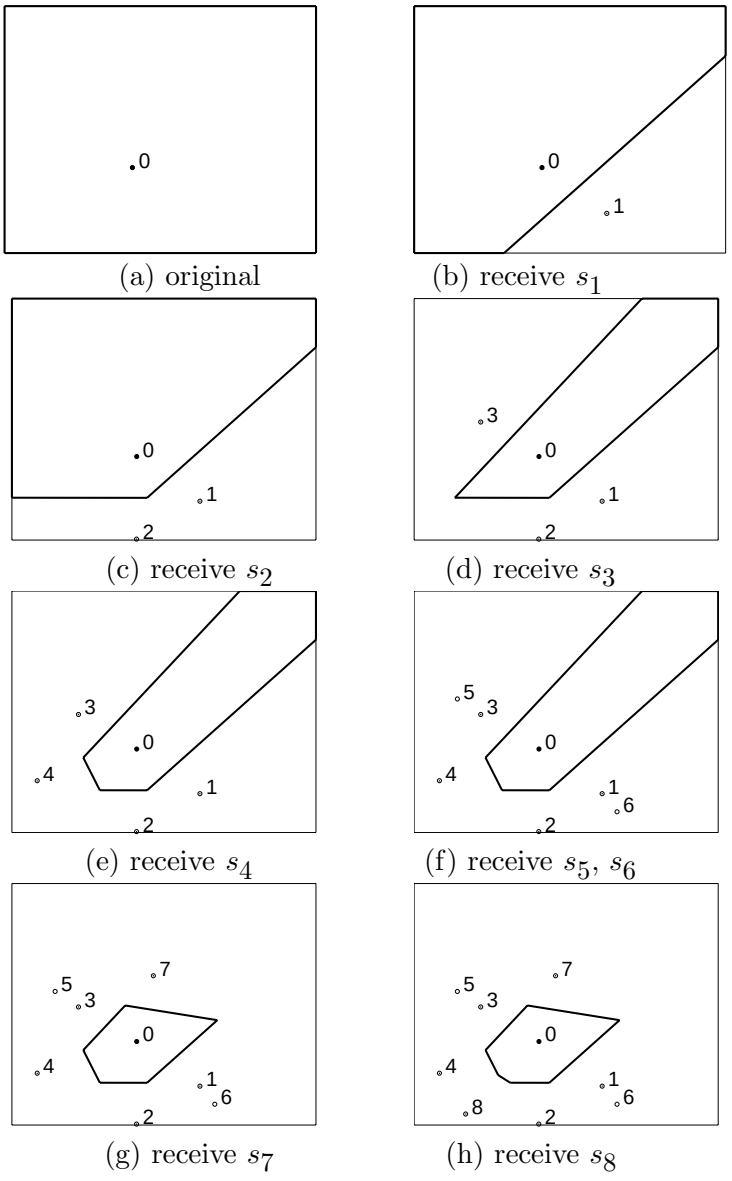


Fig. 2.2. Computing the Voronoi cell

generated. Shown in Figure 2.2 (b), the polygon including s_0 becomes the new view of s_0's Voronoi cell. Later, after s_0 receives the *hello* messages from s_2, s_3, and s_4, its Voronoi cell changes from Figure 2.2(c) to Figure 2.2(e) accordingly. The Voronoi cell will not change if the computed bisector line has no intersection with it. As shown in Figure 2.2(f), knowing s_5 and s_6 does not affect s_0's Voronoi cell. Finally, the true Voronoi cell is generated after s_0 knows the existence of s_7 and s_8.

Static sensors construct Voronoi cells considering only static neighbors and mobile neighbors which are not likely to move. These mobile sensors are detected by examining their base prices. If the base price of a mobile sensor is zero, this mobile sensor has not moved yet and most likely it will move to heal some coverage hole. Thus, when detecting coverage holes, static sensors do not consider mobile sensors which are likely to leave. To find out if a coverage hole exists, a static sensor checks whether its distance to the farthest Voronoi vertex is longer than the sensing range. If yes, then some coverage hole exists and this sensor should prepare to bid some mobile sensor to heal it.

Voronoi cells calculated in this way will not be accurate when Voronoi neighbors are far away from each other and cannot communicate with each other. The accurate calculation of Voronoi cells is not required in these cases because the coverage holes will be large. The algorithm will not mis-detect coverage holes.

2.3.2 The VECtor-based Algorithm(VEC)

VEC is motivated by the attributes of electro-magnetic particles: when two electro-magnetic particles are too close to each other, an expelling force pushes them apart. Assume $d(s_i, s_j)$ is the distance between sensor s_i and sensor s_j. d_{ave} is the

average distance between two sensors when the sensors are evenly distributed in the target area, which can be calculated beforehand since the target area and the number of sensors to be deployed are known. The virtual force between two sensors s_i and s_j will push them to move $(d_{ave} - d(s_i, s_j))/2$ away from each other. In case one sensor covers its Voronoi polygon completely and should not move, the other sensor will be pushed $d_{ave} - d(s_i, s_j)$ away. In summary, the virtual force will push the sensors d_{ave} away from each other if coverage hole exists in either of their Voronoi polygons. The virtual force exerted by s_j on s_i is denoted as $\vec{F}_{ij}$, with the direction from s_j to s_i.

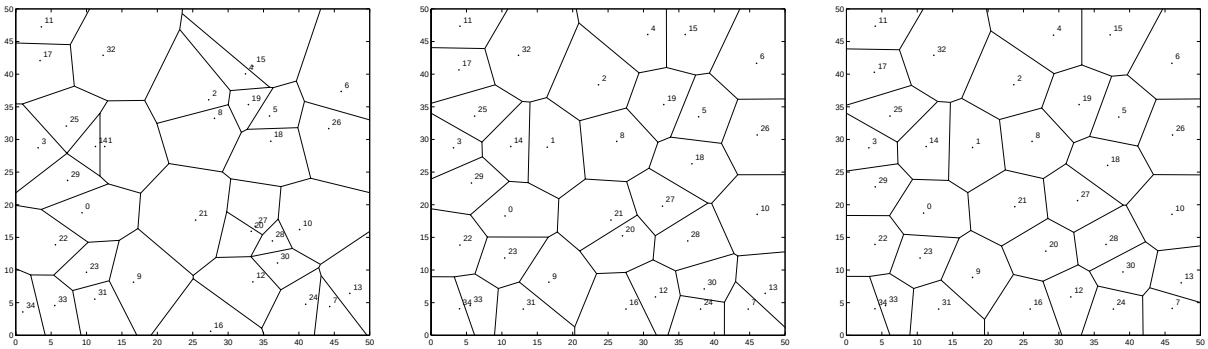
In addition to the virtual forces generated by sensors, the field boundary also exert forces, denoted as $\vec{F}_b$, to push sensors too close to the boundary inward. $\vec{F}_b$ exerted on s_i will push it to move $d_{ave}/2 - d_b(s_i)$, where $d_b(s_i)$ is the distance of s_i to the boundary. Since d_{ave} is the average distance between sensors, $d_{ave}/2$ is the distance from the boundary to the sensors closest to it when sensors are evenly distributed.

The final overall force on sensors is the vector summation of virtual forces from the boundary and all Voronoi neighbors. These virtual forces will push sensors from the densely covered area to the sparsely covered area. Thus, VEC is a "proactive" algorithm, which tries to relocate sensors to be evenly distributed.

As an enhancement, we add a *movement-adjustment* scheme to reduce the error of *virtual-force*. When a sensor determines its target location, it checks whether the local coverage will be increased by its movement. The local coverage is defined as the coverage of the local Voronoi polygon and can be calculated by the intersection of the polygon and the sensing circle. If the local coverage is not increased, the sensor should not move to the target location. Although the general direction of the movement is correct, the local

coverage may not be increased because the target location is too far away. To address this problem, the sensor will choose the midpoint or 3/4 point between its target location and its current location as its new target location. If the local coverage is increased at the new target location, the sensor will move; otherwise, it will stay.

Figure 2.3 shows an operational example of VEC. Round 0 is the initial random deployment of 35 sensors in a $50m$ by $50m$ flat space, with the sensing range of 6 meters and communication range of 20 meters. The initial coverage is 75.7%. After Round 1 and Round 2, the coverage is improved to 92.2% and 94.7%, respectively. A formal description of the VEC algorithm is shown in Figure 2.4.



(a) Round 0 (b) Round 1 (c) Round 2

Fig. 2.3. Snapshot of the execution of VEC

2.3.3 The VORonoi-based Algorithm (VOR)

Contrary to the VEC algorithm, VOR is a *pull* algorithm which pulls sensors to cover their local maximum coverage holes. In VOR, if a sensor detects the existence of

Notations:
$\mathcal{N}_i$, G_i, $\vec{F}_{ij}$, $\vec{F}_b$, d_{ave}: defined before
c_i: whether G_i is completely covered
$\vec{v}_i$: moving vector of s_i

(1) *Upon entering Discovery phase:*
(1.1) set *timer* to be *discovery_interval*
enter *Moving phase* upon timeout
(1.2) broadcast *hello* after a random time slot
(2) *Upon entering Moving phase:*
(2.1) set *timer* to be *moving_interval*,
enter *Discovery phase* upon timeout
(2.2) **if** $c_i = false$ **then**
call $VEC()$ /* call VOR and Minimax in other protocols */
(2.3) Done when satisfying stop criteria
(3) *Upon receiving a hello message from sensor* s_j*:*
(3.1) Update $\mathcal{N}_i$ and G_i
(3.2) **if** G_i is newly covered **then**
set $c_i = true$
Broadcast OK /* only for VEC */
/* The following is only for VEC, and will be replaced in VOR and Minimax */
(4) *Upon receiving an* OK *message from sensor* s_j*:*
(4.1) set $c_j = true$
(5) $VEC()$
(5.1) $\vec{v_i} = \vec{0}$
(5.2) **for** each s_j in $\mathcal{N}_i$
if $(c_j \neq true) \wedge (d_{ave} > d(s_i, s_j))$ **then**
$|\vec{F_{ij}}| = (d_{ave} - d(s_i, s_j))/2$; $\vec{v_i} = \vec{v_i} + \vec{F_{ij}}$
if $(c_j = true) \wedge (d_{ave} > d(s_i, s_j))$ **then**
$|\vec{F_{ij}}| = d_{ave} - d(s_i, s_j)$; $\vec{v_i} = \vec{v_i} + \vec{F_{ij}}$
(5.3) **if** $(d_{ave}/2 > d_b(s_i))$ **then**
$|\vec{F_b}| = d_{ave}/2 - d_b(s_i)$; $\vec{v_i} = \vec{v_i} + \vec{F_b}$
(5.4) **do** *movement adjustment*

Fig. 2.4. The VEC protocol at sensor s_i

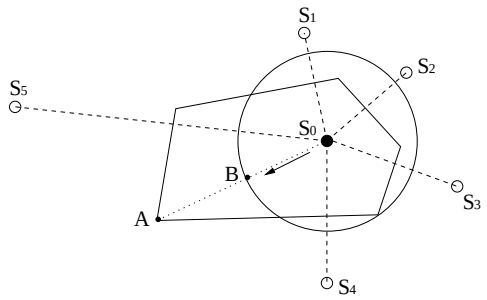


Fig. 2.5. VOR

coverage holes, it will move toward its farthest Voronoi vertex (denoted as V_{far}), and stop when the farthest Voronoi vertex can be covered. Same as VEC, in VOR, a sensor needs only to check its own Voronoi polygon. Figure 2.5 illustrates VOR. Point A is the farthest Voronoi vertex of s_0, and $d(A, s_0)$ is longer than the sensing range. To heal the hole, s_0 moves along line s_0A to Point B, where $d(A, B)$ is equal to the sensing range.

We limit the maximum moving distance to be at most half of the communication range minus the sensing range to avoid the situation shown in Figure 2.6, in which s_0 is not aware of the existence of s_1 because of communication limitations. When s_0 does not know s_1, it will calculate its local Voronoi polygon as the dotted one and view the area around A as a coverage hole. If s_0 moves toward point A and stops at a distance $d(A, B)$ (sensing range), s_0 has moved more than needed and it tries to cover the area which is already covered by s_1. It is quite possible it has to move back after it gets to know s_1. Therefore, we set the maximum moving distance such that a sensor moves towards the coverage hole step-by-step. After s_0 moves certain distance, and it gets closer to s_1, it

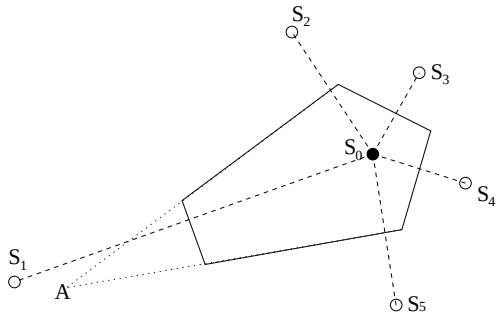


Fig. 2.6. Inaccurate Voronoi polygon

can communicate with s_1 and calculate the correct Voronoi polygon. Then the risk of moving oscillation can be greatly reduced.

VOR is a greedy algorithm which tries to fix the largest hole. Movement oscillations may occur if new holes are generated due to sensor's leaving. To deal with this problem, we add *oscillation control* which does not allow sensors to move backward immediately. Before a sensor moves, it first checks whether its moving direction is opposite to that in the previous round. If yes, it stops for one round. In addition, the movement adjustment mentioned in VEC is also applied here.

The deployment protocol using VOR is similar to the VEC Protocol, except that in line (2.2) $VEC()$ is replaced by $VOR()$, which is shown below.

Notations:

d_{max}: maximum moving distance

$\vec{v}_{i,f}$: vector from s_i to V_{far}

$VOR()$

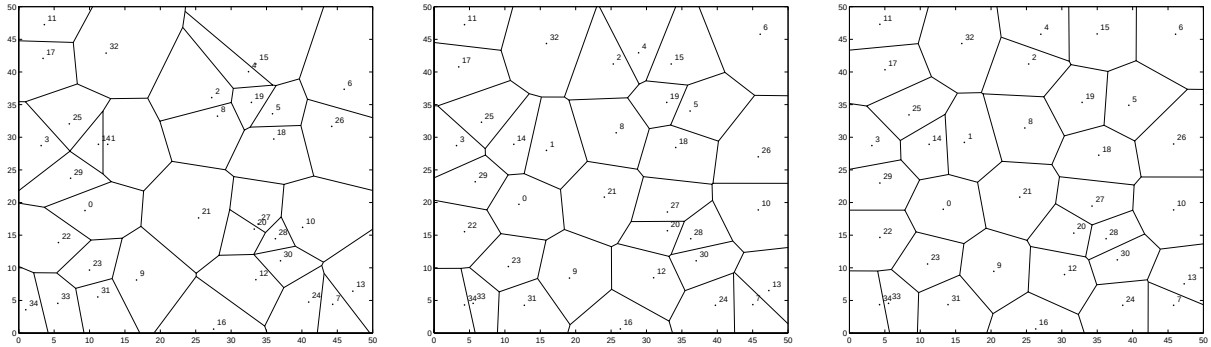
(1) $\vec{v_i} = \vec{v}_{i,f}$ - sensing range

(2) shrink $|\vec{v_i}|$ to be d_{max} if $|\vec{v_i}| > d_{max}$

(3) **do** *oscillation control*

(4) **do** *movement-adjustment*

Figure 2.7 shows an operational example of VOR. With the original coverage 75.7%, after round 1 and round 2, the coverage is improved to 89.2% and 95.6%, respectively.



(a) Round 0 (b) Round 1 (c) Round 2

Fig. 2.7. Snapshot of the execution of VOR

2.3.4 The Minimax Algorithm

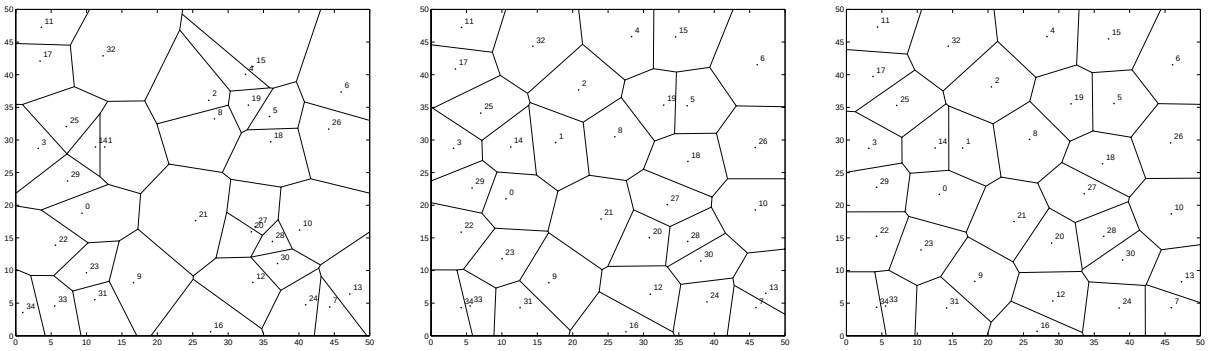
Similar to VOR, Minimax fixes holes by moving closer to the farthest Voronoi vertex, but it does not move as far as VOR to avoid situations in which a vertex that was originally close becomes a new farthest vertex. Minimax chooses the target location as the point inside the Voronoi polygon whose distance to the farthest Voronoi vertex (V_{far}) is minimized. We call this point the *Minimax point*, denoted as O_m. This algorithm is based on the belief that a sensor should not be too far away from any of its Voronoi vertices when the sensors are evenly distributed. Minimax can reduce the variance of the distances to the Voronoi Vertices, resulting in a more regular shaped Voronoi polygon, which better utilizes sensor's sensing circle. Compared with VOR, Minimax considers more information and it is more conservative. Compared with VEC, Minimax is "reactive"; it fixes the hole more directly by moving toward the farthest Voronoi vertex.

The Minimax point is the center of the smallest enclosing circle of the Voronoi vertices and can be calculated by the algorithms described in [61, 79, 82]. In the deployment protocol using Minimax, we also specify the maximum moving distance, and do oscillation control as in VOR.

Figure 2.8 shows an example of Minimax. With the original coverage 75.7%, after round 1 and round 2, the coverage is improved to 92.7% and 96.5%, respectively.

2.3.5 Termination

The algorithm terminates naturally based on the *movement-adjustment* heuristic (explained in Section 2.3.2), which does not allow sensors to move unless the local



(a) Round 0 (b) Round 2 (c) Round 3

Fig. 2.8. Snapshot of the execution of Minimax

coverage can be increased. The total coverage, bounded by 100%, increases as the local coverage increases. Based on the attributes of Voronoi diagram, the local coverage increase of one sensor does not affect the local coverage of another sensor. Thus, sensors will stop naturally when the coverage cannot be increased. The formal proof is as follows.

We denote the following terms for the proof. The location of sensor s_i in the r^{th} round is denoted as $[x_i^{(r)}, y_i^{(r)}]$. The Voronoi polygon of s_i, G_i, in the r^{th} round is denoted as $G_i^{(r)}$. G_i changes in different rounds if s_i or its neighbors move. Area of the covered part of $G_i^{(r)}$ in the r^{th} round is denoted as $A_i^{(r)}$ and that in the $(r+1)^{th}$ round is denoted as $\hat{A}_i^{(r)}$. $G_i^{(r)}$ is a fixed polygon in the target field, but the covered portion of $G_i^{(r)}$ may be changed in different rounds since sensors may move and they cover different areas if they move. Therefore, $\hat{A}_i^{(r)}$ is not equal to $A_i^{(r)}$. Also, $\hat{A}_i^{(r)}$ is not equal to $A_i^{(r+1)}$. They refer to the covered area of different polygons. The area of the covered portion of $G_i^{(r)}$ by a sensor located at $[x, y]$ is denoted as $A_i^{(r)}([x, y])$. The area of the covered portion in the whole target field in the r^{th} round is $A_{total}^{(r)}$.

LEMMA 1. *(a)* $A_{total}^{(r)} = \sum_{i=1}^{n} A_i^{(r)}$;
(b) $A_{total}^{(r+1)} = \sum_{i=1}^{n} \hat{A}_i^{(r)}$.

Proof Voronoi diagram is a partition of the target field, so (a) is obvious. With the same reason, (b) is also correct. The summation of the covered area of partitions is the whole covered area in the target field, whatever a partitioning method is used. Therefore, the summation of the covered area of the Voronoi polygons in the previous round is also the whole covered area in the current round.

LEMMA 2. $A_i^{(r)} = A_i^{(r)}([x_i^{(r)}, y_i^{(r)}])$

Proof This is the direct result of the attribute of Voronoi diagram. Every point within $G_i^{(r)}$ is closer to $[x_i^{(r)}, y_i^{(r)}]$ than to any other sensor. Any point not covered by s_i is also not be covered by any other sensor.

THEOREM 1. $A_{total}^{(r+1)} > A_{total}^{(r)}$ *before all sensors stop moving.*

Proof At the $(r+1)^{th}$ round, there may be areas in $G_i^{(r)}$ which is not covered by s_i, but is covered by other sensors, because the current Voronoi polygon of s_i is $G_i^{(r+1)}$, but not $G_i^{(r)}$. Therefore,

$$\hat{A}_i^{(r)} \geq A_i^{(r)}([x_i^{(r+1)}, y_i^{(r+1)}]) \tag{2.1}$$

By enforcing the *movement adjustment* heuristics (described in section 2.3.2), our algorithms guarantee that, if s_i moves,

$$A_i^{(r)}([x_i^{(r+1)}, y_i^{(r+1)}]) > A_i^{(r)}([x_i^{(r)}, y_i^{(r)}]). \tag{2.2}$$

Certainly, if s_i does not move,

$$A_i^{(r)}([x_i^{(r+1)}, y_i^{(r+1)}]) = A_i^{(r)}([x_i^{(r)}, y_i^{(r)}]), \tag{2.3}$$

because $[x_i^{(r+1)}, y_i^{(r+1)}] = [x_i^{(r)}, y_i^{(r)}]$.

From (2.1),(2.2),(2.3)

$$\sum_{i=1}^{n} \hat{A}_i^{(r)} > \sum_{i=1}^{n} A_i^{(r)}([x_i^{(r)}, y_i^{(r)}]), \tag{2.4}$$

if some sensor moves in the r^{th} round.

From Lemma 2

$$A_i^{(r)} = A_i^{(r)}([x_i^{(r)}, y_i^{(r)}]) \tag{2.5}$$

From(2.4),(2.5)

$$\sum_{i=1}^{n} \hat{A}_i^{(r)} > \sum_{i=1}^{n} A_i^{(r)}, \tag{2.6}$$

if some sensor moves in the r^{th} round.

By Lemma 1,

$$\begin{cases} A_{total}^{(r)} = \sum_{i=1}^{n} A_i^{(r)} \\ \\ A_{total}^{(r+1)} = \sum_{i=1}^{n} \hat{A}_i^{(r)} \end{cases} \tag{2.7}$$

From (2.7),(2.6)

$$A_{total}^{(r+1)} > A_{total}^{(r)}, \tag{2.8}$$

if some sensor moves in the r^{th} round.

COROLLARY 1. *Our distributed algorithms are convergent, and thereby terminate naturally.*

Proof Following from Theorem 1 and the fact that $A_{total}^{(r)}$ is upper bounded by the total area of the target field, our distributed algorithms converge, and terminate naturally. All sensors stop moving when no coverage increase can happen.

In some applications, the coverage requirement may be met without achieving maximum coverage. In this case, it may be prudent to terminate the deployment process before the maximum coverage is reached to save power and reduce the deployment time. To terminate the deployment procedure earlier, we use a threshold ϵ, defined as the minimum increase in coverage below which a sensor will not move. With a larger ϵ, the deployment will finish earlier. When $\epsilon = 0$, sensors stop when the best coverage is obtained.

2.3.6 Optimizations

2.3.6.1 Dealing With Message Loss

Hello messages may be lost due to collisions. Consequently, sensors may fail to know the existence of some Voronoi neighbors and mistakenly determine coverage holes. To address this problem, we associate each item in a sensor's neighbor list with a number which indicates the freshness of this item. When constructing the Voronoi polygon, sensors only consider the sensors in its neighbor list with certain freshness. For example, only sensors that have been heard within the last two rounds can be considered when constructing the Voronoi polygon. Supposing the probability of message loss is 5%,

the probability that a message is lost two consecutive times is 0.25%. Therefore, if a sensor does not hear a HELLO message from a neighbor for two consecutive cycles, it can assume that the neighbor has moved and be correct with a 99.75% chance.

This solution introduces a new problem. If a sensor actually moves to a new place, its previous neighbors cannot hear it. If these old neighbors still consider this sensor in their formation of the Voronoi polygons until the freshness threshold is violated, it will prolong the deployment process. To address this problem, we propose that a sensor broadcasts its new location before it moves so that its neighbors can react promptly if a new hole is generated by its leaving.

2.3.6.2 Dealing with Position Clustering

In some cases, the initial deployment of sensors may form clusters, as shown in Figure 2.9, resulting in low initial coverage. In this case, sensors located inside the clusters can not move for several rounds, since their Voronoi polygons are well covered initially. This problem prolongs the deployment time, as is shown in Figure 2.9, in which some sensors are still clustered together after the sixth round. To reduce the deployment time in this situation, we propose an optimization which detects whether too many sensors are clustered in a small area. The algorithm "explodes" the cluster to scatter the sensors apart. Each sensor compares its current neighbor number to the neighbor number it will have if sensors are evenly distributed. If a sensor finds the ratio of these two numbers is larger than a threshold, it concludes that it is inside a cluster and chooses a random position within an area centered at itself which will contain the same number of sensors as its current neighbors in the even distribution. The explosion

algorithm only runs in the first round. It scatters the clustered sensors and changes the deployment to be close to random.

2.4 Deployment Protocols with Virtual Movement

The basic protocols require sensors to move iteratively, eventually reaching the final destination. Other approaches can be envisioned in which the sensors move only once to their destination to minimize the sensor movement. One such approach is to let sensors stay fixed and obtain their final destinations by simulated movement. With the same round-by-round procedure, sensors calculate their target locations, virtually move there, and exchange these new virtual locations with the sensors which would be their neighbors as if they had actually moved. The real movement only happens at the last round after final destinations are determined.

We did not deploy this alternative method for two reasons. First, this approach is susceptible to poor performance under network partitions which are likely to occur in a sensor deployment. If a network partition occurs, each partition will exercise the movement algorithms without knowledge of the others. Consequently, the obtained final destination is not accurate and the required coverage cannot be reached. Using real movement, the network partitions will be healed allowing all sensors to be eventually considered in the algorithm. A second reason is the high communication overhead. To guarantee logical neighbors are reached, a network-wide broadcast is needed when using simulated mobility. If this network-wide broadcast is implemented by gossiping, the message complexity is at minimum $2rn^2$ (here r is the number of rounds needed and n is

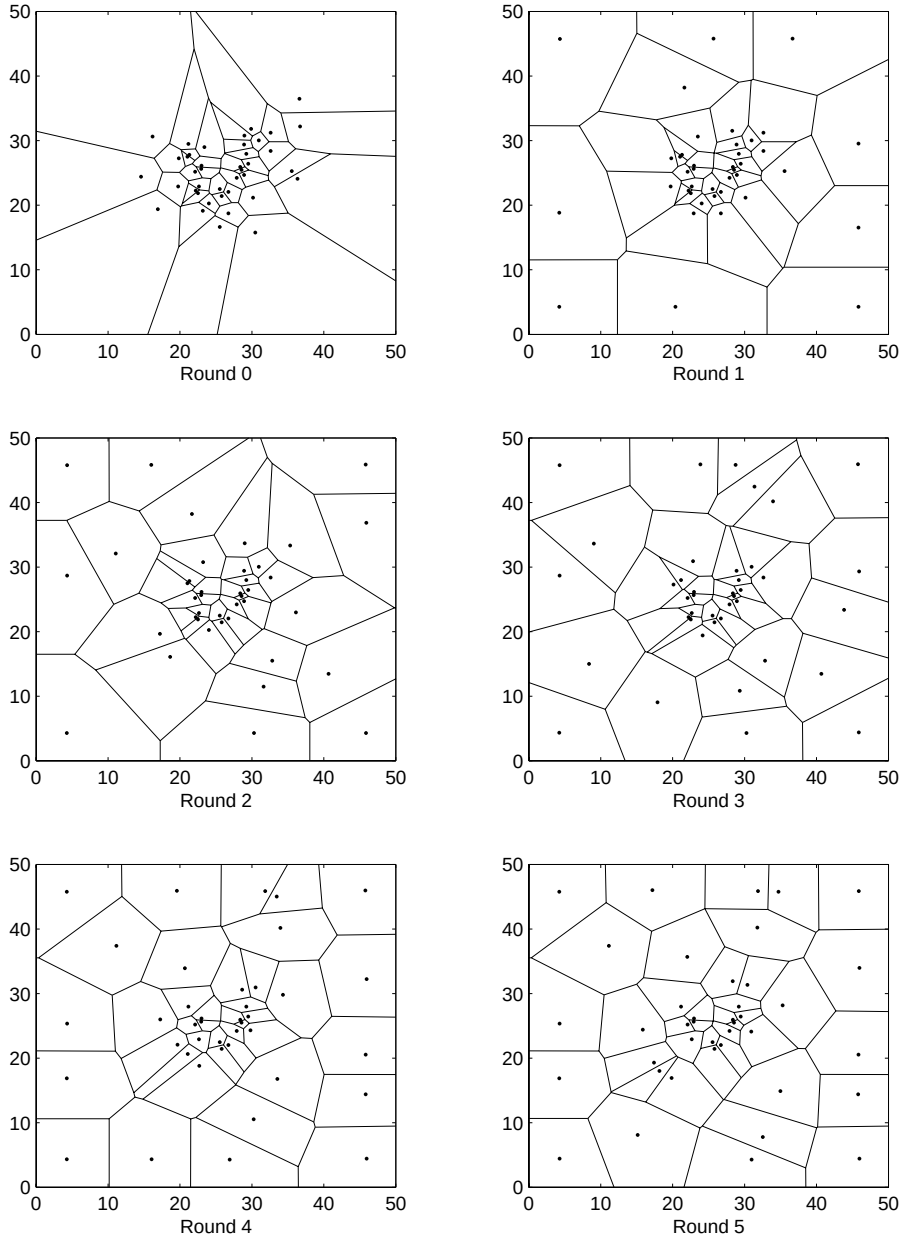


Fig. 2.9. Working procedure (VOR)

the number of sensors in the network). Using actual mobility as in the basic protocols, a much lower message complexity, $2rn$, is sufficient.

To get balance between movement and message complexity, we propose to let sensors do virtual movement when the communication cost to reach the logical Voronoi neighbors is reasonable, and do physical movement otherwise. The challenge is to determine if a sensor can reach its logical neighbors with reasonable communication cost. We propose the following heuristics.

First, if a sensor's distance to its farthest Voronoi vertex is shorter than half of the communication range, it must know all its Voronoi neighbors. In this case, one hop broadcast (same in the basic protocols) is enough to exchange the location information with its logical neighbors and physical movement is not necessary. Otherwise it is possible that some Voronoi neighbors are out of the communication range. To get the locations of these Voronoi neighbors, sensors request their neighbors within the communication range to broadcast their neighbor lists, thus obtaining the logical positions of sensors located within two broadcast hops. When the distances between the physical locations of sensors and their farthest Voronoi Vertices are larger than two times the maximum moving distance, sensors should move physically.

In realization, we divide the discovery phase into two sub-phases. In the first sub-phase, sensors broadcast hello messages; in the second sub-phase, sensors broadcast the locations of known neighbors. In one round, if a sensor's distance to its farthest Voronoi vertex is larger than half of the communication range, it will calculate the target location as in the basic schemes, and do logical movement. In the next round, it will set a flag in the hello messages, indicating that it wants its neighbors to broadcast their neighbor

Notations:
d_{max}, c_i: defined before
d_c: communication range
NL_i: the neighbor list of s_i
w_i: whether s_i wants the neighbor list of its neighbors
l_i: whether s_i needs to broadcast its neighbor list
loc_i, loc_i': the physical and logical position of s_i
p_i: loc_i' has not changed for p_i rounds.
P: s_i should move if $p_i \leq P$

(1) *Upon entering Discovery phase-I:*
  (1.1) set *timer* to be $discovery\_interval/2$
    enter *Discovery phase-II* upon timeout
  (1.2) broadcast $hello(w_i)$ after a random time slot
(2) *Upon entering Discovery phase-II:*
  (2.1) set *timer* to be $discovery\_interval/2$
    enter *Moving phase* upon timeout
  (2.2) **if** $l_i = true$ **then**
      broadcast NL_i after a random time slot
      $l_i = false$
(3) *Upon entering Moving phase:*
  (3.1) set *timer* to be $moving\_interval$,
    enter *Discovery phase-I* upon timeout
  (3.2) **if** $c_i = false$ **then**
      calculate loc_i' by VEC or VOR or Minimax
      do *oscillation control*
      do *movement adjustment*
      $p_i = 1$ if logically moves
      if $d(loc_i, V_{far}) \geq 2 * d_{max}$ **then**
        move to loc_i'; w_i = false
      else if $d(loc_i', V_{far}) \geq d_c/2$ **then**
        w_i = true;
  (3.3) **else if** $p_i \leq 1$ **then**
      $p_i = p_i + 1$
  (3.4) **else if** $p_i \leq P$
      move to loc_i'; w_i = false
(4) *Upon receiving a* $hello(w_j)$ *message from sensor* s_j*:*
  (4.2) **if** $w_j = true$ **then**
      $l_i = true$
  (4.1) Update $\mathcal{N}_i$ and G_i
  (4.2) **if** G_i is newly covered **then**
      set $c_i = true$
(5) *Upon receiving a* NL_j *message from sensor* s_j*:*
  (5.1) Update $\mathcal{N}_i$ and G_i

Fig. 2.10. Virtual movement protocols at s_i

list. Any sensor that receives a hello message with such a flag will broadcast its neighbor list in the second sub-phase of the discovery phase. In this way, the message complexity is at most two times the basic scheme in one round. The flag will not be reset until the sensor moves physically. Sensors move physically under two conditions: One is that its physical position is two times the maximum moving distance to its farthest Voronoi vertex, as discussed above. The second condition is that a sensor's logical position has not changed for several rounds. Then the sensor can determine that it has obtained its final location and it can move. The formal description of the protocol with the virtual movement is shown in Figure 2.10.

