The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

A DATA CENTRIC FRAMEWORK FOR MOBILE TARGET TRACKING AND DATA DISSEMINATION IN WIRELESS SENSOR NETWORKS

A Thesis in

Computer Science and Engineering

by

Wensheng Zhang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2005

The thesis of Wensheng Zhang has been reviewed and approved* by the following:

Guohong Cao
Associate Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

Chita R. Das
Professor of Computer Science and Engineering

George Kesidis
Associate Professor of Computer Science and Engineering

Tom F. La Porta
Professor of Computer Science and Engineering

Aylin Yener
Assistant Professor of Electrical Engineering

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

Wireless sensor networks are envisioned to be extremely useful for various environmental, civil, military and homeland security missions. Different from other paradigms of networks, sensor networks not only provide communication services but also generate sensing data to be transmitted. As sensor networks scale in size, so will the amount of sensing data generated. The large volume of sensing data coupled with the facts that the data are spread across the entire network, the sensor network is resource-constrained, and the sensor network is usually deployed in unattended and hostile environments, creates a demand for efficient and secure data collection and dissemination/access techniques to obtain useful and high-quality data from within the network.

In this thesis, we study the problem in a typical application scenario where a sensor network is deployed to monitor mobile targets in unattended environments and disseminate the obtained sensing data from dynamically changed detecting nodes to mobile sinks. We address the problem in three aspects.

Collaborative Detection and Efficient Data Collection: Since sensor nodes have limited sensing/communication ranges and are not reliable, it is necessary for multiple sensor nodes to collaborate in detecting a target. Further, due to the mobility of the targets, nodes involved in the collaboration will change over time. We propose a dynamic convoy tree-based collaboration (DCTC) framework to facilitate the collaborative detection among a dynamic set of sensor nodes, and efficient collection of data for generating high-quality sensing results. In DCTC, sensor nodes surrounding a mobile target

form a tree structure, which can be dynamically adjusted, to facilitate the collaborations among them and collect their sensing data. One big challenge in implementing DCTC is how to reconfigure the convoy tree efficiently. We formalize the problem as an optimization problem of finding a convoy tree sequence with high tree coverage and low energy consumption, and propose both an ideal scheme and several practical schemes to solve it.

Efficient Data Dissemination: To facilitate users (static or mobile) in finding sensing data of interest from within a sensor network, we propose an index-based data dissemination scheme with adaptive ring-based index (ARI). This scheme is based on the idea that sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes. To tolerate failures and balance load, the index nodes for each type of targets form a ring surrounding the location which is determined based on the target type. To improve the performance, we further propose several mechanisms to optimize the ARI scheme.

Securing Data Forwarding: In data collection and dissemination, sensing data need to be forwarded in the network. To achieve authenticity and confidentiality in sensing data forwarding, innocent sensor nodes can share group keys for data encryption and authentication. However, the group key-based techniques will become ineffective if some nodes are compromised since the adversary may obtain group keys from these compromised nodes. To deal with node compromise, the innocent nodes should update their group keys to prevent the adversary from utilizing the captured keys. Because most previously proposed group rekeying schemes have high overhead and are not

suitable for sensor networks, we design and evaluate a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes, based on the idea that future group keys can be preloaded to nodes before deployment, and neighbors can collaborate to protect and appropriately use the preloaded keys.

Table of Contents

List of Tables xi

List of Figures xii

Acknowledgments xv

Chapter 1. Introduction 1

1.1 Mobile Target Tracking and Data Dissemination in Sensor Networks 1

1.2 Thesis Overview 4

Chapter 2. Dynamic Convoy Tree-Based Collaboration for Mobile Target Detection and Sensing Data Collection 9

2.1 Introduction 9

2.2 The System Model 12

2.3 The Optimal DCTC Scheme 18

2.4 Design and Implementation of DCTC 23

2.4.1 Constructing the Initial Tree 23

2.4.2 Tree Expansion and Pruning 25

2.4.3 Tree Reconfiguration 27

2.4.3.1 The Basic Scheme 27

2.4.3.2 Sequential Reconfiguration 29

2.4.3.3 Localized Reconfiguration 32

2.5 Performance Evaluations 33
2.5.1 The Simulation Model 33
2.5.2 Simulation Results 34
2.5.2.1 Comparing the Tree Expansion and Pruning Schemes 36
2.5.2.2 Fine-Tuning the Reconfiguration Scheme 38
2.5.2.3 Comparing the Reconfiguration Schemes 39
2.6 Summary of Dynamic Convoy Tree-based Collaboration 44

Chapter 3. Data Dissemination with Ring-Based Index for Wireless Sensor Networks 46
3.1 Introduction 46
3.2 System Model 49
3.3 The Index-Based Data Dissemination 51
3.3.1 Basic Idea of the Index-based Data Dissemination 51
3.3.2 Analytical Comparison of the Data Dissemination Methods . 53
3.4 An Adaptive Ring-based Index (ARI) Scheme 56
3.4.1 Motivation 56
3.4.2 The ARI Scheme 58
3.4.2.1 Initializing an Index Ring 58
3.4.2.2 Querying and Updating an Index 60
3.4.2.3 Dealing with Node Failures 63
3.4.2.4 Ring Reconfiguration for Load Balance 65
3.5 Enhancements 65

3.5.1 Reduce the Overhead of Index Updating 65
3.5.1.1 Basic Idea of LIU 66
3.5.1.2 Dynamically Adjust the Frequency of Index Updating 67
3.5.1.3 The Protocol 68
3.5.2 Reduce the Query Overhead 70
3.6 Performance Evaluations 71
3.6.1 Simulation Model 71
3.6.2 Simulation Results 72
3.6.2.1 Comparing the performance of data dissemination schemes 72
3.6.2.2 Impact of Parameter r on The Performance of ARI 76
3.6.2.3 Tolerating Clustering Failures 78
3.6.2.4 Load Balance 79
3.6.2.5 Evaluating LIU 80
3.6.2.6 Evaluating LIQ 84
3.7 Summary of the Data Dissemination Scheme with Ring-Based Index 85

Chapter 4. Distributed Group Key Management for Data Authenticity and Confidentiality 86
4.1 Introduction 86
4.2 System Model 89
4.3 Basic Predistribution and Local Collaboration-Based Group Rekeying (B-PCGR) Scheme 90

4.3.1 Basic Idea 90
4.3.1.1 Group Key Predistribution 91
4.3.1.2 Local Collaboration-Based Key Protection 91
4.3.1.3 Local Collaboration-Based Group Key Updating . . 92
4.3.2 Detailed Description of B-PCGR 92
4.3.2.1 Predistributing g-Polynomials 92
4.3.2.2 Encrypting g-Polynomials and Distributing the Shares of e-Polynomials 93
4.3.2.3 Key Updating 94
4.3.3 Security Analysis 95
4.3.4 Parameter Selection 98
4.3.4.1 A Theoretic Model 98
4.3.4.2 Some Numeric Results 101
4.3.4.3 The Approach for Parameter Selection 104
4.3.5 Detecting False Shares 105
4.4 Enhancements 106
4.4.1 Cascading PCGR (C-PCGR) Scheme 107
4.4.1.1 The Scheme 107
4.4.1.2 Security Analysis 111
4.4.2 Random Variance-Based PCGR (RV-PCGR) Scheme 111
4.4.2.1 Basic Idea 112
4.4.2.2 The Scheme 112
4.4.2.3 Security Analysis 115

4.5 Performance Evaluations 116
4.5.1 Performance Analysis 116
4.5.1.1 Comparing the Performance of PCGR Schemes . . . 117
4.5.1.2 Comparison with Other Group Rekeying Schemes . 119
4.5.2 Simulations 123
4.5.2.1 Simulation Model 124
4.5.2.2 Simulation Results 125
4.6 Discussions 128
4.6.1 Node Isolation and New Node Deployment 128
4.6.2 Other Issues 129
4.7 Summary of Distributed Group Rekeying Schemes 129

Chapter 5. Conclusion and Future Work 131
5.1 Conclusion 131
5.2 Future Work 133

References 135

List of Tables

2.1 Simulation Parameters 35

2.2 Energy consumption per second for data collection (mW) 42

3.1 Estimating the overhead of the data dissemination schemes 54

3.2 Simulation Parameters 73

4.1 Lookup table for selecting μ ($N = 2000$,$r_f = 3.9e-7$,$r_c = 1.4e-7$,$t =$ 6 days; $P_n(\mu, t) \geq 0.9$, $P_c(\mu, t) \leq 0.01$) 104

4.2 Comparing B-PCGR, C-PCGR and RV-PCGR 120

4.3 Comparing B-PCGR with previous group rekeying schemes 122

List of Figures

1.1 Mobile target tracking and Data Dissemination 2

1.2 A data centric framework for mobile target monitoring 5

2.1 Using convoy tree to track the target 10

2.2 Dividing a sensor network into grids 13

2.3 The basic structure of DCTC 14

2.4 The optimal DCTC scheme 19

2.5 Tree expansion and pruning 26

2.6 The sequential tree reconfiguration 30

2.7 The Sequential Tree Reconfiguration Algorithm 31

2.8 Comparing the conservative scheme and the prediction-based scheme . . 35

2.9 Fine-tuning the sequential reconfiguration scheme 38

2.10 Energy consumption due to tree expansion and pruning 40

2.11 Comparing the energy consumption due to tree reconfiguration 41

2.12 Comparing the energy consumption of the sequence reconfiguration and the localized reconfiguration 43

3.1 Dividing a sensor network into grids 50

3.2 Index-based data dissemination 52

3.3 Initializing an Index Ring 59

3.4 Querying an index 62

3.5 Dealing with clustering failures 64

3.6 Lazy index updating 66

3.7 The state transition of a storing node 69

3.8 Evaluating data dissemination schemes ($v = 1.0m/s$) 74

3.9 Impact of r ($v = 3.0m/s, m = 8$) 77

3.10 Evaluating the fault tolerance feature of ARI ($r = 34m * 2, m = 8, v = 3.0m/s$) 78

3.11 Evaluating the load balance feature of ARI ($r = 34m * 2, m = 8, v = 1.0m/s$) 80

3.12 The index updating message complexity with/without LIU ($r = 34m * 2, m = 8$) 81

3.13 The query message complexity with/without LIU ($r = 34m * 2, m = 8$) . 82

3.14 The average query delay with/without LIU ($r = 34m * 2, m = 8$) 82

3.15 The total message complexity with/without LIU ($r = 34m * 2, m = 8$) . 83

3.16 The message complexity with/without LIQ ($r = 34m * 2, m = 8, v = 3.0m/s$) 84

4.1 B-PCGR: Polynomial encryption and share distribution 93

4.2 B-PCGR: Key updating 94

4.3 An Example 100

4.4 Impact of r_c 102

4.5 Impact of r_f 103

4.6 Impact of n 103

4.7 C-PCGR: Polynomial Decryption and Share Distribution 108

4.8 C-PCGR: Key Updating (C is increased from 0 to 1) 110

4.9 Basic Idea of RV-PCGR 113

4.10 Performance of the key updating scheme ($\tau_c = 10min$,$\tau_u = 100min$) . . 125

4.11 Tunning key updating interval (τ_u) 127

Acknowledgments

I am most grateful and indebted to my advisor, Professor Guohong Cao for his support throughout my thesis research. He not only created a motivating, enthusiastic, and critical research environment, but also provided valuable research experience and expertise. Professor Cao guided me through the countless problems and setbacks I met in my research. Without all his guidance and help, this thesis would not have been possible. It has been a great honor and pleasure for me to conduct this thesis under his supervision. I would also like to thank Professors Tom F. La Porta, Chita R. Das, George Kesidis, and Aylin Yener for spending their time on my committee and for providing insightful and constructive comments on this thesis.

During my time at Penn State, I met many friends, too many to list here. They helped me to develop new ideas through discussion and criticism. With them, I have a good time besides doing research at Penn State. I have been really lucky to have met all of them.

I'm also thankful to my family. Although they live far away, their love and support are always with me.

Chapter 1

Introduction

Recent advances in micro-electro-mechanics (MEMS) and wireless communications have enabled the design of small-size, low-cost sensor nodes. These nodes are equipped with sensing, computation, communication and storage units, which allow them to sense their environments and collect, process, exchange, and store the sensing data. Once being deployed in some space, these nodes can self-organize themselves into multihop wireless sensor networks [2, 30, 37, 47], which are expected to be widely applied in military and civilian applications.

1.1 Mobile Target Tracking and Data Dissemination in Sensor Networks

In many applications, e.g., battlefield surveillance [36] and habitat monitoring [9], sensor networks are deployed to unattended environments for monitoring mobile targets. Figure 1.1 shows a typical scenario of battlefield surveillance. As an enemy tank appears in the surveillance region, the sensor nodes nearby can detect the tank and monitor its surrounding area (called *monitoring region*), which may include enemy soldiers, and generates sensing data about the monitoring region. The sensing data can be pushed to some stationary data centers, which will serve the data to users; the data can also be saved locally waiting for queries from users (sinks) that could be static command

centers or moving soldiers [53]. As desired, sensing data about the monitoring area should be produced promptly and reliably, and the users interested in the information should obtain the sensing data promptly, efficiently, and securely.

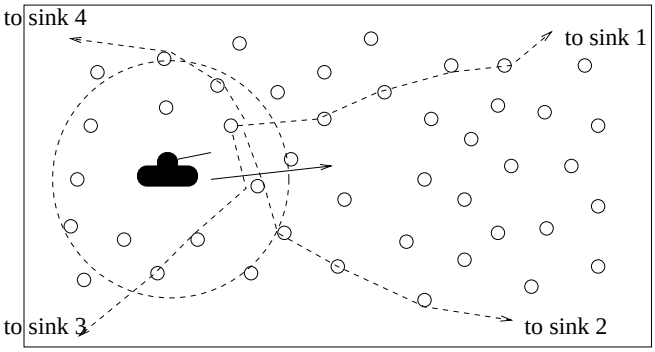


Fig. 1.1. Mobile target tracking and Data Dissemination

Deploying sensor networks for monitoring mobile targets and disseminating the sensing data from dynamically changed detecting nodes ("mobile" sources) to mobile sinks could be complicated and intriguing due to several reasons. First of all, sensor nodes are resource-constrained and not reliable. These nodes have limited energy supply since they are powered by batteries that are not replaceable or rechargeable in harsh environments. Therefore, the protocols proposed for sensor networks must be energy efficient and energy conserving in order to reduce the energy consumption and extend the life time of the networks. In addition, sensor nodes are not reliable. They may malfunction or even die out because of energy depletion or environmental reasons, and their readings may drift and lose calibration due to environmental interference. Thus, we

cannot rely on a single node to detect a target or store sensing data. Because some level of redundancy can be used to deal with failures, multiple nodes may collaborate in a sensing task. These nodes can cooperate to obtain fine-grain and high-precision sensing data, and maintain the persistence of the data to satisfy queries. If the detected target or the monitoring region is large, a large number of sensor nodes may be involved due to the limited sensing/communication range of each individual node. Further, due to the mobility of targets, nodes involved in the collaboration will change over time. All these lead to the collaborations among a large dynamic set of senor nodes. This, together with the constrained power supply in sensor nodes, poses new challenges in designing efficient schemes.

Secondly, as a sensor network scales in size, so is the amount of sensing data generated. The large volume of data coupled with the facts that the data are spread across the entire network due to the mobility of monitored targets, and that the sinks could also be mobile, creates a demand for efficient data dissemination/access techniques to find the relevant data from within the network. This problem of data dissemination in sensor networks has been studied in the past, and many schemes [23, 27, 53, 39, 20, 21] have been proposed. However, these schemes either require all sensing data be pushed to some storage nodes regardless of their usefulness, or demand the flooding of queries or the data availability information. Therefore, they are not efficient for large scale sensor networks, especially in the scenarios where a large amount of sensing data are generated but only a small portion of them will be queried.

Thirdly, as sensor networks are deployed in unattended and hostile environments, adversaries may eavesdrop the communications in the network. Even worse, the adversaries may capture and reprogram some sensor nodes, or inject their own sensor nodes into the network and make the network accept them as legitimate nodes [22]. After getting control of even a few nodes, the adversaries can mount various attacks from inside the network. For example, they may use a compromised node to inject false sensing reports or maliciously modify reports that go through it. Under such attacks, the sink may receive incorrect sensing data and make wrong decisions, which could be dangerous in scenarios such as battlefield surveillance and environmental monitoring. These security issues must be addressed in order to successfully deploy a sensor network. As the first line of defense, cryptographic (key-based) mechanisms can be employed to achieve the confidentiality and authenticity of sensing data. To effectively employ these mechanisms, however, key management is required. Considering the highly constrained system resources and the large network scale, the key management schemes must be efficient and scalable.

1.2 Thesis Overview

In this thesis work, we address the above issues by proposing a data-centric framework (as shown in Figure 1.2) which facilitates collaborative target monitoring, efficient and secure sensing data collection/dissemination. This framework includes the following three parts:

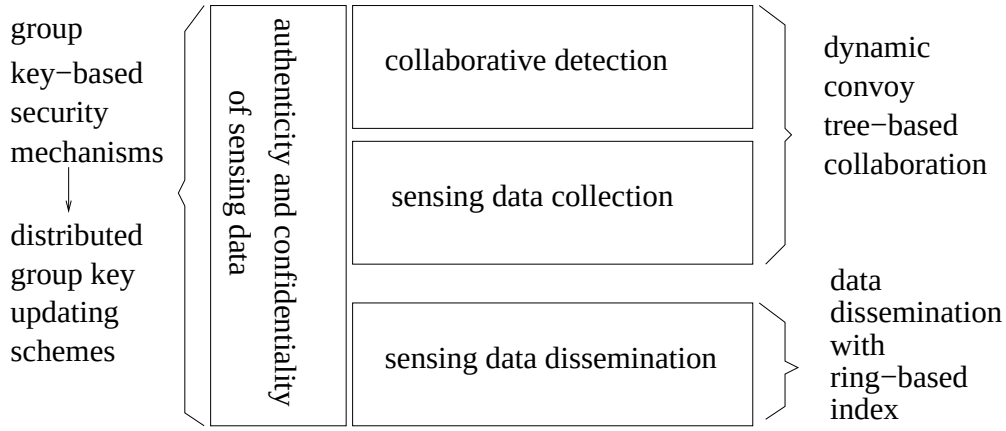


Fig. 1.2. A data centric framework for mobile target monitoring

- **Dynamic Convoy Tree-based Collaboration (DCTC) for Mobile Target Monitoring and Sensing Data Collection:** We first study how multiple sensor nodes collaborate efficiently in monitoring a mobile target and collecting their sensing data to obtain high-precision high-reliability sensing results. We propose a *Dynamic Convoy Tree-based Collaboration (DCTC)* framework [55]. DCTC employs a tree structure called *convoy tree*, which includes sensor nodes around the moving target, to facilitate the collaborations among sensor nodes and collect their sensing data. As the target move, the tree is dynamically configured to add some nodes and prune some nodes. Since a big challenge in implementing the DCTC framework is how to reconfigure the convoy tree efficiently as the target moves, we first formalize it as an optimization problem of finding a min-cost convoy tree sequence with high tree coverage, and give an optimal solution (o-DCTC) based on dynamic programming. Considering the constraints of sensor networks, we propose some practical solutions. Specifically, we propose two tree expansion and

pruning schemes: the *conservative* scheme and the *prediction-based* scheme. We also propose two tree reconfiguration schemes: the *sequential reconfiguration* and the *localized reconfiguration*. We evaluate the performance of the optimal solution and the practical implementations through extensive simulations. Based on the simulation results, when the same reconfiguration scheme is used, the prediction-based scheme outperforms the conservative scheme and it can achieve a similar coverage and energy consumption to the optimal solution. When using the same scheme for tree expansion and pruning, the localized reconfiguration scheme outperforms the sequential reconfiguration scheme when the node density is high, and the trend is reversed when the node density is low.

- **Sensing Data Dissemination with Ring-Based Index:** In this part, we study the issue of sensing data dissemination after they have been aggregated at the source side. Although many schemes [23, 27, 53, 39, 20, 21] have been proposed for disseminating sensing data, most of them are not efficient in a large scale network where a huge amount of sensing data are generated, but only a small portion of them are queried. To address the problem, we propose an index-based data dissemination scheme. With this scheme, sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes, which act as the rendezvous points for sinks and sources. To address the issues of fault tolerance and load balance, we extend the scheme with an adaptive ring-based index (ARI) technique [59], in which the index nodes for one event type form a ring surrounding the

location which is determined by the event type, and the ring can be dynamically reconfigured. Considering that frequently updating or querying index nodes may cause large overhead, we further propose a lazy index updating (LIU) mechanism and a lazy index querying (LIQ) mechanism to reduce the overhead. Analysis and simulations are conducted to evaluate the performance of the proposed index-based scheme. The results show that the index-based scheme outperforms the external storage-based scheme, the DCS scheme, and the local storage-based schemes with flood-response style. The results also show that using ARI can tolerate clustering failures and achieve load balance, and using LIU (LIQ) can further improve the system performance.

- **Distributed Group Key Management for Data Authenticity and Confidentiality:** To secure sensing data forwarding, innocent sensor nodes can share group keys for data encryption and authentication. However, the group key-based security mechanisms will become ineffective if some nodes are compromised since the adversary can obtain group keys from the compromised nodes. To deal with node compromise, the compromised nodes should be identified, and the innocent nodes should update their group keys to prevent the adversary from utilizing the captured keys. Since most existing group key updating schemes [26, 49, 51, 4] have high overhead and are not suitable for sensor networks, we propose a family of distributed group key updating schemes [48], which are based on the following ideas: (1) Future keys are preloaded to individual nodes before deployment to avoid the high overhead of securely disseminating new keys at the key updating time.

(2) Neighbors collaborate with each other to effectively protect and appropriately use the preloaded keys; the local collaboration also relieves the high cost of the centralized management. Based on thorough analysis, we show that the proposed schemes can achieve a good level of security with low overhead that affordable by current generation of sensor nodes, and they outperform most of the previously proposed group rekeying schemes. We also simulate the proposed schemes in filtering false data, and the results show that the effectiveness of filtering false data can be significantly improved.

The rest of the thesis is organized as follows. Chapter 2 presents the research on collaborative detection and efficient sensing data collection. Chapter 3 presents the research on efficient sensing data dissemination. Chapter 4 presents the research on distributed group key management for sensing data authenticity and confidentiality. Finally, Chapter 5 concludes the thesis.

Chapter 2

Dynamic Convoy Tree-Based Collaboration for Mobile Target Detection and Sensing Data Collection

2.1 Introduction

In this chapter, we study how multiple sensor nodes collaborate efficiently in detecting a target and collecting the sensing data generated by them to obtain high-precision high-reliability sensing results. We study this problem under a specific application scenario, where a sensor network is deployed for detecting and tracking a mobile target, and monitoring a particular region surrounding the target in sensor networks. As shown in Figure 2.1, the sensor nodes surrounding an adversary tank detect and track the tank and its surrounding area which may include enemy soldiers. These nodes collaborate among themselves to aggregate data about the tank as well as its surrounding area, and one of them (i.e., the root) generates a data report. The data report can be saved locally waiting for other node's query, or can be forwarded to single or multiple data centers (the sinks), which can be a static command center or moving soldiers. Other examples can be found in applications such as tracking an important target (e.g., an important person) in a parade, or a valuable animal (e.g., a panda) in the forest. As design goals, the sensor nodes surrounding the moving target should promptly provide robust and reliable status information about the mobile target and the region around it

in an energy efficient way, and the network should forward this information to the sinks in a fast and energy efficient way.

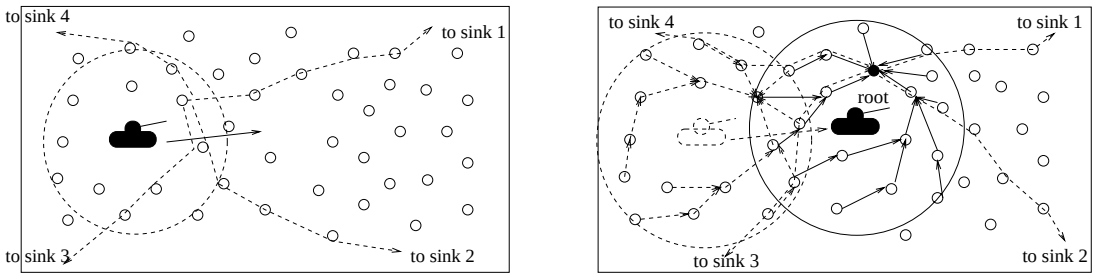


Fig. 2.1. Using convoy tree to track the target

Most existing researches in sensor networks, e.g., the directed diffusion [27, 31], LEACH [24] and TTDD [53], concentrate on finding efficient ways to forward the data report to the data center, and not much work has been done on how to detect the mobile target and generate robust and reliable reports in an energy efficient way. Recently, Chu *et al.* [12, 62] studied the problem of tracking a mobile target using an information-driven approach. However, their approach assumed that a single node close to a target can detect the status of the target, and did not consider the collaboration among nodes that can detect the target at the same time. Because sensor nodes deployed in current sensor networks do not have a large sensing distance, or a high level of sensing accuracy and node reliability, Cerpa *et al.* [10] suggested that multiple nodes surrounding the target should collaborate to make the collected information more complete, reliable and accurate. However, no concrete algorithm was given.

In this thesis, we propose a *Dynamic Convoy Tree-based Collaboration (DCTC)* framework to detect and track the mobile target and monitor its surrounding area. DCTC relies on a tree structure called *convoy tree*[1], which includes sensor nodes around the moving target, and the tree is dynamically configured to add some nodes and prune some nodes as the target moves. Figure 2.1 illustrates how to use the convoy tree to track a mobile target. As the target first enters the detection region, sensor nodes that can detect the target collaborate with each other to select a root and construct an initial convoy tree. Relying on the convoy tree, the root collects information from the sensor nodes and refine these information to obtain more complete and accurate information about the target using some classification algorithms [28, 30]. As the target moves, some nodes in the tree become far away from the target and are pruned. Since most sensor nodes stay asleep before the target arrives for power saving, the root should predict the target moving direction and activate the right group of sensor nodes so that these nodes can detect the target as soon as the target shows up. As the convoy tree reconfigures itself, the root may also need to be changed to optimize the communication overhead. Figure 2.1 (b) shows how the convoy tree reconfigures itself with a new root.

A big challenge of implementing the DCTC framework is how to reconfigure the convoy tree in an energy efficient way as the target moves. To address this problem, we first formalize it as an optimization problem of finding a min-cost convoy tree sequence with high tree coverage, and give an optimal solution (o-DCTC) based on dynamic programming. Considering the constraints of sensor networks, we propose some practical

[1] *We use the word "convoy tree" to differentiate it from the normal tree structure since convoy tree is a moving tree which tracks the target.*

solutions. Specifically, we propose two tree expansion and pruning schemes: the *conservative* scheme and the *prediction-based* scheme, and two tree reconfiguration schemes: the *sequential reconfiguration* and the *localized reconfiguration*. We also evaluate the performance of the optimal solution and the practical implementations through extensive simulations. Based on the simulation results, when the same reconfiguration scheme is used, the prediction-based scheme outperforms the conservative scheme and it can achieve a similar coverage and energy consumption to the optimal solution. When using the same scheme for tree expansion and pruning, the localized reconfiguration scheme outperforms the sequential reconfiguration scheme when the node density is high, and the trend is reversed when the node density is low.

The remainder of this chapter is organized as follows. Section 2.2 describes the system model and the basic framework of DCTC. Section 2.3 presents an optimal DCTC based on dynamic programming. In section 2.4, we propose practical solutions to improve the performance considering the constraints of sensor networks. Section 2.5 evaluates different solutions and compares them to the optimal solution. Section 2.6 concludes the chapter.