2.5 Performance Evaluations

2.5.1 Objectives, Metrics, and Methodology

We implement our deployment protocols in the ns-2 (version 2.1b9a), a standard network simulator. Our objectives in conducting this evaluation study are three-fold: first, testing the effectiveness of our protocols in providing high coverage; second, by comparing VEC, VOR and Minimax, and comparing the basic protocols and the virtual movement protocols, giving some insight on choosing protocols in different situations; finally, studying the effectiveness of controlling the tradeoff among various metrics by adjusting parameters.

We analyze the performance of our protocols from two aspects: *deployment quality* and *energy consumption*. *Deployment quality* is measured by the sensor coverage and the time (number of rounds) to reach this coverage. Deployment time is determined

by the number of rounds needed and the time of each round. The duration of each round is primarily determined by the moving speed of sensors, which is the mechanical attribute of sensors. Thus, we only use the number of rounds to measure the deployment time. Energy consumption includes two parts, mechanical movement and communication. Message complexity is used to measure the energy consumed in communication. As for movement, the energy consumed in moving a sensor n meters consists of two parts: starting/braking energy and moving energy. Therefore, we use moving distance and the number of movement as the metrics.

We run simulations under different sensor density, which determines the sensor coverage that can be reached and the difficulty to reach it. In a $100m * 100m$ target field, we distribute four different number of sensors, ranging from 120 to 180, in increments of 20 sensors. The initial deployment follows the random distribution. Most simulation results are under the termination condition that ϵ is equal to 1% divided by the number of sensors. To evaluate each metric under different parameter settings, we run 10 experiments based on different initial distribution and calculate the average results.

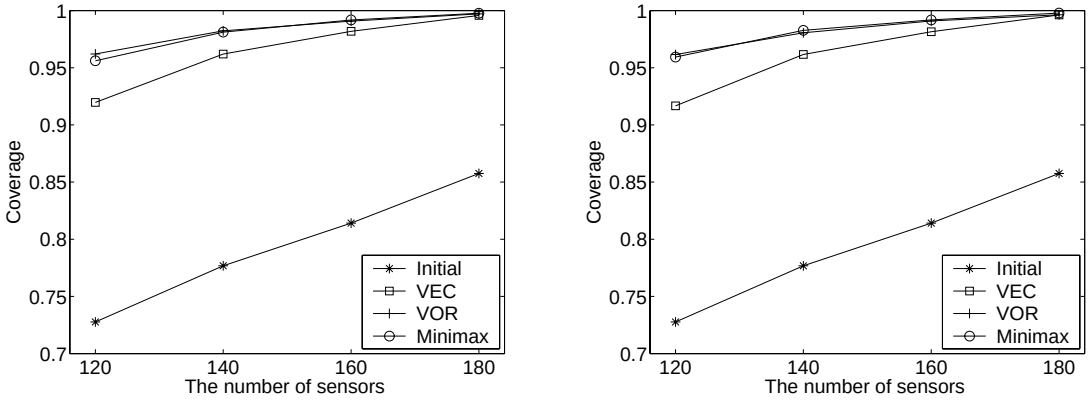
We choose 802.11 as the MAC layer protocol and DSDV as the routing protocol. The physical layer is modeled after the RF MOTE from Berkeley, with 916.5MHZ OOK 5kbps as the bandwidth and 20 meters as the transmission range. Based on the information from [2], we set the *sensing range* to be 6 meters. This is consistent with other current sensor prototypes, such as Smart Dust (U.C.Berkeley), CTOS dust, Wins (Rockwell)[3].

2.5.2 Simulation Results

2.5.2.1 Coverage

Figure 2.11 shows the coverage obtained when the coverage increase threshold ϵ is equal to 1% divided by the number of sensors. From the figure, we can see that the coverage is greatly increased by all three algorithms compared to the initial random distribution. For example, when 140 sensors are deployed, Minimax and VOR can increase the coverage to be more than 98% from 77.7%. In contrast, to obtain the same coverage under random deployment, on average, 340 sensors are required.

Also, by analyzing the trace of the basic protocols, we find the coverage increases very quickly during the first several rounds. For example, in most of the cases, the coverage can be increased to over 85% after the first round when 120 sensors are deployed and over 90% in higher sensor densities. In the virtual movement protocols, actual coverage increase happens after the real movement.



(a) Basic protocols (b) Virtual movement

Fig. 2.11. Coverage

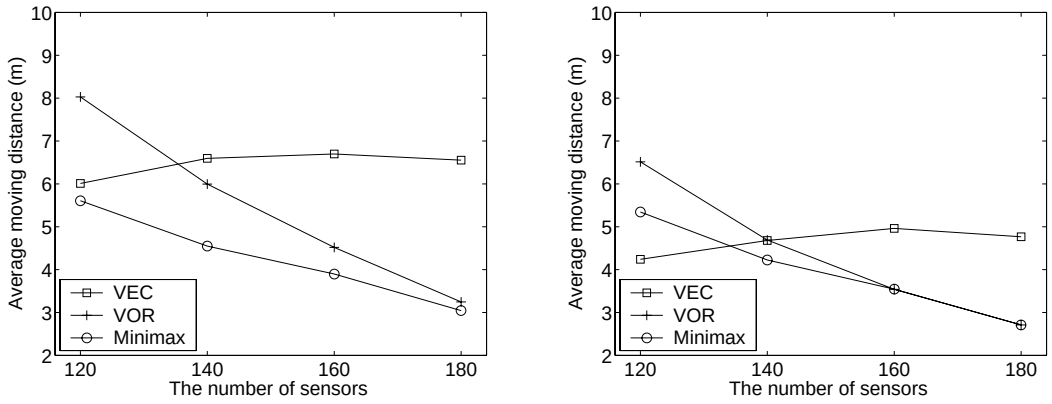
Among VEC, VOR, and Minimax, VEC performs the worst. The primary reason is that VEC is sensitive to the initial deployment. Consider an extreme situation in which sensors are located in the same line with equal spacing. In this case, no sensor will move, since the virtual forces offset each other, though there are large coverage holes. If the sensors are located in similar relative positions initially, VEC does not perform well. In addition, VEC neither considers coverage holes nor utilizes any geometric information from the Voronoi polygons when choosing the target locations. It tries to reach relatively balanced positions among the sensors, despite the difficulty of obtaining an exact global even distribution from only local information.

VOR and Minimax achieve quite similar coverage. They both move to heal the holes directly. VOR is more greedy and may move more than needed, thus generating new coverage holes. But finally, VOR will move sensors back to the correct positions if coverage can be increased by doing so.

Between virtual movement protocols and basic protocols, virtual movement protocols achieve almost the same coverage as the basic protocols as expected. We can conclude that using virtual movement will not affect the achieved coverage.

2.5.2.2 Energy Consumption

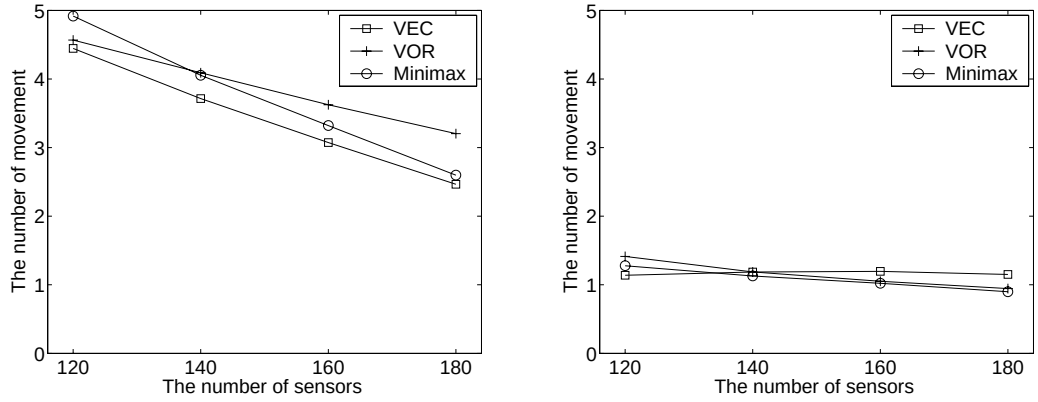
Figure 2.12 and Figure 2.13 show the moving distance and the number of movements respectively. Figure 2.14 shows the message complexity. Here message complexity is defined as the number of messages exchanged when the protocol terminates. To evaluate the energy consumption, we normalize the moving distance and the number of movements into message complexity. That is, with the same amount of energy consumed



(a) Basic protocols (b) Virtual movement

Fig. 2.12. Moving distance

in movement, how many messages can be transmitted. Calculated from Robomote [78], approximately, to move a sensor one meter consumes a similar amount of energy as transmitting 300 messages. The energy consumption in starting/braking is varied in different systems. Figure 2.15 shows the unified energy consumption when the starting/braking to one meter moving energy consumption ratio is one. (We also have plotted the case when the ratio is four. The results are similar to Figure 2.15, so we do not show it here.) From the figure, we can conclude that virtual movement protocols are much more energy-efficient than the basic protocols. The improvement is larger when the starting energy is high. Among the three algorithms to calculate the target locations, Minimax is always the most energy-efficient, except when the sensor density is quite low. At low sensor density, VEC consumes the least energy. VEC pushes sensors into relatively regular positions and does not perform the fine adjustment of their locations to reach high



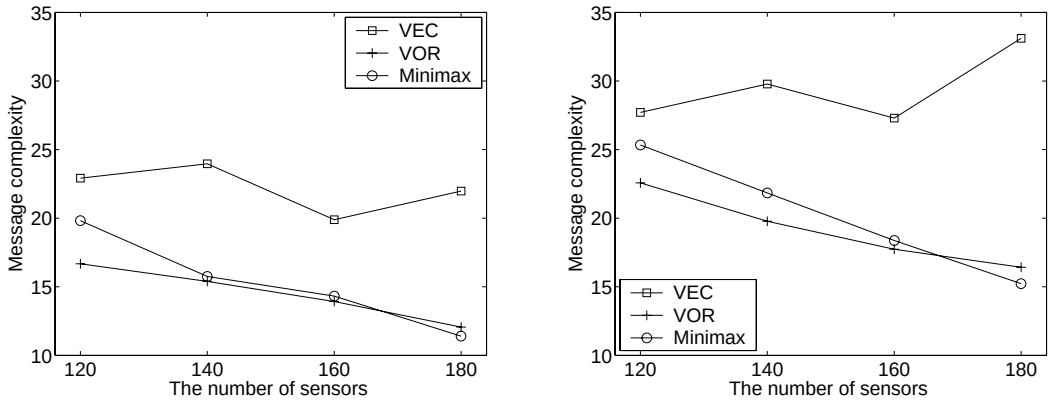
(a) Basic protocols (b) Virtual movement

Fig. 2.13. The number of movements

coverage. Therefore, VEC consumes the least energy, and reaches the lowest coverage among these three algorithms under low density.

Between VOR and Minimax, Minimax moves less in most cases. Minimax is proposed to address the aggressive feature of VOR and the simulation results verify its effectiveness. Because Minimax is sensitive to the construction of Voronoi polygon, it is sensitive to message loss. In our previous chapter [41], we showed that if message loss is not accounted for, Minimax has the largest moving distance. This is because when message loss occurs, the calculated Voronoi polygon is not correct.

From Figure 2.12 we can observe an interesting phenomena of VEC: the moving distance is similar under different sensor densities. This is because VEC fixes coverage holes by pushing sensors into a relatively even distribution. In VEC, sensors are pushed by the virtual forces, which are determined by the difference between the average distance of sensors when they are evenly distributed and the individual inter-distances. Both

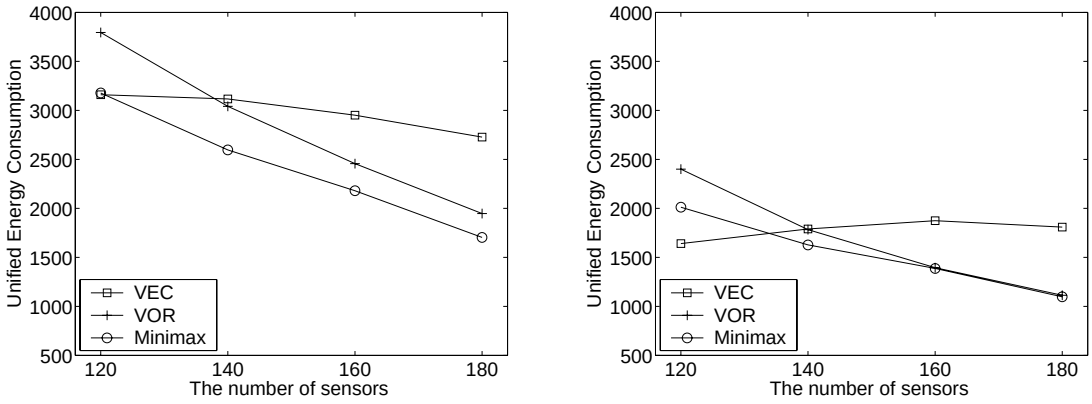


(a) Basic protocols (b) Virtual movement

Fig. 2.14. Message Complexity

values increase with a low density and decrease with a higher density. Thus, VEC is not sensitive to sensor density. In contrast, Minimax and VOR relocate sensors by measuring the coverage holes, which are larger under lower density and smaller under high density. Therefore, the moving distance in VOR and Minimax is decreased with a higher sensor density.

Figure 2.13 shows the number of movements using basic protocols and virtual movement protocols, respectively. We can see that the number of movements is greatly reduced by using virtual movement. In particular, sensors move about once when the number of nodes is more than 140, and less than 1.5 times when the number of nodes is 120. This shows the effectiveness of the heuristic to determine when to move physically. In the basic protocols, sensors move several times on average. They move less with a higher node density and fewer coverage holes.



(a) Basic protocols (b) Virtual movement

Fig. 2.15. Unified Energy Consumption (energy consumed in starting/braking is equal to moving one meter)

As described in Section 2.2.2, path planning is required to overcome obstacles. We expect a greater negative impact on moving distance when obstacles must be overcome on the protocols that require the larger number of movements.

Figure 2.14 shows the message complexity. Message complexity is primarily determined by the number of rounds to finish the deployment process and the number of messages in each round. Within one round, VEC transmits more messages than VOR and Minimax since sensors need to send one message to notify neighbors that their Voronoi polygons are well covered. In addition, VEC needs more rounds to terminate (explained in the next section). Therefore, VEC has the highest message complexity. Between Minimax and VOR, Minimax needs more rounds to terminate and has higher message complexity.

2.5.2.3 Convergence Time

In this section, we evaluate the convergence time of our protocols. We set ϵ to be 0. Figure 2.16 shows the coverage in each round when the number of sensors is 140. From the figure, we can see that the coverage increases very quickly during the first several rounds. Here for each algorithm, we run 50 experiments. After 10 rounds, the algorithms achieve at least 98% of the best coverage they can reach, in all these experiments and the algorithms achieve at least 99% of the best coverage, in about 99.33% of these experiments. From the experiments, we can conclude that our algorithm can quickly converge.

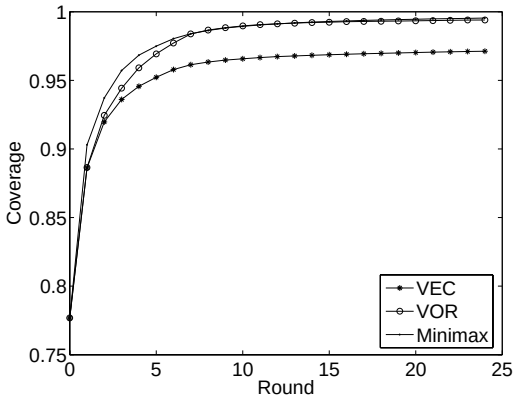


Fig. 2.16. Convergence time

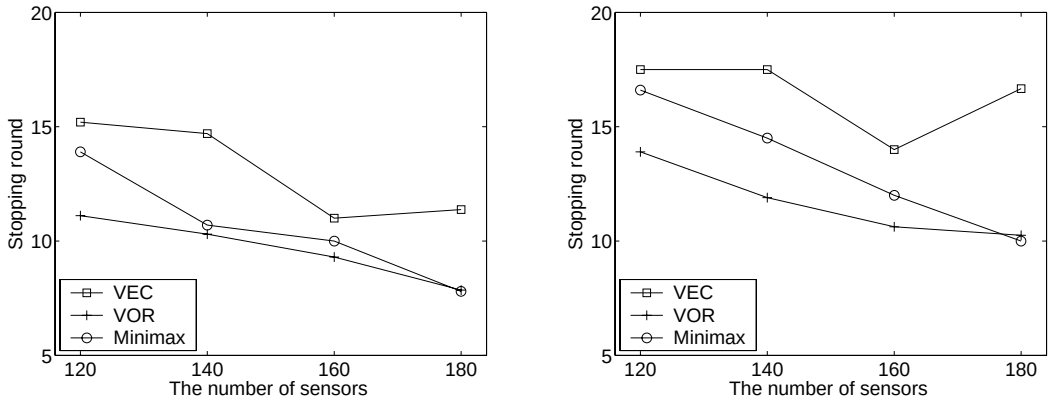
Our algorithm resembles the steepest descent algorithm [55], in that both try to move along the direction that is locally optimal. Unfortunately, the convergence theory of steepest descent is not satisfactory from a theoretical point of view. It is shown in

[55] that it converges if the objective function satisfies certain conditions and proper step lengths are taken. However, no result on the convergence rate of steepest descent for general objective functions can be found in the literature. For most applications, steepest descent is the best choice if no global information of the objective function is available.

2.5.2.4 Termination

Figure 2.17 shows deployment time under different sensor density. As expected, virtual movement protocols require more time to terminate. In virtual movement, each sensor waits several rounds before real movement.

In general, the deployment procedure can be roughly divided into two parts: One is overall re-distribution, which moves a group of sensors from a dense area to a sparse area to achieve a relatively even distribution of sensors. Another is the minor adjustment of positions in the local area to achieve better coverage. When the sensor density is not high, time consumed in the overall re-distribution is the major factor of deployment. When the sensor density is very high, no large-scale movement is needed and the position adjustment is the major factor. VOR is good at the overall re-distribution because of its aggressive feature. It terminates the quickest in low or medium sensor density. Minimax calculates the target locations to heal coverage holes most accurately, and it finishes the quickest in very high density. VEC pushes sensors away from dense area by virtual force. The sensors in a sparse area may not move for a long time since no sensor is present to push them. These sensors only move after the sensors from a dense area are propagated into their area. This propagation process may take a long time.



(a) Basic protocols (b) Virtual movement

Fig. 2.17. Termination

2.5.2.5 Impact of Coverage Increase Threshold ϵ

In this section, we study the effectiveness of controlling the tradeoff between coverage and deployment time by adjusting ϵ, the coverage increase threshold. TABLE 2.1 shows the termination round, and the value of various metrics when the protocol reaches the termination round. The metrics include coverage, moving distance, the number of movements, and moving efficiency, measured by the ratio of moving distance to the distance between the initial position and the final position. We can see that, with a smaller ϵ, a higher coverage can be reached, while the deployment cost and the deployment time is also increased. By properly setting this threshold, we can save time and energy by trading off a small amount of coverage. For example, in Minimax, when ϵ is increased from 0.5% to 1%, the deployment time can be shortened by 7 rounds (32%), while the ultimate coverage achieved is only reduced by 0.4%.

Coverage			
$\epsilon * n$	VEC	VOR	Minimax
0.5%	96.41	98.52	98.65
1%	96.16	98.06	98.27
2%	95.79	97.46	97.87
Stopping Round			
$\epsilon * n$	VEC	VOR	Minimax
0.5%	20.60	14.70	21.20
1%	17.50	11.90	14.50
2%	11.40	10.20	10.70
Moving Distance (m)			
$\epsilon * n$	VEC	VOR	Minimax
0.5%	4.72	4.78	4.30
1%	4.68	4.69	4.22
2%	4.63	4.61	4.15
Average Number of Movements			
$\epsilon * n$	VEC	VOR	Minimax
0.5%	1.21	1.24	1.17
1%	1.19	1.19	1.13
2%	1.17	1.16	1.10
Moving Efficiency			
$\epsilon * n$	VEC	VOR	Minimax
0.5%	1.10	1.06	1.05
1%	1.10	1.05	1.04
2%	1.09	1.04	1.03

Table 2.1. Impact of ϵ ($n = 140$)

Among VEC, VOR, and Minimax, the termination time is quite different when ϵ is small, and similar when ϵ is larger. For example, when ϵ is equal to 2%, the three algorithms terminate in a similar time. As for other metrics, Minimax performs the best. Therefore, Minimax is the best choice if ϵ can be set to be a relatively large number.

2.6 Conclusion and Discussions

This chapter addressed the problem of moving sensors in a target field to get high coverage. Based on Voronoi diagrams, we designed two sets of distributed protocols to iteratively move mobile sensors from densely deployed areas to sparsely deployed areas. Simulation results verified the effectiveness of our protocols and provided a baseline for performance under ideal conditions.

The virtual movement protocols can significantly reduce mechanical movement with a cost of less than two times an increase in the message complexity over the basic protocols. In each set of the protocols, three algorithms to calculate the target location were proposed: VEC, VOR and Minimax. Minimax is the best choice in most cases. VEC moves least at a low sensor density and can be deployed when the coverage requirement is not high. VOR terminates the earliest when the sensor density is not very high, and it can be deployed when both the deployment time requirement and coverage requirement are strict.

Below, we discuss some open issues.

2.6.1 Distributed Scheme versus Centralized Scheme

To address the problem of mobile sensor deployment, we propose that sensors calculate their target locations in a distributed fashion. We did not deploy a *centralized* approach for the following reasons. First, a central server architecture may not be feasible in some deployments. Further, the centralized approach suffers from the problem of a single point of failure.

Although a *centralized* approach is not feasible for many scenarios, it is interesting to study it and compare it with our distributed algorithms. We consider a centralized approach in an ideal case, in which the optimal positions to place sensors is decided a priori. For n sensors, there are n regular positions. It does not matter which sensor is placed in which position. There exists a central server, which can collect the locations of sensors and direct them to move, such that the average moving distance is minimized. Basically, this is a bi-party matching problem and the classic Hungarian method [68] can be used to calculate how to allocate sensors from their initial positions to the target locations such that the moving distance is minimized. We have compared the average moving distance of our distributed protocols with this ideal case. When the node density is lower than about 110 sensors per $100m * 100m$, the centralized scheme outperforms our scheme; otherwise, our distributed algorithms are better. (Here, we choose Minimax under virtual movement for comparison). This is because in our scheme sensors only move when there are coverage holes, while in the centralized scheme sensors always move to an optimal point even if it does not improve coverage. In our scheme, when the node density is high, sensors need only move a short distance to reach high coverage.

Under low density when most sensors are required to move, the centralized scheme results in a lower average moving distance. In terms of coverage, the centralized approach can always guarantee the optimal coverage since the optimal positions are decided a priori.

2.6.2 Sensing Area

In this chapter, the sensing area of each sensor is assumed to be a disk with radius $6m$. This is the ideal case which provides us with a baseline of the sensor placement problem. In future work, we will address varying sensing ranges. Here, we discuss these issues.

Our protocols can deal well with the case of a larger or smaller sensing radius if the sensing area is uniformly a disk. The performance of the protocols depends more on the ratio of communication range to sensing range than the absolute sensing range. As the sensing range decreases with regard to the communication range, our protocols will perform very well because they can accurately construct the Voronoi diagrams. As the sensing range increases, we need to enlarge the broadcast hops to better construct the Voronoi polygons.

If the sensing area is an irregular shape, instead of a disk, sensors can still check their Voronoi polygons to determine the coverage holes. In this case, we can decrease the sensing range used in our protocols to account for the reduced coverage. In future work, we will study our protocol's sensitivity to the sensing area.

2.6.3 Sensitivity to Communication Range

In our previous chapter [41], we evaluated the impact of communication range on the basic protocols and found that when communication is more than two times of the sensing range, the performance is similar to the results presented here (6m sensing range and 20m communication range). This requirement is reasonable and most hardware can satisfy it. In situations in which this requirement cannot be satisfied, we can increase the broadcast hop limit. In the virtual movement protocols, the broadcast hop can be increased accordingly if the communication range is short and the performance will not be affected.

Chapter 3

Bidding Protocols for Deploying Mobile Sensors

3.1 Introduction

When the coverage requirement is not very strict, or if sensors can be scattered in the target field relatively uniformly, it is unnecessary to equip every sensor with a motion base. We propose to deploy a mixture of mobile and static sensors to construct sensor networks, such that a balance between sensor cost and coverage can be achieved.

In this chapter, we design two distributed bidding protocols for the placement of mobile sensors in a sensor network composed of both mobile and static sensors: a basic bidding protocol and a proxy-based bidding protocol, which is an improvement on the basic bidding protocol. In the protocols, mobile sensors are treated as *servers* to heal *coverage holes*. Coverage holes are locations not covered by any sensor. Each mobile sensor has a *base price*, which is related to the size of any new hole generated by its movement. This represents the cost of its movement in terms of coverage. Static sensors detect coverage holes locally and estimate their sizes as *bids*. The static sensors *bid* the mobile sensors that have a base price lower than the hole to be covered. In the *basic bidding protocol*, mobile sensors choose the highest bids and thus move to heal the largest coverage holes. Using this process, sensors will only move to cover holes larger than those generated by their movements. After moving to the holes, mobile sensors raise their base prices to reflect the new coverage cost, and re-enter the bidding process.

This process iterates until no static sensor can give a bid higher than the base price of any mobile sensor.

To reduce the moving distances of mobile sensors, the *proxy-based* bidding protocol proposes that mobile sensors perform virtual movements from small holes to large holes and only perform physical movements after the final destinations are identified. Simulation results show our protocols can achieve high sensor coverage at low cost.

The rest of the chapter is organized as follows. Section 3.2 analyzes this sensor placement problem from theoretical perspective. We present the basic bidding protocol in section 3.3, and present the proxy-based bidding protocol in section 3.4. Section 3.5 evaluates the performance of the proposed protocols. We conclude the chapter in section 3.7.

3.2 Theoretical Analysis

3.2.1 Problem Statement

When a portion of deployed sensors are mobile, the deployment problem can be described as follows: given a target field covered by a number of circles (the sensing circles of the static sensors), but still having some uncovered areas, how to place a certain number of additional circles (the sensing circle of the mobile sensors) to maximize the overall coverage.

3.2.2 Reducing to the Vertex Covering Problem

This problem is a NP-hard problem, which can reduce to the vertex covering problem [68]. The detailed proof of NP-Completeness is shown as follows.

To prove that this problem in NP-hard, we will reduce the following question:

> Given a cubic planar graph G and integer k, does G have an independent set of size k?

to

> Given a square target field partially covered with a number of unit circles (*i.e.* of radius 1) and integer k, can we obtain a complete coverage of the target field by adding k unit circles?

The first question is NP-complete [36, 35].

The second question can be clearly reduced to our optimization problem.

The first stage in translating the problem is to draw the given graph G on an integer grid. We will request that each node has both coordinates divisible by, say, 10.

For each node of the original graph we have a point $(10i, 10j)$; near that point we center a unit circle and inside we create an uncovered area as shown in the figure below; note that this area has 3 special points, and that nearby are another 3 points, each in distance 1 from a corresponding special point; we call them *outer special points*. Outer special points will be located at lines in which at least one coordinate is divisible by 10.

Each edge of the original graph corresponds to a line in which points have at least one coordinate which is an integer divisible by 10. We cover this path with points that are in distance, say, between 1.5 and 1.75 from each other, so that such a "trail of points" starts at one of the special points of a node gadget and ends at an outer special nodes of another gadget. The trails of points of each edge must be disjoint, and each must have an odd number of points.

We create a little uncovered area around each points on our trails. We finish the construction by covering all areas that we explicitely did not wish to leave uncovered. It follows from the figure that it can be done.

Now, suppose that we had n nodes and m edges in the original graphs and the trails of points of the edges together contain $2K - m$ points. Then we ask if we can achieve the complete coverage by adding $K + n - k$ circles.

Suppose that the original question has answer "YES", *i.e.* there exists an independent set with k nodes, and thus a vertex cover with $n - k$ nodes. In gadgets corresponding to the vertices of the vertex cover we cover the central area (with the 3 inner special nodes) with a single circle. In gadgets corresponding the the vertices of the independent sets we use three circles to cover the central area and the areas of the outer special points. Now consider an edge; it corresponds to a trail of, say, $2h - 1$ points, two of them being outer special points. One of these points is covered by circle used because of the independent set, so we have $2h - 2$ points left, and the uncovered areas around these points are placed in such a way that we can cover pairs of them in one circle; thus we use $h - 1$ circles; to these circles we can add the circle placed by the independent set rule, so we attribute the use of h circles to this edge. If we add together all circles used that way we get K circles. Hence, we covered the entire area with $K + n - k$ circles and the new question has answer "YES".

Now, suppose that the new question has answer "YES", so that we obtain a complete coverage by adding $K + n - k$ circles. We will change the placement of the new circles without changing their number to assure some good properties of the placement.

Consider a vertex gadget and its three inner special points, shown in Figure 3.1.

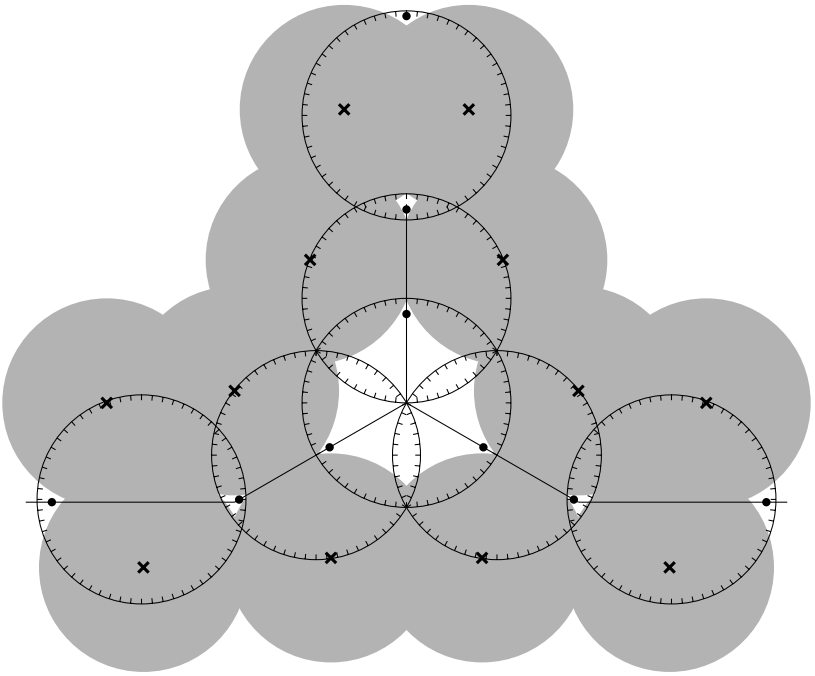


Fig. 3.1. Gadget for NP-completeness proof. Dots indicate the special points used in the proof; crosses are the locations of already placed sensors.

Suppose that some two of these points are covered with a single circle; then this circle cannot cover any of the outer special points. We move this circle so it covers the entire central area, in particular, all three inner points. We had to have three circles outer special points; we move them so they cover the area of these points as well as the areas of the adjacent points on their respective trails. We call such a vertex a *cover vertex*.

Suppose that the inner special points were covered with three different circles. We move these circles so they cover the central area and the areas of the outer special points. Consider a trail of points of an edge, say, with $2h-1$ points. Suppose that these points are covered with $h+1$ (or more) circles. Then we remove all the circles that cover the areas of these points, we cover them together with the entire inner area and the areas of the outer special points. We call such a vertex an *independent vertex*.

Now consider a trail of points of an edge $\{u, v\}$, say with $2h-1$ points. Suppose both u and v are independent; in this case both of the outer special points that are at the ends of this trail are covered together with their respective inner points, leaving $2h-3$ points to cover. We remove the circles that cover these points, as well as the outer special point of v that is on the trail; because the latter was covered together with its respective inner point, we are removing the cover of at least $2h-1$ points, and they are placed in such a way that we surely remove at least h circles. Now we cover $2h-2$ points on the trail with $h-1$ circles, and we use one more circle to cover three inner points of the gadget of v. As a result, we changed the classification of v to *cover vertex* without increasing the number of circles.

Now independent vertices form an independent set and cover vertices form a vertex cover. When we consider a trail of points of an edge with $2h-1$ points, we cover them with h circles, together with an inner special point of the incident independent point (if any). Thus we cover the trails of points of edges, together with the gadgets of the independent points, with K circles, and the remaining $n-k$ circles cover gadgets of the cover vertices, which in turn form a vertex cover. Hence the answer to the original question is "YES", we do have an independent set of k nodes.

3.2.3 Greedy Approximation

Although our problem is a fundamentally difficult problem, and there is no optimal solution, we can still find some practical solutions to approximate the optimal solution based on heuristics. Similar to the greedy algorithm, which is a commonly used heuristic for the vertex covering problem, we can place mobile sensors to the largest coverage holes.

3.3 Basic Bidding Protocol

In this section, we present the basic bidding protocol. We evaluate its performance in terms of coverage, energy consumption, and deployment time in Section 3.5. The description of this protocol provides a basic understanding and insight into our solution. We improve upon this protocol with optimizations described in Section 3.4 to reduce the required moving distance for each mobile sensor.