2.2 The System Model

A sensor network can be modeled as a graph $G(V, E)$, where each vertex represents a sensor node, and there is an edge between two nodes when they are within each other's communication range. Each node is aware of its own location by using GPS [1] or other techniques such as triangulation [8]. Each sensor node can be in the active mode or sleep mode. In the active mode, the node can sense the target, send and receive data. An active

node can select its transmission power from the following n levels: $u_1 \leq u_2 \leq \cdots \leq u_n$. Corresponding to the power level it selects, it has one of the following communication ranges: $d_1 \leq d_2 \leq \cdots \leq d_n$. Here, the relation between the power level and the communication range satisfies the following energy model [23]: $u_i = a * d_i^2 + c$, where a and c are constants.

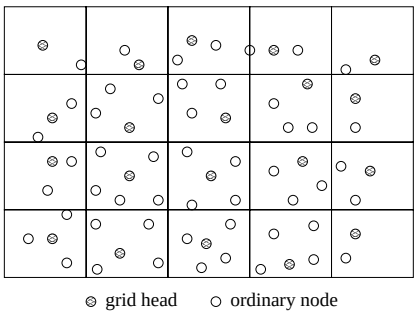


Fig. 2.2. Dividing a sensor network into grids

To save power, the sensor nodes stay in the sleep mode most of the time based on the GAF protocol [52]. As shown in Figure 2.2, the sensor network is divided into grids, where each pair of nodes in neighboring grids can communicate directly with each other. When there is no target close to a grid, only the grid head is required to be awake, and other nodes only need to wake up periodically. Each node maintains some information about the nodes in the same grid, e.g., the locations. Through the discovery process of the GAF protocol, each node can learn the information about other nodes in the same

grid. When a mobile target[2] enters the detection region, the grid head will wake up all nodes inside the grid.

Let t_s (t_e) denote the time when the target enters (leaves) the detection region of the network. At any time t ($t_s \leq t < t_e$), the set of sensor nodes that should participate in detecting the target is denoted as S_t. Note that, S_t is application specific, and may be affected by several factors such as the sensing capacity of sensor nodes, the precision and reliability demands on the sensing results, the size of the target, *etc.* In this paper, we assume that S_t includes all the nodes whose distance to the current location of the target is less than d_s. Also, we call the circle that is centered at the current location of the target and has a radius of d_s as the *monitoring region* of the target at time t.

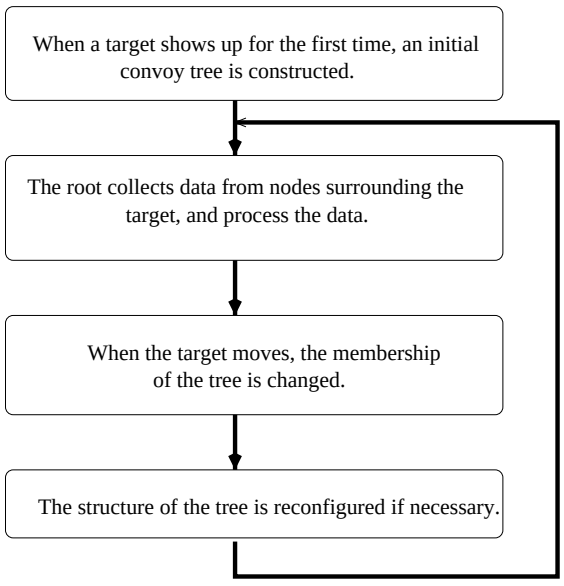


Fig. 2.3. The basic structure of DCTC

[2]In this paper, we only consider the problem of detecting a single mobile target at one time.

Figure 2.3 illustrates the basic design of DCTC. With DCTC, a *convoy tree* of a target is formed as soon as the target enters the detection region of the network. Every certain time interval[3], each node in the tree generates a sensing report and sends the report to its parent, which forwards the report to the root. After the root has received all the reports, it processes them to generate a final report that will be sent to the sinks. From the processing results, the root obtains the current location of the target. Based on this information, it may add some new nodes, prune some existing nodes, or reconfigure itself. The convoy tree at time t is denoted as $T_t(R)$, which is formally defined as follows:

DEFINITION 1. *(convoy tree): Convoy tree* $T_t(R) = \langle V_t, E_t \rangle$, *where* V_t *is the set of nodes that are awake and participate in detecting the target at time* t, E_t *is the set of edges connecting nodes to their parents, and* R *is the root of the tree.*

One important attribute of a tree T_t is the *tree coverage*, which is defined as

$$\frac{\mid V_t \cap S_t \mid}{\mid S_t \mid}.$$

Intuitively, the tree coverage is the number of nodes in the tree as a fraction of the number of nodes that can detect the target. We use $V_t \cap S_t$ instead of V_t, because some nodes in the tree cannot detect the target but haven't been pruned yet. When all nodes that can detect the target are in the tree, the tree coverage reaches 100%. Increasing the tree coverage can increase the number of nodes involved in the target detection, which helps obtain more accurate information about the target. Some nodes may only detect part of the target, or some angle of the target. Some sensor nodes may be malicious or

[3]To make easy the presentation, we assume that time is discrete and the time interval is 1.

have malfunction. In such an environment, redundancy is helpful. With some level of redundancy, the root can obtain more accurate information about the target. Note that the root will remove some redundant information and only send the essential information to the data center. Certainly, the root may fail or even be a malicious node. To deal with this kind of problem, other nodes need to monitor what happens with the root. When the root fails, its neighboring nodes need to collaborate to elect a new one. They may even periodically send some encrypted information to the data center directly if the root is suspected to be malicious. Since this is not the major concern of the paper, we will not further discuss it.

The other important attribute of tree $T_t(R)$ is the energy consumed for collecting the sensing data via this tree. Let r denote the size of the sensing data generated by node i, $e(i, R)$ denote the total energy consumed by transmitting one unit of information from node i to R through the path on the tree. The amount of energy consumed for data collection is defined as

$$E^d(T_t(R)) = r * \sum_{i \in V_t} e(i, R).$$

The convoy tree reconfigures itself as the target moves. A collection of trees at different time interval form a *convoy tree sequence*, which is denoted as

$$\Gamma(t_s, t_e, T_s(R)) = \langle T_s(R_s), T_{t_s+1}(R_{t_s+1}), \cdots, T_{t_e}(R_{t_e}) \rangle,$$

where t_s is the start time of the first data collection process. When the parameter $T_s(R)$ is not necessary, it will be removed for simplicity.

The average tree coverage of $\Gamma(t_s, t_e)$ is defined as

$$\frac{\sum_{t=t_s}^{t_e} \mid V_t \cap S_t \mid / \mid S_t \mid}{t_e - t_s + 1}.$$

The total energy consumption of $\Gamma(t_s, t_e, T_s)$ is defined as

$$E(\Gamma(t_s, t_e, T_s)) = E^d(T_s) + \sum_{t=t_s+1}^{t=t_e} [E^t(T_{t-1}, T_t) + E^d(T_t)]$$

where $E^t(T_{t-1}, T_t)$ is the energy consumed by the transformation from T_{t-1} to T_t. A convoy tree sequence $\Gamma(t_s, t_e, T_s)$ is a *min-cost convoy tree sequence if* $(\forall \Gamma'(t_s, t_e, T_s))$ $(E(\Gamma(t_s, t_e, T_s)) \leq E(\Gamma'(t_s, t_e, T_s)))$.

Generally speaking, there is a tradeoff between tree coverage and energy consumption, and we should not optimize only one of them without considering the constraint of the other. This can be further explained by an example. If there is no constraint on the energy consumption, the tree coverage can reach 100% by adding all nodes to the tree. Similarly, without considering the constraints of the tree coverage, finding a min-cost convoy tree sequence is trivial. However, it is a challenge to find a min-cost convoy tree sequence whose average coverage is larger than a certain value. In the next section, we study a special case of this problem: finding a min-cost convoy tree sequence that can reach 100% coverage, which can only be solvable under ideal situations.

2.3 The Optimal DCTC Scheme

If each node has the information about the network topology, and the moving trace of the target is known, we can design an optimal DCTC (o-DCTC) to find a min-cost tree with the constraint that the tree coverage is 100%. Since the whole target trace is known, the tree coverage can reach 100% by expanding the convoy tree to include all nodes and only those nodes that are included in the monitoring region of the target. Also, each of these nodes can be pruned from the tree as long as they are out of the monitoring region.

Different from the general problem of finding a min-cost convoy tree sequence, in this section, we consider convoy trees that contain and only contain nodes that are within the monitoring region of a target. To help understand o-DCTC, we provide the formal definitions for this special kind of trees and the min-cost convoy tree sequences that contain only this kind of trees.

DEFINITION 2. *(min-convoy tree): A convoy tree $T_t(R)$ is a min-convoy tree if and only if $T_t(R)$ contains and only contains nodes in S_t.*

DEFINITION 3. *(min-convoy tree sequence): A convoy tree sequence $\Gamma(t_s, t_e)$ is a min-convoy tree sequence if and only if $\forall \Gamma'(t_s, t_e)(E(\Gamma(t_s, t_e) \leq E(\Gamma'(t_s, t_e) \wedge$ trees in $\Gamma(t_s, t_e)$ and $\Gamma'(t_s, t_e)$ are all min-convoy trees).*

In the rest of the section, we present a centralized algorithm that uses dynamic programming to compute a min-convoy tree sequence in Figure 2.4, and prove some properties of this algorithm. The algorithm is based on the following recursive relation:

Notations:

- Φ_t: a set of convoy tree sequence.

The *o*-$DCTC$ algorithm:

```
Function DCTC(t, t_e, Φ_t)
1       if t = t_e then
2             find Γ ∈ Φ_t, such that, ∀Γ' ∈ Φ_i(E(Γ) ≤ E(Γ'));
3             return E(Γ);
4       else
5             Φ_{t+1} = ∅;
6             for each T_{t+1} which is a min-convoy tree at time t + 1
7                   min_eng = ∞;
8                   for each Γ_t ∈ Φ_t
9                         append T_{t+1} to Γ_t to get Γ_{t+1};
10                        if E(Γ_{t+1}) ≤ min_eng then
11                              min_eng = E(Γ_{t+1}); Γ' = Γ_{t+1};
12                  Φ_{t+1} = Φ_{t+1} + {Γ'};
13            DCTC(t + 1, t_e, Φ_{t+1})
begin
      DCTC(t_s, t_e, {⟨T_s⟩})
end.
```

Fig. 2.4. The optimal DCTC scheme

$$o\text{-}DCTC(t, t_e, T) = \begin{cases} E^d(T), & \text{if } t = t_e; \\ MIN_{T_{t+1} \in \mathcal{T}_{t+1}}\{E^d(T_t) + E^t(T_t, T_{t+1}) & \\ +o\text{-}DCTC(t+1, t_e, T_{t+1})\}, & \\ & \text{if } t < t_e. \end{cases} \tag{2.1}$$

Here $\mathcal{T}_t$ is the set of all min-convoy trees at time t. We show the correctness of the o-DCTC algorithm by proving that $o\text{-}DCTC(t, t_e, T_t)$ in Equation 2.1 can compute a min-convoy tree sequence starting from T_t.

THEOREM 1. *Let $\Gamma_m(t, t_e, T_t)$ be a min-convoy tree sequence, o - $DCTC(t, t_e, T_t) = E(\Gamma_m(t, t_e, T_t))$, where o-DCTC is defined in Equation (2.1).*

Proof by induction on t.

(1) Base: When $t = t_e$, $E(\Gamma_m(t, t_e, T_t)) = E(\langle T_t \rangle) = E^d(T_t)$, and $o\text{-}DCTC(t, t_e, T_t) = E^d(T_t)$. Therefore, $E(\Gamma_m(t, t_e, T_t)) = o\text{-}DCTC(t, t_e, T_t)$.

(2) Induction: Assume that $o\text{-}DCTC(t, t_e, T_t) = E(\Gamma_m(t, t_e, T_t))$ for $t > t_e - n, n \geq 1$. We want to prove that it holds when $t = t_e - n$.

(a) Let $\Gamma_m(t, t_e, T_t) = \langle T_t, T'_{t+1}, \cdots, T'_{t_e} \rangle$. Thus,

$E(\Gamma_m(t, t_e, T_t))$

$= E^d(T_t) + E^t(T_t, t'_{t+1}) + E(\langle T'_{t+1}, \cdots, T'_{t_e} \rangle)$

$\leq E^d(T_t) + E^t(T_t, t'_{t+1}) + E(\Gamma_m(t+1, t_e, T'_{t+1}))$, where $\Gamma_m(t+1, t_e, T'_{t+1})$ is a min-convoy tree sequence.

$= E^d(T_t) + E^t(T_t, t'_{t+1}) + o\text{-}DCTC(t+1, t_e, T'_{t+1})$, according to the induction assumption.

$\leq o\text{-}DCTC(t, t_e, T_t)$, according to Equation (2.1).

(b) On the other hand,

Let $o\text{-}DCTC(t, t_e, T_t) = E^d(T_t) + E^t(T_t, T''_{t+1}) + o\text{-}DCTC(t+1, t_e, T''_{t+1})$. Thus,

$o\text{-}DCTC(t, t_e, T_t)$

$= E^d(T_t) + E^t(T_t, T''_{t+1}) + E(\Gamma_m(t+1, t_e, T''_{t+1}))$, where $\Gamma_m(t+1, t_e, T''_{t+1})$ is a min-convoy tree sequence.

$= E(\langle T_t, T''_{t+1}, \cdots, T''_{t_e} \rangle)$

$\leq E(\Gamma_m(t, t_e, T_t))$.

Therefore, $\Gamma_m(t, t_e, T_t) = o\text{-}DCTC(t, t_e, T_t)$.

Tree Expansion/Pruning and Reconfiguration in o-DCTC

Till now, we are discussing the o-DCTC scheme in an abstract way, and have not mentioned how to calculate the energy consumed for transforming one tree to another. To do this, we need to know how nodes are added/pruned and how a tree is reconfigured in the ideal situation. Note that the method for calculating the energy consumption for data collection is already defined in the previous section.