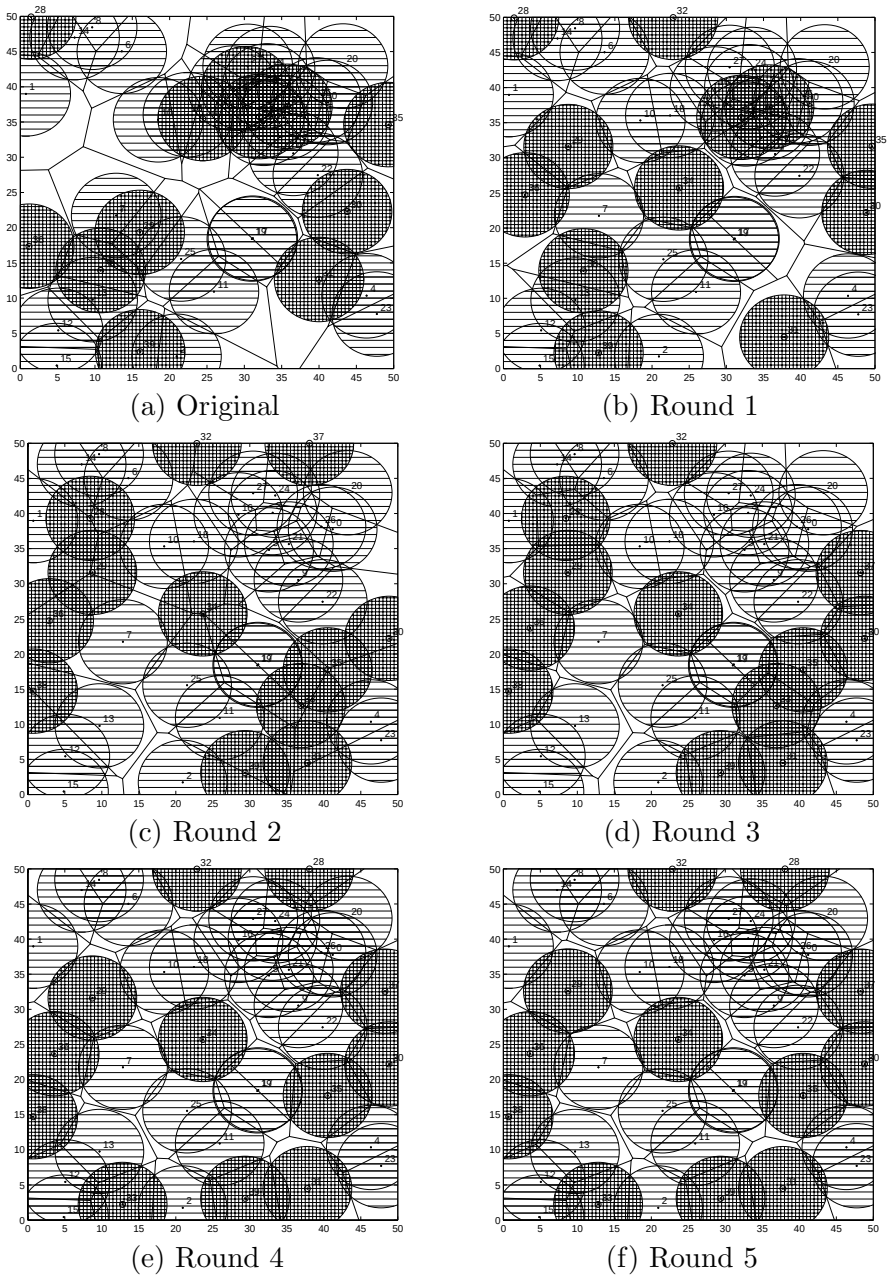
3.3.1 Bidding Protocol Overview

According to the greedy heuristic for this NP-hard problem, mobile sensors should move to the area where the most additional coverage can be obtained. After a mobile sensor leaves its original location to cover (heal) another coverage hole, it may generate a new hole in its original location. Thus, a mobile sensor only moves to heal another hole if its leaving will not generate a larger hole than that to be healed. However, due to lack of global information, mobile sensors may not know where a coverage hole exists. Even with the location of the coverage hole, it is still a big challenge to find the target position inside the coverage hole which can bring the most additional coverage when a mobile sensor is placed there compared to other positions. We propose to let the static sensors detect the coverage holes locally, estimate the size of these holes, and determine the target position inside the hole. Based on the properties of the Voronoi diagram, static sensors can find the coverage holes locally and provide a good way to estimate the target location of the mobile sensors.

The roles of mobile and static sensors motivate us to design a bidding protocol to assist the movement of the mobile sensors. We view a mobile sensor as a hole healing server. Its service has a certain base price, which is the estimate of any generated coverage hole after it leaves the current place. Static sensors are the bidders of the coverage hole healing services. Their bids are the estimated sizes of the holes they detect. Static sensors bid mobile sensors that have a base price lower than their bid. Mobile sensors choose the highest bid and move to the target locations provided by the static sensors.

The bidding protocol runs round by round after the initialization period. During the initialization period, all static sensors broadcast their locations and identities locally. We choose the broadcast radius to be two hops, with which sensors can construct the Voronoi diagram in most cases. After the initialization period, static sensors broadcast this information again only when new mobile sensors arrive and need this information to construct their own Voronoi cells.

Each round consists of three phases: *service advertisement*, *bidding*, and *serving*. In the advertisement phase, mobile sensors broadcast their base prices and locations in a local area. The base price is set to be zero initially. By the end of the service advertisement phase, each static sensor has a *service list*, which is a list of mobile sensor IDs along with their location and base price. In the bidding phase, static sensors detect coverage holes locally by examining their Voronoi cells. If such holes exist, they calculate the bids and the target locations for the mobile sensors. Examining the service list, the static sensor chooses a mobile sensor whose base price is lower than its bid, and sends a bidding message to this mobile sensor. We will present how to determine the mobile sensor to bid if there are multiple mobile sensors whose base price is lower than the bid of the static sensor. In the serving phase, the mobile sensor chooses the highest bid and moves to heal that coverage hole. The accepted bid will become the new base price of the mobile sensor. After the serving phase, the mobile sensors broadcast their new locations and new base prices and a new round begins. Because the base price increases monotonically, when no static sensors can give out a bid higher than the base price of the mobile sensors, the protocol terminates.



(a) Original (b) Round 1

(c) Round 2 (d) Round 3

(e) Round 4 (f) Round 5

Fig. 3.2. Snapshot of the execution of the bidding protocol

Before getting into the technical details of the bidding protocol, we first use an example to show how the protocol works. As shown in Figure 3.2, the circles with a striped shadow represent the sensing coverage of the static sensors, and the circles with grid shadow are that of the mobile sensors. Initially, 40 sensors are randomly placed in a $50m \times 50m$ flat field, among which 30% are mobile sensors. The initial coverage is 82%. The protocol terminates in the fifth round when the coverage reaches 93%. The sixth round has the same topology as the fifth round.

3.3.2 Bid Estimation

In the bidding message, static sensors provide the estimated coverage hole size as the bid and the target location to which the mobile sensor should move. This information is calculated based on their Voronoi cells. If there exists a coverage hole, the static sensor chooses the farthest Voronoi vertices as the target location of the coming mobile sensor. Inside one coverage hole, there are many positions that a mobile sensor can be located. If the mobile sensor is placed at the position farthest from any nearby sensors, the gained coverage is the highest since the overlap of the sensing circles between this new coming mobile sensor and existing sensors is the lowest. As shown in Figure 3.3, sensor s_a chooses its farthest Voronoi vertex O as the target location of the mobile sensor for which it bids.

From the global point of view, using the greedy heuristic to choose the largest coverage hole may not be optimal in some cases. As shown in Figure 3.4, A is the farthest Voronoi vertex of s_a. Although a high additional coverage can be obtained by placing a mobile sensor at A, it is not globally optimal since it leaves some scattered

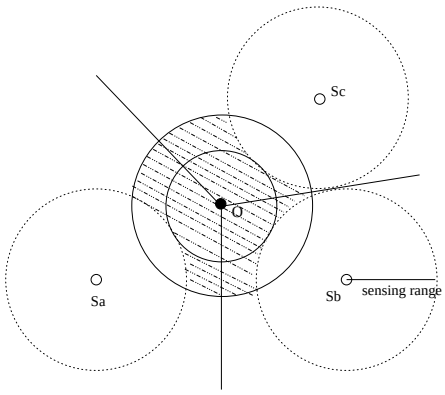


Fig. 3.3. Bid estimation

coverage holes which are hard to cover by placing additional mobile sensors. To deal with this problem, we propose an optimization which puts a limit on the maximum distance between the calculated target location and the bidder. As shown in Figure 3.4, by setting this maximum distance, a mobile sensor will be placed at B so that another mobile sensor can move to point C, to achieve better coverage. This maximum distance, denoted by d_{limit}, is a function of sensing range. We choose d_{limit} to be $\sqrt{3} * sensing\_range$, since to place sensing circles in a hexagonal relative position will minimize overlapping and maximize the coverage. Under these conditions, the distance between the centers of the sensing circles is $\sqrt{3} * sensing\_range$.

Having determined the target location of the mobile sensor it bids, static sensors calculate the bid as: $\pi * (d - sensing\_range)^2$, where d is the distance between the bidder and the target location. As shown in Figure 3.3, s_a's bid is the area of the inner circle centered at O, which is not the actual additional coverage to be obtained. The actual additional coverage is the shadow area, which is difficult to calculate since it involves

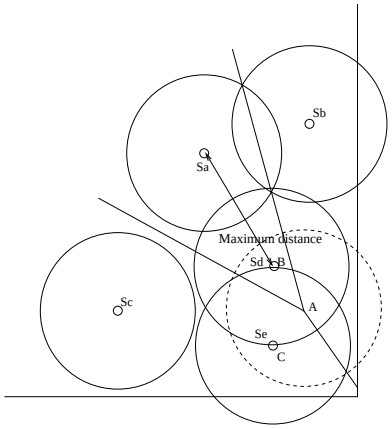


Fig. 3.4. Optimize the greedy heuristic

the union of circles. Using the inner circle as the bid simplifies the calculation, and can be used to approximate the actual additional coverage, which is the sensing circle minus the overlapping area of the sensing circles. The larger the overlapping area, the smaller the inner circle. Thus, the bid used can represent the relative size of the coverage holes.

Note that the maximum base price (or bid) is $\pi * (d_{limit} - sensing\_range)^2$, which is $\sqrt{3} * \pi * sensing\_range^2/2$.

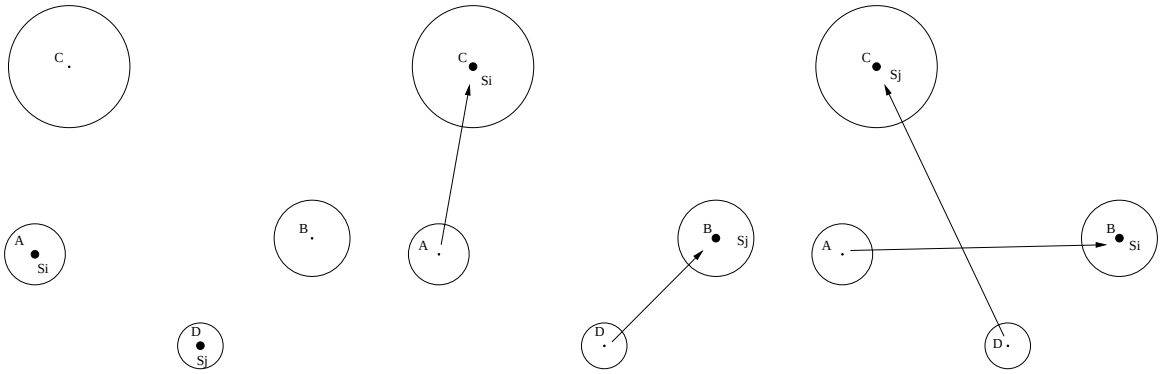
The property of the Voronoi diagram guarantees that the shadow area is always the additional coverage. This can be explained as follows. The points inside one Voronoi cell are closest to the sensor in this cell. The points in the Voronoi edge are closest to these two sensors besides this edge. The Voronoi vertex is the point closest to the sensors which contribute to the existence of this vertex. The sensing circle centered at the Voronoi vertex must only overlap with the sensing circles of the sensors which contribute to the construction of this Voronoi vertex. Thus we guarantee that the shadow area

shown in Figure 3.3 is always the additional coverage brought by placing a mobile sensor at O.

In addition to the static sensors, mobile sensors with a base price larger than zero also act as bidders. This is necessary because mobile sensors with a relatively larger price are essentially acting as static sensors. At this point, they can assist the movement of other mobile sensors.

3.3.3 Criteria of Choosing Mobile Sensors to Bid

After the service advertisement phase, each sensor has a list of mobile sensors, their locations and their base prices. A bidder needs to determine which mobile sensor to bid among those having a lower base price than its bid. We propose two criteria for choosing mobile sensors: *distance-based and price-based*. In the distance-based approach, a bidder chooses the closest mobile sensor to bid; in the price-based approach, a bidder chooses the cheapest mobile sensor to bid. The advantage of the distance-based approach is shown in Figure 3.5. We use a dashed circle to represent the coverage hole. The center of the circle is the target position of the mobile sensor to heal this hole. Initially, s_i is located in hole A and s_j is in hole D. The base price of s_i is higher than s_j, since the size of A is larger than D. In the distance-based approach, hole C will bid s_i and hole B will bid s_j. The movement of s_i and s_j is shown in Figure 3.5(b). In the price-based approach, both holes C and B will bid s_j and hole C wins. Then hole B bids s_j. Their movement is shown in Figure 3.5(c). As can be seen, the average moving distance of s_i and s_j is shorter in the distance-based approach, because the distance-based approach helps sensors move to their closest holes.



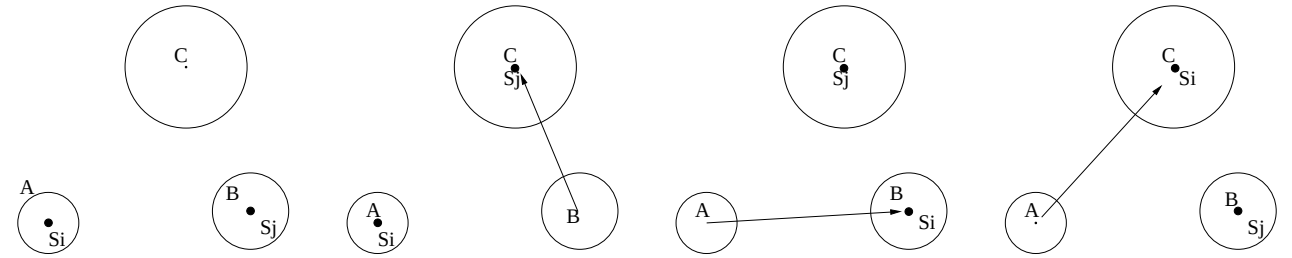
(a) Original situation (b) Distance-based (c) Price-based

Fig. 3.5. Distance-based vs. price-based

The advantage of the price-based approach is shown in Figure 3.6. With the price-based approach, hole C bids s_i since it has a lower base price. s_i moves once and no other sensor needs to move. But with the distance-based approach, hole C bids s_j since it is closer. After sensor s_j moves to hole C, hole B needs a sensor and it will bid s_i. In this way, both s_i and s_j have to move.

3.3.4 Multiple Healing Detection

Due to the limited service advertisement radius, static sensors may have different knowledge about the mobile sensors. Therefore, it is possible that several static sensors independently bid different mobile sensors for the same coverage hole since the cheapest mobile sensor or the closest mobile sensor in their views is different. If more than one succeeds in bidding, multiple mobile sensors will move to heal the same hole, which is not necessary. Figure 3.7(a) shows one example. A is the farthest Voronoi vertex of s_a



(a) Original situation (b.1) Distance-based (b.2) Distance-based (c) Price-based

Fig. 3.6. Distance-based vs. price-based

and B is the farthest Voronoi vertex of s_b. Both s_a and s_b bid mobile sensors to their farthest Voronoi vertices. When both biddings are accepted, a multiple healing occurs.

We propose a self-detection algorithm for mobile sensors to solve this problem. A mobile sensor has a knowledge of the locations and base prices of other mobile sensors in its neighborhood after the service advertisement phase. If it finds out that some other mobile sensors have a higher base price than its own, it will run the detection algorithm to check whether a multiple healing has occurred. If yes, the mobile sensor will lower its base price to zero and most likely some sensor will bid it to cover a different hole.

In the detection algorithm, the detecting mobile sensor calculates a **detecting threshold**, equal to $\pi * (d_{min} - sensing\_range)^2$, where d_{min} is the distance to its closest neighbor. If the detecting threshold is smaller than its new base price, or d_{min} is smaller than the sensing range, a multiple healing has occurred, since without multiple healing, the calculated value should be the same as its new base price. As shown in Figure 3.7(b), s_e and s_f, located in A and B respectively, are the mobile sensors bid by s_a and s_b. s_f's new base price, the bid put forward by s_b, is calculated without considering s_e, which is $\pi * (d_{b,f} - sensing\_range)^2$, where $d_{b,f}$ is the distance between

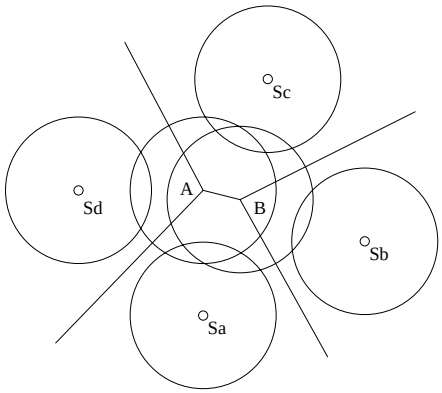
s_b and s_f. Without a multiple healing, $d_{b,f}$ is just d_{min}, and the calculated detecting threshold should be the same as the new base price. If multiple healing has occurred, $d_{e,f}$ is d_{min}, which is smaller than $d_{b,f}$, and the detecting threshold is smaller than the new base price.

3.4 Proxy-based Bidding Protocol

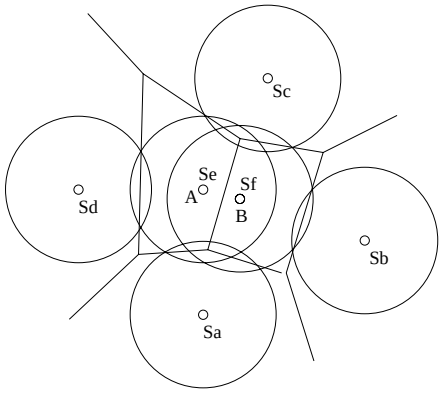
In this section, we present the proxy-based bidding protocol. This protocol improves the performance of the basic bidding protocol in terms of energy efficiency and load balance. In this protocol, sensors only move after their final location is determined; all calculations with respect to multiple healing detection and optimization are carried out through the exchange of messages before movement. The key tradeoff is the increased number of messages *vs.* the decreases in required movement. Because movement is typically much more expensive than exchanging messages, this protocol provides a more efficient solution than the basic bidding protocol.

3.4.1 General Idea: Logical Movement

Although the basic bidding protocol can achieve a high coverage, there is still room for improvement in terms of energy efficiency and load balance. In the basic bidding protocol, mobile sensors move iteratively to heal large and larger holes. Most likely, mobile sensors will move in an irregular pattern, which consumes more energy than moving directly from their initial location to the final destination. Also, in the basic bidding protocol, some sensors are penalized by being required to move a long distance. These phenomena are illustrated by the following example, shown in Figure



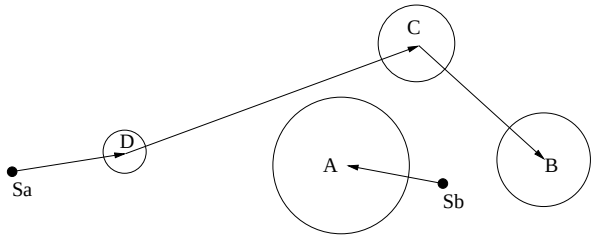
(a) The duplicate healing problem



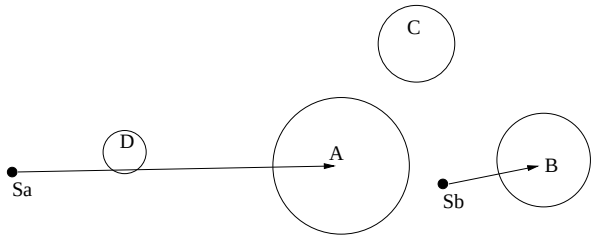
(b) Fix the duplicate healing problem

Fig. 3.7. Duplicate healing

3.8(a). In the first round, holes A, B, and C bid mobile sensor s_b, and hole D bids s_a. Hole A and hole D win due to their large size, and these two sensors move. In the second round, hole C bids for sensor s_a; hole B does not bid in this round since it does not know the existence of s_a due to the limited advertisement radius. In round three, hole B knows of s_a and bids it. s_a moves the third time to reach its final location, resulting in a much longer moving distance than s_b. Ideally, s_a shall move to heal hole A and s_b move to heal hole B, as shown in Figure 3.8(b). The comparison between the basic bidding protocol and the ideal solution motivates us to propose the proxy-based bidding protocol to better allocate mobile sensors to coverage holes such that the overall moving distance is shortened and no sensor is penalized.



(a) The characteristic of the basic bidding protocol



(b) Ideal solution

Fig. 3.8. Motivation of proxy-based bidding protocol

Following the same bidding framework, the proxy-based bidding protocol deploys the idea of *virtual movement*. Instead of moving physically in each round, mobile sensors perform virtual movements once they accept a bid. They only perform physical movements after they determine their final destinations. In this way, mobile sensors will not move in an irregular pattern. Also, virtual movement enables the possibility for mobile sensors to exchange their coverage holes to further shorten the moving distance since it does not matter which sensor heals which hole when all the largest holes are to be healed. For example, as shown in Figure 3.8, through logical movement, s_a identifies that hole B as its final destination and s_b identifies hole A. Before they perform the physical movements, they can exchange their destinations, i.e., s_a moves to hole B and s_b moves to hole A, such that an ideal allocation of mobile sensors to holes is obtained. In addition, with virtual movement, we can do multiple-healing detection before sensors physically move, and the vain movements of the sensors involved can be saved. In the following sections, we present the details of this protocol.

3.4.2 Proxy Sensor

To implement virtual movement, the first problem to be addressed is how to advertise services to the neighborhood of those virtual positions when mobile sensors do not move. One intuitive solution is to perform a network-wide broadcast. However, this may significantly increase the communication overhead. To keep the same communication overhead and let the sensors in the neighborhood of the virtual position of the mobile sensor receive the advertisement messages, we propose to use *proxy sensors*,

which are static sensors located closest to the virtual positions of the mobile sensors, to advertise the services and process the bidding messages for those mobile sensors.

The sensor closest to a mobile sensor's virtual position should be the bidder who detects the coverage hole and bids this mobile sensor since sensors detect coverage holes locally by checking their Voronoi polygons. Therefore, we choose the winning bidder as the proxy of the mobile sensor who accepts its bid.

The proxy of a mobile sensor is not fixed during the lifetime of the mobile sensor. When a mobile sensor accepts a bid in the first round, it sends a *delegate* message to the bidder and the bidder becomes the first proxy of this mobile sensor. In the next round, the bidder (proxy) advertises the virtual position and the new base price of the mobile sensor. In the view of other sensors, the mobile sensor has moved to its virtual position and the Voronoi diagram is computed based on the new virtual position of the mobile sensor. Based on the new base price of the mobile sensor, static sensors can still bid for the mobile sensor. Their bidding messages, if any, will be sent to the proxy instead of the real mobile sensor. Based on the received bidding messages, the proxy determines which new hole should be healed. If a new bid is accepted, the proxy delegates the proxy role to that bidder who will become the new proxy of the mobile sensor. In this way, the physical movement of the mobile sensor is replaced by delegating the role of proxies between static sensors, thus realizing virtual movement. When a proxy sensor does not receive any bidding messages for a *waiting threshold* of n rounds, it will notify the mobile sensor to perform physical movement. Experimentally we determined that $n = 2$ provides good results.

In addition to virtual movement, by using a proxy sensor, *multiple healing* can be detected before it happens in many situations. After the service advertisement, proxy sensors have a service list which contains the information of the virtual positions of mobile sensors. A proxy sensor can act as the mobile sensor it represents and detect whether a *multiple healing* would happen by examining its service list, with the same method of *multiple healing* detection presented in Section 3.3.4. A proxy sensor calculates the Voronoi cell without considering its mobile sensor, as if its bid in the previous round had failed. Then it checks whether the original coverage hole remains; if the same hole exists, no multiple healing has occurred since its mobile sensor is required to heal the hole; otherwise, some neighbor has bid a mobile sensor to heal the same hole and a multiple healing has occurred. If the proxy discovers that a multiple healing has occurred, it reduces the base price of its delegated mobile sensor to zero, and re-advertises the new service in the subsequent rounds.

To avoid all proxies from detecting the same multiple-healing and reducing the base prices of their delegated mobile sensors to zero, the proxies check whether the moving distance of its delegate from its current position to this hole is the shortest among those mobile sensors that heal the same hole. If not, it reduces the base price of its delegate to zero; otherwise, it waits for other mobile sensors to leave.

3.4.3 Coverage Hole Exchange

Coverage hole exchange is proposed to reduce the overall moving distance and to reduce the chance that an individual sensor is penalized by moving a long distance. It is performed by proxy sensors. A proxy sensor checks the service list obtained after the

service advertisement phase and determines with which mobile sensor to exchange the virtual position of its mobile sensor. The exchange criteria will be described in depth in the next paragraph. If an exchange is necessary, the proxy sensor sends a request to the proxy of the mobile sensor with which it wants to exchange position. A proxy sensor which receives multiple exchange requests chooses one by the same criteria and sends back a confirm message. Then these two proxy sensors exchange delegation of their mobile sensors, and the two mobile sensors exchange the proxies and their associated coverage holes.

Before presenting the exchange criteria, we introduce the following notations. We use d_i to represent the moving distance of s_i before exchange, and $\hat{d}_i$ the moving distance after exchange. d_{max} is a maximum moving distance threshold. All exchanges between s_i and s_j must satisfy the following prerequisites; otherwise, the exchange will not be performed.

$$\begin{cases} \hat{d}_i + \hat{d}_j \leq d_i + d_j \\ \hat{d}_i \leq max[d_i, d_{max}] \\ \hat{d}_j \leq max[d_j, d_{max}] \end{cases} \tag{3.1}$$

Shown in (3.1), all exchanges must reduce the overall distances. Also, the exchanges must not increase the moving distance of a single sensor to be longer than d_{max} if it is not so before the exchange, and must not further increase the moving distance of a single sensor if its moving distance is already longer than d_{max}.

Among the exchanges which satisfy the prerequisites shown in (3.1), we give higher priority to those which can release or mitigate node penalization. We first check

exchanges, in which one or both involved sensors have to move longer than d_{max} before the exchange, and choose the one which can reduce the overall moving distance the most.

Formally, the exchange is chosen as follows:

$$[s_i, s_j] = argmin_{\hat{d}_k + \hat{d}_l - d_k - d_l} \{[s_k, s_l] : d_k \geq d_{max} \vee d_l \geq d_{max}\} \tag{3.2}$$

Here, $[s_i, s_j]$ indicates mobile sensor s_i exchanges its virtual position with s_j. If there is no such exchange, we choose the exchange which can reduce the overall moving distance the most. That is:

$$[s_i, s_j] = argmin_{\hat{d}_k + \hat{d}_l - d_k - d_l} \{[s_k, s_l] : d_k \leq d_{max} \wedge d_l \leq d_{max}\} \tag{3.3}$$

Without hole exchange, proxy sensors can notify mobile sensors to move if they do not receive bidding messages for the *waiting threshold* of n rounds. Hole exchange complicates the decision of when to tell a sensor to move. As shown in Figure 3.8, if s_b moves physically in the third round, s_a has no sensor with which to exchange its virtual position after it virtually moves to hole B. To solve this problem, s_b should wait for more rounds before movement. In general, a mobile sensor that gets a high base price in the first two rounds should wait for additional rounds before physically moving so that other sensors have an opportunity to perform hole exchange. Through extensive experiments,

we determined that $n = 5$ for sensors that receive high bid prices in the first two rounds, and $n = 2$ for other sensors, yields good results.

There is an exception to this general principle. For very large holes, i.e., holes bigger than the sensing range of a single sensor, as shown in Figure 3.4, two mobile sensors (or more) are needed for healing. In Figure 3.4, static sensor s_a bids mobile sensor s_d to move, and it is s_b which bids another mobile sensor s_e to heal the same hole. Normally, sensors that move first to heal the hole act as bidders in the next rounds to bid more sensors to heal the same hole. These sensors, like s_d, which move first and have the maximum base price $\sqrt{(3)} * pi * sensing\_range^2/2$ (described in section 3.3.2), should move immediately because they will act as bidders.

3.4.4 Protocol Specification

As with the basic bidding protocol, the proxy-based bidding protocol runs round by round until mobile sensors obtain their final locations and move there directly. Each round consists of four phases: service advertisement, bidding, virtual movement, and hole-exchange. (1) In the service advertisement phase, proxy sensors advertise the virtual locations, physical locations and base prices for their delegated mobile sensors. In the first round, a mobile sensor does not have a proxy and advertises its physical location and base price by itself. (2) In the bidding phase, static sensors calculate their Voronoi polygons based on the virtual positions of mobile sensors. They detect coverage holes by examining the Voronoi polygons, estimate the hole size, choose the closest or cheapest mobile sensor, and send bidding messages to its proxy or the mobile sensor itself if the mobile sensor has no proxy. (3) In the virtual movement phase, proxy sensors (or mobile

sensors without a proxy) choose the highest bid and send a delegate message to the bidder. The bidder becomes the new proxy. The base price of mobile sensors is updated by their new proxies. Also, proxy sensors need to check whether hole-exchange is needed. If yes, they choose the mobile sensor suitable for exchange and send out an exchange request to the proxy of that mobile sensor. (4) In the hole-exchange phase, proxy sensors check the received requests, choose one with the highest priority and return the confirm message to the requester. Then the mobile sensors delegated by these two proxy sensors exchange the hole to heal.

The protocol terminates naturally when all the largest holes are healed and no more hole exchanges are necessary. Through the bidding process, when no sensors can raise a bid higher than the lowest base price of mobile sensors, all the largest holes are healed. This process terminates naturally as presented in Section 3.3. For hole exchange, we require that all the exchanges must reduce the overall moving distance. There is a lower bound of the overall moving distance, and hole exchange will finish naturally. Through this iterative 4-phase process, proxy sensors notify mobile sensors to move and the deployment process terminates. We show the formal algorithm in the following.

We show an operational example to illustrate the advantage of the proxy-based protocol over the basic bidding protocol. 40 sensors, of which 30% are mobile, are randomly distributed in a 50m*50m field. The initial distribution is shown in Figure 3.10(a); the distribution after deployment is shown in Figure 3.10(b). In this example (and most others), the proxy-based bidding protocol and the basic bidding protocol get the same distribution of sensors after deployment. Figure 3.10(c) shows the moving trace of mobile sensors in the proxy-based bidding protocol. The mobile sensors move

Notations:

S_i: the service list received by sensor s_i.

loc_i, loc'_i: physical position and logical position of s_i

$\mathcal{P}_i$: the proxy of s_i

(0) *Upon entering Advertising phase:*

set *timer* to be *advertise_interval* and enter *Bidding phase* upon timeout

if s_i is a mobile sensor without proxy **then**

broadcast $service\langle s_i, loc_i, loc_i, base\_price_i, s_i\rangle$

if $s_i = \mathcal{P}_j$ **then**

broadcast $service\langle s_j, loc_j, loc'_j, base\_price_j, s_i\rangle$

(1) *Upon receiving* $service\langle s_j, loc_j, loc'_j, base\_price_j, \mathcal{P}_j\rangle$*:*

add $\langle s_j, loc_j, loc'_j, base\_price_j, \mathcal{P}_j\rangle$ to S_i

(2) *Upon entering Bidding phase:*

set *timer* to be *bidding_interval* and enter *Logical movement phase* upon timeout

calculate bid_i if hole exists

choose the closest/cheapest sensor s_j from S_i where $base\_price_j < bid_i$

send $bidding\langle s_i, loc'_j, bid_i\rangle$ to $\mathcal{P}_j$

(3) *Upon receiving* $bidding\langle s_j, loc'_k, bid_j\rangle$*:*

record it if it has the highest bid

(4) *Upon entering Logical movement phase:*

set *timer* to be *logical_interval*, and enter *Hole-exchange phase* upon timeout

if has recorded $bidding\langle s_k, loc'_i, bid_k\rangle$ **then**

send $delegate\langle s_i, loc_i, loc'_i, bid_k\rangle$ to s_k

else if $s_i = \mathcal{P}_j$ and has recorded $bidding\langle s_k, loc'_j, bid_k\rangle$

send $delegate\langle s_j, loc_j, loc'_j, bid_k\rangle$ to s_k

else if $s_i = \mathcal{P}_j$

if hole-exchange with s_m is needed

send $request\langle s_i, s_j, loc_j, priority\rangle$ to $\mathcal{P}_m$

else if it is time for s_j to move **then**

send $notice\langle loc'_j, base\_price_j\rangle$ to s_j

(5) *Upon receiving* $delegate\langle s_j, loc_j, loc'_j, base\_price_j\rangle$*:*

$\mathcal{P}_j = s_i$; record loc_j, loc'_j, $base\_price_j$.