First, we introduce an algorithm used to calculate a min-cost convoy tree. It is similar to the *Dijkstra' algorithm* for solving the single-path problem in a weighted graph [13]. The correctness proof is also similar to that provided in [13]. Note that all the notations used in this algorithm have been defined before.

ALGORITHM 1. *MinCostTree(V,e,R)*

1 $Q = V$

2 While $Q \neq \emptyset$

3 *find* $u \in Q$ *such that* $(\forall v \in Q)(e(u,R) \leq e(v,R))$;

4 **for** *each* v *that is a neighbor of* u

5 **if** $e(v,u) + e(u,R) < e(v,R)$ **then**

6 $e(v,R) = e(v,u) + e(u,R)$; *change* v*'s parent to be* u.

7 $Q = Q - \{u\}$.

At time $t = t_s, t_s+1, ..., t_e$, the convoy tree $T_t(R)$ needs to expand itself to include nodes that will detect the target at time $t+1$. The tree expansion procedure is defined as follows:

1. R defines the new set of nodes to join the tree as $S_{new} = S_{t+1} - V_t$.

2. R multicasts a message $wake\_up(L_{t+1})$ to the heads of the grids that contain nodes in S_{new}. Receiving the *wake_up*, the grid head wakes up those nodes.

3. After a node wakes up, it calls $MinCostTree(S_{t+1}, e, R)$ to compute the min-cost convoy tree and joins the tree by attaching to the parent node computed by the algorithm.

The tree pruning process is similar to the tree expansion process. Here, the notification message is called *power_down*, and only nodes in $S_{old} = V_t - S_{t+1}$ are notified. When a node receives the *power_down* message, it leaves the tree and returns to sleep if it is not the grid head.

A tree reconfiguration procedure which transforms $T_{t+1}(R)$ to $T_{t+1}(R')$, where $R' \in V_{t+1}$, is defined as follows:

1. R' multicasts a message $reconf(R', L_{t+1})$ to the heads of the grids that contain the nodes in S_{t+1}. Receiving the message, the grid head broadcasts the message to those nodes.

2. After a node receives $reconf(R', L_{t+1})$, it calls $MinCostTree(S_{t+1}, e, R')$ to compute the min-cost convoy tree and changes its parent to the one computed by the algorithm.

From the description of the above processes, we know that the overhead for reconfiguring a tree (i.e, $E^t(T_t(R), T_{t+1}(R'))$) includes the energy consumed for transmitting the messages *wake_up*, *power_down* and *reconf* during the reconfiguration.

2.4 Design and Implementation of DCTC

Although the o-DCTC scheme can optimize the energy consumption and achieve a 100% tree coverage, it is not practical since it assumes that the moving target trace is known as *a priori* and each node has knowledge about the network topology. In this section, we propose some practical solutions to implement the DCTC framework.

2.4.1 Constructing the Initial Tree

When a target first enters the detection region of a sensor network, the sensor nodes that are awake and close to the target can detect it. Then, these sensor nodes need to construct an initial convoy tree by first selecting a node to be the root of the tree.

Many leader election algorithms [46] can be used to select the root. Since this is not the major concern of this paper, we only present the basic idea of a simple solution, which selects the node closest to the target as the root. If two or more nodes have the same distance to the target, node *id* (can be the MAC address) is used to break the tie. This selection criterion is based on the following intuitive reasons: First, if the root is close to the target, i.e., close to the geographic center of the nodes in the tree, the tree should have a short height and small energy consumption during data collection. Second, since there may be multiple moving sinks [53] distributed in the network, selecting a root that is far away from the sensing nodes may not be a good solution. Note that this is different from some existing work [27] where collected data will be forwarded to the same sink. In some cases, the new root may be some special powerful node, which can cache the data and wait for other nodes to query the data. In this case, the root selection process will be much simpler.

This algorithm has two phases. In the first phase, each node i broadcasts to its neighbors an $election(d_i, id_i)$ message with its distance to the target (d_i) and its own *id*. If a node does not receive any election message with (d_j, id_j) that is smaller than (d_i, id_i), it becomes a root candidate. Otherwise, it gives up and selects the neighbor with the smallest (d_j, id_j) to be its parent. Since the nodes may not be within one-hop communication range of each other[4], two or more root candidates may exist after the first phase. In the second phase, each root candidate i floods a $winner(d_i, id_i)$ message to other nodes that can detect the target. When a root candidate i receives a winner message with (d_j, id_j) that is smaller than (d_i, id_i), it gives up the candidacy

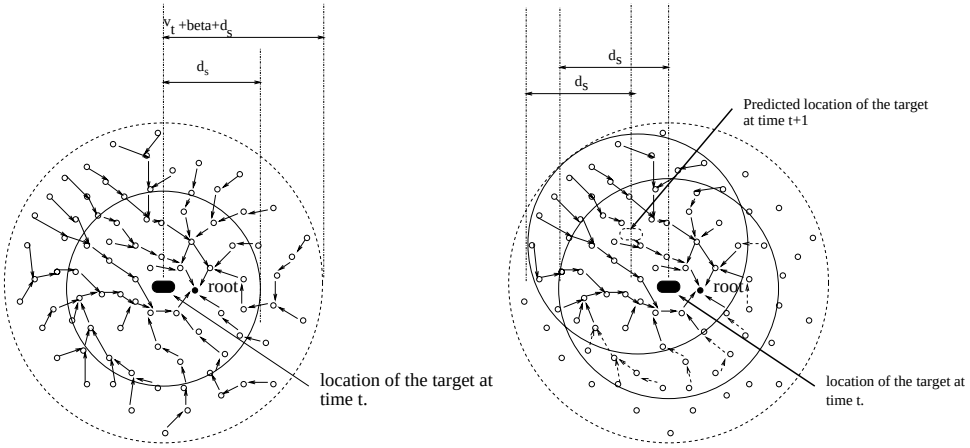
[4]We assume there is no network partition.

and constructs a path to node j by reversing the path on which the winner message is sent. Eventually, all root candidates give up but one with the smallest (d_i, id_i) becomes the root. These election and winner messages are limited to only nodes that can detect the target. The selection process only takes a very short time, so there should not be a case where new nodes keep adding into the root election process. The initial convoy tree constructed in this way may not be a min-cost convoy tree, and may have low coverage. The tree expansion and reconfiguration schemes, which will be presented later, can be used to reconstruct a better convoy tree with high coverage and low cost.

2.4.2 Tree Expansion and Pruning

For each time interval, the root adds some nodes and removes some nodes as the target moves. To identify which node to be added and removed, we study two schemes: the *conservative* scheme and the *prediction-based* scheme.

In the conservative scheme, as shown in Figure 2.5(a), the nodes that should be in the tree are those whose distance to the current location of the target is less than $v_t + \beta + d_s$, where v_t is the current velocity of the target, β is a parameter, and d_s is the radius of the monitoring region. This scheme assumes that the target may move in any direction with a velocity less than $v_t + \beta$. Obviously, β should be large enough in order to achieve a high tree coverage. However, if β is too large, the expanded tree may contain a lot of redundancy; i.e., many nodes which should not be in the tree are added to the tree. This will increase the power consumption since these nodes cannot be in the power save mode.



(a) The conservative scheme (b) The prediction-based scheme

Fig. 2.5. Tree expansion and pruning

The prediction-based scheme is proposed to reduce the redundant nodes added by the conservative scheme. This scheme assumes that certain movement prediction techniques can be used to predict the future location of the target. Therefore, only nodes located within the estimated monitoring region are added to the tree. As shown in Figure 2.5(b), only nodes whose distance to the predicted location are less than d_s are added. Since the prediction may not be accurate, the tree coverage may be reduced. However, considering the target moving direction is not frequently changing, the chance of correctly predicting the target moving direction is high. Hence, the advantage of using prediction outweighs its negative effects. Due to space limit, we do not present the details of the movement prediction technique, which can be found in [32, 3].

After the nodes that should be added to the tree and those that should be pruned from the tree are identified, the root notifies them by sending notification messages to the heads of the grids that contain these nodes. A grid head receiving a prune message

immediately broadcasts it to the nodes in the grid, and the nodes receiving the message leave the tree and return to sleep. When a grid head receives a join message, it wakes up the sensor nodes in its grid. It also requests for the locations and costs of the nodes within its neighboring grids that are closer to the root. After receiving all these information, they are broadcast to the nodes in the grid, which find the appropriate parents to minimize their costs of sending packets to the root. The algorithm for finding the parent is the same as that used in the tree reconfiguration schemes, which will be presented next.

2.4.3 Tree Reconfiguration

2.4.3.1 The Basic Scheme

When the target moves, the sensor nodes that participate in the detection change accordingly. If many nodes become far away from the root, a large amount of energy may be wasted for them to send their sensing data to the root. To reduce the energy consumption, the root should be replaced by a node closer to the center of the monitoring region (i.e., the moving target), and the tree should be reconfigured. Due to the reconfiguration cost, the tree should not be reconfigured too frequently, especially when the current root is still close to the target. In our solution, the reconfiguration is triggered by a simple heuristic, which is based on the distance between the root and the current location of the target. If this distance is larger than a threshold, the tree should be reconfigured; otherwise, it should not. We set the reconfiguration threshold to be

$$d_m + \alpha * v_t$$

where d_m is a parameter to specify the minimum distance that triggers the reconfiguration, v_t is the velocity of the target at time t, and α is a parameter to specify the impact of the velocity.

If the tree should be reconfigured based on the heuristic, a new root should be selected. This new root should be a node close to the current location of the target. When multiple nodes exist, one of them can be randomly chosen. We already explained part of the reason in Section 4.1. Choosing a node close to the target is helpful for constructing a tree with short height and less energy consumption during data collection. This will be a good option when multiple sinks exist, or when the data has to be cached in the network, waiting for other nodes to query the data. The process for root migration is briefly described as follows after the new root has been chosen. The current root first finds the grid that covers the current location of the target. Then, it sends a migration request to the grid head. Receiving the request, the grid head forwards the message to the new root.

After the root has been changed, the tree needs to be reconfigured to minimize the energy consumption for data collection. In Section 3, we presented the $MinCostTree$ algorithm to construct a min-cost convoy tree under the assumption that each node is aware of the locations of all nodes in the tree. Due to the high overhead of maintaining these information, this algorithm may not be suitable for a real sensor network.

Another option is based on broadcasting. In this approach, the new root first broadcasts a $reconf$ message to its neighbors. A node receiving the $reconf$ rebroadcasts the message and attaches its location and the cost of sending a packet to the root. When any other node receives the $reconf$, it waits for certain time interval and collects the

reconf messages sent during this interval. After that, it changes its parent to be a selected neighbor, so that the cost of sending packets to the root via this neighbor is smaller than via any other known neighbors. After this, it broadcasts a *reconf* attaching its location and the cost of sending a packet to the root. This process will continue until all nodes that are within the monitor region are added to the tree. Using this approach, a node does not need to maintain the information about other nodes since the information can be obtained from neighbors during the reconfiguration process. However, the broadcasting overhead is too high, and it is difficult for a node to decide how long it should wait to collect the information from its neighbors. To address this issue, we present two practical tree reconfiguration schemes: the *sequential reconfiguration* scheme, and the *localized reconfiguration* scheme.

2.4.3.2 Sequential Reconfiguration

The proposed sequential reconfiguration scheme is based on the grid structure. The main idea has been illustrated in Figure 2.6. As shown in Figure 2.6(a), the new root (in grid g_0) initiates the reconfiguration by broadcasting a *reconf* message to the nodes in the its own grid, and sending the message to the heads of its neighboring grids (i.e., g_1, g_2, g_3 and g_4). The *reconf* includes information about nodes in its own grid such as the locations of these nodes and the cost of sending data to the root. Having received the *reconf*, the grid head needs to do the following: First, it broadcasts the messages to the nodes in its own grid. Second, based on the information attached with these messages, it finds the parent for each node in the grid, so that the cost of sending packets from the node to the root is minimized. Finally, as shown in Figure 2.6(b), it

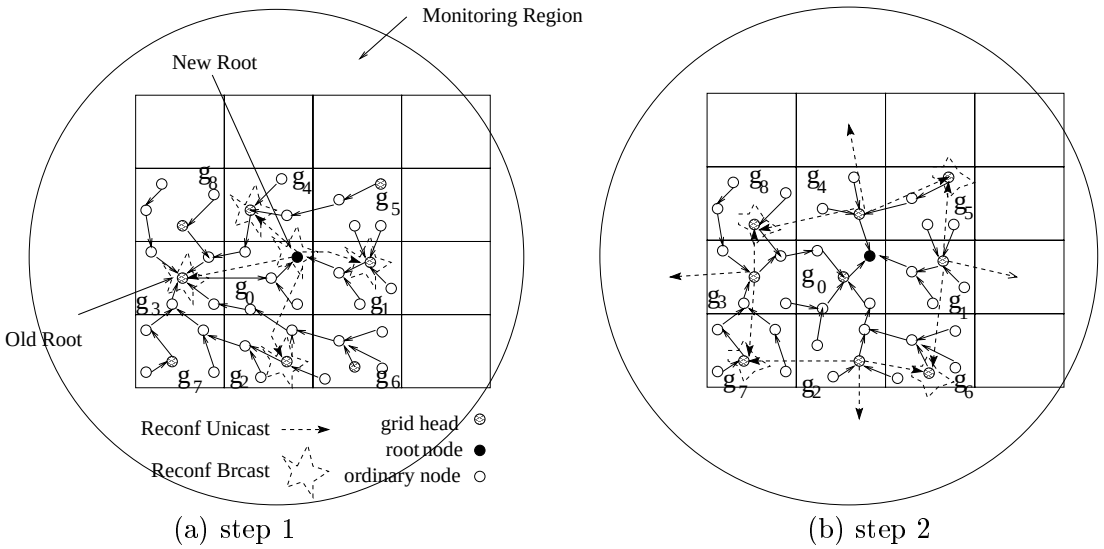


Fig. 2.6. The sequential tree reconfiguration

sends a *reconf* which includes the location and cost of each node in the grid, to the heads of its neighboring grids that are far away from the root, and these grid heads will repeat the same process until all nodes in the monitor region are added to the tree. Based on the property of the grid structure, the grid head of g_0 may not be within the communication range of the grid head of g_5, g_6, g_7, or g_8. Hence it only sends *reconf* to the grid heads of g_1, g_2, g_3, g_4, and let them to further propagate the *reconf* requests. Also, each ordinary node needs to find its parent that can minimize its cost to send packets to the root based on the information gathered from the grid head. A formal description of this algorithm is shown in Figure 2.7.