(6) *Upon receiving* $notice\langle loc'_i, base\_price_i\rangle$*:*

move to loc'_i and record $base\_price_i$

(7) *Upon receiving* $request\langle s_j, s_k, loc_k, priority\rangle$*:*

record it if it has the highest priority

(8) *Upon entering Hole-exchange phase:*

if $s_i = \mathcal{P}_m$ and has recorded $request\langle s_j, s_k, loc_k, priority\rangle$

send $confirm\langle s_m, s_k, loc_m\rangle$ to s_j

$\mathcal{P}_k = s_i$; $loc'_k = loc'_m$; $base\_price_k = base\_price_m$; record loc_k

(9) *Upon receiving* $confirm\langle s_m, s_k, loc_m\rangle$*:*

$\mathcal{P}_m = s_i$; $loc'_m = loc'_k$; $base\_price_m = base\_price_k$; record loc_m

Fig. 3.9. The proxy-based protocol at sensor s_i

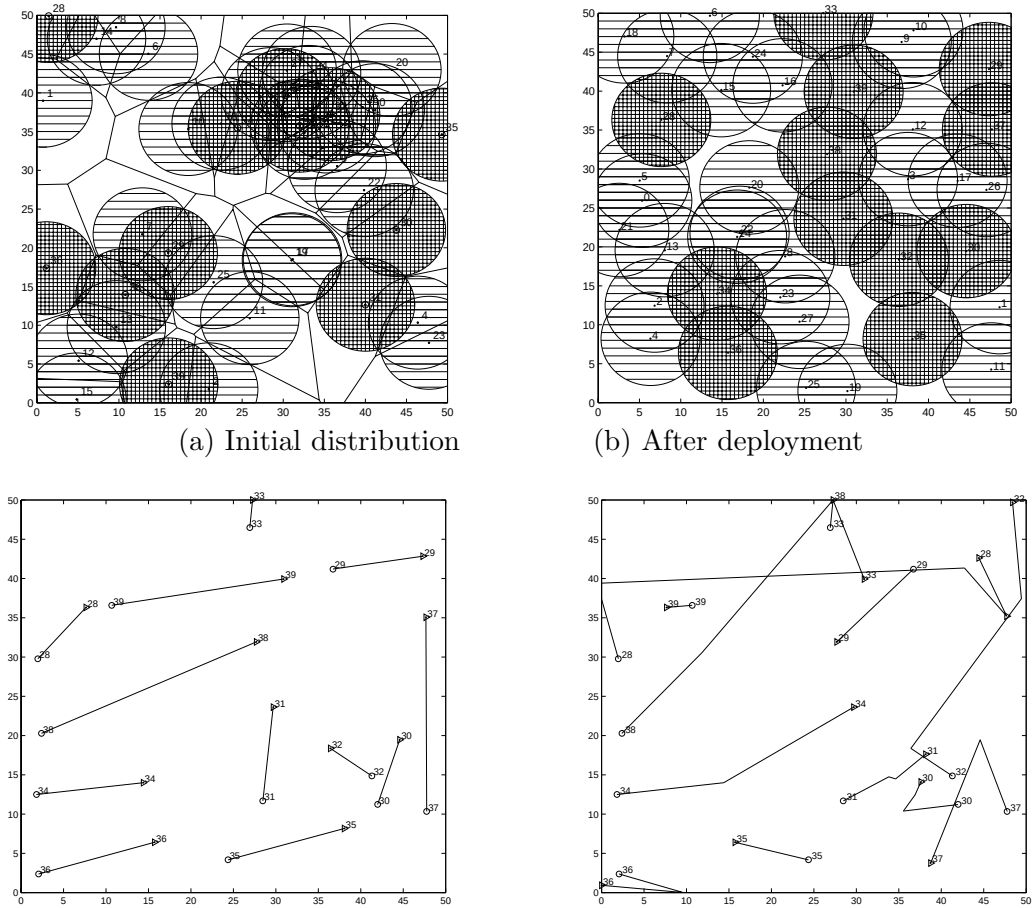
13.65m on average. Sensor 38 moves the longest distance 27.85m. Figure 3.10(d) shows the moving trace of mobile sensors in the basic bidding protocol. The average moving distance is 23.77m. Sensor 28 has the longest moving distance. It moves 5 times for a total distance of 68.68m. From this example, we can see that the proxy-based protocol is more energy-efficient and load-balanced.

3.5 Performance Evaluations

3.5.1 Objectives, Metrics, and Methodology

We implement our deployment protocols in the ns-2 (version 2.1b9a). Our objectives in conducting this evaluation study are three-fold: first, justifying our proposal of constructing sensor networks with both mobile and static sensors to balance cost and sensing coverage; second, testing the effectiveness of our bidding protocols in providing high coverage; finally, comparing the basic bidding protocol and the proxy-based bidding protocol, and giving some insight on choosing deployment protocols.

We analyze the performance of our schemes from three aspects: *sensor cost*, *deployment quality*, and *energy consumption*. Sensor cost is measured by the money used to construct the network. Deployment quality is measured by the sensor coverage and the time (number of rounds) to reach this coverage. Deployment time is determined by the number of rounds needed and the duration of each round. The duration of each round is primarily determined by the moving speed of sensors, which is the mechanical attribute of sensors. Thus, we only use the number of rounds to measure the deployment



(a) Initial distribution (b) After deployment

(c) Trace of the proxy-based protocol (d) Trace of the basic bidding protocol

Fig. 3.10. An operational example

time. Energy consumption includes two parts, mechanical movement and communication. Message complexity is used to measure the energy consumed in communication. As for movement, the energy consumed in moving a sensor n meters consists of two parts: starting/braking energy and moving energy. Therefore, we use moving distance and the number of movements as the metrics.

We run simulations for different compositions of sensor networks, and determine the coverage that can be reached. In a $60m * 60m$ flat field, we randomly distribute 60 sensors. Among these sensors, we assign a percentage of sensors to be mobile. This percentage varies from 10% to 50%, with an increment of 10%. The mobile sensors are chosen randomly. To evaluate each metric under different parameter settings, we run 50 experiments based on different initial distributions and calculate the average results.

We choose 802.11 as the MAC layer protocol and DSDV as the routing protocol. The physical layer is modeled after the RF MOTE from Berkeley, with 916.5MHZ OOK 5kbps as the bandwidth and 20 meters as the transmission range. Based on the information from [2], we set the *sensing range* to be 6 meters. This is consistent with other current sensor prototypes, such as Smart Dust (U.C.Berkeley), CTOS dust, Wins (Rockwell)[3].

In the following sections, we show the simulation results.

3.5.2 Tradeoff between cost and coverage

In order to evaluate the tradeoff between sensor cost and coverage, we consider three cases of network composition: all the sensors are mobile; all the sensors are static; a percentage of sensors are mobile. When all the sensors are static, random deployment

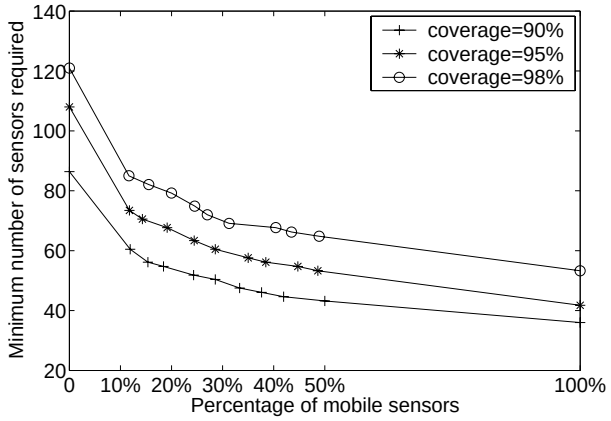
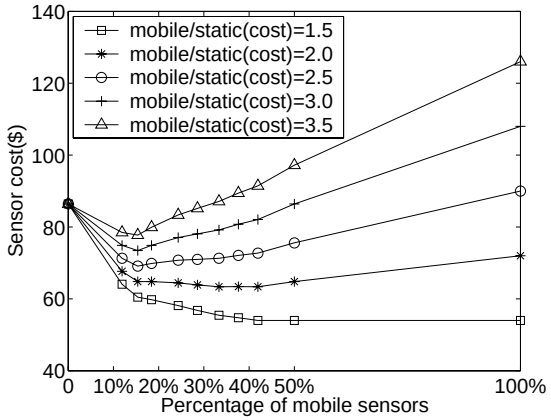


Fig. 3.11. The number of sensors needed to reach certain coverage under different mobile percentage

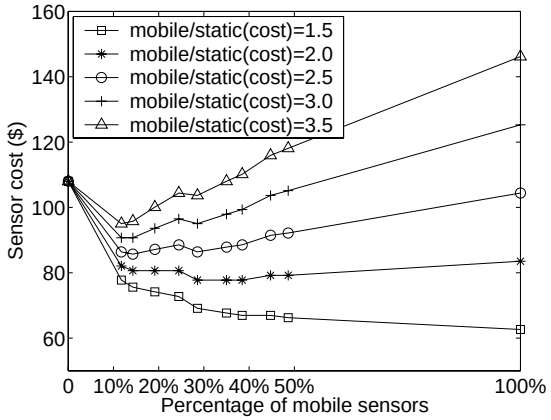
is used. When all the sensors are mobile, the VOR protocol, presented in Chapter 2, is used for sensor deployment. When a percentage of sensors are mobile, our basic bidding protocol (using the distance-based criteria) is used. Figure 3.11 shows the total number of sensors needed to reach certain coverage with under different network compositions. 0% of mobile sensors means that all sensors are static.

As shown in Figure 3.11, to reach a certain coverage, random deployment of static sensors uses the most number of sensors; as the percentage of mobile sensors increases, the required number of sensors to reach a certain coverage decreases; a deployment of 100% mobile sensors requires the fewest sensors. However, the cost of mobile sensors may be high.

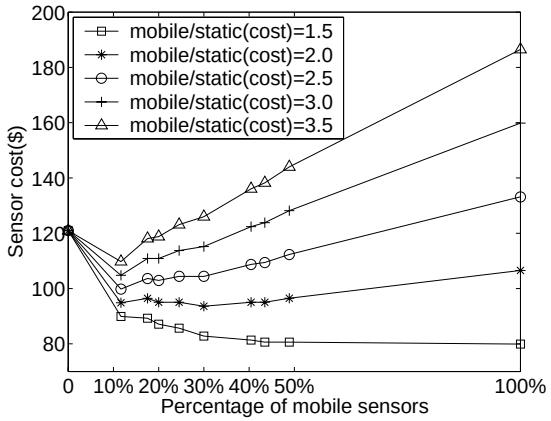
Compared to random deployment, the basic bidding protocol can significantly reduce the number of sensors required to reach a certain coverage. For example, to reach a 90% coverage, with only 10% of mobile sensors, the basic protocol needs 30% fewer



(a) To reach 90% coverage



(b) To reach 95% coverage



(c) To reach 98% coverage

Fig. 3.12. The cost of sensors to reach certain coverage

sensors; when 50% of the sensors are mobile, the required number of sensors is reduced by 50%.

Compared to the case in which 100% of the sensors are mobile, to reach 90% coverage, the basic bidding protocol requires 40% fewer mobile sensors in the case in which 50% sensors are mobile. Note that the cost of mobile sensors are higher than static sensors, so the overall cost of using a percentage of mobile sensors may be reduced even though more sensors in total are used.

Figure 3.12 shows the sensor cost of these three protocols to reach a certain sensor coverage. Based on the *cost ratio* between the mobile sensor and the static sensor, the overall sensor cost of these three protocols may be different. Intuitively, if the cost ratio is low (e.g., 1.5), increasing the percentage of mobile sensors can reduce the overall sensor cost. On the other hand, if the mobile sensors are very expensive, using only static sensors may have the lowest sensor cost (not shown in the figure). When the cost ratio is somewhere in the middle, the basic protocol which has a mix of mobile and static sensors can achieve the lowest sensor cost. For example, when the cost ratio is 3.5, to reach 95% coverage, the basic protocol has the lowest cost when 10% sensors are mobile. Currently, the cost of a static sensor prototype Motes is about $100 and the cost of a mobile sensor prototype is about $200 [24]. The ratio is expected to increase under mass production. Based on this figure, we can see there is a tradeoff between cost and coverage. The basic protocol can achieve a balance between these two most of the time.

3.5.3 Comparing the Protocols

We consider four cases: both the proxy and basic bidding protocols using both distance and price-based criteria. In the figures showing the simulation results, we use 'Proxy-distance', "Proxy-price', 'Basic-distance' and 'Basic-price' to represent them, respectively.

3.5.3.1 Coverage

Figure 3.13 shows the coverage obtained by our protocols under different mobile sensor percentage. We can make two observations from the figure. One is that our bidding protocols can increase the coverage significantly. The other is that all four cases we consider achieve very similar coverage. All the four cases follow the same bidding framework and heal the largest holes. In terms of coverage, there is no preference between the basic-bidding protocol and the proxy-based bidding protocol; there is no preference between distance-based criteria and price-based criteria to choose mobile sensors.

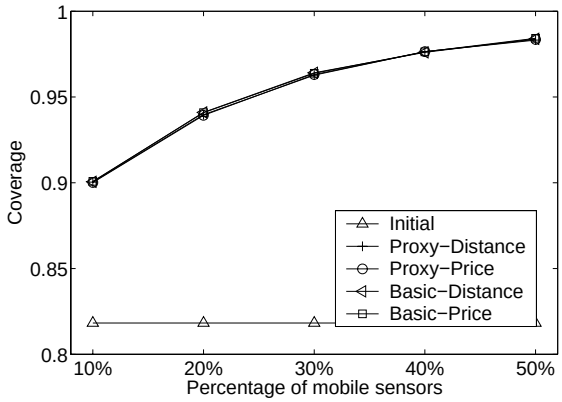


Fig. 3.13. Coverage

3.5.3.2 Termination

When all the largest holes are healed and no sensor can give a higher bid than the lowest base price of mobile sensors, the protocols terminate. Figure 3.14 shows the number of rounds that the protocols have run when the protocols terminate. As expected, the proxy-based bidding protocol requires more rounds to terminate. In the proxy-based bidding protocol, each sensor waits several rounds before physical movement and sensors spend a number of rounds on exchanging holes. However, because physical movements will likely dominate the recursion time, and the proxy-based protocol reduces movement, it may still terminate in the shortest time. In both the proxy-based protocol and the basic protocol, using distance-based criteria or price-based criteria does not significantly affect the termination time.

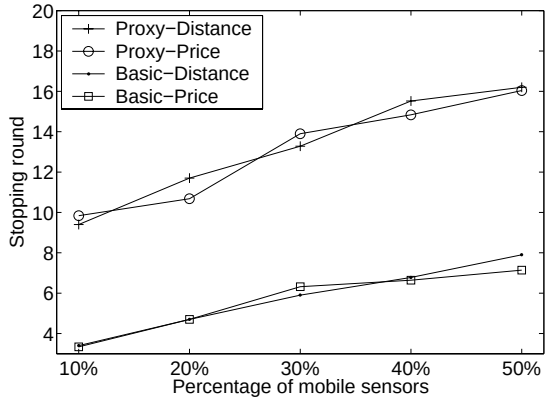


Fig. 3.14. Termination

The deployment rounds are increased when the mobile sensor percentage increases. With more mobile sensors available, the allocation of mobile sensors to coverage holes is more complicated and needs more rounds.

3.5.3.3 Energy Consumption

Energy consumption includes two parts, mechanical movement and communication. We use message complexity to measure the energy consumed in communication; we use the number of movements and moving distance to measure the energy consumption in movement. We first show the performance of our protocols in these three metrics. Then we show a unified energy consumption considering all these metrics.

Figure 3.15 shows the moving distance. As expected, the moving distance is much lower when the proxy-based bidding protocol is used. Between the distance-based criteria and price-based criteria, the moving distance is quite similar when using the proxy-based bidding protocol, and it is shorter when the latter criteria is used in the basic bidding protocol. The figure tells us that the phenomena shown in Figure 3.6 are dominant compared to those shown in Figure 3.5. In the proxy-based bidding protocol, these two criteria achieve similar performance. For most cases, the hole exchange and virtual movement change the situations illustrated in Figure 3.6 and Figure 3.5 to an ideal case. Therefore, these two criteria achieve a similar performance.

When considering the number of movements *vs.* the percentage of mobile sensors (not shown), we find that the number of movements required does not change as the percentage of mobile sensors increases. In addition, both the distance-based criteria and price-based criteria perform the same when using the proxy-based protocol (about

1.1). When using the basic bidding protocol, the price-based criteria achieves a smaller number of movements (about 1.45) than the distance-based criteria (about 1.6) for the same reason as presented in the above paragraph.

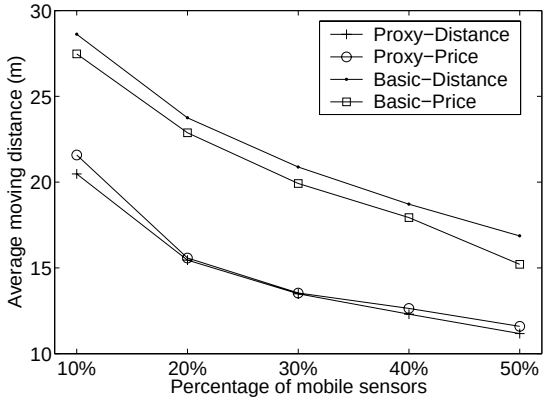


Fig. 3.15. Moving distance

Figure 3.16 shows the message complexity. The proxy-based protocol has higher message complexity than the basic protocol since it needs more rounds to terminate and needs to negotiate how to exchange holes. As mobile sensor percentage increases, the number of rounds increases, and message complexity increases accordingly.

To get a clear picture of energy consumption, we normalize the moving distance and the number of movements into message complexity. That is, with the same amount of energy consumed in movement, how many messages can be transmitted. Calculated from Robomote [78], approximately, to move a sensor one meter consumes a similar amount of energy as transmitting 300 messages. Also, we set the energy consumption

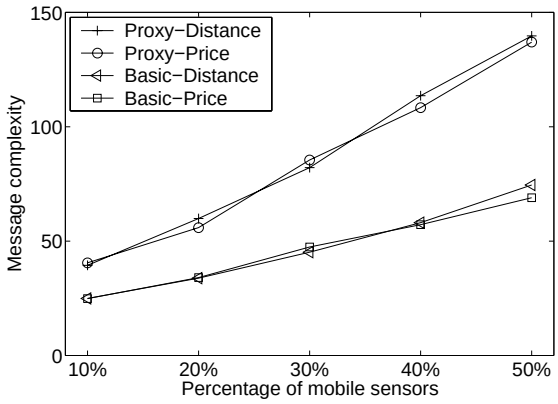


Fig. 3.16. Message complexity

in starting/braking to be the same as that in moving one meter. Figure 3.17 shows the unified energy consumption. As expected, the proxy-based bidding protocol consumes much less energy than the basic bidding protocol. Though sensors spend more energy in communication, they save much energy in movement. Mechanical movement is the dominant factor in energy consumption. Thus, the proxy-based protocol is much more energy efficient than the basic bidding protocol.

3.5.4 Load Balance

To evaluate whether individual sensors are penalized in terms of moving distance, we show the maximum moving distance among the mobile sensors in Figure 3.18. As expected, the maximum moving distance in the proxy-based protocol is much shorter than that in the basic bidding protocol.

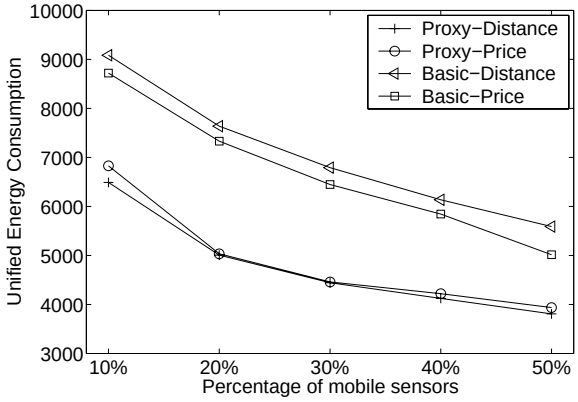


Fig. 3.17. Unified energy consumption

3.6 Related Work

In this section, we introduce related work in sensor coverage, static sensor deployment, mobile sensor deployment, and relay node placement in sensor networks.

3.6.1 Coverage

Meguerdichian, *et al.*, presented several interpretations of coverage in sensor networks, including deterministic coverage and stochastic coverage [63]. Also, the authors proposed a centralized polynomial time algorithm for coverage calculation. Another metric of sensor coverage, exposure, was defined in [62]. The authors also designed a centralized algorithm for calculating the minimal exposure paths.

3.6.2 Sensor Deployment

All previous work on sensor deployment either assumes all sensors are static or assumes all sensors are mobile. In the following, we first introduce chapters on static

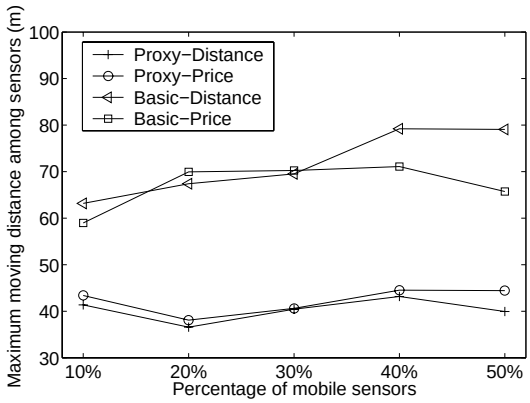


Fig. 3.18. Maximum moving distance

sensor deployment followed by chapters on mobile sensor deployment. Deployment of static sensor networks has been addressed in [18, 25]. Clouqueur, *et al.*, proposed to deploy sensors in several steps and assumed random deployment in each step [18]. The number of sensors in each step and the cost of deployment were used as a cost function. The authors proposed algorithms to determine the number of steps of sensor deployment such that the cost is low and the desired distribution is obtained. Dhillon, *et al.*, proposed a centralized polynomial-time algorithm to determine sensor distribution such that a minimum number of sensors are deployed and a minimum amount of data are transmitted.

Deployment of mobile sensors has been addressed in [41, 39, 49, 49, 90, 40, 42]. The work in [90] assumes that a cluster head is available to collect the sensor location and determine the target location of the mobile sensors. Howard, *et al.*, proposed an algorithm to deploy mobile sensors into a building from outside, in which sensors are deployed iteratively one by one, utilizing the location information obtained from the

previous deployment [49]. The same authors proposed algorithms based on potential field to maximize the monitoring field in [50]. Wang, *et al.*, proposed three algorithms, VEC, VOR, Minimax, and two protocols, to deploy mobile sensors to increase the coverage considering energy efficiency and deployment time. The authors gave insight on how to choose the algorithm and the protocol under different system requirements [41, 39].

The only work, to our knowledge, that addressed a mixed of mobile and static sensors, is our preliminary result of [40, 42].

3.6.3 Other Related Work

Other related work includes the study of heterogeneous networks in which not all sensors are the same, for example, networks that have both sensor nodes and relay nodes, which only have communication capability. Hou, *et al* proposed a centralized polynomial-time heuristic algorithm for relay node placement to increase network lifetime [48]. Patel, *et al* designed centralized deployment strategies for sensor nodes, relay nodes, and base stations considering connectivity and coverage [69].

3.7 Conclusions

In this chapter, we proposed to use a mix of mobile and static sensors to construct sensor networks to balance cost and sensor coverage. We identified the problem of deploying mobile sensors in a mixed sensor network as a NP-Complete problem, and designed bidding protocols to tackle this problem in a distributed fashion, in which static sensors act as bidders and mobile sensors act as hole-healing servers. Intensive simulation justified our idea of deploying both mobile and static sensors. Users can

determine the percentage of mobile sensors to get the most economical deployment of sensors to construct a network satisfying the coverage requirement. Simulation results also showed the efficiency and effectiveness of our proxy-based bidding protocol in placing mobile sensors to achieve high coverage.

In the future, we will work on the deployment of mobile sensors for non-uniform coverage requirements, or for purposes other than coverage. In many applications, some locations are more important than others and may require more sensors for coverage. The bidding protocols presented here can be adapted to this scenario by modifying the rules of assigning base and bid price. In addition, we believe the bidding protocol can be used in many other applications, such as distributed resource allocation.

Chapter 4

Optimizing Sensor Movement Planning for Energy Efficiency

4.1 Introduction

In a mobile node, mechanical actuation has much higher power consumption than communication, sensing and computation [24]. Therefore, energy efficient movement is very important for mobile nodes to increase their lifetime when energy recharge is difficult. Most applications utilizing mobile sensors reduce energy consumption by reducing the number of movements and the moving distance of each sensor.

In this chapter, we study the problem of minimizing the entire actuation energy consumption in mobile sensors. To accurately characterize this energy consumption, we adopt a complete energy model, which includes not only the required mechanical energy for accomplishing the movement command, but also various energy dissipation due to acceleration, heating, viscous damping, internal motor friction, etc. Unlike previous models, this model considers the entire energy consumption from the energy source, e.g., battery, to accomplish the moving requirement of the application. Based on this model, we propose an optimal velocity schedule for mobile nodes when the road condition is uniform. For sensors moving under variable road conditions, we propose to use continuous-state dynamic programming to calculate a near optimal velocity schedule. We can postulate a desired approximation level. We propose a method to minimize the

computation complexity to achieve the desired approximation. We also study the variety in motion hardware. Specifically, we design one velocity schedule for simple microcontrollers, in which the acceleration can not be specified and one velocity schedule for relatively complex microcontrollers, in which acceleration can be specified. Simulation results show the effectiveness of our velocity planning.

The rest of the chapter is organized as follows. In section 4.2, we introduce a technical preliminary on actuation. We present the energy efficient motion planning in section 4.3 and draw conclusions in section 4.4

4.2 Technical Preliminary

Generally, the motion base of a mobile node is composed of three main parts: (1) driving/moving devices, e.g., the wheels; (2) motors, which transfer the electrical energy to mechanical energy and drive the wheels; (3) microcontroller, which controls mobile sensors to move as desired.

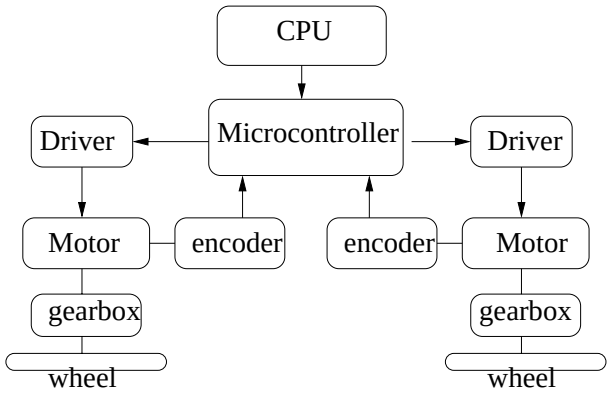


Fig. 4.1. Architecture of the Motion Base

The driving devices include wheels, caterpillars and walking legs. The current mobile sensor prototype [78] is wheel-based. Many small mobile robot platforms, which are likely to be adopted in mobile sensors, use wheels as well for their low cost and easy manipulation. In this chapter, we consider wheeled mobile sensors.

Motors transform electrical energy supplied by the battery to mechanical energy needed to rotate the wheels and run the mobile sensors. The angular velocity of a motor determines that of the wheel it drives, and consequently determines the velocity of the mobile sensor. The output power of a motor is the product of its angular velocity and the torque it applies to run the wheel. The higher the torque, the heavier the load that a motor can support. For example, in a rough ground, a higher torque is needed to run a mobile sensor than that in a smooth ground. Normally, there is a gearbox between a motor and the wheel it drives, which can reduce the angular velocity and increase the torque supplied. In this way, with the same output power, more load can be supported.

By adjusting the voltage applied to a motor, it can be accelerated or decelerated. To make a motor rotate at a desired speed or with a desired acceleration, a feedback PID (Proportional Integration Derivation) controller is normally used to calculate the voltage which should be applied to the motor. The feedback controller takes the current velocity of the motor as input, computes its difference to the desired velocity, and calculates the best voltage to minimize this difference. This best voltage is expressed as a logical signal and will be amplified by the motor driver. Then the real desired voltage will be supplied to the motor. The current velocity of a motor is monitored by an encoder. This whole feedback control process is taken rather frequently, depending on the accuracy of the encoder. The relationship between these parts in the motion base is shown in Figure 4.1.

4.2.1 Movement Mechanism and Kinematics

We consider mobile sensors with differential drives, the same kind as Robomote [24], the mobile sensor prototype mentioned above. Small mobile robot platforms, khepera [4] and FIRA [56], are also similar. Differential drive moves two coaxial wheels with two independent electric motors. The moving direction and velocity of a mobile sensor is determined by the rotation speed of these two drive wheels. Let ν be the velocity in the forward direction and ω_s be the angular velocity around the ICR (Instantaneous Center of Rotation) of the mobile sensor, ω_1 and ω_2 be the angular velocity of two wheels, r be the radius of the wheel, and d be the distance between the two wheels. ν and ω_s has the following relationship with ω_1 and ω_2 [56]:

$$\begin{cases} \nu = r(\frac{\omega_1+\omega_2}{2}). \\ \omega_s = r(\frac{\omega_1-\omega_2}{d}). \end{cases} \tag{4.1}$$

There are other kinds of wheeled movement mechanisms, including omnidirectional movement and car-like nonholonomic mechanism. For those kinds of motion mechanism, ν and ω_s can also be written as a linear combination of the angular velocities of the wheels, as in equation (4.1). Our method of energy efficient motion planning can be easily extended for the usage in those cases.

4.3 Energy Efficient Motion Planning

We deal with the problem of energy-efficient motion planning of a mobile sensor given a movement task from the application: "move to target location D". To fulfill

this task, the mobile sensor needs to determine the path to the target location and the velocity and acceleration along the path.

Suppose the distance between a sensor's current location and its target location is L, and the heading position of this mobile sensor is θ with respect to its target location. To make a short moving path, the mobile sensor can first turn to the direction of its target location and then move straight to its target location. According to equation (4.1), a mobile sensor can turn by making its two drive wheels spin in the opposite directions with the same rate and can move straight by making its wheels spin in the same directions with the same rate. Since the wheels spin at the same rate when turning, the energy consumed is the same as moving straight by a distance $\theta * d/2$. Therefore, we concentrate on how to determine a velocity schedule of motors (which is proportional to that of the wheels), such that the energy consumption is minimized. In the following sections, we first introduce the power model of a motor. Then we describe how to determine the velocity schedule under a constant load (uniform road condition). Finally, we present the continuous-state dynamic programming method to calculate the velocity schedule under variable load.

4.3.1 Power Model of Motion

To accurately model the energy consumption for motion, we consider not only the mechanical energy required for movement, but also the energy dissipation inside the motor due to various reasons, such as internal friction, etc. We choose the energy model described in [19]:

$$P(t) = V(t)I(t); \tag{4.2}$$

$$V(t) = RI(t) + K_e\omega(t); \tag{4.3}$$

$$I = \frac{1}{K_T}[(J_m + J_L)\frac{d\omega(t)}{dt} + T_L(t) + T_f + D\omega(t)] \tag{4.4}$$

Here, $P(t)$ is the power consumption. $V(t)$ is the voltage supplied to the motor and $I(t)$ is the current flow through the rotor of the motor. R is the armature resistance. K_e is the back EMF (electormotive force) constant. $\omega(t)$ is the angular velocity of the motor at time t. K_T is the torque constant of the motor. J_m and J_L is the inertia of motor and load, respectively. $T_L(t)$ is the load torque at time t. T_f is the friction torque of the motor. D is the viscous damping, which represents the coefficient of speed-dependent power dissipation.

4.3.2 Energy Efficient Velocity Planning with Constant Load

As shown in our power model, the energy consumed in a motion is determined by many factors. K_e, K_T, D, T_f and J_m are the characteristics of the motor. J_L is a characteristic of the mobile sensor; load torque T_L is determined by the environments in which sensor moves, e.g, the surface friction; angular velocity is specified by the users; depending on the complexity of the microcontroller, accelerations may also be determined by the users. We aim to calculate a velocity schedule optimal for energy consumption in case (a) only velocity can be specified and case (b) both velocity and acceleration can be specified.

In this section, we consider a constant load torque during the course of movement, which corresponds to the assumption that the sensor moves over a flat ground with uniform friction. In the next section, we address the velocity planning under variable load torque.

Since unnecessary accelerations cost more energy, the optimal velocity schedule should be accelerating to ω, moving uniformly at ω, and decelerating to zero at the target location. We will find the best ω for minimizing the energy consumption if we are not allowed to set the acceleration α_a and deceleration α_d; and find the best triple $\langle \alpha_a, \omega, \alpha_d, \rangle$ if we are allowed to do so.

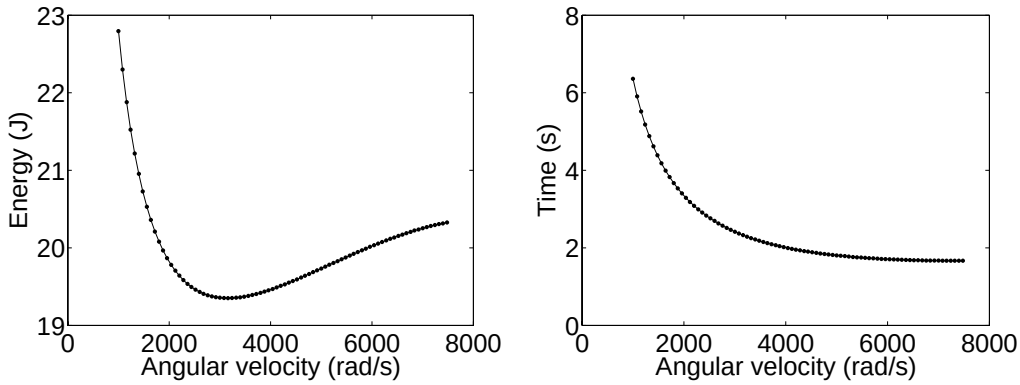
The energy consumption for a mobile sensor to move distance L is the sum of the input energy of the two drive motors, which can be expressed as $E(L) = 2\int_0^T P(t)\,dt$. T is the time for movement, which is a function of ω, α_a, α_d and L. Let T be $f_t(\omega, \alpha_a, \alpha_d, L)$. Then the energy consumption is

$$E(L) = 2\int_0^{f_t(\omega,\alpha_a,\alpha_d,L)} P(t)\,dt. \tag{4.5}$$

4.3.2.1 Calculating Optimal ω

By computation, $E(L)$ can be expressed as a polynomial of ω, α_a, α_d. The detailed calculation and expression of $E(L)$ is shown in Appendix A in the end of this chapter. In case we can not specify α_a and α_d to the microcontroller, we want an ω that minimizes $E(L)$. The necessary condition for ω to be a local minimizer of $E(L)$ is $\frac{dE(L)}{d\omega} = 0$. By taking the derivative of E(L) with respect to ω, we get a polynomial

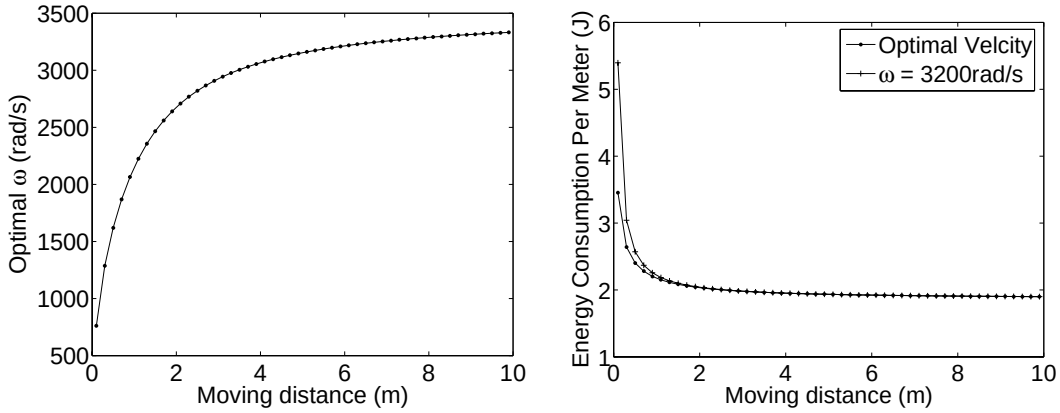
equation of degree four. Polynomial equations of degree four can be solved analytically in constant time.



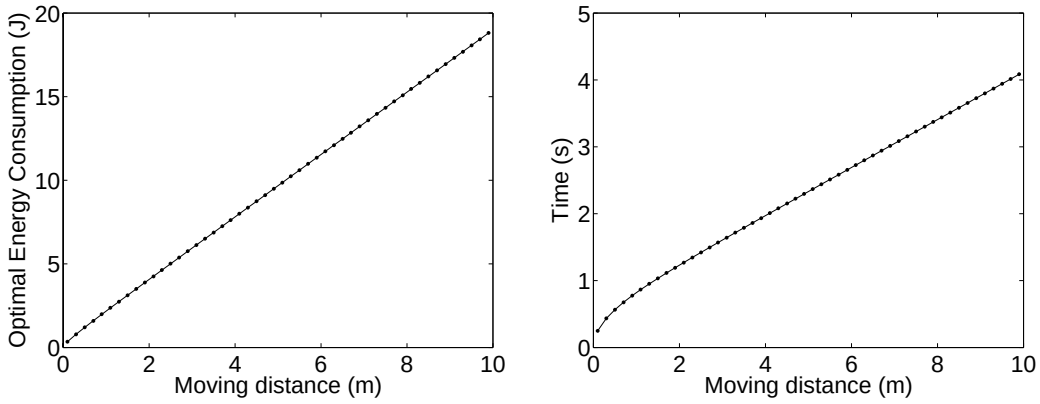
(a) Energy consumption (b) Moving time

Fig. 4.2. Impact of ω

In Figure 4.2, we plot the energy consumption and the moving time for a mobile sensor to move 5 meters as the angular velocity varies. The acceleration/deceleration is set to $9000 rad/s^2$. The load torque is 10 times the friction torque. For simulating the behavior of a motor, we use the data of a DC Micromotor (Serious 1319 012S) from Micromo Electronics [5]. For other parameters, as gear ratio, we use those compatible with with *Robomote* and *Khepera*. The detailed value of each parameter is shown in Appendix B in the end of this chapter. From the figure, we can see that, by choosing an appropriate velocity, energy can be saved significantly. For example, by setting ω at its optimal value, $0.7J$ can be saved compared with setting it to $6000 rad/s$, and the saved energy can be used to send hundreds of messages.



(a) Angular velocity (b) Energy consumption per meter



(c) Energy consumption (d) Moving time

Fig. 4.3. Impact of moving distance

In Figure 4.3, we plot the optimal value of ω for energy efficiency, the corresponding energy consumption, and moving time, as the moving distance varies. From 4.3(a), we can see that the optimal angular velocity is not linearly related to the moving distance. Instead, optimal velocity approaches a fixed value when the moving distance increases. We choose an ω, $3200 rad/s$, which is compatible to the optimal velocity for moving distance longer than $2.5m$, and plot the energy consumption per meter under this velocity and the optimal velocity in Figure 4.3(b). As shown in the figure, the energy consumption per meter is almost constant for moving distance longer than 2 meters. We can see that, if the sensor moves on a uniform surface and the movements over very small distances are rare, we can obtain very satisfactory results with a fixed ω. Otherwise, calculating optimal velocity schedule can achieve significant savings of energy, on the order of even $40 - 50\%$.

As shown in Figure 4.3(c), the optimal energy consumption has a near-linear relationship with the moving distance. Also, when the moving distance approaches zero, the optimal energy consumption approaches a number greater than zero, which means there is always a "startup" energy. Figure 4.3(d) shows the moving time under optimal velocity. We can see that this moving time also has a near-linear relationship with the moving distance and there is a "startup" time.

4.3.2.2 Calculating Optimal $\langle \alpha_a, \omega, \alpha_d \rangle$

Some microcontrollers, such as the MCDC2805 motion controller from *Micromo Electronics*, allow users to specify acceleration and deceleration in addition to the angular velocity. In this case, we want to optimize triplet $\langle \alpha_a, \omega, \alpha_d \rangle$ to minimize $E(L)$. The

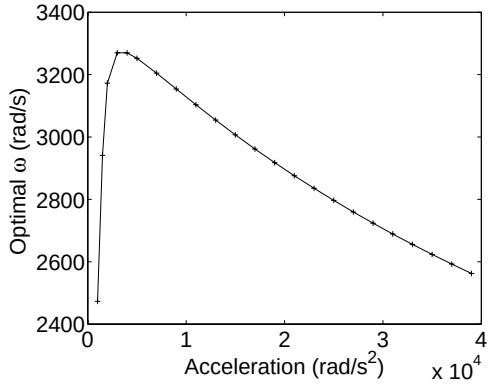
necessary condition for this triplet to be a minimizer of $E(L)$ is as follows:

$$\begin{cases} \frac{\partial E(L)}{\partial \omega} &= 0 \\ \frac{\partial E(L)}{\partial \alpha_a} &= 0 \\ \frac{\partial E(L)}{\partial \alpha_d} &= 0 \end{cases} \tag{4.6}$$

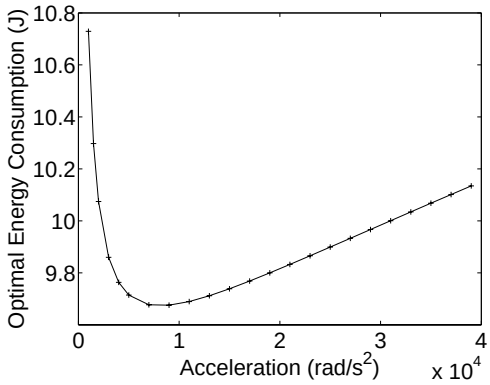
By taking the partial derivative of E(L) with respect to ω, α_a, and α_d, we get a system of three equations. Using two of these three equations, α_a and α_d can be eliminated. In fact, α_a and α_d have the same magnitude. After the elimination, we get a polynomial equation of degree eight in ω. This equation can be solved by computing the eigenvalues of the companion matrix [6]. The computation complexity is $O(n^2)$, where n is the degree of the polynomial (here $n = 8$).

Figure 4.4 plots the optimal angular velocity for a sensor to move 5 meters when α varies (here we choose $\alpha_d = \alpha_a$), the corresponding energy consumption and moving time. From the figure, we can see that an optimally chosen α can not only reduce the energy consumption significantly, but also reduce the moving time.

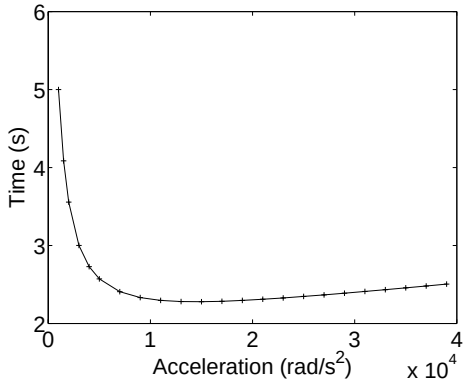
Figure 4.5 shows the optimal ω and α ($\alpha_a = \alpha_d$ in the optimal setting) for energy efficiency, the achieved energy consumption under such setting, and the corresponding movement time, under different moving distance. We can see, by properly setting ω and α, the energy consumption and the moving time is approximately linear to the moving distance.



(a) Angular velocity

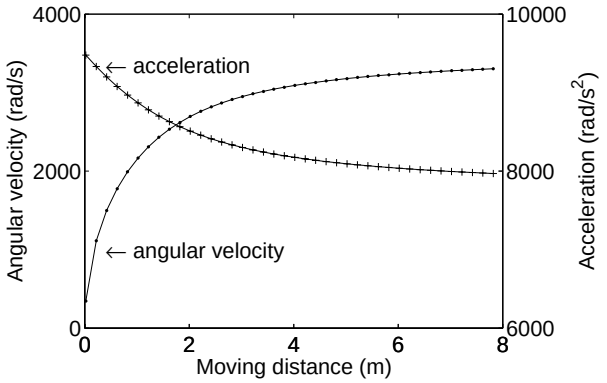


(b)Energy consumption

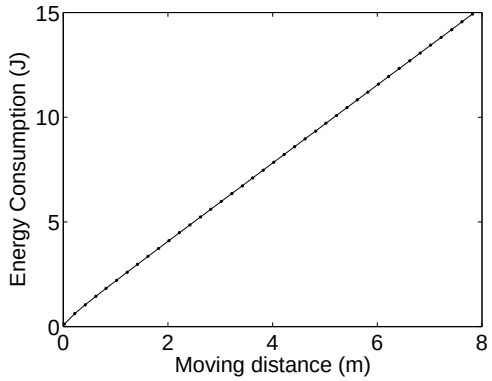


(c)Moving time

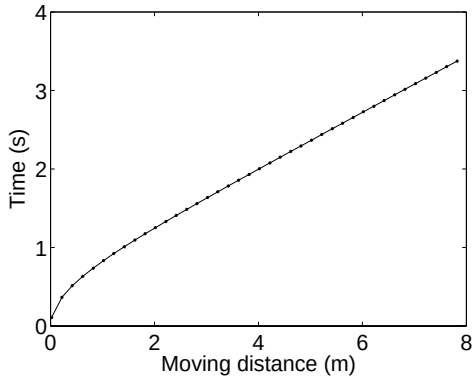
Fig. 4.4. Impact of Setting Acceleration/Deceleration



(a) Optimal ω and α



(b) Energy consumption

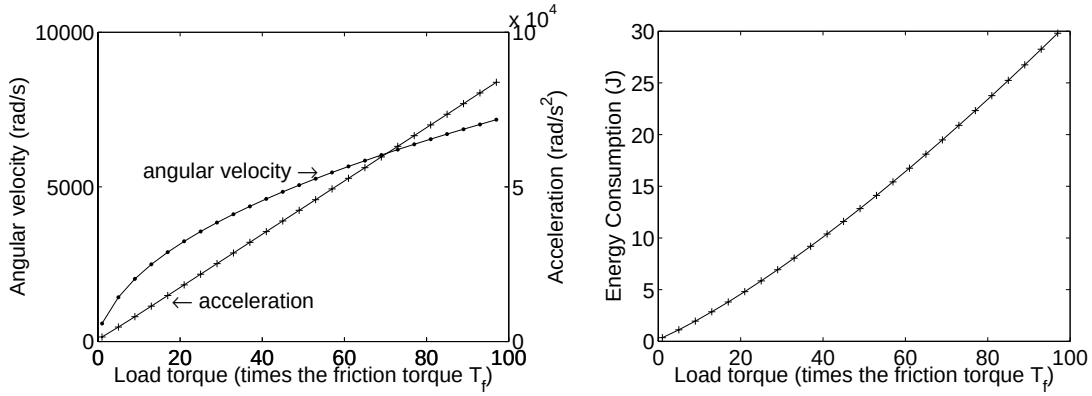


(c) Moving time

Fig. 4.5. Optimal setting of $\langle \alpha_a, \omega, \alpha_d \rangle$

4.3.3 Energy Efficient Velocity Planning With Variable Load

In this section, we consider the situation that load torque changes during movement. When load torque changes, the format of the velocity schedule for constant load, *i.e.*, accelerating, moving uniformly, and decelerating, may not be an optimal solution as in the scenario of constant load torque. As shown in Figure 4.6, the optimal angular velocity and acceleration for energy consumption change when the load torque varies. In these situations, minimization of energy consumption may require additional accelerations in the middle of movement.



(a) Optimal ω and α (b) Energy consumption

Fig. 4.6. Impact of load torque

As explained in section 4.3.2, frequent accelerations should be avoided. In addition to starting and stopping, accelerations should only occur around the place where load torque changes. The whole path with variable condition can be segmented into S_1, S_2,

..., S_m, such that each segment has uniform load condition. For example, a mobile node moves from a concrete road onto the dirt field. Its path can be segmented into two parts. In segment S_i with uniform load torque, mobile node will have velocity schedule $\langle \overline{\omega_i}, \alpha_i, \omega_i, \alpha_i', \overline{\omega_{i+1}} \rangle$, where $\overline{\omega_i}$ and $\overline{\omega_{i+1}}$ are the initial and final speed, which are the transition speed between segment S_{i-1} and S_i, and between S_i and S_{i+1}, ω_i is the uniform velocity in segment S_i, α_i is the initial acceleration and α_i' is the final acceleration. Therefore, the velocity schedule for a path with m segments should have the following format:

$$\langle 0 = \overline{\omega_1}, \alpha_1, \omega_1, \alpha_1', \overline{\omega_2}, \cdots, \overline{\omega_m}, \alpha_m, \omega_m, \alpha_m', \overline{\omega_{m+1}} = 0 \rangle$$

Certainly, it is quite possible that α_i or α_i' is equal to zero and $\overline{\omega_i}$ is equal to ω_{i-1} or ω_i in the optimal velocity planning.

An optimal velocity schedule can be obtained by calculating the total energy consumption expressed in terms of the variables described above, taking partial derivatives, and constructing and solving a system of equations. However, this method is not scalable since the system of equations changes when the number of segments changes.

To achieve scalability, we propose dynamic programming to optimize the velocity schedule. In our dynamic programming problem, the transition velocity between segments is the "state variable". Given the transition velocity of one segment, we can calculate the optimal velocity schedule for minimizing energy consumption in this segment by using a similar method as in section 4.3.2. Let the function to do this calculation be G, where $\langle \alpha_i, \omega_i, \alpha_i' \rangle = G(\overline{\omega_i}, \overline{\omega_{i+1}}, S_i)$ (argument S_i provides the length and load

torque of this segment), and $E(\overline{\omega_i}, \overline{\omega_{i+1}}, S_i)$ is the respective energy consumption. Let $Ec(\overline{\omega}, i)$ be the minimum energy consumption for segments S_1, S_2, ..., S_i, assuming that the final angular velocity equals $\overline{\omega}$. We have the following recurrence:

$$Ec(\overline{\omega}, i) = \begin{cases} E(0, \overline{\omega}, 1), & if\ i = 1 \\ \min_{\overline{\omega_p}}(Ec(\overline{\omega_p}, i-1) + E(\overline{\omega_p}, \omega, S_i)), & otherwise. \end{cases}$$

Calculating $Ec(\overline{\omega}, 1)$, $Ec(\overline{\omega}, 2)$, $\cdots$, $Ec(\overline{\omega}, i-1)$ in sequence, we obtain the minimized $Ec(0, m)$ and the corresponding velocity schedule. This actually is a process of finding the transition velocity $\overline{\omega_i}$ (for $i = 2, \cdots, m$) to minimize the total energy consumption. However, our problem is a *continuous-state* dynamic programming problem since velocity is a continuous variable and there are infinite possible values. We need to discretize the velocity space, such that the finite-state dynamic programming can be applied. In this way, we can obtain the approximately optimal solution.

Suppose the space of the angular velocity is $\{0, \omega_{max}\}$. One possible discretization is to choose a number n and discretize the velocity space into $\{0, \omega_{max}/n, 2\omega_{max}/n, \cdots, (n-1)\omega_{max}/n, \omega_{max}\}$. For this discretization, we need to run function G $m(n+1)^2$ times, for a fineness of ω_{max}/n. Suppose ω_{max} is $17000 rad/s$ and we want a fineness of $10 rad/s$. Then n is 1701 and we need to run function G $1701^2 m$ times. To reduce the computation complexity, we discretize the velocity space iteratively. We use a small n in each step, discretize the velocity space into $(n+1)$ grid points and get the best transition velocities within these grid points. Then we reduce velocity space into $2/n$ of the original space

centered at the calculated transition velocity. After l iterations, we obtain the fineness of $\omega_{max} * (2/n)^l$ and run the function G $m * l * (n+1)^2$ times.

The remaining task is to find a n such that we can minimize the computation cost for the same fineness. That is, we want to find a n which can get a finer discretization with the same computation. Let x be the number of evaluations of function G, and $f(x)$ be the fineness of the discretization. In each iteration, we do $m(n+1)^2$ computations, and reduce the velocity space into $2/n$ of the original. After l iterations, we will have executed function G $lm(n+1)^2$ times and reduced the velocity space into $(2/n)^l$ of the original space. Here, $x = lm(n+1)^2$ and $f(x) = (2/n)^l$. Then f(x) can be expressed as:

$$f(x) = [(\frac{2}{n})^{\frac{1}{(n+1)^2}}]^{\frac{x}{m}} \tag{4.7}$$

Therefore, we want to minimize $(\frac{2}{n})^{\frac{1}{(n+1)^2}}$ to obtain smallest velocity space with the same computation complexity. When $n = 4$, we get the best solution. For the example mentioned above, by setting n to four, we get a fineness of $8.3rad/s$ after 11 iterations, and run the function G $275m$ times, as compared with 1701^2m times execution of function G for a fineness $10rad/s$.

4.4 Conclusion

In this chapter, we address the problem of minimizing movement-related energy consumption. We adopt a complete energy model. Based on the model, we propose approaches for calculating the velocity schedule to minimize the energy consumption for mobile nodes equipped with various kinds of motion controllers and moving in various

load situations. Experimental results show that energy saving is significantly by using the velocity schedule.

Appendix A

First, we describe some notations. t_{acc} is the time used to accelerate, t_{con} is the time of moving uniformly at ω, and t_{dec} is the time of deceleration. d_{acc}, d_{con} and d_{dec} are the moving distances in these three phases, respectively. v is the speed of the mobile sensor and N_r is the gear ratio of the gear box. $v = r\omega/N_r$.

The time T to finish the movement can be expressed as

$$\begin{aligned} T &= t_{acc} + t_{con} + t_{dec}, where \\ & t_{acc} = \omega/\alpha_a; d_{acc} = \tfrac{1}{2}.\tfrac{\omega}{\omega_a}.\tfrac{r\omega}{N_r} \\ & t_{dec} = \omega/\alpha_d; d_{dec} = \tfrac{1}{2}.\tfrac{\omega}{\omega_d}.\tfrac{r\omega}{N_r} \\ & t_{con} = d_{con}/v = (L - d_{acc} - d_{dec})/v = \tfrac{N_r * L}{r * \omega} - \tfrac{\omega}{2\alpha_a} - \tfrac{\omega}{2\alpha_d} \end{aligned} \tag{4.8}$$

Plugging equation (4.3), (4.4) and (4.8) into equation (4.5), we get

$$
\begin{aligned}
E(L) = \;& b3 * w^3 + b2 * w^2 + b1 * w + b0 + b\_1 * w^{-1}, where \\
b3 = \;& (-1/(6 * \alpha_a) - 1/6\alpha_d)((RD^2 + K_e K_T D)/K_T^2 \\
b2 = \;& (K_e K_T + 2RD)(\alpha_a J + T_L + T_f)/(2\alpha_a K_T^2) + \\
& (K_e K_T + 2RD)(\alpha_d J + T_L + T_f)/(2\alpha_d K_T^2) - \\
& (K_e K_T + 2RD)(T_L + T_f)/(2\alpha_d K_T^2 + 2\alpha_d K_T^2)) \\
b1 = \;& R(\alpha_a J + T_L + T_f)^2/(\alpha_a K_T^2) + \\
& R(\alpha_d J + T_L + T_f)^2/(\alpha_d K_T^2) - \\
& R(T_L + T_f)^2(1/(2\alpha_a) + 1/(2\alpha_d))/K_T^2 + \\
& (RD^2 + K_e K_T D) N_r L/r K_T^2 \\
b0 = \;& (K_T K_e + 2RD)(T_L + T_f) N_r L/r K_T^2 \\
b\_1 = \;& N_r L R (T_L + T_f)^2/r
\end{aligned}
$$

Appendix B: Experimental Setting

For simulating the behavior of motor, we use the data of a DC Micromotor (Serious 1319 012S) from *Micromo Electronics*, which are listed below.

R	22 Ω
J_m	$3.824 * 10^{-6}$ z-in-sec^2
K_e	0.682 mv/rpm
K_T	0.923 oz-in/A
T_f	0.014 oz-in

Table 4.1. **Parameters of the Motor**

The viscous damping D is not provided from the datasheet of this Micromotor. We set D to be $4.85 * 10^{-8}$Nm/(rad/s), which is compatible to similar DC micromotors. We use the same gear ratio, 25 : 1, as the mobile sensor prototype developed by USC, Robomote. Since the load inertia after the gear reduction should be smaller than 5 times the load inertia, here we choose it to be three times the motor inertia, which is $1.1472 * 10^{-5}$ z-in-sec^2. The wheel radius is set to be $0.02m$ and the distance between the two drive wheels is $0.05m$, which are compatible with *Robomote*, *Khepera* and *Mica*.

Chapter 5

Sensor Relocation in Mobile Sensor Networks

5.1 Introduction

During the lifetime of sensor networks, sensor nodes may fail, requiring nodes to be moved to overcome the coverage hole created by the failed sensor. In this chapter we address the problem of *sensor relocation*, i.e., moving previously deployed sensors to overcome the failure of other nodes, or to respond to an occurring event that requires that a sensor be moved to its location. This sensor relocation is different from sensor deployment. Compared with sensor deployment, sensor relocation has many special difficulties. First, sensor relocation may have a strict response time requirement. For example, if the sensor monitoring a security-sensitive area dies, another sensor should move to replace it as soon as possible. Second, relocation should not affect the application currently using the sensor network, which means that the relocation should minimize its effect on the current sensing topology. Finally, since movement may be much more expensive in terms of energy than computation and communication, any algorithm must balance energy costs with response time. In particular, care must be taken to balance the energy costs of an individual node with the overall network energy cost to ensure maximum network lifetime.

In this chapter, we propose a framework for relocating mobile sensors in a timely, efficient, and balanced manner, and at the same time, maintaining the original sensing

topology as much as possible. In our framework, sensor relocation consists of two phases: the first is to find the redundant sensors in the sensor network; the other is to relocate them to the target location. For the first phase, we propose a *Grid-Quorum* based solution to quickly locate the redundant sensors with low message overhead. For the second phase, we propose efficient heuristics to achieve good balance between energy efficiency and relocation time when determining the sensor relocation path. Simulation results show that the proposed heuristics are very effective in reducing the relocation time and the energy consumption.

The rest of the chapter is organized as follows. In section 5.2, we define the sensor relocation problem and the grid-based system model. Section 5.3 presents the Grid-Quorum solution and Section 5.4 presents our solution for relocating sensors. Performance evaluations are presented in Section 5.5. Section 5.6 concludes this chapter.

5.2 Problem Statement

In theory, the two protocols we previously proposed in Chapter 2 and Chapter 3 can be used for sensor relocation. For example, after a sensor failure, the sensors neighboring the failed node can execute the algorithms. After several rounds, the neighbor sensors will move to cover the area initially covered by the failed sensor. However, moving neighbor sensors may create new holes in that area. To heal these new holes, more sensors must move. This process continues until some area having redundant sensors is reached and the sensors leaving this area do not create new holes. Using the method, sensors may move several times, wasting energy. In addition, since many sensors are

involved, it may take a long time for the algorithm to terminate. Based on this observation, we propose to first find the locations of the redundant sensors, and then design an efficient route for them to move to the destination.

To determine which sensor(s) is redundant is a challenging problem. It is hard for a single sensor to independently decide whether its movement will generate a coverage hole. To make such a decision, the sensor requires information about whether its neighbors will move or not. More specifically, a number of sensors located closely must determine the redundant sensors among themselves.

A *grid-based* architecture is a natural solution for this problem. We can divide the target field into grids. The grid head is responsible for collecting the information of its members, and determining the existence of redundant sensors based on their locations. For redundant sensors located on the boundary of the grid, grid heads coordinate to make decisions. The grid head can also monitor its group members and initiate a relocation process in case of new event or sensor failure.

A grid-based architecture is feasible in a network in which nodes are relatively regularly deployed, for example as would be the case after the termination of previously proposed sensor deployment algorithms presented in Chapter 2, and Chapter 3. This is because, unlike the case of a network in which nodes are randomly deployed, the cost of organizing sensors into grids is low. Further, this organization can facilitate data aggregation, routing, etc., [86, 85] in addition to finding the redundant sensors. Since many existing techniques on grid (cluster) maintenance [47, 83] can be directly applied, we will not address these issues in this chapter.

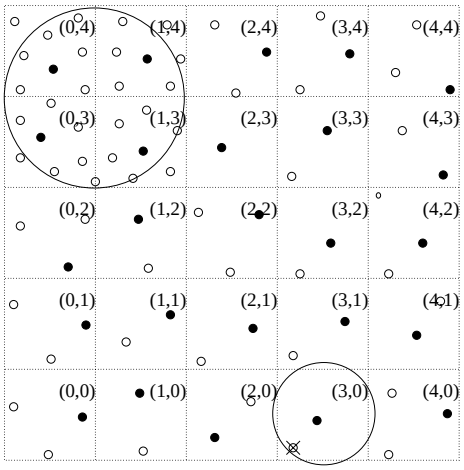


Fig. 5.1. The system model

With the grid-based model, the sensor relocation problem can be reduced to two sub-problems: finding the redundant sensors and then relocating them to the target location. Figure 5.1 illustrates the sensor relocation problem when grids are used; the black nodes are used to represent grid heads. Each grid is indexed by a tuple, whose first number is used to represent the column and the second number is used to represent the row. Grids (1,3), (0,3), (1,4) and (0,4) have redundant sensors. When a sensor at grid (3,0) dies, resulting in a coverage hole, its grid head first needs to locate the redundant sensor and then relocate some sensor to fix the coverage hole. For the first problem, we propose a Grid-Quorum solution to quickly identify the redundant sensors. For the second problem, we propose a cascaded movement solution to relocate sensors in a timely and energy efficient way.

5.3 Finding the Redundant Sensors

In this section, we first give the background and motivation of the Grid-Quorum idea. Then, we present the detailed solution and illustrate its advantage in terms of message complexity and response time.

5.3.1 Background and Motivation

The problem of finding redundant sensors has some similarity to the publish/subscribe problem [30, 37, 16], where the publisher advertises some information and the subscriber requests the information. Mapping the terminology to our problem, the grids that need more sensors are the subscribers, and the grids that have redundant sensors are the publishers. In the publish/subscribe system, the matching of a request to an advertisement is called *matchmaking*.

Generally, there are three types of solutions for matchmaking. (1) Matchmaking occurs at the subscriber, which is referred as "broadcast advertisement" [30]. In our problem, this is similar to letting the grids having redundant sensors flood this information. Later, when some grid needs redundant sensors, it can get the information quickly. (2) Matchmaking occurs at the publisher, which is referred as "broadcast request" [30]. In our problem, this is similar to letting the grids that need sensors flood the request. The grid that has redundant sensors replies after receiving the request. (3) Matchmaking happens in the middle of the network [16, 37, 87]. In our problem, this is similar to letting the supply grid advertise the information to some intermediate grids from which the demand grid obtains the information.

Unlike the traditional publish/subscribe problem, the information in our system is not reusable. The information about the redundant sensor can only be used once, since it may be changed after the redundant sensor moves to the requesting place. Due to this special property, the message complexity will be very high if we use the broadcast advertisement approach. This is due to the reason that two network-wide broadcasts are needed for each redundant sensor: one for advertisement and the other for data update after the redundant sensor moves.

For the broadcast request approach, the delay is relatively long since it is on-demand. Therefore, we prefer the third solution, which can achieve a balance between message complexity and response time. In this type of solution, a structure like that in [37, 87], can be used to facilitate the matchmaking between the advertisement and the request. Since the data may not be re-used, this structure should be simplified compared to that in [37, 87]; otherwise the benefit may not be worth the cost. Therefore, we need a simple and low-cost structure for matchmaking.

Our solution is motivated by the concept of quorum [14], which is defined as follows. Given a nonempty set $\mathcal{U}$, a coterie $\mathcal{C}$ is a set of $\mathcal{U}$'s subsets. Each subset $\mathcal{P}$ in $\mathcal{C}$ is called a quorum. The following condition must be held for quorums in a coterie $\mathcal{C}$ under $\mathcal{U}$:

- $(\forall \mathcal{P} \in \mathcal{C} :: \mathcal{P} \neq \emptyset \wedge \mathcal{P} \subseteq \mathcal{U})$
- Minimality Property: $(\forall \mathcal{P}, \mathcal{Q} \in \mathcal{C} :: \mathcal{P} \not\subset \mathcal{Q})$
- Intersection Property: $(\forall \mathcal{P}, \mathcal{Q} \in \mathcal{C} :: \mathcal{P} \bigcap \mathcal{Q} \neq \emptyset)$

By organizing grids as quorums, each advertisement and each request can be sent to a quorum of grids. Due to the intersection property of quorums, there must be a grid which is the intersection of the advertisement and the request. The grid head will be able to match the request to the advertisement. A simple quorum can be constructed by choosing the nodes in a row and a column. Instead of flooding the network with advertisements or requests, the request and the advertisement are only sent to nodes in a row or column. For example, as shown in Figure 5.1, suppose grid (0,3) has redundant sensors, it only sends the advertisement to grids in a row ((0,3), (1,3), (2,3), (3,3), (4,3)) and a column ((0,4), (0,3), (0,2), (0,1), (0,0)). When grid (3,0) is looking for redundant sensors, it only needs to send a request to grids in a row ((0,0), (1,0), (2,0), (3,0), (4,0)) and a column ((3,4), (3,3), (3,2), (3,1), (3,0)). The intersection node (0,0) will be able to match the request to the advertisement. Suppose N is the number of grids in the network. By using this quorum based system, the message overhead can be reduced from $O(N)$ to $O(\sqrt{N})$. Although the message overhead is very low compared to flooding, we can further reduce the message overhead by observing the specialty of our problem.

5.3.2 The Grid-Quorum Solution

In our Grid-Quorum system, we do not require the intersection of any two quorums. Instead, we deploy two coterie, called *supply coterie* and *demand coterie* separately, and only require that the quorum belong to the supply coterie intersects with all quorums in the demand coterie, and vice versa.

The formal definition is as follows. Given a nonempty set $\mathcal{U}$, there is a supply coterie $\mathcal{C}_s$ and a demand coterie $\mathcal{C}_d$, which are the sets of $\mathcal{U}$'s subsets. Each subset $\mathcal{P}_s$

in coterie $\mathcal{C}_s$ is called a supply quorum and each subset $\mathcal{P}_d$ in coterie $\mathcal{C}_d$ is called a demand quorum. Suppose coterie $\mathcal{C}_s$ has m quorums, and coterie $\mathcal{C}_d$ has n quorums. The following condition must be hold for quorums in coterie $\mathcal{C}_s$ and $\mathcal{C}_s$ under $\mathcal{U}$:

- $\bigcup_{i=1}^{m} \mathcal{P}s_i = \mathcal{U}$

  $\bigcup_{i=1}^{n} \mathcal{P}d_i = \mathcal{U}$

- Minimality Property:

  $(\forall \mathcal{P}_s, \mathcal{Q}_s \in \mathcal{C}_s :: \mathcal{P}_s \not\subset \mathcal{Q}_s)$

  $(\forall \mathcal{P}_d, \mathcal{Q}_d \in \mathcal{C}_d :: \mathcal{P}_d \not\subset \mathcal{Q}_d)$

- Intersection Property:

  $(\forall \mathcal{P}_s \in \mathcal{C}_s, \forall \mathcal{P}_d \in \mathcal{C}_d :: \mathcal{P}_s \bigcap \mathcal{P}_d \neq \emptyset)$

  $(\forall \mathcal{P}_d \in \mathcal{C}_d, \forall \mathcal{P}_s \in \mathcal{C}_s :: \mathcal{P}_d \bigcap \mathcal{P}_s \neq \emptyset)$

To construct a Grid-Quorum, the grid heads belong to the grids in one row are organized into one quorum, called *supply quorum* and the grid heads belong to the grids in a column are organized into one quorum, called *demand quorum*. All the supply quorums compose the supply coterie, and the demand quorums compose the demand coterie. In this way, the natural geographic relation ensures that every supply quorum has intersection with all the demand quorums and vice versa. When a grid has redundant sensors, the grid head propagates this information through the supply quorum to which it belongs. When any grid wants more sensors, the grid head needs only to search its demand quorum. Since every demand quorum has intersection with all supply quorums,

the grid head can get all the information about redundant sensors. We can see that using the geographic information reduces the cost of building Grid-Quorum to almost zero.