Notations:

- g_i: a grid. g_0 is the grid of the new root.
- l_i: a list recording the locations of the nodes in grid g_i and the cost for sending packets to the root. Formally, $l_i = \{\langle k, e(k,R)\rangle \mid k \text{ is a node in } g_i\}$.
- $reconf(R, l_i)$: a reconfiguration message sent by the head of g_i.
- $d_{i,j}$: the distance between node i and node j.

Procedure $MinEngPath(R, l_1, l_2)$

Sort items in l_1 in the increasing order of $d_{i,R}$, where $\langle i, e(i,R)\rangle$ is an item in l_1.
for each $\langle i, e(i,R)\rangle \in l_1$
find $\langle j, e(j,R)\rangle \in l_2$, such that $(\forall\langle k, e(k,R)\rangle \in l_2)(e(i,j)+e(j,R) \leq e(i,k)+e(k,R))$;
node i's parent is set to j, $e(i,R) = e(i,j) + e(j,R)$;
$l_2 = l_2 + \{\langle i, e(i,R)\rangle\}$;

The Sequential Tree Reconfiguration Algorithm

(C.1) R sends $reconf$ to nodes in its grid, g_0. Those nodes call $MinEngPath(R, l_0, \emptyset)$ to compute their costs and parents.

(C.2) R sends $reconf(R, l_0)$ to the head of each neighboring grid.

(C.3) When the head of g_i receives $reconf(R, l_j)$ and it has some member nodes within the monitor range:
if g_i has received $reconf$ from any neighboring grids that are closer to R **then**
call $MinEngPath(R, l_i, \bigcup l_j)$
send $reconf(R, l_i)$ to the head of each neighboring grid that is far away from R.

Fig. 2.7. The Sequential Tree Reconfiguration Algorithm

2.4.3.3 Localized Reconfiguration

The sequential reconfiguration algorithm has some drawbacks. During each reconfiguration process, large lists containing the information about all nodes in a grid are transmitted between grids, and are broadcast within grids. This may create significant amount of traffic, especially when the node density is very high or the reconfiguration is frequent. In this section, we propose a localized reconfiguration scheme which provides two optimizations to address this problem.

In the first optimization, we can remove the cost information from the reconfiguration message since a heuristic can be used to estimate the cost, as long as the distance between two nodes is known. Based on the results from [45], the cost of sending one unit of information from node i to node j can be estimated by $\sigma * v(d_{i,j})$, where $\sigma > 1$ is a system parameter used to estimate the relationship between the minimum cost and the real cost. A large σ means a high cost. In our simulations, σ is set to 2.0. $v(x)$ is a function defined as $c * (a/c)^{1/2} * x + a * (a/c)^{-1/2} * x$, where a and c are parameters related to the power model. It has been proved that $v(d_{i,j})$ is the minimum cost for sending one unit of information from node i to node j [45]. To implement this idea, $MinEngPath$ of Figure 2.7 should be modified as follows:

MinEngPath(R, i, l_2**)**

(1) find $\langle j, e(j,R)\rangle \in l_2$, such that

$$(\forall \langle k, e(k,R)\rangle \in l_2)\{e(i,j) + \sigma * v(d_{j,R}) \leq e(i,k) + \sigma * v(d_{k,R})\}$$

(2) node i changes its parent to be j; $e(i,R) = e(i,j) + \sigma * v(d_{j,R})$.

In the second optimization, the information about the node locations can be removed from the messages, if the receiver has got this information about the sender recently. This can be easily implemented by caching the location information about other nodes for some time, and certainly under the assumption that nodes do not move frequently.

Comparing to the sequential reconfiguration scheme, this scheme can greatly reduce the reconfiguration overhead. This overhead reduction can be significant when the node density is high or when the reconfiguration is frequent. However, the tree constructed by this scheme may have high data collection overhead in terms of energy when compared to the sequential reconfiguration, especially when the node density is low. Thus, there is tradeoff between the reconfiguration overhead and data collection overhead, and we will use simulations to evaluate the overall energy consumption of both schemes under different system settings.

2.5 Performance Evaluations

2.5.1 The Simulation Model

We developed a simulator based on $ns2$ (version 2.1b8a) [38], to evaluate and compare the performance of the o-DCTC scheme and the proposed practical solutions. In this simulator, the MAC protocol is based on IEEE 802.11, and the tworay ground propagation model is adopted. The transmission range of each node can be $10m$, $20m$, $30m$ and $40m$, and the transmission power used to support each of the above transmission range is calculated using the model in [23]. Sensor nodes are distributed over a $500 \times$

$500m^2$ flat field, which is divided into $17 \times 17m^2$ grids. In each experiment, 5000 or 1500 sensor nodes are deployed to simulate a dense setting or a sparse setting.

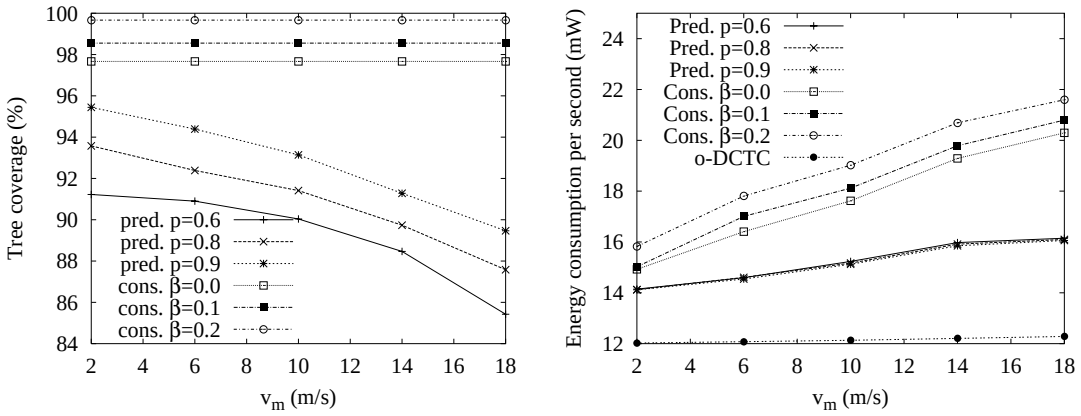
We use a mobility model similar to [3] to simulate the movement of the target. At the beginning of the simulation, the target shows up at a random location on the border of the field with an initial moving direction and velocity. Its moving direction can be one of the following: north (N), northeast (NE), east (E), southeast (SE), south (S), southwest (SW), west (W) and northwest (NW). Its velocity is uniformly distributed between 0 and v_m. Every $10s$, the target may change its moving direction and/or velocity. Specifically, with a probability of 0.6, the direction and velocity of the target keep unchanged. With a probability of 0.05, the moving direction will change 45^o or 90^o, and the velocity will be reselected from $[0, v_m]$. When evaluating the prediction-based scheme, the movement of the target can be predicted with an accuracy of p, varying from 0.6 to 0.9. A mis-prediction happens when the acceleration of the target is changed, but predicted to be unchanged. Table 3.2 lists most of the simulation parameters.

2.5.2 Simulation Results

In this section, we compare different tree expansion, pruning and reconfiguration schemes in terms of tree coverage and energy consumption. In most cases, the 95% confidence interval for the measured data is less than 6% of the sample mean. Only in some cases (Figure 10), the 95% confidence interval is about 10% of the sample mean and has been shown in the figure.

Table 2.1. Simulation Parameters

Parameter	Values
field size (m^2)	500×500
number of nodes	5000 (dense setting) or 1500 (sparse setting)
communication range (m)	10.0, 20.0,30.0 or 40.0
radius of the monitoring region surrounding a target (m)	50.0
simulation time for each scenario (s)	25.0
maximum speed of a mobile target: v_m (m/s)	$2.0 - 20.0$
prediction accuracy (p)	$0.6 - 1.0$
size of a control message ($join$, $prune$, $reconf$) ($byte$)	10
size of the information about a node used in tree expansion and reconfiguration ($byte$)	16 (sequential scheme) or 12 (localized scheme)
size of a sensing report: s_r ($byte$)	40 or 100



(a) Tree coverage (b) Energy Consumption

Fig. 2.8. Comparing the conservative scheme and the prediction-based scheme

2.5.2.1 Comparing the Tree Expansion and Pruning Schemes

We first compare the performance of the conservative scheme, the prediction-based scheme and the o-DCTC scheme in terms of average tree coverage and energy consumption, and then study the impact of some system parameters on the system performance. When comparing the conservative scheme and the prediction-based scheme, the same tree reconfiguration scheme (i.e., the sequential reconfiguration scheme with parameters $d_m = 5m$ and $\alpha = 0.5$) is used.

As shown in Figure 2.8 (a), the conservative scheme achieves larger tree coverage than the prediction-based scheme. However, as shown in Figure 2.8 (b), the prediction-based scheme consumes much less energy than the conservative scheme. This is due to the fact that the conservative scheme conservatively expands a convoy tree to include all nodes whose distance to the current target is less than $d_s + v_t + \beta$, whereas the prediction-based scheme relies on prediction to include only a small group of nodes in the tree. With complete knowledge of the network topology and the movement trace of the target, the o-DCTC scheme expands the convoy tree to include and only include the nodes that are within the monitoring region of the target. Thus, it achieves a coverage of 100%, and consumes less energy than the other two schemes. Although the o-DCTC scheme has the best performance, it is not practical in real sensor networks. Among the two practical schemes, we prefer the prediction scheme, because it saves lots of energy when compared with the conservative scheme, and it can achieve a high tree coverage and low energy consumption close to the o-DCTC scheme.

Both the tree coverage and the energy consumption of the conservative scheme are affected by parameter β. As described in the conservative scheme, β specifies the scope of velocity change that can be tolerated by the scheme, and affects the number of nodes that should be included in the tree. If d_s and v_t are fixed, a large β results in a large tree coverage and large energy consumption. As shown in Figure 2.8, both the coverage and the energy consumption of the conservative scheme increase as β increases. From the description of the conservative scheme, we know that the velocity of the target is another factor affecting the number of nodes that should be included in the convoy tree. If d_s and β are fixed, the number of nodes in the tree increases as v_t increases. This has been verified in Figure 2.8 (b), where the energy consumption of the conservative scheme increases as the velocity increases.

Prediction accuracy affects the tree coverage and the energy consumption of the prediction-based scheme. As shown in Figure 2.8 (b), the tree coverage becomes smaller as the prediction accuracy decreases, since some nodes within the monitoring region of the target may not be included in the tree. Prediction accuracy also affects the energy consumption of the prediction-based scheme. After a mis-prediction is found, nodes that should not be added to the tree will be pruned, and nodes within the monitoring region but not in the tree will be added. Since these operations increase the energy consumption, as shown in Figure 2.8 (b), the energy consumption increases as the prediction accuracy drops.

The velocity of the target affects the tree coverage and the energy consumption of the prediction-based scheme. When the movement direction of a target is mis-predicted, a larger velocity results in a larger distance between the predicted location and the

actual location of the target, and a larger distance in turn results in a larger decrement in the tree coverage. This has been verified by Figure 2.8 (a), where the tree coverage decreases as the velocity increases. On the other hand, as shown in Figure 2.8 (b), when the velocity increases, the membership of the tree changes more frequently and the tree needs to be reconfigured more frequently, which increases the energy consumption.

2.5.2.2 Fine-Tuning the Reconfiguration Scheme

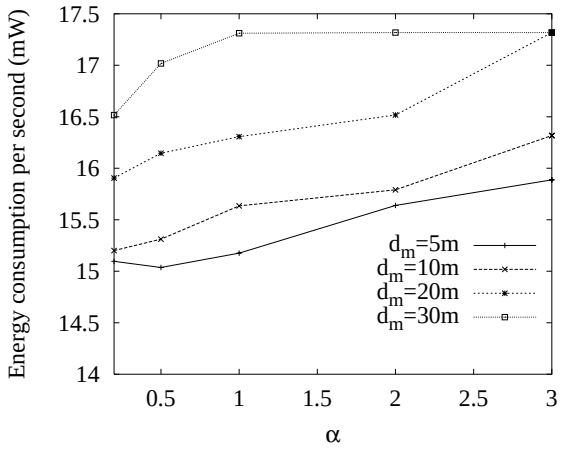


Fig. 2.9. Fine-tuning the sequential reconfiguration scheme

Before evaluating the performance of the tree reconfiguration scheme, we need to choose the appropriate values for parameters d_m and α. Since it is difficult to find the values suitable for all scenarios, we tune the parameters to minimize the energy consumption only in some representative scenarios. In these simulations, the prediction-based scheme (with prediction accuracy of 0.8) is used for tree expansion and pruning.

The results of fine-tuning the sequential reconfiguration scheme are shown in Figure 2.9. We also get similar results when fine-tuning the localized reconfiguration scheme (not shown).