Still using the example of Figure 5.1, Grids (0,4), (1,4), (0,3) and (1,3) have redundant sensors, while grid (3,0) needs more sensors. The grid head of (1,3) propagates its redundant sensor information through its supply quorum ((1,4), (1,3), (1,2), (1,1), (1,0)). The grid head in grid (3,0) searches its demand quorum ((0,0), (1,0), (2,0), (3,0), (4,0)). Grid (1,0) can reply the information about redundant sensors. Compared to using the quorum in the last example, using grid-quorum cuts the message by half.

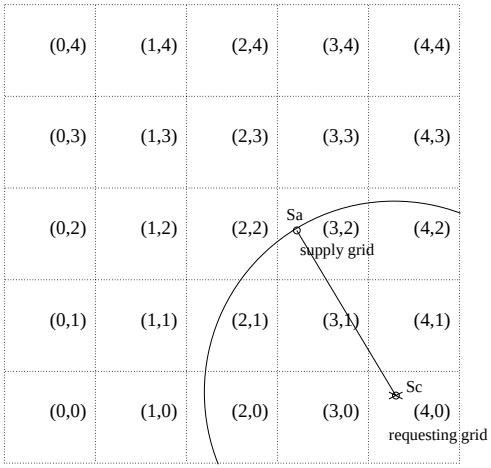
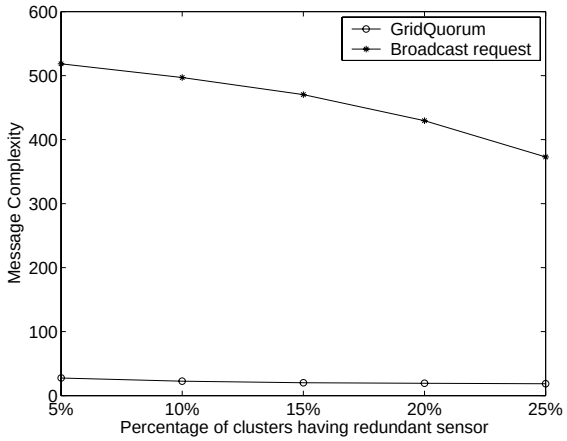


Fig. 5.2. Stopping criteria

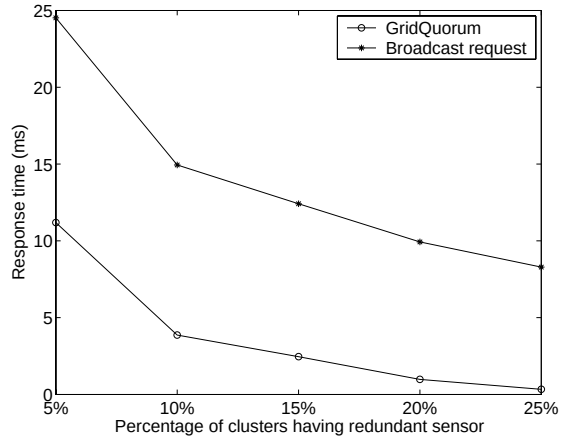
Optimization: To further reduce the message complexity, we add a stopping criteria for propagating request messages through the demand quorum. Since we want the closest redundant sensor, the propagation should be stopped after we already get the

best solution. To implement the idea, the already visited grid head can piggyback the information in its supply quorum before forwarding the query message to the next grid in the same row. Then every grid head in this demand quorum can check whether a better result can be found for the demanding grid. If not, this grid head can reply the query grid immediately without further propagating the request. For example, in Figure 5.2, sensor s_c dies and grid (3,2) has redundant sensor s_a. All sensors located outside the circle with center s_c and distance $d(s_a, s_c)$ must be farther to s_c than s_a. Cluster head of grid (3,0) attaches the location of s_a in the requesting message before forwarding it. When the grid head in grid (2,0) receives the request message, it will not forward it further since no closer redundant sensor can be found.

Figure 5.3 compares the performance between the Grid-Quorum solution and the "Broadcast Request" method in a $10 * 10$ grid. The results were obtained from a simple ns2 simulation; more details on the simulation environment are given later. Cluster heads of neighbor grids communicate through a gateway sensor. Suppose there is no other traffic in the network. From the figures, we can see that the Grid-Quorum solution can significantly reduce the message complexity compared to the "Broadcast request" approach.



(a) Message complexity



(b) Response time

Fig. 5.3. Comparison between the Grid-Quorum solution and the "Broadcast Request" approach

5.4 Sensor Relocation

5.4.1 General Idea: Cascaded Movement

Having obtained the location of the redundant sensor, we need to determine how to move the sensor to the target location (destination). Moving it directly to the destination is a possible solution. However, it may take a longer time than the application requirement. For example, a sensor monitoring a strategic area dies and the application specifies that the maximum tolerable time for such a sensing hole is thirty seconds. If the redundant sensor is 100 meters away and it takes at least one minute for the sensor to reach its destination, the application requirement cannot be met. Moreover, moving a sensor for a long distance consumes too much energy. If the sensor dies shortly after it reaches the destination, this movement is wasted and another sensor has to be found and relocated.

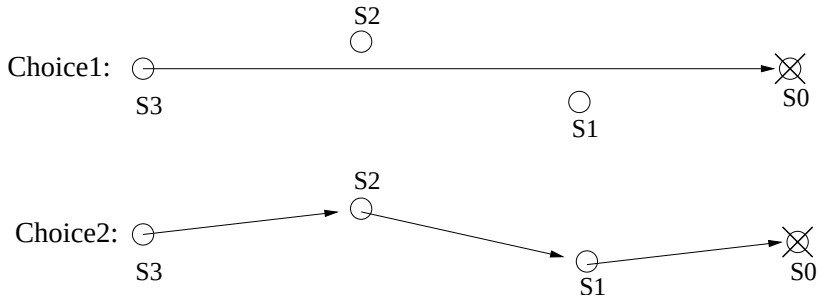


Fig. 5.4. Cascaded movement

We propose to use a *cascaded movement* to address the problem. The idea is to find some cascading (intermediate) nodes, and use them for relocation to reduce the delay and balance the power. As shown in Figure 5.4, instead of letting the redundant sensor s_3 move directly to the destination, s_1 and s_2 are chosen as cascading nodes. As a result, s_3 moves to replace s_2, s_2 moves to replace s_1, and s_1 moves to the destination. Since the sensors can first exchange communication messages (i.e., logically move), and ask all relevant sensors to (physically) move at the same time, the relocation time is much shorter.

A node s_i which moves to replace another node s_j, is referred to as s_j's *successor*, and s_j is referred to as s_i's *predecessor*. In Figure 5.4, s_3 is s_2's successor and s_2 is s_3's predecessor. We also introduce a virtual node s_0, which is used to represent the target location. It may represent the failed sensor or the location where an extra sensor is needed to increase the detection accuracy. In Figure 5.4, we say s_0 is s_1's predecessor and s_1 is s_0's successor.

Selecting cascading nodes is not easy since the sensor nodes may be used by some application and their movement may affect the sensing or communication tasks they are performing. To ensure that this effect is within application's requirement, each sensor s_i is associated with a *recovery delay* T_i. After s_i's movement, its successor must take its place within T_i. T_i is determined by the application based on the critical level of s_i's sensing task, the size of the coverage hole generated by s_i's movement, and other application factors. We use T_0 to represent the recovery delay of the relocation event. It can be the maximum recovery delay of the failed sensor or the time limit for an additional sensor being placed at s_0.

The T value imposes restrictions on the spatial relationship and departure time of the cascading nodes. We use t_i to denote the departure time of s_i and d_{ij} to denote the distance between s_i and s_j. The following Inequality must be satisfied if s_j is s_i's successor.

$$d_{ii+1}/speed - (t_i - t_{i+1}) \leq T_i \tag{5.0}$$

For simplicity, t_i is normalized to be the time period after the relocation request is sent and t_0 (for s_0) is set to be 0.

Based on Inequality (5.4.1), whether s_j can be the the successor of s_i is not determined solely by its distance to s_i, but also s_i's departure time. If s_i moves at t_0 (0), s_j must be within $speed * T_i$ from s_i; if s_i moves after another t minutes, s_j can be farther away from s_i as long as $d(s_i, s_j) \leq speed * (T_i + t)$. Whether s_i can stay at its place for this t minutes or must move immediately is determined by its own predecessor. For example, if s_i is the successor of s_0, and d_{i0} is shorter than $speed * T_0$, s_i can flexibly move between $(0, T_0 - d_{i0}/speed)$. In this case, we normally let s_i move at $T_0 - d_{i0}/speed$ (the upper limit) such that more sensors can be chosen as s_i's successor and we can choose the best one.

The set of cascading nodes for a relocation and their departure time together is defined by a *cascading schedule*. For example, in Figure 5.4, Choice 2, $s_3(t_3) \rightarrow s_2(t_2) \rightarrow s_1(t_1) \rightarrow s_0$ is a cascading schedule, which can be used to recover a sensor failure. Certainly, the cascading scheduling should make sure that the recovery delay is

satisfied; i.e., Inequality (5.4.1) is satisfied. For example, Choice 1 is not a cascading schedule since the sensor failure cannot be recovered within the required time.

5.4.2 The Metrics to Choose Cascading Nodes

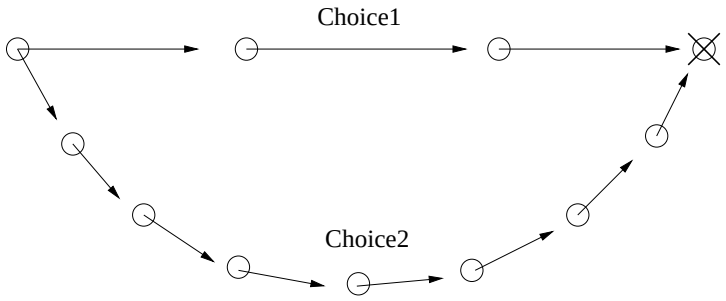


Fig. 5.5. Cascaded movement

The cascading schedule should minimize the total energy consumption and maximize the minimum remaining energy so that no individual sensor is penalized. However, in most cases, these two goals cannot be satisfied at the same time. As shown in Figure 5.5, suppose all sensors have the same amount of power. Choice1 consumes less energy, but the involved sensors will have lower remaining energy. Sensors in Choice2 have higher remaining energy, but the total energy consumption of Choice 2 is higher than that in choice1. There is a tradeoff between minimizing the total energy consumption and maximizing the minimum remaining energy, and we want to find a balance between them.

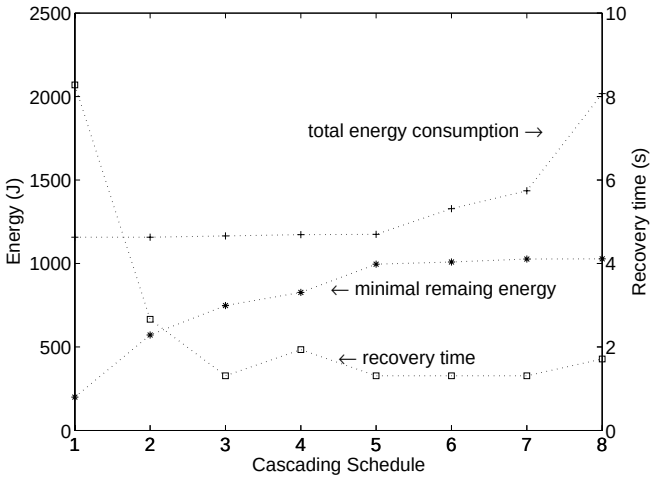


Fig. 5.6. Tradeoff

Before presenting our solution, we first make observations. Based on the sensor deployment result generated by running VOR [41, 39], we randomly choose some sensor and deplete its energy. Then, all cascading schedules to recover the failed sensor are enumerated and compared in terms of the total energy consumption and the minimum remaining energy. Here, the recovery delay ($T_i, i \neq 0$) is relaxed for better observation, but the relocation time (T_0) is calculated for reference. The cascading schedule which is worse than some other schedule in both metrics (total energy consumption and minimum remaining energy) will be ignored; that is, we only keep the cascading schedules which perform better than others at least in terms of one metric.

Figure 5.6 shows the total energy consumption and the minimum remaining energy of these schedules in an increasing order. As shown in the figure, the total energy consumption is almost flat at the beginning and then significantly increased, whereas the minimum remaining energy has a steep increase at the beginning and then becomes

flat. This observation motivates us to achieve a good balance between minimizing the total energy consumption and maximizing the minimum remaining energy.

From the observation, we can see that it is possible to continuously spend a little more energy for a much higher minimum remaining energy until a turning point after which the cost is higher but the gain is less. The cascading schedule just before this turning point should be the best schedule. In other words, the best schedule is the schedule with *the minimum difference between the total energy consumption and the minimum remaining power*. This new metric can be explained in a mathematical way. Suppose there are two cascading schedules with E_1 and E_2 as their total energy consumption, and $Emin_1$ and $Emin_2$ as their minimum remaining energy. Schedule 1 is chosen since $E_1 - Emin_1 \leq E_2 - Emin_2$. This inequality can also be expressed as $E_1 - E_2 \leq Emin1 - Emin_2$; i.e., the cascading schedule with more advantage and less disadvantage should be chosen.

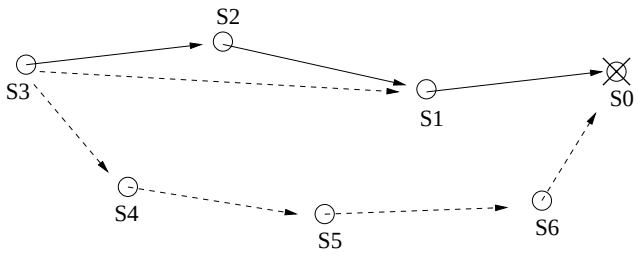


Fig. 5.7. An example

Figure 5.7 uses an example to further explain the reason. In Figure 5.7, moving s_3 directly to the target location is the most energy efficient way. However, in this way, s_3 will be penalized, and its minimum remaining energy will be significantly reduced. If s_1 is added as a cascading node, the load of s_3 can be shared and the minimum remaining energy can be improved. Since the total length of the zigzag line $s_3s_1s_0$ is only a little bit longer than the length of s_3s_0, only a slightly more power is needed. If more sensors close to the line s_3s_0 are chosen as cascading nodes, the load can be further shared and the minimum remaining power can be further improved. Certainly, if some sensor close to this line has very low energy, it should not be selected for cascading. When all eligible sensors close to this line have been chosen as cascading nodes, a balanced and efficient schedule is obtained. Starting from this point, if we want to further improve the minimum remaining energy, faraway sensors such as s_4, s_5 and s_6, have to be chosen. However, in this way, the total energy consumption will be significantly increased, and then it may not be a good solution.

In the remainder of the chapter, the cascading schedule with minimum difference between the total energy consumption and minimum remaining power is referred to as the *best* cascading schedule. The cascading schedule with the least total energy consumption is referred to as the *shortest* schedule.

5.4.3 The Algorithm

Before presenting the algorithm for calculating the best cascading schedule, we first introduce some notations, and describe how to modify Dijkstra's algorithm to calculate the shortest cascading schedule.

The sensor network can be modeled as a complete weighted graph $G(V, E)$, where vertices correspond to the sensor nodes. There are edges between any pair of nodes, and the weight of edge $s_i s_j$ is the distance between s_i and s_j. The remaining power of s_i before relocation and after relocation is denoted by P_i and P_i' separately. We represent the energy as the distance that the sensor can move with this energy. In this way, if s_i moves to s_j in the relocation, $P_i' = P_i - d_{ij}$.

To calculate the shortest cascading schedule, we cannot simply apply Dijkstra's algorithm due to the constraint of the recovery delay. Unlike the shortest path problem, not all the edges with a finite weight in the graph can be selected as a segment of the path. Also, we cannot set the weight of an edge to be infinity because this is not just determined by its length and there is no way to determine if two nodes can be successor and predecessor of each other. To solve the problem, we add a *DeleteEdge* operation to Dijkstra's algorithm to guarantee the delay constraint. *DeleteEdge(s_i)* deletes edge $s_j s_i$ if s_j can not become the successor of s_i; i.e., $d_{ij} \geq (T_i + t_i) * speed$. The new algorithm, as shown in Figure 5.8, is referred to as the *Modified Dijkstra's algorithm*.

To calculate the best cascading schedule, we first calculate the shortest cascading schedule and record its total energy consumption E and its minimum remaining energy $Emin$. Then, we delete all the edges $s_i s_j$ if $P_i - d_{ij} \leq Emin$ and a new graph is generated. This process continues and a new shortest cascading schedule is calculated as long as the difference between the total energy consumption and the minimum remaining energy is increased compared to the previously calculated cascading schedule. When the process terminates, the schedule calculated before the last schedule is the best schedule; i.e., the schedule with the smallest difference between the last two schedules. As shown in

ModifiedDijkstra(Graph G(V,E), Vertex s_0)
Initialization: $S = \{s_0\}$, $Q = V$
DeleteEdge(s_0)
while not Empty (Q)
1. Let $\mathcal{F} = \{\langle s_k, s_l\rangle \mid \langle s_k, s_l\rangle \in S \times Q, d_{kl} \leq (T_k + t_k) * speed\}$
2. Find $\langle s_i, s_j\rangle \in \mathcal{F}$ such that $\forall \langle s_k, s_l\rangle \in \mathcal{F}, d_{ij} \leq d_{kl}$
3. s_j.predecessor $= s_i$
4. $t_j = T_i + t_i - d_{ij}/speed$
5. $P'_j = P_j - d_{ij}$
6. Add s_j to S
7. DeletedEdge(s_j)

end

Fig. 5.8. Modified Dijkstra's algorithm

Figure 5.6, schedule 1 is calculated first, and then 2, ..., 5 and 6. The energy difference of schedule 6 is larger than schedule 5. Thus, the algorithm stops and schedule 5 is chosen as the best cascading schedule. Figure 5.9 shows the formal description of the algorithm.

5.4.4 Distributed Protocol

In this section, we describe how to implement the algorithm presented above in a distributed way. We first present a distributed protocol to calculate the shortest cascading schedule and then describe how to use it to get the best cascading schedule.

To calculate the shortest cascading schedule, the grid head of s_0 initiates a dynamic programming computation by broadcasting a request message, which includes T_0, t_0, the redundant sensor s_r, E_0, and $Emin_0$, where E_0 and t_0 are set to 0, and $Emin_0$ is set to infinity. A node s_i receiving the request first determines if it can be the successor of the sender s_j. If the answer is yes, it sets $E_i = d_{ij} + E_j$, $Emin_i = min(P_i - d_{ij}, Emin_j)$,

Initialization: $E = 0, Emin = -2, E' = 0, Emin' = -1$
while (1)
1. find the shortest cascading schedule using the Modified Dijkstra's algorithm
2. record the minimum remaining power as $Emin'$
3. delete all edges $s_i s_j$ if $P_i - d_{ij} \leq Emin'$
4. **if** $E' - Emin' < E - Emin$ **then**
   $E = E', Emin = Emin'$
   else
   return the previously calculated schedule

Fig. 5.9. Algorithm to calculate the best cascading schedule

and $t_i = T_j + t_j - d_{ij}/speed$. Then, it rebroadcasts T_i, t_i, s_r, E_i and $Emin_i$, and remembers its predecessor s_j. If a node s_j receives several such messages, it will choose the one from s_k which can minimize E_j, which is the energy consumption of the shortest cascading schedule from s_j to s_0. Then, s_j calculates the other fields of the message, broadcasts the message and sets its predecessor to be s_k. Finally, when the request arrives at the redundant sensor s_r, the distributed calculation terminates.

To make the broadcast-based protocol work, we have to address one issue: how can a node determine it has received the message which can minimize the total energy consumption before broadcasting it? There are two intuitive solutions. In the first method, when a node receives a request, it calculates those fields. If E is lower than that calculated from the previously received message, it rebroadcasts the updated version. This method has relatively high message complexity since each node may broadcast several times. In another method, each node waits for a period of time before broadcasting the message. However, if the time threshold is low, there may not be enough information to decide the lowest E value; if the time threshold is high, the delay may be increased.

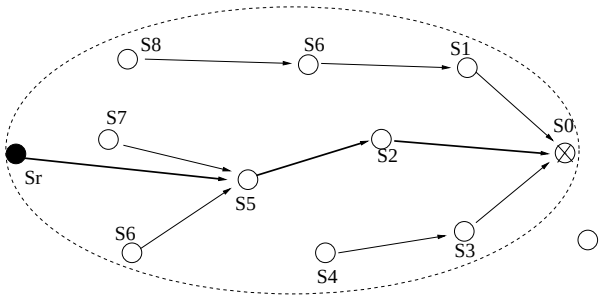


Fig. 5.10. The distributed protocol

We propose to use the geographic information to ensure that nodes make correct decisions before broadcasting with a high probability. As a result, in most cases, each sensor only broadcasts once. Our solution needs two data structures: the *primary search area* and the *waiting list*. The primary search area is determined based on the location of the redundant sensor and the event, and it should have a high probability to encompass all the cascading nodes. The primary search area in Figure 5.10 is elliptic. It can also be other shapes like rectangle. Recall that the cascading nodes should be close to the line connecting the redundant sensor and the event location. Therefore, it is not necessary to involve nodes faraway. Each sensor node determines a *waiting list* after it receives the relocation request for the first time. Only after receiving all the messages from the nodes in the waiting list, a sensor can broadcast the message. The waiting list of s_i includes all the neighbors of s_i which are within the primary search area and are further away from the redundant sensor than s_i. As shown in Figure 5.10, s_2's waiting list includes s_1 and s_3. It only broadcasts the message after receiving the message of s_1 and s_3.

Another issue of this broadcast-based protocol is that the potential successor node may be out of the communication range of the sender. Due to the limitation of the communication range, the successor may not be a communication neighbor. For example, in Figure 5.10, the redundant sensor s_r should be the successor of s_5, but they are not within each other's communication range. s_r may not know the existence of s_5 and can not get the shortest schedule. To address this problem, s_7 piggybacks s_5's message when broadcasting its own message if it finds that s_r may use it. In general, if s_i receives some message from s_k, it will check whether other node may need it. If so, it piggybacks T_k, t_k, E_k, and $Emin_k$ in its message. The formal description of the distributed calculation of the shortest cascading schedule is shown in Figure 5.11.

To calculate the best cascading schedule, we only need to execute the distributed calculation of the shortest schedule iteratively similar to the algorithm shown in Figure 5.9 after the following modifications: (1) The minimum remaining energy calculated in the previous iteration, $Emin'$ is attached. When s_i receives message from s_j, it will check whether $P_i - d_{ij} \leq Emin'$. If yes, s_i and s_j cannot be the successor and predecessor of each other. This change is similar to step 3 in Figure 5.9. (2) In addition to the current predecessor, each node also needs to record the predecessor in the previous iteration.

5.4.5 Optimization: Prepositioning

In some situations, the recovery delay constraint may result in a bad relocation schedule in terms of energy efficiency. Figure 5.5 shows such an example. If the recovery delay is too strict, we have to use choice2. Also, letting each cascading node move synchronously may result in temporary coverage holes though these holes may be within

Notations:
$listw_i$: waiting list of s_i
$listo_i$: list of nodes from which s_i has received messages
loc_i: location of s_i
M_j: $\langle t_j, T_j, loc_j, E_j, Emin_j \rangle$
$listm_i$: list of M

At cluster head
Upon receiving the information of redundant sensor s_r
1. add $M_0\langle t_0, T_0, loc_0, 0, 0\rangle$ to $listm_0$
2. broadcast $request\langle listm_0, s_r, loc_r, loc_0\rangle$

At node s_i
Upon receiving $request\langle listm_j, s_r, loc_r, loc_0\rangle$ from s_j
1. **if** receive such message for the first time **then**
   determine $listw_i$, $E_i = \infty$
2. add s_j to $listo_i$ if it is in $listw_i$
3. **for** each $M_k\langle t_k, T_k, loc_k, E_k, Emin_k\rangle$ in $listm_j$
   3.1 $d_{ik} = distance(loc_i, loc_k)$
   3.2 **if** $(d_{ik} \le (T_k + t_k) * speed)$ **then**
   add M_k to $listm_i$
   if $(E_i \ge E_k + d_{ik})$ **then**
   $E_i = E_k + d_{ik}$
   $Emin_i = min\{Emin_k, P_i - d_{ik}\}$
   $t_i = T_k + t_k - d_{ik}/speed$
   s_i.predecessor $= s_k$
4. **if** $(listw_i = listo_i)$ **then**
   4.1 add $M_i\langle t_i, T_i, loc_i, E_i, Emin_i\rangle$ to $listm_i$
   4.2 broadcast $request\langle listm_i, s_r, loc_r, loc_0\rangle$

At redundant sensor s_r
Upon receiving $request\langle listm_j, s_r\rangle$ from s_j
1. **if** receive such message for the first time **then**
   determine $listw_r$, $E = \infty$
2. add s_j to $listo_r$ if it is in $listw_r$
3. **for** each $M_k\langle t_k, T_k, loc_k, E_k, Emin_k\rangle$ in $listm_j$
   3.1 $d_{rk} = distance(loc_r, loc_k)$
   3.2 **if** $(d_{rk} \le (T_k + t_k) * speed \wedge E \ge E_k + d_{rk})$
   $E = E_k + d_{rk}$
   $Emin = min\{Emin_k, P_r - d_{rk}\}$
   s_r.predecessor $= s_k$
4. **if** $(listw_r = listo_r)$ **then**
   return the shortest cascading schedule

Fig. 5.11. Distributed calculation of the shortest schedule

the application's tolerance. These problems can be addressed if the relocation request can be predicted. For example, if the grid head detects that a group member has very low remaining energy, it can initiate a relocation request to the redundant sensor. In this way, a best relocation schedule in terms of energy efficiency can be found, and the recovery relay constraint can be relaxed. If time permits, the redundant sensor can first move to its predecessor node. After it arrives there, its predecessor starts to move. In this way, no temporary coverage hole will be introduced.

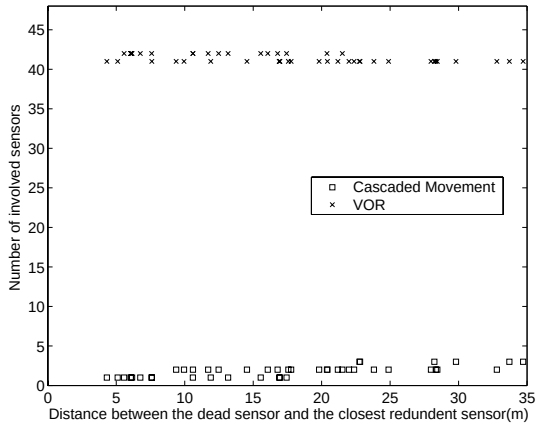
5.5 Performance Evaluation

The evaluation includes three parts: First, the effectiveness of the proposed solution is compared with the VOR scheme Chapter 2. Second, the effectiveness of the cascaded movement is evaluated. Finally, the effectiveness of the metric for choosing the cascading schedule is evaluated.

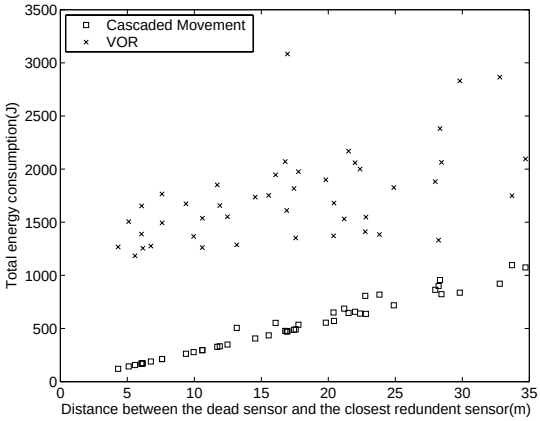
5.5.1 Effectiveness of our sensor relocation solution

The sensor relocation solution presented in this chapter first searches the closest redundant sensor and then relocates it to the target location. Other solutions can be based on the idea of relocating the nearby sensor to the target location. The representative scheme is VOR 2. VOR runs round by round. In each round, it detects the coverage hole and moves nearby sensors to heal it. This process continues until the target field is well covered.

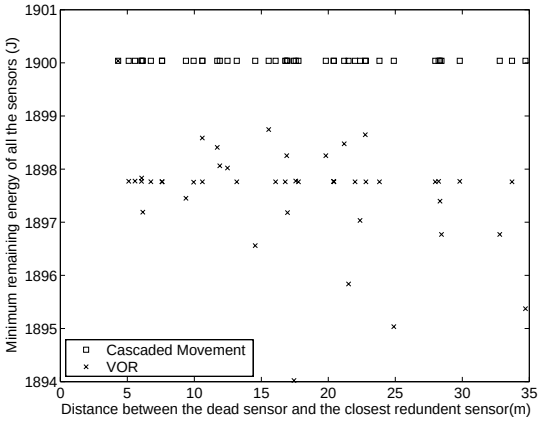
We use sensor failure recovery as an example to evaluate the proposed solution and the VOR scheme. The simulation environment is set as follows. 48 sensors are



(a) Number of sensors moved



(b) Total energy consumption



(c) Minimum remaining energy

Fig. 5.12. Comparison between our solution and VOR

randomly deployed in a $60m * 60m$ target field. VOR is used to deploy them into a well-covered sensor network. The speed of the mobile sensor is $2m/s$ and the recovery delay is $10s$. The energy consumption per meter is $27.96J$, which is calculated based on a mobile sensor prototype [78]. Each sensor is randomly assigned a remaining energy between $1900J$ and $2000J$. We randomly choose a sensor, deplete its energy, and generate a coverage hole. Then, the proposed sensor relocation and VOR are executed to recover the sensor failure. We measure the performance of both approaches by three metrics: the number of sensors moved, the total energy consumption, and the minimum remaining energy.

Simulation results are shown in Figure 5.12. Since the distance between the failed sensor and the redundant sensor directly affect the energy consumption, we use the distance as the x-axis. From the figure, we can see that our solution outperforms VOR in all three metrics. In VOR, nearby sensors move to heal the coverage hole. Since their movement results in new holes, more and more sensors are involved. The propagation of the sensor movement is not directed to the redundant sensor, but to all directions, resulting in moving oscillation. From Figure 5.12(a), we can see that many sensors are involved in the relocation although the redundant sensor is nearby. As a result, the total energy consumption is very high (see Figure 5.12(b)) since a lot of movement is wasted. Also, in VOR, when a coverage hole is detected, nearby sensors are moved even when their remaining energy is low. This results in a much lower minimum remaining energy as shown in Figure 5.12(c).

The simulation results verified the advantage of our solution, where only a minimum number of nodes are involved and many other sensors are not affected. The energy

consumption is low and the remaining energy is high. In summary, although VOR is effective in deploying mobile sensors, first finding the redundant sensor and then relocating it to the target location is much better for sensor relocation.

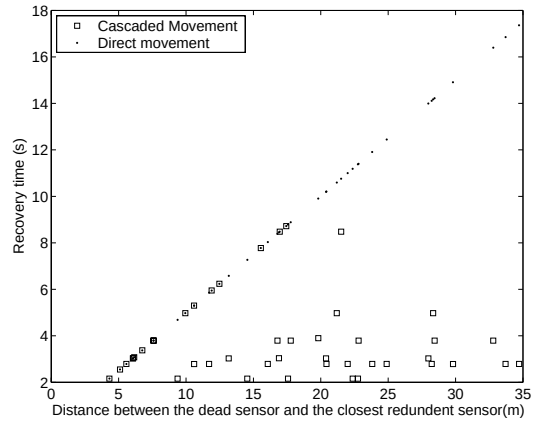
5.5.2 Cascaded Movement vs Direct Movement

In this section, with the same simulation setup, we compare the cascaded movement approach to the direct movement approach, which moves the redundant sensor directly to the target location.

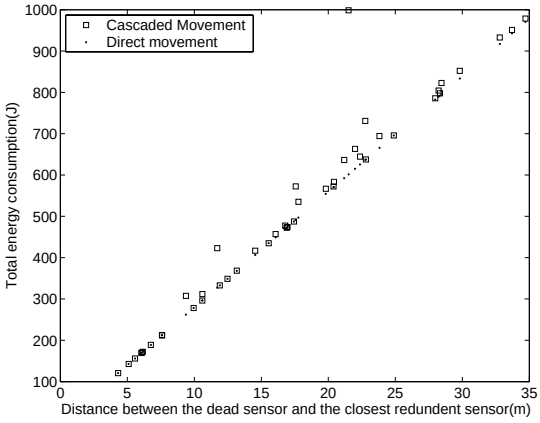
Simulation results are shown in Figure 5.13. As can be seen (Figure 5.13(a)), the relocation time can be significantly reduced in the cascaded movement approach. As for energy consumption, direct movement is better, but its its advantage over cascaded movement is very limited (Figure 5.13(b)). This proves that cascaded movement is energy efficient. On the other hand, the minimum remaining energy of using cascaded movement is much better than that of direct movement. If the redundant sensor has relatively high power, moving it directly to the target location may not affect the minimum remaining power; otherwise, it may significantly reduce the minimum remaining power, especially when the moving distance is long. This explains why the minimum remaining energy drops proportionally as the distance increases in the direct movement approach (see Figure 5.13(c)).

5.5.3 The metric to choose the cascading schedule

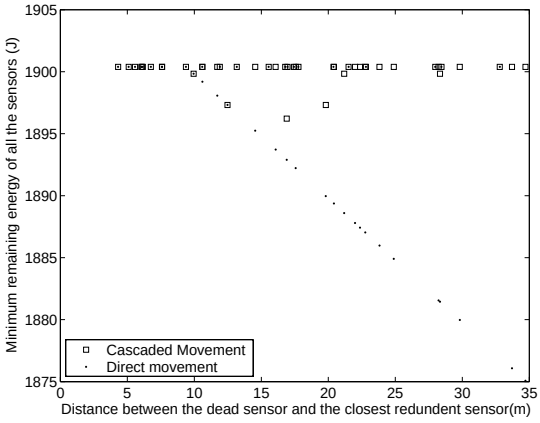
In our solution, the metric used to get the best cascading schedule is to minimize the difference between the total energy consumption and the minimum remaining power.



(a) Relocation time

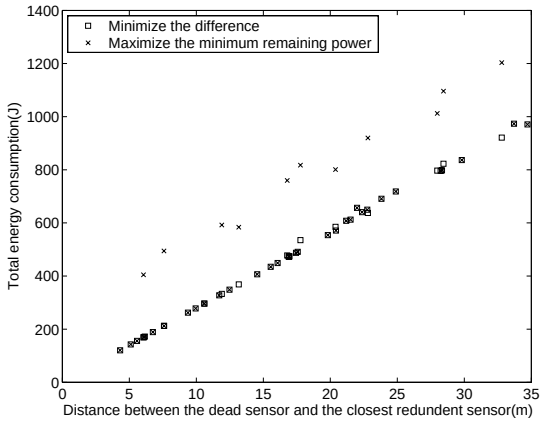


(b) Total energy consumption

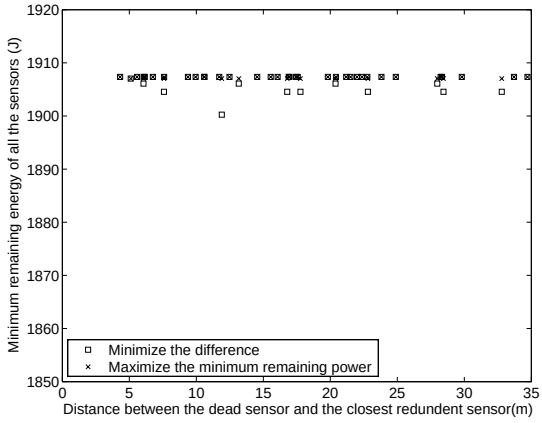


(c) Minimum remaining energy

Fig. 5.13. Comparison between cascaded movement and direct movement

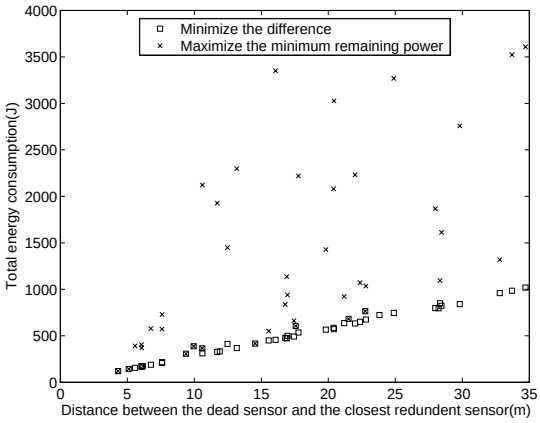


(a) Total energy consumption

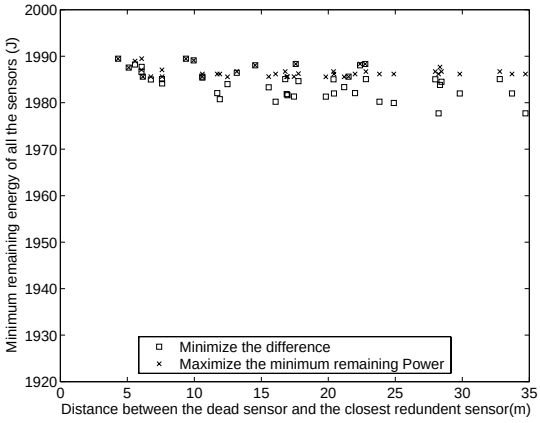


(b) Minimum remaining energy

Fig. 5.14. Comparisons when the remaining energy is very different



(a) Total energy consumption



(b) Minimum remaining energy

Fig. 5.15. Comparisons when the remaining energy is similar

Since there are other objective metrics existing, we compare our solution to other alternatives. Since we have shown (see Figure 5.13(b)) that the total energy consumption of our approach is similar to the direct movement approach, which is optimal, we only compare our approach with another alternative that maximizes the minimum remaining power.

The simulation setup is the same as before except for the energy level before relocation. Two cases are considered. One is that sensors have similar remaining power, which is randomly distributed between $1990J$ and $2000J$. The maximum difference of the remaining power among sensors is $10J$ and the variance is about 11.8. In the other case, the remaining power is very different. The sensor remaining power is randomly distributed between $1900J$ and $2000J$. The maximum difference of the remaining power among sensors is $100J$ and and the variance is about 855.

As shown in Figure 5.14 and Figure 5.15, for both settings, our solution has much lower total energy consumption compared to the approach of maximizing the minimum remaining energy. In the extreme case, our solution saves about $2000J$. Meanwhile, the minimum remaining energy is at most $10J$ lower than its alternative.

Between these two settings, our approach saves more energy when the remaining energy is similar. The reason is as follows. Sensors with relatively more energy must be involved to maximize the minimum remaining energy. When the remaining energy is similar, it is not likely to find nearby sensors with high energy. Then, faraway sensors are more likely to be involved, and more energy will be consumed compared to our solution. On the other hand, when the remaining energy is similar, the disadvantage of our solution is a little bit larger since only nearby sensors are involved in the relocation.

These sensors may become the sensors with minimum remaining energy after relocation and the minimum remaining energy among all the sensors is reduced consequently.

When the remaining energy is very different, both approaches have similar minimum remaining energy since a sensor with minimum remaining energy is more likely not involved in the relocation and the minimum remaining energy of the network does not change after the relocation.

5.6 Conclusions

In this chapter, we defined the problem of sensor relocation, which can be used to deal with sensor failure or response to new events. To effectively relocate sensors and minimize the effect on the application, we proposed a two-phase sensor relocation solution: redundant sensors are first identified and then relocated to the target location. We proposed a Grid-Quorum solution to quickly locate the closest redundant sensor with low message complexity, and proposed to use cascaded movement to relocate the redundant sensor. Since the sensors can first exchange communication messages (i.e., logically move), and ask all relevant sensors to (physically) move at the same time, the cascaded movement solution can significantly reduce the relocation time. Further, a distributed protocol has been proposed to find the best cascading schedule to minimize the difference between the total energy consumption and the minimum remaining power. Simulation results verified that the proposed solution outperforms others in terms of relocation time, total energy consumption, and minimum remaining energy.

Chapter 6

Pair-wise Key Establishment for Wireless Sensor Networks with Key Migration

6.1 Introduction

Secure operation is a basic requirement of many applications (e.g. military surveillance) of wireless sensor networks. However, the open channel of wireless communication and the typically unattended working environment make sensor networks vulnerable to various attacks such as eavesdropping, and node capture and compromise. It is important for sensor networks to provide a secure communication infrastructure and the most fundamental component of such infrastructure is that neighboring nodes share a pairwise key to secure their node-to-node communications. The establishment of pairwise keys for wireless sensor networks is the topic of this chapter.

Establishing pairwise keys for wireless sensor networks is challenging due to the following reasons. First, sensor nodes have very limited computation and storage capabilities, which makes it infeasible to use public-key cryptosystem to bootstrap the pairwise key establishment, as in ad hoc networks [88, 54, 52, 67]. Second, for many applications, sensor networks are deployed in unattended fields. In such environments, sensor nodes are vulnerable to node capture, and after a node is captured, all the secret information stored in it can be read out and disclosed. Certainly, adding tamper-resistance to

sensor nodes can protect the security information. However, due to the low-cost requirements, sensor nodes usually do not have tamper-resistance. Therefore, a pairwise key establishment scheme must deal with node compromise. Third, manual deployment is impossible in many applications and there is no *a priori* knowledge about sensor distribution. Without such knowledge, we cannot predict which pair of nodes will be neighbors after deployment, and thus no deterministic direction exists for preloading keys. Fourth, we can not rely on an on-line KDC (Key Distribution Center) to help in key establishment due to high overhead for maintaining such a server and the problem of single point of failure in such a configuration.

Considering the above issues, the following criteria can be used to judge whether a pairwise key establishment scheme is effective and practical for sensor networks:

- *Resilience to node compromise:* When some nodes have been compromised, the pairwise keys shared by non-compromised node pairs should be revealed as little as possible. Ideally, any such key should not be revealed to an adversary, no matter how many nodes and which nodes have been compromised.

- *Probability for key establishment:*

  For any two neighboring nodes, the probability for them to be able to establish a pairwise key should be as high as possible; a probability of 100% is desired. This feature affects the efficiency of the communications that must be protected by pairwise keys. For example, if two nodes cannot set up a pairwise key, a message to be sent securely from one node to the other may have to travel a multi-hop path, of which each link is protected by a key.

- *Probability for direct key establishment:*

  A secure pairwise key establishment should not involve any third party node. If a third party node is involved, the established key is exposed to the node, and thus the adversary may be able to obtain the key after compromising the node. In addition, indirect key establishment relies on the existence of appropriate third party nodes, which may not be available all the time.

- *Node-to-node authentication:*

  Authenticity of node identity is a fundamental security requirement. Ideally, any two nodes can successfully set up a pairwise key only when their identity can be verified by each other.

- *Storage requirements:*

  The size of the memory and storage in sensor nodes is very limited. This capacity must accommodate various application code, data, etc. Therefore, the memory requirement for pairwise key establishment should be small.

- *Computation and communication efficiency:*

  A pairwise key establishment scheme should be efficient in terms of both communication and computation.

- *Scalability:*

  In many applications of sensor networks, a large network scale is required. Therefore, a pairwise key establishment scheme must be applicable to such environments.

There have been many works on establishing pairwise keys in a distributed fashion based on randomized key pre-distribution [28, 17, 71, 59, 26, 81]. A representative approach proposed by Eschenauer and Gligor [28] works as follows: each node receives a random subset of keys from a large key pool before being deployed; two nodes find one common key from their subsets, and use it as their pairwise key. However, the network topology after deployment can not be accurately estimated when the keys are preloaded. Therefore, it is quite possible that some neighboring node-pairs cannot find a common key, and thus are unable to set up a pairwise key directly. In addition, after a number of nodes are compromised (and thus the keys held by these nodes are compromised) the pairwise keys shared by some non-compromised nodes may also be exposed. The works [17, 71, 59, 26] following this scheme aim to increase the resiliency to node compromise or increase the probability of two nodes establishing a key by different ways.

For example, Chan, *et al.*, propose that two nodes can only establish a pairwise key if they share more than a threshold number of keys. However, these schemes still are not resilient to large number of node compromise. To meet certain requirements on node resiliency, the supportable network size has to be small. Moreover, the probability for any two nodes to set up a pairwise key directly is still lower than desirable, especially when the network has a large scale. Although two nodes failing in direct key establishment may set up a key with the help of other nodes, this is not secure since the established key is exposed to other nodes.

In this chapter, we propose a novel pairwise key establishment scheme for static sensor networks. The key establishment protocol supports the addition of new nodes during network operation. Compared with existing schemes, our pairwise key establishment scheme has the following advantages:

- *Perfect resilience to node compromise.*

  All the links between non-compromised nodes are still safe no matter what percentage and what subsets of nodes are compromised.

- *Full connectivity with direct key establishments and supporting large networks.*

  When applied to Berkeley Mica Motes [20], our schemes can support $64,000$ nodes and full connectivity at the same time. That is, we guarantee that any pair of nodes can setup a pairwise key directly without involving a third party. For European Eyes Motes [29], we can support even more nodes with full connectivity.

- *Low storage requirements.*

  Our scheme has a very low storage requirement after the key establishment phase. In particular, each node only needs to store the established keys used for communication and one additional key for the addition of new nodes.

- *Node authentication.*

  With our scheme, if and only if two nodes can set up a key, they can ascertain the identity of each other.

- *Highly Efficient.*

  Our scheme is efficient in terms of both communication and computation.

As far as we know, this is the first work which can satisfy the above attributes simultaneously.

The rest of this chapter is organized as follows. Section 6.2 introduces the system model. Section 6.3 gives details of our schemes and their analysis. We evaluate its performance in section 6.5. Section 6.6 reviews existing techniques related to pairwise key establishment. Finally, section 6.7 concludes the chapter.

6.2 System Model

We consider a wireless sensor network which is composed of a large number of sensor nodes, e.g., the Berkeley Mica Mote [20]. These nodes have limited power supply, storage space, and computation capability. Therefore, public key-based operations cannot be afforded. On the other hand, each node has a certain amount of storage for data. This is a reasonable assumption: all the current sensor prototypes have a *measurement flash* for storing sensing data. We also assume that sensor nodes are indexed by a number starting from from 0 to $n-1$ (n is the total number of nodes in the network).

There is an off-line Key Distribution Center (KDC) for the network. The KDC is responsible for generating keys and preloading nodes with security information before the sensors are deployed.

6.3 New Pair-wise Key Establishment Schemes

In this section, we present our schemes for pairwise key establishment. As with the existing key predistribution schemes, in our schemes, nodes are preloaded with security information, e.g., keys, before being deployed by the off-line KDC; after deployment,

nodes communicate with each other to setup pairwise keys. Our schemes differ from existing schemes in the following ways:

Exclusive key ring. The preloaded key ring of each node is exclusive to one another. After the key set-up phase, nodes only have keys used to encrypt their communications with neighbors, and the keys are not shared with any other nodes. In this way, capturing any number of nodes will only affect communication with their neighbors, which share keys with them. By preloading exclusive key ring, our schemes can achieve close to 100% resiliency to node compromise.

Decrypting key and Encrypted keys. Since the preloaded key rings are exclusive to each other, nodes do not share common keys. To reach a key agreement between two neighboring nodes, we propose that nodes send encrypted versions of their keys to their neighbors. To enable a message receiver to decrypt the message and retrieve the key, we use a unique key in the receiver to encrypt the key of the sender. In detail, each node is preloaded with a *decrypting key*, in addition to the aforementioned exclusive key ring (each key in this key ring is called an *explicit* key in the remainder of this chapter). Also, each node is preloaded with an encrypted version of their explicit keys, which are encrypted by the decrypting keys of other nodes. Ideally, when two nodes s_0 and s_1 want to set up a common key, node s_0 sends to s_1 one of its encrypted keys, and s_1 uses its decrypting key to decrypt the encrypted key from s_0. More details for this process will be described in our schemes.

Data storage. After the key setup phase, only the established pairwise keys and the decrypting key are stored. The decrypting key will be used for further node additions. All the other security information, such as the encrypted keys and those unused explicit

keys can be deleted to save the storage. In this way, we can utilize the *data storage* in a sensor node. Previous schemes can not utilize data memory since they can not delete any key considering future node additions. Our schemes will delete the keys not used after key setup phase, so we can utilize this storage before the network is put into use. We can preload much more security information by using this memory and thus support a very large network.

In the following sections, we first present a basic scheme. Then we present an advanced scheme using key chain, by which the supportable network size can be nearly doubled. After that, we provide some discussion on the tradeoff between the probability of establishing pairwise key and supportable network size.

6.3.1 Basic Scheme

The basic scheme has three steps: key generation and key pre-distribution, which are off-line procedures; pairwise key setup, which is performed after nodes are deployed; and post-deployment operation, which is performed after nodes set up pairwise keys with each other. Before presenting the details of this scheme, we introduce the following notations:

- n: network size
- m: size of the key ring
- k_{D_i}: decrypting key of node s_i.
- $k_{i,j}$: the key shared by s_i and s_j. Note that $k_{i,j}$ is the same as $k_{j,i}$
- $k_i[k_j]$: result of encrypting k_j by k_i

6.3.1.1 Key Generation and Key Distribution

To guarantee that any pair of nodes can establish a pairwise key in a network with n nodes, at least $n * (n-1)/2$ keys are needed. To evenly distribute these keys among nodes, each node should be preloaded with $(n-1)/2$ keys. The off-line KDC randomly generates $n * (n-1)/2$ keys and pre-loads each node with a key ring of size $m = (n-1)/2$. Note that the key rings are exclusive to one another. When n is even, $m = (n-1)/2$ is not an integer. We let $m = \lfloor (n-1)/2 \rfloor$ for node s_i if i is odd, and $m = \lceil (n-1)/2 \rceil$ if i is even.

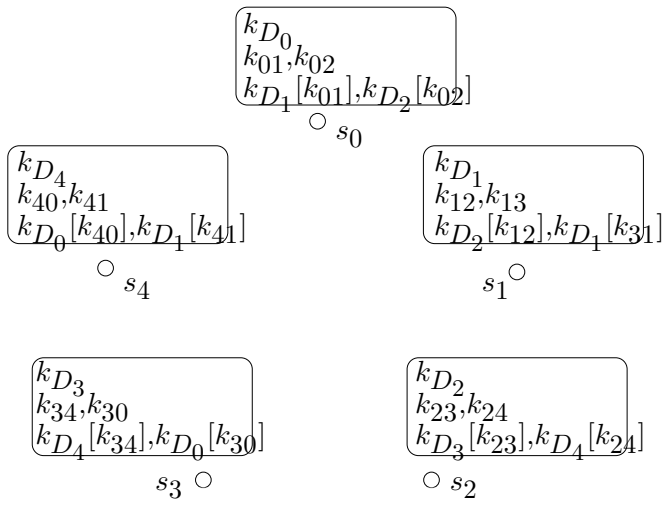


Fig. 6.1. Key Pre-distribution

The m keys in a node s_i will be used as the pairwise keys between s_i and the nodes with IDs $(i+1)$ mod $n, \ldots, (i+m)$ mod n. In this way, any two nodes s_i and s_j

can find their pairwise key in either s_i or s_j. Without loss of generality, we let $i > j$. Then $k_{i,j}$ is preloaded into s_i if $i - j < m$ and it is preloaded into s_j otherwise.

We use one example to illustrate the allocation of pairwise keys. As shown in Figure 6.1, the network has five nodes: s_0, s_1, s_2, s_3, and s_4. According to our key allocation method, s_0 has k_{01} and k_{02}; s_1 has k_{12} and k_{13}; s_2 has k_{23} and k_{24}; s_3 has k_{34} and k_{30}; and s_4 has k_{40} and k_{41}. Thus, the link between any pair of nodes can be secured by a unique key.

For $k_{i,j}$ preloaded into s_i, the off-line KDC uses k_{D_j} to encrypt $k_{i,j}$ and loads $k_{D_j}[k_{i,j}]$ into s_i.

In Summary, a node s_i is preloaded with the following information:

k_{D_i}

$\{k_{i,j} | j = (i+1) \bmod n, \ldots, (i+m) \bmod n\}$

$\{k_{D_j}[k_{i,j}] | j = (i+1) \bmod n, \ldots, (i+m) \bmod n\}$

6.3.1.2 Pair-wise Key Setup

After nodes are deployed into the target field, they will broadcast their IDs and meanwhile, get a neighbor list. For each neighbor s_j, s_i will first check whether $j \in \{(i+1) \bmod n, \ldots, (i+m) \bmod n\}$. If so, s_i has $k_{i,j}$. It sends $k_{D_j}[k_{i,j}]$ to s_j. After receiving $k_{D_j}[k_{i,j}]$, s_j can use its decrypting key k_{D_j} to decrypt it and get $k_{i,j}$. Then s_i and s_j can use $k_{i,j}$ to communicate with each other.

If s_i does not have $k_{i,j}$, then s_j must have it. s_i will wait for s_j to send it $k_{D_i}[k_{i,j}]$. After receiving $k_{D_i}[k_{i,j}]$, s_i uses its decrypting key k_{D_i} to decrypt the message and retrieve $k_{i,j}$. Let us use an example to illustrate this procedure. As shown in Figure 6.2,

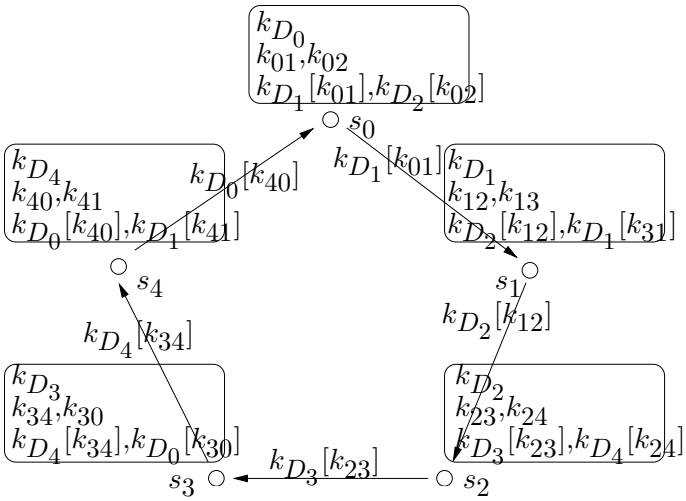


Fig. 6.2. Key Setup

after receiving the *hello* messages from neighbors, nodes know their neighbors. Taking s_0 as an example, s_0 discovers that s_4 and s_1 are its communication neighbors. It has k_{01} and it sends $k_{D_1}[k_{01}]$ to s_1. For the other neighbor s_4, it does not have k_{40}. s_4 sends $k_{D_0}[k_{40}]$ to s_0 and it uses k_{D_4} to decrypt it and get k_{40}. The final secure connection is shown in Figure 6.3 as the dotted lines.

6.3.1.3 Post-Deployment

After establishing pairwise keys with neighbors, nodes delete the preloaded keys to free the storage. They only keep the established pairwise keys and the decrypting keys. As shown in Figure 6.3, after key establishment phase, only one additional key (decrypting key) is kept in the memory other than the pairwise keys.

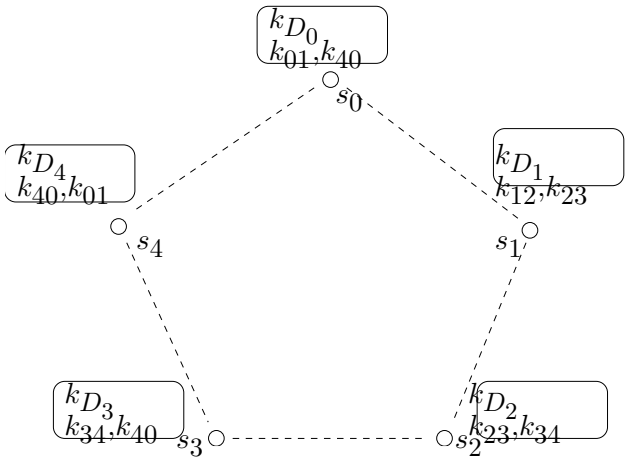


Fig. 6.3. Final Result

6.3.1.4 Node Addition

Suppose the network is currently composed of n nodes, and n' nodes are to be added. To guarantee that any new nodes can establish a pairwise key with any existing node, at least $n' * n$ keys are needed. These keys are to be preloaded into the new nodes; thus each new node is preloaded n with keys. To guarantee that any two new nodes can establish a pairwise key, at least $n' * (n' - 1)/2$ keys are needed totally. To evenly distribute these keys to new nodes, each new node is preloaded with $(n' - 1)/2$ keys. In summary, the off-line KDC generates all these keys, and pre-loads each new node with a key ring of $(n + (n' - 1)/2)$ keys. Note that the key rings of new nodes are exclusive to one another, and are exclusive to the key rings of existing nodes.

Inside the key ring of node s_i, n keys $k_{i,0}, \ldots, k_{i,n-1}$ are used as pairwise keys shared by s_i and existing nodes. $(n' - 1)/2$ (denoted as m') keys are used as pairwise

keys shared by s_i and s_j, where $j \in \{(i+1) \bmod (n+n'), \ldots, (i+m') \bmod (n+n')\}$. In this way, any two new nodes s_i and s_j can find their pairwise key in either s_i or s_j. Without loss of generality, we specify $i > j$. Then $k_{i,j}$ is preloaded into s_i if $i - j < m'$ and it is preloaded into s_j otherwise.

For any key k_{ij} preloaded into a new node s_i, the off-line KDC uses k_{D_j} to encrypt it and also preloads $k_{D_j}[k_{i,j}]$ into s_i.

To illustrate the key preloading of new nodes, we continue with the same example shown in Figure 6.3. Currently, the network has five nodes, $s_0, \ldots, s_4$. Three new nodes, s_5, s_6 and s_7 are to be added, shown in Figure 6.4. For each node s_i, it is preloaded with keys k_{ij}, where $j = 0, \ldots, 4$. These keys will be the pairwise keys shared with existing nodes $s_0, \ldots, s_4$. s_i is also preloaded with keys k_{ij}, where $j = (i+1) \bmod 8 + 5$. For example, s_5 is preloaded with $k_{5,6}$ and s_7 is preloaded with $k_{7,5}$.

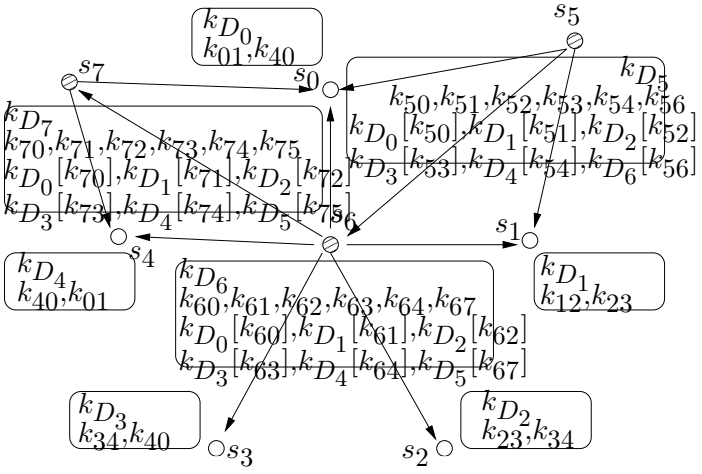


Fig. 6.4. Node Addition

After the new nodes are distributed into the target field, they broadcast their ID. Existing nodes will also broadcast their ID once they hear the messages from the new nodes. The new node learns its neighbors, either existing nodes or new nodes. It sends the encrypted keys directly to the neighbors of existing nodes. For example, s_5 sends $k_{D_0}[k_{50}]$ to s_0 and sends $k_{D_1}[k_{51}]$ to s_1. For the neighbors that are also new nodes, it first checks whether it has the pairwise keys. If so, it sends the encrypted keys to its neighbors; otherwise, it waits for its neighbors to send it the keys. For example, s_5 sends $k_{D_6}[k_{56}]$ to s_6 and waits for s_7 to send it the encrypted k_{75}. After receiving $k_{D_5}[k_{75}]$, s_5 uses k_{D_5} to decrypt it and retrieve k_{75}.

After the key establishment phase, new nodes also delete those pairwise keys shared with nodes that are not their neighbors, and only keep one decrypting key and the pairwise keys that will be used. Figure 6.5 shows the result of node addition. Dotted lines indicate the secure links.

6.3.2 Advanced Scheme

We propose an advanced scheme in order to reduce the amount of preloaded security information, and thus support a larger network. To achieve this goal, we adopt a technique based on a key chain that can compress the key rings. By using this technique, we can shrink the key ring of each node to be only two keys. In general, we use a seed key and a hashing key. All the keys in the key ring can be generated by these two keys. Therefore, each node is only preloaded with these two keys. When individual keys are needed, a sequence of computations are performed to calculate the keys.

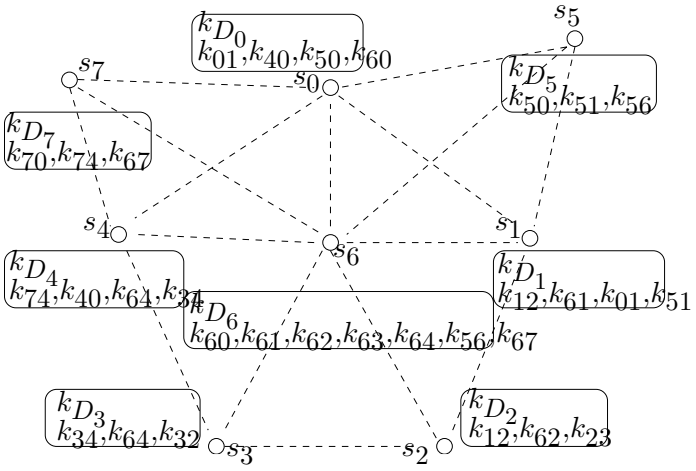


Fig. 6.5. Result of Node Addition

In detail, the off-line KDC generates one seed key and one hashing key for each node. We use k_C to denote the hashing key and k_S to denote the seed key. Suppose the size of the key ring is m. We use $k_1, \ldots, k_m$ to denote these keys. They are calculated in the following way:

$$k_1 = k_S$$

$$k_2 = hash(k_1, k_C)$$

$$\ldots$$

$$k_m = hash(k_{m-1}, k_C)$$

These m keys are used as pairwise keys between the key holder s_i and its possible neighbors in the same way as described in section 6.3.1.1 and section 6.3.1.4. Taking s_5 in Figure 6.4 as an example, $k_{50} = k_{S_5}$; $k_{51} = hash(k_{50}, k_{C_5})$; $\ldots$; $k_{54} = hash(k_{53}, k_{C_5})$; and $k_{56} = hash(k_{54}, k_{C_5})$.

For any key k_{ij} which is generated by the hashing key and seed key of s_i, the off-line KDC uses k_{D_j} to encrypt it and preload $k_{D_j}[k_{ij}]$ into s_i.

After a node is deployed into the target field, it will broadcast its ID and discovers its communication neighbors. For each neighbor s_j, s_i will first check whether it has $k_{D_j}[k_{i,j}]$. If so, it sends the encrypted k_{ij} to s_j for it to decrypt and retrieve k_{ij}. After sending the encrypted keys to its neighbors, it can delete all the encrypted keys and calculate the keys it needs by performing a sequence of hashing functions. Otherwise, it waits for s_j to send it $k_{D_i}[k_{ji}]$. After receiving $k_{D_i}[k_{ji}]$, it will use k_{D_i} to decrypt it and get k_{ji}.

6.3.3 Discussion

In our schemes, each node must store its decrypting key, but the pairwise key shared by each pair of nodes can be stored in either of them. In addition, the encrypted keys can be stored in other nodes. This attribute enables the flexibility in organizing the network. Suppose a heterogeneous network is used as a distributed database [87, 81], some nodes may have a much larger storage for data. These nodes can be preloaded with more encrypted keys. In this way, the capability of nodes can be fully utilized.

6.4 Analysis

In this section, we analyze the attributes of our pairwise key establishment schemes.

6.4.1 Resilience to Node Capture and Compromise

Our schemes have close to perfect resilience to node compromise. In our schemes, each node is only preloaded with a decrypting key and the pairwise keys shared by itself and other nodes. If a node is captured after it is deployed but before it establishes pairwise keys with its neighbors, only this information is disclosed. The decrypting key can only be used to decrypt the pairwise keys associated with it. Therefore, capturing one node during this period does not reveal the pairwise keys shared by other nodes. If a node is captured after the key setup phase, no information about other nodes is disclosed with the same reason.

If a node is captured, an adversary will learn the keys in that node. It can thus eliminate these keys from the list of possible keys in all other nodes since the key rings are exclusive. Thus, the space in which an adversary must guess keys in other nodes is slightly decreased. However, the impact of this decrease in space is virtually neglectable.

With a key length of l bits, the size of key space is 2^l. The possibility of guessing one key successfully is $1/2^l$. If m nodes are captured before the key establishment phase, $m * (n-1)$ keys are disclosed, and the size of the unknown key space is reduced to $2^l - m * (n-1)$. The possibility of guessing one key successfully in the un-compromised nodes increases to $1/(2^l - m*(n-1))$. Because $2^l >> m*(n-1)$, this is almost equal to $1/2^l$ even for very large sensor networks. For example, with a key length of 128 bits, in a network composed of 100,000 nodes, if 4,999,950 keys are compromised, the probability of guessing is the same to at least 55 significant bits (calcualted by Matlab 7.0 for the precision).

6.4.2 Supportable Network Size

In this section, we first calculate the network size when any pair of nodes are guaranteed to establish a pairwise key, and then the case that any pair of nodes has certain probability to setup a key. Suppose the data storage of each node is x $Byte$s and the key length is l $Byte$s. The data storage can accommodate x/l keys. To ease the computation, we assume that decrypting keys, hashing keys and seed keys are preloaded into the configuration EEPROM. (Since data storage will be reset after the network is coming into use and collecting data, all remained keys will be removed from the configuration EEPROM anyway.) All data storage is used to store encrypted keys and explicit keys, which has the same order as the network size.

The supportable network size under full connectivity depends on how nodes are deployed. We consider two extreme cases first: one is that all the nodes are deployed at one time. Another is that nodes are deployed one by one. Then we consider a general case.

6.4.2.1 One-time Deployment

When all nodes are deployed at one time, the supportable network size in the advanced scheme is calculated as follows. The data storage of each node can accommodate x/l keys. Under the advanced scheme, the whole storage is used to store the encrypted keys. The size of key ring is $(n-1)/2$. Therefore, $(n-1)/2 = x/l$, and $n = 2x/l + 1$. The network can support $(2x/l + 1)$ nodes.

In the basic scheme, since both encrypted keys and explicit keys (determined by the network size) have to be stored in the data storage, so $2*(n-1)/2 = x/l$. A network can support $x/l + 1$ nodes.

6.4.2.2 Incremental Deployment

In this case, nodes are deployed one by one. Each time a node is deployed, it needs to send encrypted keys to existing nodes and no existing node has any encrypted keys. To calculate the network size, we consider the last added node. To guarantee the establishment of pairwise keys with all previously deployed nodes, the key ring size should be $(n-1)$. In the advanced scheme, $n-1 = x/l$. A network can support $(x/l+1)$ nodes. In the basic scheme, $2 * (n-1) = x/l$ and a network can support $(x/(2l) + 1)$ nodes.