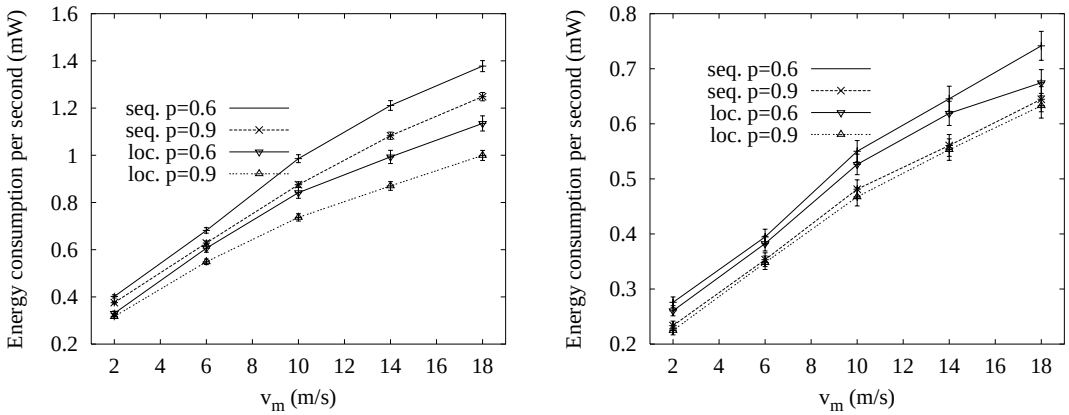
Figure 2.9 shows the results of tuning the sequential scheme when the maximum velocity of the target is $10m/s$ and the sensing report size is $100bytes$. From this figure, we can see that the energy consumption increases as the parameter d_m increases. This is due to the fact that, when d_m is large, the tree can not be reconfigured (optimized) promptly, resulting in large energy consumption for data collection. Due to the same reason, as shown in the figure, the energy consumption increases as α increases, except when both d_m and α are small. The energy consumption is minimized when d_m and α are both small ($d_m = 5m$ and $\alpha = 0.5$ in this case).

2.5.2.3 Comparing the Reconfiguration Schemes

In this subsection, we compare the performance of the sequential reconfiguration scheme and the localized reconfiguration scheme in terms of energy consumption, using the same tree expansion and pruning scheme (i.e., the prediction-based scheme). As we know, the total energy consumption includes three parts: tree expansion and pruning, tree reconfiguration, and data collection. In the following, we compare the energy consumption of the reconfiguration schemes in detail.

Energy consumed by tree expansion and pruning:

Figure 2.10 compares both reconfiguration schemes in terms of the energy consumed by tree expansion and pruning. As the velocity increases, more nodes should be added to or pruned from the tree. This increases the energy consumed by tree expansion



(a) Dense setting (b) Sparse setting

Fig. 2.10. Energy consumption due to tree expansion and pruning

and pruning. Thus, as shown in Figure 2.10, the energy consumption of each scheme increases as the velocity increases. From the figure, we can also see that the energy consumption of the localized scheme is less than the sequential scheme. This is due to the fact that, when adding nodes to the tree, the localized reconfiguration scheme consumes less energy than the sequential reconfiguration scheme, since it requires less information transmitted between grids and less information broadcast within grids. The energy saving becomes more significant when the velocity or the node density increases, since more nodes need to be added to the tree in these two cases.

Energy consumed by tree reconfiguration:

Figure 2.11 compares both schemes in terms of the energy consumed by tree reconfiguration. When the velocity of the target increases, as shown in Figure 2.11, the energy consumed by tree reconfiguration also increases because the tree is reconfigured more frequently. From Figure 2.11, we can also see that, the localized scheme consumes

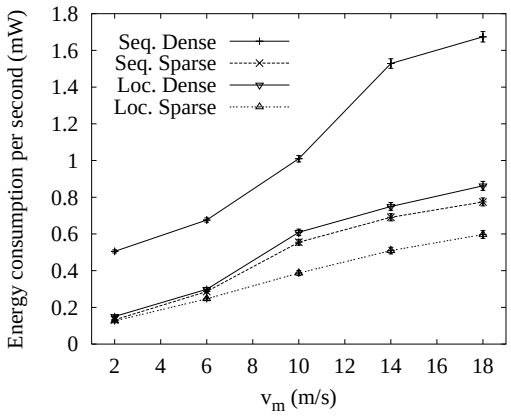


Fig. 2.11. Comparing the energy consumption due to tree reconfiguration

less energy for tree reconfiguration than the sequential scheme, since it requires less information transmitted between grids and less information broadcast within grids.

Energy consumed by data collection:

Table 2.2 compares both reconfiguration schemes in terms of the energy consumed by data collection. In the sequential reconfiguration scheme, each node is aware of the cost of the nodes in its own grid and its neighboring grids that are close to the root. Based on the knowledge, this scheme guarantees that each node can find an optimal path that has the minimum cost of sending reports to the root. However, when using the localized scheme, a heuristic is used for a node to find its parent node, and this approach can not guarantee that each node can find the optimal path that has the minimum cost of sending reports to the root. Thus, the data collection part consumes less energy in the sequential reconfiguration than that in the localized scheme.

The performance difference between these two schemes is affected by the node density. Under dense setting, the difference is small because the heuristic used in the

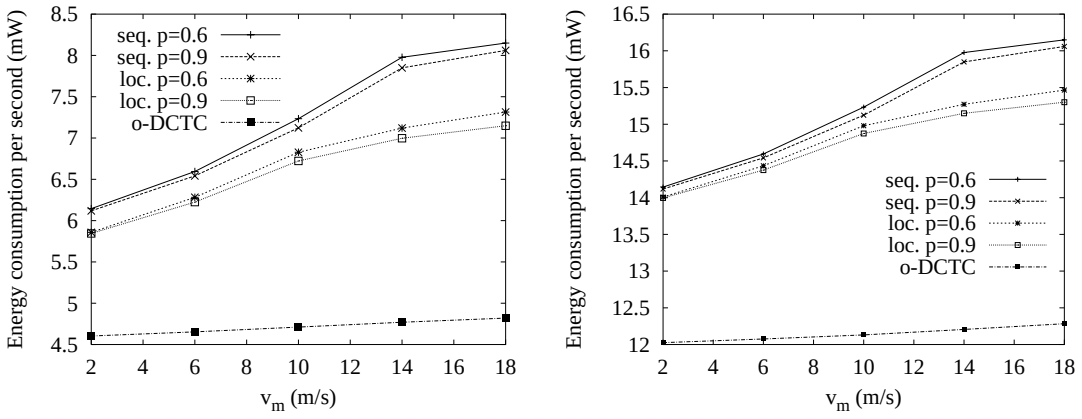
Table 2.2. Energy consumption per second for data collection (mW)

	The Sequential Reconfiguration	The Localized Reconfiguration
Dense, $s_r = 40bytes$	5.137884 ± 0.036342	5.377100 ± 0.038034
Dense, $s_r = 100bytes$	13.137774 ± 0.092927	13.528830 ± 0.095693
Sparse, $s_r = 40bytes$	2.300998 ± 0.048827	2.729444 ± 0.057918
Sparse, $s_r = 100bytes$	5.733158 ± 0.121657	6.577812 ± 0.139580

localized scheme is effective, due to the reasons explained in Section 2.4.3.3. As shown in Table 2.2, the localized scheme consumes only $3\% - 5\%$ more energy than the sequential scheme. However, the difference becomes larger under sparse setting, where the localized scheme consumes $15\% - 20\%$ more energy than the sequential scheme.

Overall energy consumption:

We now compare both reconfiguration schemes in terms of the overall energy consumption. Under dense setting, as shown in Figure 2.12 (a) and Figure 2.12 (b), the performance of the localized scheme is better than the sequential scheme, because the localized scheme significantly outperforms the sequential scheme in terms of the energy consumption for tree expansion and pruning (as shown in Figure 2.10 (a)) and tree reconfiguration (as shown in Figure 2.11). Although the sequential scheme outperforms the localized scheme in terms of the energy consumed by data collection (as shown in Table 2.2), the difference is small. These figures also show that the energy saving becomes larger as the velocity of the target increases, because the energy saving of the localized reconfiguration scheme for tree expansion, pruning and tree reconfiguration increases as



(a) Dense setting, $s_r = 40bytes$ (b) Dense setting, $s_r = 100bytes$

(c) Sparse setting, $s_r = 40bytes$ (d) Sparse setting, $s_r = 100bytes$

Fig. 2.12. Comparing the energy consumption of the sequence reconfiguration and the localized reconfiguration

the velocity gets larger, and the energy consumed by data collection is not affected by velocity changes.

Figure 2.12 (c) and Figure 2.12 (d) show the results under sparse setting. Different from the results under dense setting, the sequential reconfiguration scheme outperforms the localized reconfiguration scheme, because the localized scheme consumes much larger energy for data collection ($10\% - 18\%$) than the sequential scheme. At the same time, the energy saving of the localized scheme for tree expansion/pruning (as shown in Figure 2.10 (b)) and tree reconfiguration (as shown in Figure 2.11) is small. However, if the velocity is large and the sensing report size is small, as shown in Figure 2.12 (c), the performance difference between the localized scheme and the sequential scheme becomes less significant, because the localized scheme can save large amount of energy in tree reconfiguration when the velocity is large (as shown in Figure 2.11).

2.6 Summary of Dynamic Convoy Tree-based Collaboration

In this chapter, we proposed the dynamic convoy tree-based collaboration (DCTC) framework, which can be applied to detect and track a moving target. As the target moves, how to reconfigure the convoy tree in an efficient way is a big challenge in implementing DCTC. We formalized the problem as an optimization problem which needs to find a convoy tree sequence with high tree coverage and low energy consumption. We first proposed an optimal solution based on dynamic programming. Considering the real constraints of a sensor network, we then proposed several practical implementation techniques. For tree expansion and pruning, we proposed two schemes: the conservative

scheme and the prediction-based scheme. For tree reconfiguration, the sequential reconfiguration scheme and the localized reconfiguration scheme were presented. Simulation results showed that the prediction-based scheme outperforms the conservative scheme, and it can achieve a relatively high coverage and low energy consumption close to the optimal solution. When the same tree expansion and pruning scheme is used, the localized reconfiguration performs better when the node density is high, and the trend is reversed when the node density is low.

Chapter 3

Data Dissemination with Ring-Based Index for Wireless Sensor Networks

3.1 Introduction

In the past several years, many data dissemination schemes [23, 27, 53, 39, 20, 21] have been proposed for sensor networks. One widely adopted scheme is the *external storage-based (ES)* data dissemination. It relies on a centralized base station, which is external to the sensor network, for collecting and storing sensing data. In this scheme, sensor data are always collected to the base station from the network no matter they are queried or not. In addition, if many queries are issued from nodes within the network [53], the data must be sent back and forth between the sensors and the base station. Ratnasamy *et al.* [39] proposed a *data-centric storage-based (DCS)* data dissemination scheme, in which the sensing data are stored at certain nodes within the network. In this scheme, however, data are still pushed in a predefined manner regardless of queries. Hence, it lacks flexibility and may introduce many unnecessary data transfers when the querying rate is low.

To avoid unnecessarily transferring the sensing data, *local storage-based (LS)* data dissemination schemes, e.g., directed diffusion [27] and two-tier data dissemination (TTDD) [53], have been proposed. In these schemes, a source sends data to a sink only when the sink has sent a query for the data. These schemes need a sink-source

matching mechanism to facilitate a sink to find the source holding the data of interest. The matching mechanisms adopted by most LS schemes follow a *flood-response* pattern [21], which inherently needs to flood certain control messages. For example, in directed diffusion, a sink floods its query over the whole network; the source(s) with the requested data then knows where to send the data. In TTDD, the source detecting a certain event floods the advertisements of the event to the network, and the sinks interested in the event can send their queries directly to the source. Considering the large number of nodes in a sensor network, the network-wide flooding may introduce significant traffic.

Due to the drawbacks mentioned above, most existing data dissemination schemes, which rely on an external base station, or advertise the availability of data, may not work well in large scale sensor networks, especially in scenarios where a large amount of sensing data are generated, but only a small portion of them will be queried. To address this problem, we propose an index-based data dissemination scheme. In this scheme, the sensing data of an event are stored at the detecting nodes themselves or some nodes close to them (these nodes are called *storing nodes*). A storing node only sends data to a sink when it receives a query from the sink. Also, the location information (called *index*) of the storing nodes are pushed to and maintained at some nodes (called *index nodes*) based on the event type related to the stored data. Hence, queries for a particular event are routed to the appropriate index nodes. The index-based scheme is more attractive than the existing data dissemination schemes since it avoids both unnecessarily transferring the sensing data and flooding control messages to the whole network. Certainly, this scheme introduces additional overhead for maintaining index

nodes. However, as demonstrated by the analysis and evaluation results in this chapter, this scheme can improve the overall system performance.

One major challenge of implementing the index-based data dissemination is how to maintain the indices in the network, such that the indices are highly accessible to sinks and the index nodes are not overloaded. To achieve these goals, we propose an implementation based on the *ring* structure, and call it the *Adaptive Ring-based Index (ARI)* scheme [58]. In ARI, the index nodes for a particular event type are a set of nodes surrounding one or more particular locations (called *index centers* of the event type). The index centers are determined by applying a predefined hash function, e.g., Geographic Hash Table (GHT) [39], on the event type. Note that the hash function maps an event type to one or more locations within the detecting region of the sensor network. The index nodes for the same event type are connected via some forwarding nodes to form a ring (called *index ring*). The number and locations of the index nodes on an index ring, as well as the shape of the ring, can be adaptively changed to achieve load balance and optimize the system performance.

With ARI, a source needs to update its location at the index nodes whenever its location changes. If a target frequently moves, its source node also frequently changes, and the overhead of index updating may become very large. Furthermore, most of the updates may be useless if the source is seldom queried. On the other hand, some sinks may continuously monitoring a moving target by frequently querying the source associated with the target. Using ARI, the sink needs to query the index nodes before querying the storing node. Therefore, the overhead of query can also be large. To deal with the above problems, we further propose a *lazy index updating (LIU)* mechanism

and a *lazy index query (LIQ)* mechanism to reduce the overhead of index updating and index querying, and hence improve the overall system performance.

Extensive analysis and simulations are conducted to analyze and evaluate the performance of the proposed index-based data dissemination scheme, and its performance is compared to the existing data dissemination schemes. The results show that, the index-based scheme outperforms external storage-based schemes, the DCS scheme, and local storage-based schemes in many scenarios. The results also show that using the ARI scheme can tolerate clustering failures and achieve load balance, and the proposed optimization mechanisms can further improve the system performance.