6.4.2.3 General Case

Consider that currently the network has $n1$ nodes, and $n2$ nodes are to be added. These $n2$ nodes will saturate the network. That is, if a new node is added after this node addition, it cannot be guaranteed to establish pairwise keys with existing nodes. The network then can support $n1 + n2$ nodes in total. These $n2$ nodes needs to be preloaded with $n1+(n2-1)/2$ keys. In the advanced scheme, $n1+(n2-1)/2 = x/l$. The supportable network size is $n\ (n1 + n2) = x/l + 1/2 + n2/2$. We can see that, with a larger $n2$, the supportable network size is also larger. In the basic scheme, $n1 + (n2 - 1)/2 = x/(2l)$, and supportable network size is $x/(2l) + 1/2 + n2/2$.

6.4.2.4 Tradeoff Between Efficiency and Network Size

When we need a larger network size than calculated above, the connectivity may be sacrificed. Let p be the probability that two nodes can setup a pairwise key. Considering a general case, in the advanced scheme, $p * (n1 + (n2 - 1)/2) = x/l$, and the supportable network size can be increased to $n = x/(lp) + 1/2 + n2/2$. In the basic scheme, $n = x/(2lp) + 1/2 + n2/2$.

6.4.3 Memory

After the pairwise key setup phase, each node can delete the preloaded keys to free the storage except the decrypting key and the pairwise keys with its neighbors. Suppose a node has n_l neighbors. It needs only $(n_l + 1) * l$ space for storing the security information. The ideal case is that only pairwise keys with communication neighbors are stored in memory. In our schemes, we only require one more key (decrypting key) than the optimal case.

6.4.4 Other Metrics

- *Node Authentication.* Any key is shared and known only to a pair of nodes. Therefore, with a key, the communicating parties must know the identity of each other.

- *Direct Establishment.* Our scheme on pairwise key establishment does not require the involvement of any third party. This is more secure compared to existing key predistribution schemes.

6.5 Evaluation

We have two objectives in conducting this performance evaluation. First, we aim to study the performance our schemes under different sensor prototypes and different existing encryption algorithms. Second, we aim to compare our schemes with previous schemes. We choose supportable network size and storage efficiency as the evaluation metrics. Four popular sensor prototypes are considered: the Berkeley Mica Mote [20], the European Eyes Mote [29], the Intel imote [77], and the TMote Sky [65]. Five widely used encryption algorithms, RC4, IDEA, RC5, MD5, and SHAI, are considered.

6.5.1 Supportable Network Size

Table 6.1 and 6.2 show the supportable network size under one-time deployment and incremental deployment under the advanced scheme, respectively. Our scheme outperforms existing pairwise key schemes in terms of supportable network size. For example, Chan, *et al.*, has shown that q-composite scheme [17] can support $1,415$ nodes and Eschenauer and Gligor's scheme [28] can support $1,159$ nodes, when $p = 0.33$ and a parameter f_m is equal to 0.1. f_m measures the fraction of broken links after some nodes are captured. In our scheme, when $p = 0.33$, we can support $192,000$ nodes and we guarantee perfect node resilience, which is about 100 times larger than theirs.

6.5.2 Storage

In TABLE 6.3, we show the memory used by our scheme under different node density and encryption algorithms. The key lengths of the encryption algorithms are cited from [34].

Connectivity	Motes	Measurement Flash	Supportable Network Size
Full connectivity	Mica Motes	$512KB$	$64,001$
	Eyes Motes	$8MB$	$800,001$
	Intel iMote	$512KB$	$64,001$
	TMote Sky	$1024KB$	$128,001$
$p = 0.5$	Mica Motes	$512KB$	$128,001$
	Eyes Motes	$8MB$	$1,600,001$
	Intel iMote	$512KB$	$128,001$
	TMote Sky	$1024KB$	$256,001$
$p = 0.33$	Mica Motes	$512KB$	$192,001$
	Eyes Motes	$8MB$	$2,400,001$
	Intel iMote	$512KB$	$192,001$
	TMote Sky	$1024KB$	$384,001$

Table 6.1. Supportable Network Size under One-time Deployment

Our scheme outperforms existing schemes in terms of storage efficiency. To support a much smaller network, the pairwise key establishment schemes [28, 17] requires 200 keys, and $3,200B$ storage.

In fact, we achieve a near-optimal memory occupation. In addition to keys used for securing the communication, only one additional key is stored in the space.

6.6 Related Work

In this section, we first introduce previous work on pairwise key establishment schemes in sensor networks. Then we survey related key establishment schemes in ad hoc networks.

6.6.1 Pair-wise Key Establishment Schemes in Sensor Networks

In the past years, a variety of pairwise key establishment schemes have been proposed. Most of these schemes fall into the following categories:

Connectivity	Motes	Measurement Flash	Supportable Network Size
Full connectivity	Mica Motes	$512KB$	$32,001$
	Eyes Motes	$8MB$	$200,001$
	Intel iMote	$512KB$	$32,001$
	TMote Sky	$1024KB$	$64,001$
$p = 0.5$	Mica Motes	$512KB$	$64,001$
	Eyes Motes	$8MB$	$800,001$
	Intel iMote	$512KB$	$64,001$
	TMote Sky	$1024KB$	$128,001$
$p = 0.33$	Mica Motes	$512KB$	$96,001$
	Eyes Motes	$8MB$	$1,200,001$
	Intel iMote	$512KB$	$96,001$
	TMote Sky	$1024KB$	$192,001$

Table 6.2. Supportable Network Size under One-by-one Deployment

Centralized Schemes: One instance is SNEP proposed by Perrig *et al.* [70], in which an online base station is used for helping two nodes establish a pairwise key. This scheme is resilient to the capture of any number of ordinary sensor nodes. However, the property is based on the assumption that the online base station will not be compromised, which is difficult to achieve in practice. In addition, any two nodes must go through the base station for key establishment, which makes the base station a performance bottleneck in a large-scale network.

Distributed Schemes: To deal with the limitations inherent in the centralized schemes, a number of distributed schemes have been proposed. In LEAP proposed by Zhu *et al.* [89], all nodes are preloaded with an initial network key before their deployment, and after node deployment, each pair of neighboring nodes derive their pairwise keys based on the initial key. This scheme however requires no node compromise before pairwise

Node density	Encryption Algorithm	Key length	Memory requirements
20	RC4	$128b$	$336B$
	IDEA	$128b$	$336B$
	RC5	$64b$	$168B$
	MD5	$128b$	$336B$
	SHAI	$128b$	$336B$
40	RC4	$128b$	$656B$
	IDEA	$128b$	$656B$
	RC5	$64b$	$328B$
	MD5	$128b$	$656B$
	SHAI	$128b$	$656B$

Table 6.3. Memory requirements (Node density is measured by the average number of communication neighbors of each node)

key establishment completes. Otherwise, the adversary will obtain the initial key and will be able to derive the pairwise key between any pair of nodes.

Considering that nodes could be compromised before pairwise key establishment completes, researchers have proposed schemes based on the ideas of both key predistribution and probabilistic key sharing. Eschenauer and Gligor [28] proposed the first such scheme, in which each node is preloaded with a random subset of keys. Any two nodes, after deployment, can set up a pairwise key if they have one common key preloaded. Since each node is randomly preloaded with a key ring from the key pool, it cannot be guaranteed that any two nodes can find a common key. In this case, they will not be able to set up a pairwise key directly. [28] has also shown that the probability for direct key establishment is affected by the key pool size, key ring size and network size. Specifically, as the key pool size increases, the key ring size decreases, and/or the network size increases, the probability is reduced. When two nodes cannot set up a pairwise key directly, they may still establish a pairwise key indirectly, if there exists a path connecting

them such that any two neighboring nodes on the path share a pairwise key. The indirect key establishment, however, introduce a new security risk: if a node is compromised, the pairwise keys that are established with the help of the node are also compromised.

Eschenauer and Gligor's scheme [28] was enhanced by Chan *et al.*[17], who proposed a q-composite random key pre-distribution scheme and a random pairwise keys scheme. It was further enhanced by Liu and Ning, who proposed a random subset assignment scheme and a grid-based scheme [59] based on Blundo's polynomial key predistribution protocol [13]. Du *et al.* [26] also proposed a similar scheme based on Blom's key predistribution scheme [12]. Traynor *et al.* [81] studied the pairwise key establishment problem in heterogeneous networks in which a subset of nodes are equipped with tamper-resistant hardware. These advanced schemes provide improved connectivity and resiliency. But they still cannot guarantee that any two nodes can set up a pairwise key. Also, they are not resilient to a large number of node compromises.

Location-aware Schemes: Du *et al.* [27], Liu *et al.* [60], and Huang *et al.* [53] proposed location-aware key establishment schemes, in which location information is employed to facilitate sensor nodes in establishing pairwise keys. These schemes can further improve the network connectivity, assuming the knowledge on the possible locations of sensor nodes before they are deployed. However, they cannot provide guarantee on pairwise key establishment, either.

6.6.2 Pair-wise Key Establishment Schemes in Ad hoc Networks

During recent years, a great deal of research has been conducted in key management for ad hoc networks. Zhou and Haas [88] proposed a solution based on threshold

cryptography. In their approach, the certificate authority (CA) service is distributed over a certain number of nodes called servers. These servers collectively maintain the public/private key pairs for all mobile nodes. Hubaux *et al.* [54] proposed to let users issue certificates for each other based on their personal acquaintances. An algorithm called "shortcut hunter" was presented to construct local certificate repositories such that any pair of nodes can find certificate chains to each other in their merged repository with high probability. Asokan *et al.* [9] proposed password authenticated key exchange protocols and multi-party Diffie-Hellman key exchange protocols to establish a group key among a dynamic group of mobile nodes. Unlike the above schemes, our approach does not use public key infrastructure or public algorithms. Capkun *et al.* [15] proposed a scheme in which two node can directly establish a security association when they move to the vicinity of each other and are able to exchange some secret material via a secure out-of-band channel. Two nodes without direct security association can set up an association with the help of "friends". In contrast, our scheme does not require out-of-band channel for key establishment or the help from any third party nodes. Basagni *et al.* [11] proposed to combine mobility-adaptive clustering and a probabilistic method for selecting key-generation nodes. Based on these techniques, this scheme periodically updates the symmetric keys used by all the nodes in the network. Unlike this scheme, our approach is localized, without involving any network-wide communication.

6.7 Conclusion

In this chapter, we propose a pairwise key establishment scheme through encrypted key migration. Compared to existing schemes, our scheme has perfect resilience

to node compromise, supports very large networks, provides high connectivity, and has very low overhead in terms of memory and communication.

Chapter 7

Related Work

In this chapter, we introduce work related to this thesis research.

7.1 Coverage

Meguerdichian, *et al.*, presented several interpretations of coverage in sensor networks, including deterministic coverage and stochastic coverage [63]. Also, the authors proposed a centralized polynomial time algorithm for coverage calculation. Another metric of sensor coverage, exposure, was defined in [62]. The authors also designed a centralized algorithm for calculating the minimal exposure paths.

7.2 Sensor Deployment

We first introduce papers on static sensor deployment, then papers on mobile sensor deployment, and finally node placement in other context of sensor networks. Deployment of static sensor networks has been addressed in [18, 25]. Clouqueur, *et al.*, proposed to deploy sensors in several steps and assumed random deployment in each step [18]. The number of sensors in each step and the cost of deployment were used as a cost function. The authors proposed algorithms to determine the number of steps of sensor deployment such that the cost is low and the desired distribution is obtained. Dhillon, *et al.*, proposed a centralized polynomial-time algorithm to determine sensor distribution

such that a minimum number of sensors are deployed and a minimum amount of data are transmitted.

Deployment of mobile sensors has been addressed in [49, 49, 90]. The work in [90] assumes that a cluster head is available to collect the sensor location and determine the target location of the mobile sensors. Howard, *et al.*, proposed an algorithm to deploy mobile sensors into a building from outside, in which sensors are deployed iteratively one by one, utilizing the location information obtained from the previous deployment [49]. The same authors proposed algorithms based on potential field to maximize the monitoring field in [50].

Relay nodes placement in heterogeneous networks has been addressed in [48, 69]. Hou, *et al* proposed a centralized polynomial-time heuristic algorithm for relay node placement to increase network lifetime [48]. Patel, *et al* designed centralized deployment strategies for sensor nodes, relay nodes, and base stations considering connectivity and coverage [69].

7.3 Motion Planning For Energy Efficiency

Energy efficient motion planning has been addressed in the robotics field. Sun and Reif worked on finding the most energy efficient path from a source point to a destination point [80]. They assumed there was no acceleration and turning during the movement and adopted the energy model in which only energy loss due to friction and gravity was considered. A similar model is also used in [73, 74, 75]. This model considers only the output mechanical energy of the motor, instead of the real energy consumption from the battery. In [64], power consumption is modeled as a polynomial of motor's angular

velocity, neglecting the effect of acceleration. Unlike mobile robots, which may move continuously performing certain tasks like carpet cleaning, mobile sensor may move only a short distance, stop and move again for application's needs. In this way, acceleration and deceleration must be taken into consideration. Energy efficient motion planning has also been addressed for walking robots with legs [33]. But this moving mechanism is not likely to be adopted for mobile sensor for their relatively high cost.

Low power design in static sensor networks has been intensively studied. Inside the network stack, there are power aware routing protocols [47], MAC protocols [23, 84], and power aware applications [90]. There is also work in lower power sensor platform [38].

7.4 Pair-wise Key Establishment Schemes in Sensor Networks

In the past years, a variety of pairwise key establishment schemes have been proposed. Most of these schemes fall into the following categories:

Centralized Schemes: One instance is SNEP proposed by Perrig *et al.* [70], in which an online base station is used for helping two nodes establish a pairwise key. This scheme is resilient to the capture of any number of ordinary sensor nodes. However, the property is based on the assumption that the online base station will not be compromised, which is difficult to achieve in practice. In addition, any two nodes must go through the base station for key establishment, which makes the base station a performance bottleneck in a large-scale network.

Distributed Schemes: To deal with the limitations inherent in the centralized schemes, a number of distributed schemes have been proposed. In LEAP proposed by Zhu *et al.*

[89], all nodes are preloaded with an initial network key before their deployment, and after node deployment, each pair of neighboring nodes derive their pairwise keys based on the initial key. This scheme however requires no node compromise before pairwise key establishment completes. Otherwise, the adversary will obtain the initial key and will be able to derive the pairwise key between any pair of nodes.

Considering that nodes could be compromised before pairwise key establishment completes, researchers have proposed schemes based on the ideas of both key predistribution and probabilistic key sharing. Eschenauer and Gligor [28] proposed the first such scheme, in which each node is preloaded with a random subset of keys. Any two nodes, after deployment, can set up a pairwise key if they have one common key preloaded. Since each node is randomly preloaded with a key ring from the key pool, it cannot be guaranteed that any two nodes can find a common key. In this case, they will not be able to set up a pairwise key directly. [28] has also shown that the probability for direct key establishment is affected by the key pool size, key ring size and network size. Specifically, as the key pool size increases, the key ring size decreases, and/or the network size increases, the probability is reduced. When two nodes cannot set up a pairwise key directly, they may still establish a pairwise key indirectly, if there exists a path connecting them such that any two neighboring nodes on the path share a pairwise key. The indirect key establishment, however, introduce a new security risk: if a node is compromised, the pairwise keys that are established with the help of the node are also compromised.

Eschenauer and Gligor's scheme [28] was enhanced by Chan *et al.*[17], who proposed a q-composite random key pre-distribution scheme and a random pairwise keys scheme. It was further enhanced by Liu and Ning, who proposed a random subset

assignment scheme and a grid-based scheme [59] based on Blundo's polynomial key predistribution protocol [13]. Du *et al.* [26] also proposed a similar scheme based on Blom's key predistribution scheme [12]. Traynor *et al.* [81] studied the pairwise key establishment problem in heterogeneous networks in which a subset of nodes are equipped with tamper-resistant hardware. These advanced schemes provide improved connectivity and resiliency. But they still cannot guarantee that any two nodes can set up a pairwise key. Also, they are not resilient to a large number of node compromises.

Location-aware Schemes: Du *et al.* [27], Liu *et al.* [60], and Huang *et al.* [53] proposed location-aware key establishment schemes, in which location information is employed to facilitate sensor nodes in establishing pairwise keys. These schemes can further improve the network connectivity, assuming the knowledge on the possible locations of sensor nodes before they are deployed. However, they cannot provide guarantee on pairwise key establishment, either.

7.5 Pair-wise Key Establishment Schemes in Ad hoc Networks

During recent years, a great deal of research has been conducted in key management for ad hoc networks. Zhou and Haas [88] proposed a solution based on threshold cryptography. In their approach, the certificate authority (CA) service is distributed over a certain number of nodes called servers. These servers collectively maintain the public/private key pairs for all mobile nodes. Hubaux *et al.* [54] proposed to let users issue certificates for each other based on their personal acquaintances. An algorithm called "shortcut hunter" was presented to construct local certificate repositories such that any pair of nodes can find certificate chains to each other in their merged repository

with high probability. Asokan *et al.* [9] proposed password authenticated key exchange protocols and multi-party Diffie-Hellman key exchange protocols to establish a group key among a dynamic group of mobile nodes. Unlike the above schemes, our approach does not use public key infrastructure or public algorithms. Capkun *et al.* [15] proposed a scheme in which two node can directly establish a security association when they move to the vicinity of each other and are able to exchange some secret material via a secure out-of-band channel. Two nodes without direct security association can set up an association with the help of "friends". In contrast, our scheme does not require out-of-band channel for key establishment or the help from any third party nodes. Basagni *et al.* [11] proposed to combine mobility-adaptive clustering and a probabilistic method for selecting key-generation nodes. Based on these techniques, this scheme periodically updates the symmetric keys used by all the nodes in the network. Unlike this scheme, our approach is localized, without involving any network-wide communication.

Chapter 8

Conclusion

In this thesis, we studied the problem of improving the dependability of sensor networks. We have worked on providing required coverage during network initilization and maintaining dependability during the remaining network lifetime. Specifically, we adopted two methods. One is to utilize mobile sensors to get and maintain desired coverage, and to achieve fault tolerance. The other is to secure sensor networks by pair-wise keying.

During network intialization, we proposed three algorithms to reach high coverage when all sensors are mobile; we also proposed bidding protocols to deploy mobile sensors when only a portion of sensors are mobile. Simulation results have shown that our algorithms are efficient and effective.

During the network operational time, to provide fault torelance, we have proposed a two-phase sensor relocation solution: redundant sensors are first identified and then relocated to the target location. We have proposed a Grid-Quorum solution to quickly locate the closest redundant sensor with low message complexity, and propose to use cascaded movement to relocate the redundant sensor in a timely, efficient and balanced way. Simulation results have verified that the proposed solution outperforms others in terms of relocation time, total energy consumption, and minimum remaining energy.

We have also addressed the problem of how to optimize sensor movement for energy efficiency. We have adopted a complete energy model to characterize the entire energy consumption in movement. Based on the model, we have proposed an optimal velocity schedule for minimizing energy consumption when the road condition is uniform; and a near optimal velocity schedule for the variable road condition by using continuous-state dynamic programming. Considering the variety in motion hardware, we have also designed one velocity schedule for simple microcontrollers, and one velocity schedule for relatively complex microcontrollers, respectively. Simulation results have shown that our velocity planning may have significant impact on energy conservation.

To secure sensor networks by pair-wise keys, we have proposed a novel pair-wise key establishment scheme based on the idea of key migration. Our scheme outperforms existing schemes in terms of resilience to node failure, supportable network size, efficiency, and memory requirements. Extensive analysis has been conducted to analyze the proposed scheme, and the results have shown that the scheme has the following advantages: (1) near perfect resiliency to node compromise; (2) supporting a large network (e.g., supporting up to 64001 nodes when applied to Berkeley Mica notes); (3) guaranteeing pair-wise key establishment between any two nodes; (4) low storage requirements after key setup (i.e., only established pair-wise keys plus one extra key to be stored); (5) enabling node-to-node authentication.

References

[1] US Naval Observatory (USNO) GPS Operations. *http://tycho.usno.navy.mil/gps.html*, April 2001.

[2] Berkeley Sensor and Actuator Center. *http://www-bsac.eecs.berkeley.edu*, 2004.

[3] Wireless Sensing Networks. *http://wins.rsc.rockwell.com*, 2004.

[4] Mobile robots. *http://www.k-team.com/robots/khepera/index.html*, 2005.

[5] Motor. *http://www.micromo.com*, 2005.

[6] A. Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64:763–776, 1995.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, pages 102–114, August 2002.

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

[9] N. Asokan and P. Ginzboorg. Key Agreement in Ad Hoc Networks. *Computer Communications*, 23:1627–1637, 2000.

[10] F. Aurenhammer. Voronoi Diagrams — A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23:345–405, 1991.

[11] S. Basagni, K. Herrin, E. Rosti, and D. Bruschi. Secure Pebblenets. *ACM MobiHoc*, 2001.

[12] R. Blom. An optimal class of symmetric key generation systems. *Advances in Cryptology: Proceedings of EUROCRYPT 84, Thomas Beth, Norbert Cot, and IngemarIngemarsson, eds. Lecture Notes in Computer Science, Springer-Verlag*, 209:335–338, 1985.

[13] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung. Perfectly-Secure Key Distribution for Dynamic Conferences. *Lecture Notes in Computer Science*, 740:471–486, 1993.

[14] G. Cao and M. Singhal. A Delay-Optimal Quorum-Based Mutual Exclusion Algorithm for Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(12):1256–1268, 2001.

[15] S. Capkun, J. Hubaux, and L. Buttyan. Mobility Helps Security in Ad Hoc Networks. *ACM MobiHoc*, pages 46–56, June 2003.

[16] A. Carzaniga, D. Rosenblum and A. Wolf. Design and Evaluation of a Wide-area Event Notification Service. *ACM Transactions on Computer Systems*, 19, August 2001.

[17] H. Chan, A. Perrig, and D. Song. Random Key Predistribution Schemes for Sensor Networks. *IEEE Symposium on Research in Security and Privacy*, May 2003.

[18] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan and K. K. Saluja. Sensor Deployment Strategy for Target Detection. *First ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[19] Electro-Craft Corporation. *DC Motors, Speed Controls, Servo Systems, An Engineering Handbook*. Pergamon Press, 1977.

[20] CROSSBOW TECHNOLOGY INC. Wireless sensor networks. *http://www.xbow.com/Products/ Wireless_Sensor_Networks.htm*, 2005.

[21] D. Koditschek. Planning and Control via Potentical Functions. *Robitcs Review I*, pages 349–367, 1989.

[22] D. Niculescu and B. Nath. Ad Hoc Positioning Systems (APS) Using AoA. *IEEE INFOCOM*, 2003.

[23] T. Dam and K. Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Network. *SenSys*, 2003.

[24] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal and G. S. Sukhatme. Robomote: Enabling Mobility in Sensor Network. *CRES-04-006 CRES Tech Report*.

[25] S. Dhillon, K. Chakrabarty and S. Iyengar. Sensor Placement for Grid Coverage under Imprecise Detections. *Proceedings of the International Conference on Information Fusion*, 2002.

[26] W. Du, J. Deng, Y. Han, and P. Varshney. A Pairwise Key Pre-distribution Schemes for Wireless Sensor Networks. *The 10th ACM Conference on Computer and Communications Security (CCS)*, 2003.

[27] W. Du, J. Deng, Y. Han, and S. Chen. A key management scheme for wireless sensor networks using deployment knowledge. *IEEE INFOCOM*, March 2004.

[28] L. Eschenauer and V. Gligor. A Key-management Scheme for Distributed Sensor Networks. *The 9th ACM Conference on Computer and Communications Security (CCS)* , pages 41–47, November 2002.

[29] EU, TU Berlin. Eyes: Energy Efficient Sensor Networks. *http://www.eyes.eu.org/index.htm*, 2005.

[30] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[31] F. Zhao and L. Guibas. *Wireless Sensor Networks*. Morgan Kaufmann, 2004.

[32] S. Fortune, D. Du and F. Hwang. Voronoi Diagrams and Delaunay Triangulations. *Euclidean Geometry and Computers*, 1992.

[33] Y. Fujimoto. Trajectory Generation of Biped Running Robot with Minimum Energy Consumption. *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[34] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu. Analyzing and modeling encryption overhead for sensor network nodes. *Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003.

[35] M. R. Garey and D. S. Johnson. *Computers and Intractability*. 1979.

[36] M. R. Garey, D. S. Johnson, and L. Stockmeyer. *Some Simplified NP-Complete Graph Problems*. 1976.

[37] Z. Ge, P. Ji, J. Kurose and D. Towsley. Matchmaker: Signalling for Dynamic Publish/Subscribe Applications. *11th IEEE International Conference on Network Protocols(ICNP)*, November 2003.

[38] S. Gregori, Y. Li, H. Li, J. Liu and F. Maloberti. 2.45 GHz power and data transmission for a low-power autonomous sensors platform. *In Proc. of ISLPED*, 2004.

[39] Guiling Wang, Guohong Cao and Tom La Porta. Sensor Deployment Protocols. *To appear in IEEE Transaction on Mobile Computing*.

[40] Guiling Wang, Guohong Cao and Tom La Porta. A Bidding Protocol for Deploying Mobile Sensors. *The 11th IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[41] Guiling Wang, Guohong Cao and Tom La Porta. Movement-Assisted Sensor Deployment. *IEEE INFOCOM*, March 2004.

[42] Guiling Wang, Guohong Cao and Tom La Porta. Proxy-Based Sensor Deployment for Mobile Sensor Networks. *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, October 2004.

[43] Guiling Wang, Guohong Cao, Tom La Porta and Piotr Berman. Bidding Protocols for Deploying Mobile Sensors. *Under revision to IEEE Transaction on Mobile Computing.*

[44] Guiling Wang, Guohong Cao, Tom La Porta and Wensheng Zhang. Sensor Relocation in Mobile Sensor Networks. *IEEE INFOCOM*, March 2005.

[45] Guiling Wang, Mary Jane Irwin, Piotr Berman, Haoying Fu and Tom La Porta. Optimizing Sensor Movement for Energy Efficiency. *International Symposium on Low Power Electronics and Design (ISPLED)*, August 2005.

[46] Guiling Wang, Srimat Chakradhar, Wensheng Zhang, Tom La Porta, and Piotr Berman. Pair-wise Key Establishment for Wireless Sensor Networks with Key Migration. *Under submission to a conference.*

[47] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocal for Wireless Microsensor network. *Proc. of the Hawaii International Conference on System Sciences*, January 2000.

[48] Y. T. Hou, Y. Shi, H. D. Sherali, and S. F. Midkiff. Prolonging Sensor Network Lifetime with Energy Provisioning and Relay Node Placement. *Proceedings of IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2005.

[49] A. Howard, M. J. Mataric and G. S. Sukhatme. An Incremental Self-Deployment Algorithm for Mobile Sensor Networks. *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, September 2002.

[50] A. Howard, M. J. Mataric and G. S. Sukhatme. Mobile Sensor Networks Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem. *the 6th International Symposium on Distributed Autonomous Robotics Systems*, June 2002.

[51] L. Hu and D. Evans. Localization for Mobile Sensor Networks. *ACM MobiCom*, 2004.

[52] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Wireless Ad Hoc Networks. *ACM MobiCom*, September 2002.

[53] D. Huang, M. Mehta, D. Medhi, and L. Harn. Location-aware key management scheme for wireless sensor networks. *Proceeding of ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2004.

[54] H. Hubaux, L. Buttyan, and S. Capkun. The Quest for Security in Mobile Ad Hoc Networks. *ACM MobiHoc*, pages 146–155, 2001.

[55] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[56] J. H. Kim, D. -H. Kim, Y. -J. Kim and K. -T. Seow. *Soccer Robotics*. Springer, 2004.

[57] J. Lengyel, M. Reichert, B. Donald, and D. Greenberg. Real-time Robot Motion Planning using Rasterizing Computer Graphics Hardware. *Proc. SIGGRAPH*, 1990.

[58] Q. Li, M. De Rosa, and D. Rus. Distributed Algorithms for Guiding Navigation Across a Sensor Network. *ACM MobiCom*, 2003.

[59] D. Liu and P. Ning. Establishing Pairwise Keys in Distributed Sensor Networks. *The 10th ACM Conference on Computer and Communications Security (CCS)*, 2003.

[60] D. Liu and P. Ning. Location-based pairwise key establishment for static sensor networks. *Proceeding of ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2003.

[61] N. Megiddo. Linear-time Algorithms for Linear Programming in R^3 and Related Problems. *SIAM J. Computing 12*, pages 759–776, 1983.

[62] S. Meguerdichian, F. Koushanfar, G. Qu and M. Potkonjak. Exposure In Wireless Ad-Hoc Sensor Networks. *ACM MobiCom*, 2001.

[63] S. Meguerdichian, F. Koushanfar, M. Potkonjak and M. B. Srivastava. Coverage Problems in Wireless Ad-hoc Sensor Network. *IEEE INFOCOM*, 2001.

[64] Y. Mei, Y. H. Lu, C. Lee and Y. C. Hu. Energy-Efficient Motion Planning for Mobile Robots. *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[65] MoteIV. TMote Sky. *http://www.moteiv.com*, 2005.

[66] N. Patwari and A. Hero III. Using Proximity and Quantized RSS for Sensor Location in Wireless Location in Wireless Networks. *Workshop on Wireless Sensor Networks and Applications*, 2003.

[67] P. Papadimitratos and Z Haas. Secure Routing for Mobile Ad Hoc Networks. *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, Jan. 2002.

[68] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[69] M. Patel, R. Chandrasekaran, and S. Venkatesan. Energy Efficient Sensor, Relay and Base Station Placements for Coverage, Connectivity and Routing. *Proceedings of 24th IEEE International Performance, Computing and Communications Conference*, 2005.

[70] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. Spins: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, September 2002.

[71] R. Pietro, L. Mancini, and A. Mei. Random key assignment for secure wireless sensor networks. *Proceeding of ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2003.

[72] G. J. Pottie. Wireless Sensor Networks. *ITW, Killamey, Ireland*, June 1998.

[73] N. C. Rowe. Obtaining optimal mobile-robot paths with non-smooth anisotropic cost functions using qualitative-state reasoning. *International Journal of Robotics Research*, 16(3):375–399, June 1997.

[74] N. C. Rowe and Y. Kanayama. Near-minimum-energy paths on a vertical-axis cone with anisotropic friction and gravity effects. *International Journal of Robotics Research*, 13(5):408–433, Oct. 1994.

[75] N. C. Rowe and R. S. Ross. Optimal Grid-free Path Planning across Arbitrarily Contoured Terrain with Anisotropic Friction and Gravity Effects. *IEEE Transactions on Robotics and Automation*, 6(5):540–553, Oct. 1990.

[76] A. Savvides, C. Han and M. B. Strivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. *ACM MobiCom*, 2001.

[77] Sensor Nets. Intel Mote. *http://www.intel.com/research/exploratory/motes.htm*, 2005.

[78] G. T. Sibley, M. H. Rahimi and G. S. Sukhatme. Robomote: A Tiny Mobile Robot Platform for Large-Scale Sensor Networks. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[79] S. Skyum. A Simple Algorithm for Computing the Smallest Enclosing Circle. *Information Processing Letters*, 37:121–125, 1991.

[80] Z. Sun and J. Reif. On Energy-minimizing Paths on Terrains for a Mobile Robot. *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[81] P. Traynor, H. Choi, G. Cao, S. Zhu and T. L. La Porta. Establishing pair-wise keys in heterogeneous sensor networks. *IEEE INFOCOM*, April 2006.

[82] E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). *New Results and New Trends in Computer Science, volume 555 of LNCS*, pages 359–370, 1991.

[83] Y. Xu, J. Heidemann and D. Estrin. Geography Informed Energy Conservation for Ad Hoc Routing. *ACM MobiCom*, July 2001.

[84] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. *IEEE INFOCOM'02*, June 2002.

[85] M. Younis, M. Youssef and K. Arisha. Energy-Aware Management for Cluster-Based Sensor Networks. *Computer Networks: The International Journal of Computer and Telecommunicat ions Networking*, 43(5):649–668, December 2003.

[86] W. Zhang and G. Cao. Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks. *IEEE INFOCOM*, 2004.

[87] W. Zhang, G. Cao and T. La Porta. Data Dissemination with Ring-Based Index for Wireless Sensor Networks. *The 11th IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[88] L. Zhou and Z. Haas. Securing Ad Hoc Networks. *IEEE Network*, 13(6):24–30, November/December 1999.

[89] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. *The 10th ACM Conference on Computer and Communications Security*, 2003.

[90] Y. Zou and K. Chakrabarty. Energy-Aware Target Localization in Wireless Sensor Networks. *Proc. IEEE Int. Conf. Pervasive Computing and Communications*, 2003.

[91] Y. Zou and K. Chakrabarty. Sensor Deployment and Target Localization Based on Virtual Forces. *IEEE INFOCOM*, 2003.

Vita

Guiling Wang was born in China. She received the B.S. degree in Computer Science from the Nankai University, Tianjin, China, in July 1999. She enrolled in the Ph.D. program in Computer Science and Engineering at The Pennsylvania State University in August 2000. She is a student member of the IEEE.

SELECTED PUBLICATIONS

- Guiling Wang, Guohong Cao and Tom La Porta. "Movement-Assisted Sensor Deployment," to appear in IEEE Transactions on Mobile Computing
- Guiling Wang, Mary Jane Irwin, Piotr Berman, Haoying Fu, and Tom La Porta. "Optimizing Sensor Movement Planning for Energy Efficiency," in Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), August 2005
- Guiling Wang, Guohong Cao, Tom La Porta and Wensheng Zhang. "Sensor Relocation in Mobile Sensor Networks," IEEE INFOCOM, March 2005
- Guiling Wang, Guohong Cao and Tom La Porta. "Movement-Assisted Sensor Deployment," IEEE INFOCOM, Hong Kong, March 2004
- Guiling Wang, Guohong Cao and Tom La Porta. "A Bidding Protocol for Sensor Deployment," in Proceedings of the IEEE International Conference on Network Protocols (ICNP), Georgia, November 2003