The rest of the chapter is organized as follows. Section 3.2 presents the system model. Section 3.3 describes the index-based data dissemination scheme and analyzes the overhead of several data dissemination schemes. Section 3.4 describes the proposed ARI scheme in detail, and Section 3.5 presents some enhancements to ARI. Performance evaluations are presented in Section 3.6, and Section 3.7 concludes the chapter.

3.2 System Model

We consider an application scenario as follows: A sensor network is deployed for monitoring many targets moving within a vast region. The sensor nodes in the network detect the status of each target, and periodically generate sensing data. Many users are also moving within the region. From time to time, a user may issue a query via a sensor node (sink) for data about the current status of a target.

We assume that the sensor nodes are stationary, and are aware of their own locations using GPS [1] or other techniques such as triangulation [8]. To save power, the

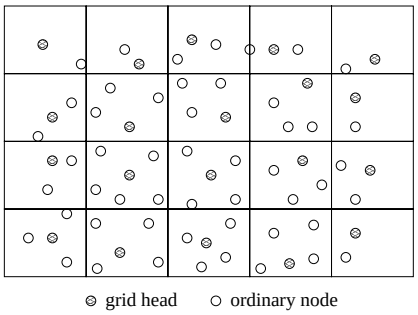


Fig. 3.1. Dividing a sensor network into grids

nodes stay in the sleep mode most of the time based on the Geographical Adaptive Fidelity (GAF) protocol [52]. Using this protocol, as shown in Figure 3.1, the sensor network is divided into grids, where each pair of nodes in neighboring grids can communicate directly with each other. Grid heads are responsible for forwarding the messages, and other nodes only need to wake up periodically.

Our proposed data dissemination scheme is built on top of Greedy Perimeter Stateless Routing (GPSR) [7, 29], a well-known geographic routing system for multi-hop wireless networks. GPSR uses two distinct algorithms for routing. One is a greedy forwarding algorithm which forwards packets progressively closer to the destination at each hop. The other is a perimeter forwarding algorithm that forwards packets where greedy forwarding is impossible. In this system, which uses both the GAF protocol and the GPSR protocol, packets are forwarded by grid heads. A grid head first tries to forward a packet to its neighboring grid head that is closest to the destination. On failing to find such a node, it forwards the packet to one of the neighboring grid heads based on the perimeter forwarding algorithm.

In our proposed data dissemination scheme, the targets of interest to the users are classified into several types, each of which has a unique key. With a certain hash function $H(.)$, a key is mapped to one or more locations within the detecting region of the sensor network. Formally, $(\forall e \in E)H(e) = L_e \in \mathcal{R}$, where E is the set of types and $\mathcal{R}$ is the detecting region.

3.3 The Index-Based Data Dissemination

In this section, we first describe the basic idea of the index-based data dissemination. Then, we analytically compare the overhead of different data dissemination schemes, and identify the scenarios in which the index-based scheme performs the best.

3.3.1 Basic Idea of the Index-based Data Dissemination

The basic idea of the index-based data dissemination is illustrated in Figure 3.2 and is described as follows: A node detecting a target periodically generates sensing data about the target, and stores the data in a storing node, which can be itself or some node nearby. When the target moves, as shown in Figure 3.2 (b) and (c), the detecting node is changed accordingly. However, the new detecting node still stores the sensing data at the same storing node, until the storing node is far away from the detecting node, and then a new storing node is selected. When a new storing node of a target has been selected, the old storing node processes the data it previously stored and generates a summary of small size; sends the summary to the new storing node. Note that the aggregation is not required by the index-based data dissemination scheme. If the previous data are not summarized, the new storing node may keep a pointer which points to the old one, such

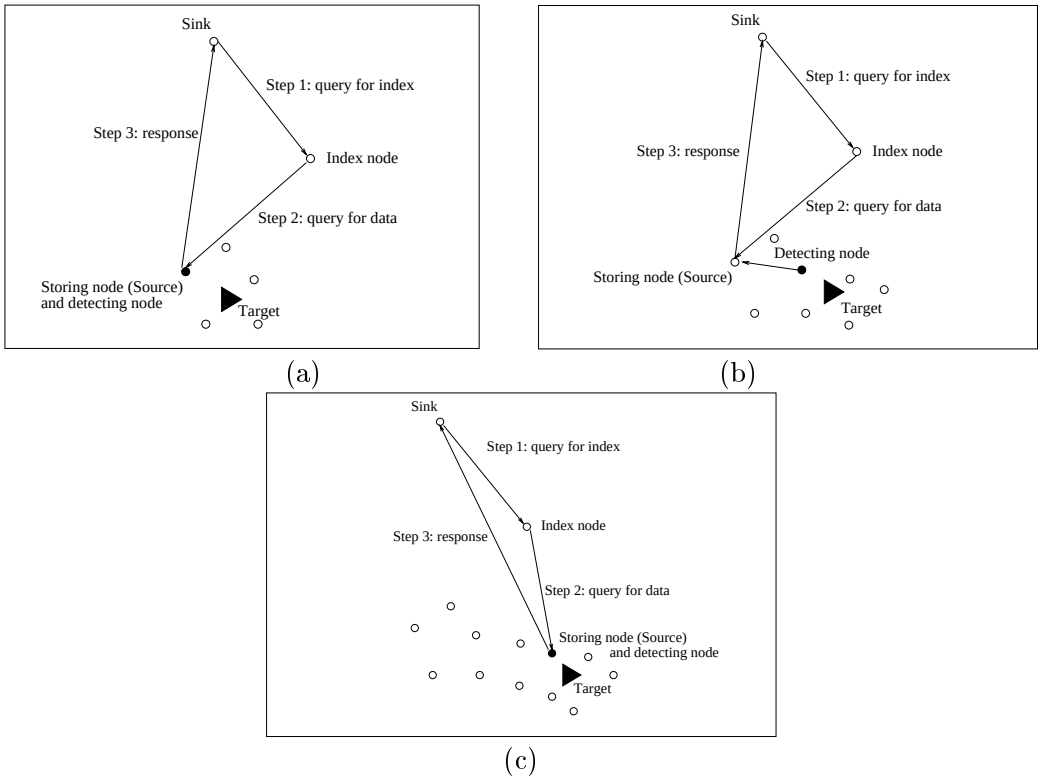


Fig. 3.2. Index-based data dissemination

that the previous data can also be accessed from the current storing node. Also, the new storing node should register its location at the index nodes for the target. When a sink wants to query the sensing data of a target, it sends a query message to an index node for the desired target. On receiving the message, the index node forwards the request to the storing node which sends a response directly to the querying sink.

3.3.2 Analytical Comparison of the Data Dissemination Methods

To compare the overhead of different data dissemination schemes, we introduce the following notations:

- N: the total number of sensor nodes in the network. Similar to [39, 20], we use $O(N)$ to represent the message complexity of flooding a message to the whole network, and use $O(\sqrt{N})$ to represent the message complexity of sending a message between two nodes in the network.

- T, n: T is the number of types, and n is the number of targets of each type.

- r_d, r_q, r_c, r_s: r_d is the rate of updating the sensing data of a target. r_q is the rate of querying a target. r_c is the rate that the detecting node of a target is changed. r_s is the rate that the storing node of a target is changed, where $r_d \geq r_c \geq r_s$. In most cases, $r_d > r_c > r_s$.

- s_d, s_q, s_i: s_d is the size of a data report, s_q is the size of a query message, and s_i is the size of an index update message. In most cases, $s_d \gg s_q$ and $s_d \gg s_i$.

We consider the following metrics when comparing the overhead of the data dissemination schemes:

- **Overall message complexity**: the number of messages generated in the whole network.
- **Hotspot message complexity**: the maximum number of messages sent, received and forwarded by one single node in the network.
- **Overall traffic**: the amount of data transferred in the whole network.
- **Hotspot traffic**: the maximum amount of data sent, received, and forwarded by one single node in the network.

Table 3.1. Estimating the overhead of the data dissemination schemes

Scheme	Overall message complexity	Hotspot message complexity
ES	(1.1) $O((\sqrt{N} * r_d + 2 * \sqrt{N} * r_q) * T * n)$	(1.2) $O((r_d + 2 * r_q) * T * n)$
DCS	(2.1) $O((\sqrt{N} * r_d + 2 * \sqrt{N} * r_q) * T * n)$	(2.2) $O((r_d + 2 * r_q) * n)$
LS	(3.1) $O((N * r_c + 2 * \sqrt{N} * r_q) * T * n)$	(3.2) $O(r_c * T * n + 2 * r_q)$
Index	(4.1) $O((\sqrt{N} * r_s + 3 * \sqrt{N} * r_q) * T * n)$	(4.2) $O((r_s + 2 * r_q) * n)$
Scheme	Overall traffic	Hotspot traffic
ES	(1.3) $O(\{\sqrt{N} * r_d * s_d + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(1.4) $O(\{r_d * s_d + r_q * (s_q + s_d)\} * T * n)$
DCS	(2.3) $O(\{\sqrt{N} * r_d * s_d + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(2.4) $O(\{r_d * s_d + r_q * (s_q + s_d)\} * n)$
LS	(3.3) $O(\{N * r_c * s_i + \sqrt{N} * r_q * (s_q + s_d)\} * T * n)$	(3.4) $O(r_c * s_i * T * n + r_q * (s_q + s_d))$
Index	(4.3) $O(\{\sqrt{N} * r_s * s_i + \sqrt{N} * r_q * (2 * s_q + s_d)\} * T * n)$	(4.4) $O(\{r_s * s_i + r_q * (s_q + s_d)\} * n)$

According to the basic operations of the index-based scheme and the other data dissemination schemes [39], we can estimate the overhead of the data dissemination

schemes in terms of the overall message complexity, hotspot message complexity, overall traffic and hotspot traffic. The estimated results are shown in Table 3.1.

Based on the estimations, we can obtain the following observations:

Observation 1: The index-based scheme has smaller overall message complexity than the other schemes, if:

(1) N is large enough.

(2) $r_s + r_q < r_d$, which can be derived by comparing (4.1) to (1.1) and (2.1).

Observation 2: The index-based scheme has smaller hotspot message complexity than the other schemes, if

(1) $\frac{r_q}{r_c} < T/2$. This relationship is derived as follows:

Let $(r_s + 2 * r_q) * n < r_c * T * n + 2 * r_q$ and $r_s = c * r_c$, where $c < 1$.

Thus, $\frac{r_q}{r_c} < \frac{n*(T-c)}{2*(n-1)} \approx T/2$.

(2) $r_s < r_d$, which can also be derived by comparing (4.2) to (1.2) and (2.2).

Observation 3: The index-based scheme has smaller overall traffic than the other schemes, if:

(1) N is large enough.

(2) $r_s * s_i + r_q * s_q < r_d * s_d$, which can be derived by comparing (4.3) to (1.3) and (2.3).

Observation 4: The index-based scheme has smaller hotspot traffic than the other schemes, if:

(1) $\frac{r_q}{r_c} < \frac{s_i * T}{s_q + s_d}$. This relationship is derived as follows:

Let $\{r_s * s_i + r_q * (s_q + s_d)\} * n < r_c * s_i * T * n + r_q * (s_q + s_d)$ and $r_s = c * r_c$, where $c < 1$.

Thus, $\frac{r_q}{r_c} < \frac{s_i * n * (T-c)}{(s_q + s_d) * (n-1)} \approx \frac{s_i * T}{s_q + s_d}$.

(2) $r_s * s_i < r_d * s_d$, which can be derived by comparing (4.4) to (1.4) and (2.4).

3.4 An Adaptive Ring-based Index (ARI) Scheme

We now propose an adaptive ring-based index (ARI) scheme. We first present the motivations of the scheme, and then describe the operations including index querying/updating, failure management, and adaptive ring reconfiguration in detail.

3.4.1 Motivation

The index-based data dissemination is similar to the problem of locating contents in the peer-to-peer systems such as Chord [44], Pastry [42], Tapestry [61], CAN [40], *etc.* In theses systems, each node is assigned a numerical or a d-dimensional identifier, and it is responsible for owning pointers to objects (e.g., files), whose identifiers map to the node's identifier or region. The set of object identifiers assigned to a nodes is changed as some nodes join or leave the system. Compared with these systems, sensor networks have their unique characteristics. For example, sensor nodes do not have unique identifiers, but can be aware of their own locations. Also, nodes are prone to failures and are constrained in energy supply. Consequently, one challenge of implementing the

index-based data dissemination is how to distribute, update, maintain and query the indices in the network, such that the following requirements are satisfied:

- Fault tolerance: Since sensor nodes are prone to failures [2], we should not rely on a single node to maintain an index. Techniques such as replication should be employed to achieve a certain level of fault tolerance.
- Load balance: Some index nodes may become overloaded. The scheme should remove the overloaded index nodes and add some lightly-loaded index nodes.
- Efficiency: The scheme should not introduce too much overhead since bandwidth and energy are scarce resources in a sensor network.

One simple scheme can be derived directly from the basic DCS scheme. In this scheme, the index nodes for a target are the nodes which form the smallest perimeter surrounding the location calculated. However, the index nodes can not be changed except when they are failed, and all queries and index updates for the target are processed by these nodes. Hence, the index nodes may become overloaded soon, especially when the query rate or the index update rate for the target is very high. Also, since the index nodes for an event type are close to each other, the scheme can not tolerate clustering failures, which may destroy all the index nodes for an event type. A hierarchical scheme similar to the structured replication DCS (SR-DCS) [39] or the resilient DCS (R-DCS) [20] can be used to tolerate clustering failures. In these schemes, the whole detecting region is divided into several subregions, and there is one index node in each subregion. Using these schemes, the index nodes in some subregion can still be overloaded if the number of queries issued from that subregion is very high. In addition, a query may

be routed through several levels of index nodes before reaching the valid index node. Tree-based replication of indices [41], which is already applied in peer-to-peer networks, may also be used in sensor networks. However, this scheme may introduce significant maintenance overhead. For example, the failure of the root or some intermediate nodes in the tree may cause the tree to be reconfigured. Furthermore, the nodes should be aware of the addition, removal and migration of index nodes. Thus, it is a challenge to design index-based schemes to satisfy the fault tolerance, load balance and efficiency requirements.

3.4.2 The ARI Scheme

To achieve the goals of fault tolerance, load balance and efficiency, we propose the *Adaptive Ring-based Index (ARI)* scheme. The ARI scheme is based on the structure of ring, which includes the index nodes for the targets of the same type, and the forwarding nodes that connect the index nodes. The basic modules in the ARI scheme are ring initialization, index querying, index updating, failure management and ring reconfiguration, which will be explained in this section.

3.4.2.1 Initializing an Index Ring

We first describe how to initialize an index ring for the targets of a certain type. As mentioned in Section 3.2, the index center for the target is calculated using a hash function, which maps a target to a location within the detecting region. Initially, the index nodes for the targets are m selected nodes whose distance to the index center is equal to or larger than r, where m and r are system parameters. These nodes are

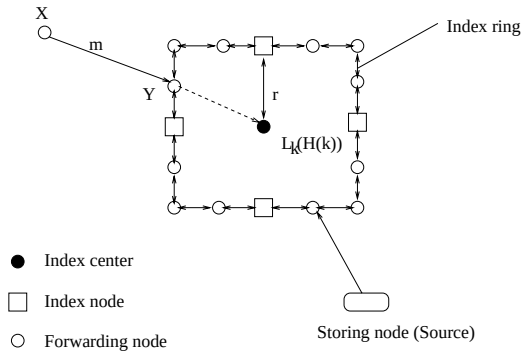


Fig. 3.3. Initializing an Index Ring

connected via some forwarding nodes to form an index ring surrounding the index center. Each node has two pointers which point to its clockwise neighbor and its counterclockwise neighbor on the ring respectively. The index ring must satisfy the following **conditions**:

(*C1*) The distance between the index center and any node on the ring should be larger than or equal to r.

(*C2*) Any message sent by a node outside of the ring-encircled region and destined to the index center, will be intercepted by some node on the ring.

Figure 3.3 shows an example of an initial index ring, where $m = 4$. The ring satisfies (*C1*), since the distance between each node on the ring and the index center is at least r. Due to the assumption that each node can only forward messages to the nodes in its neighboring grids (refer to Section 3.2), a message sent by a node outside of the ring-encircled region and destined to the index center, must pass some nodes on the ring. For example, as shown in Figure 3.3, message m sent by node X can be intercepted by node Y. Thus, the ring also satisfies (*C2*). Note that, the index ring may be changed

later, due to failures or load balance, but the resulted ring **must also** satisfy the above two conditions.

Parameters m and r affect the system performance. As m becomes large, the number of initial index nodes increases. Hence, the overhead for querying is decreased, at the cost that the overhead for storing and updating index is increased. When r becomes large, the overhead for queries issued by nodes far away from the index center is reduced. Also, more nodes are included in the ring, which improves the fault tolerance level. However, the overhead for queries issued by nodes within the region encircled by the ring is increased, as is the overhead for index update.

3.4.2.2 Querying and Updating an Index

The sink S_i can query the index of a target as follows: it first calculates the distance to the index center (denoted as L_k) of the target. If the distance is larger than r, the query message is forwarded along the direction of $S_i \rightarrow L_k$. Otherwise, the message is forwarded along the direction of $L_k \rightarrow S_i$. When a node receives a message, it performs differently based on the following cases:

(1) If the node is an index node for the queried target: the query is forwarded to the storing node of the target if there exists such a storing node.

(2) If the node is a forwarding node on the index ring for the queried target: the query is forwarded to its clockwise neighboring node on the ring.

(3) If the forwarding direction of the message is $S_i \rightarrow L_k$, and the distance between the node and the index center of the target is smaller than r: the forwarding direction

of the query message is changed to be $L_k \rightarrow S_i$, and the message is forwarded in the new direction.

(4) Otherwise: the message is forwarded in the specified direction.

Next, we use Figure 3.4 to further illustrate the querying process. Figure 3.4 (a) shows the case in which sink S_i is outside of the region encircled by the index ring of the queried target. In this case, the query issued by the sink is forwarded along the direction of $S_i \rightarrow L_k$, until it is intercepted by some node on the index ring. The node intercepting the query passes the message on the index ring in the clockwise direction, and the query is finally received and served by an index node, which further forwards the message to the storing node of the queried target.

Figure 3.4 (b) shows the case in which S_i is within the ring-encircled region, and its distance to L_k is smaller than r. In this case, the query issued by the sink is forwarded along the direction of $L_k \rightarrow S_i$ before it is intercepted by some node on the ring. The subsequent process is the same as the previous case.

Figure 3.4 (c) shows a complicated case, where S_i is inside the ring-encircled region, and its distance to L_k is larger than r. In this case, the query is first forwarded along the direction of $S_i \rightarrow L_k$. When the message arrives at a node whose distance to L_k is less than r, the forwarding direction of the message is changed to $L_k \rightarrow S_i$. The subsequent process is the same as the previous case.

When the storing node of a target changes, the location of the new storing node must be updated at the index nodes for the target. The process of transferring an index update message to the index ring is similar to that of transferring an index query

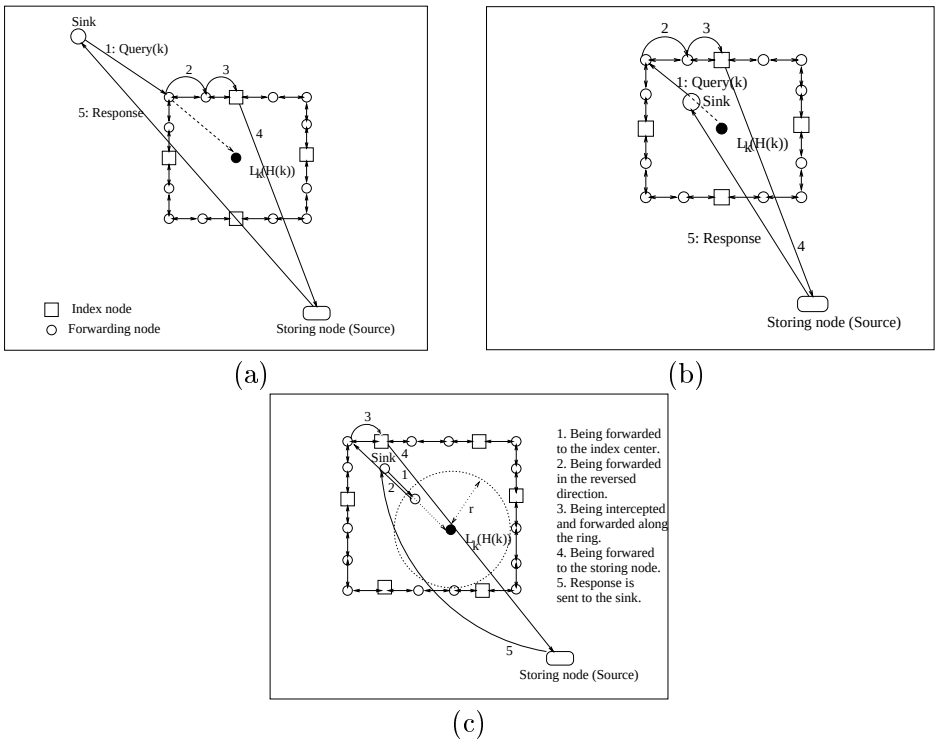


Fig. 3.4. Querying an index

message. When the message arrives at an index node on the ring, the node updates its index, and forwards the message along the circle in the clockwise direction. If the node is a forwarding node, it is simply sent to the counterclockwise neighbor. The message is dropped when it is forwarded back to a node that has already received it.

3.4.2.3 Dealing with Node Failures

To detect the failures of index nodes, neighboring nodes on an index ring should monitor each other by exchanging beacon messages periodically. In this thesis, we consider the following failures:

Individual Node Failures:

Due to energy depletion or hardware faults, an individual node may fail. This kind of failures can be tolerated by the GAF protocol, in which a failed grid head is detected and replaced by other nodes in the same grid. When a new grid head is elected after the old grid head (which is on an index ring) fails, the information (e.g., some indices and pointers) held by the old grid head is lost. However, the new head can receive beacon messages from its neighboring nodes on the ring. From these messages, it knows that it is a node on the index ring, and can get the lost information from the neighbors.

Clustering Failures:

Due to some environmental reasons, the nodes within a certain region may all fail or be isolated. This kind of failures is called *clustering failures*. Clustering failures may break an index ring. Figure 3.5 shows a scenario where an index ring is broken after a clustering failure occurs. In this scenario, nodes B, C and D are failed and can not be replaced by other nodes in their grids, since all the nodes in these grids are dead.

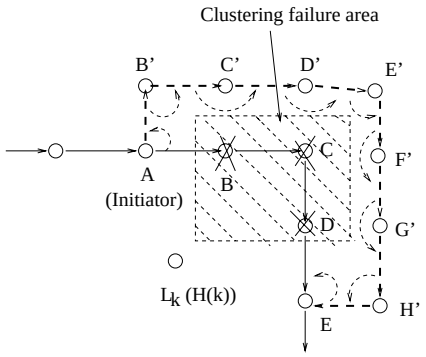


Fig. 3.5. Dealing with clustering failures

The failure can be detected by the neighbors of the failed nodes (i.e., nodes A and E), who do not receive beacon messages from the failed nodes for certain time interval. The counterclockwise neighbor (i.e., node A) of the failed nodes initiates a process to repair the ring.

In the repairing process, the initiator A sends out a *recovery* message, and the message is forwarded around the failed region based on the *right hand rule* [29], until the message arrives at a functioning node on the ring. The process is depicted in Figure 3.5, and is further explained as follows: The initiator A forwards the *recovery* message to B', which is the next neighbor of A and is sequentially counterclockwise about A from edge $\langle A, B\rangle$. On receiving the message, B' forwards the message to its functioning neighbor C', which is the next one sequentially counterclockwise about B' from edge $\langle B', A'\rangle$. The process continues, until the message arrives at E, which is a functioning node on the ring. When E receives the *recovery* message, it consumes the message, and sends an *ack* message back to A along the reversed path $\langle E, H', G', F', E', D', C', B', A\rangle$. When

A receives the *ack* message, the repairing process completes, and the broken fragment of the ring, i.e., $\langle A, B, C, D, E\rangle$, is replaced by a new fragment $\langle A, B', C', D', E', F', G', E\rangle$.

3.4.2.4 Ring Reconfiguration for Load Balance

It is possible that some nodes on the index ring become overloaded compared to others. For the purpose of load balance, these nodes should be replaced by some lightly-loaded nodes. In the ARI scheme, we propose a simple algorithm to support load balance-oriented ring reconfiguration: Each node maintains a *willing flag* to indicate whether it is willing to be a node on an index ring. Initially, the flag is turned on. Each node i periodically exchanges its residual energy level (denoted as e_i) with its neighbors, and calculates the average residual energy level of all its neighbors (denoted as $\overline{e(i)}$). Node i turns off its willing flag when: $e_i < \alpha * \overline{e(i)}$, where α is a system parameter. When a node on the ring turns off its willing flag, it sends a *quit* message to its counterclockwise neighbor. On receiving the message, the neighbor initiates a process similar to the ring repairing process (refer to 3.4.2.3) to remove the quitting node and reconfigure the ring.

3.5 Enhancements

3.5.1 Reduce the Overhead of Index Updating

With the ARI scheme, a source (i.e., the storing node for a target) needs to update its location at the index nodes when its location changes. If a target frequently moves, its storing node also frequently changes, and the overhead of index updating may become very large. Further, if the source is only seldom queried, most of the updates

may be useless. Based on the above observations, we propose a *lazy index updating (LIU)* mechanism to reduce the overhead of index updating.

3.5.1.1 Basic Idea of LIU

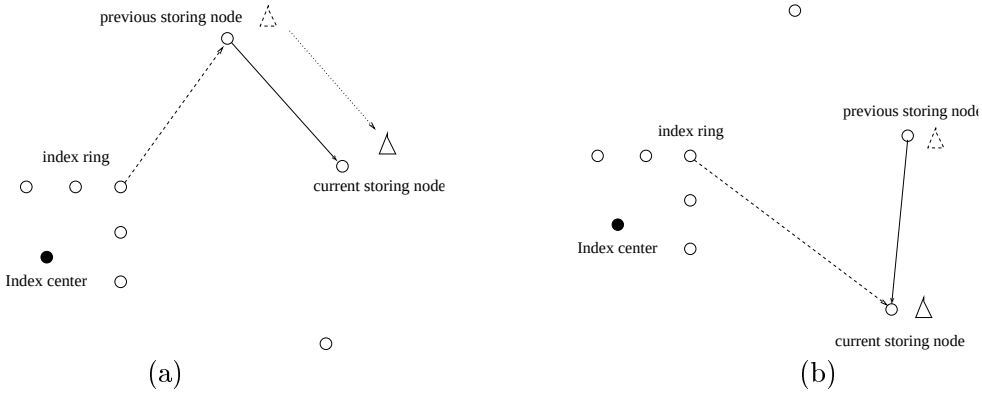


Fig. 3.6. Lazy index updating

The basic idea of LIU is illustrated in Figure 3.6. When a source changes, as shown in Figure 3.6 (a), it may not immediately update its location at the index nodes if the new location is not far away from the location registered at the index nodes. Only when the distance exceeds a certain threshold, as shown in Figure 3.6 (b), the source updates its location at the index nodes.

Since the index nodes of a source may not know the current location of the source, some measures should be taken to guarantee that a query for the source can still reach the source. For this purpose, as shown in Figure 3.6 (a), each old storing node temporarily keeps a pointer to the next storing node. So that, when the old storing node receives a

data query, it can redirect the query to the next storing node, and the process continues until the query reaches the current storing node.

Before using the lazy index updating mechanism, we need to solve the following problems: (1) whether or not updating the source location at the index nodes when the location is changed. (2) how long an old storing node should keep a pointer to the next storing node. Next, we first study these problems in theory, and then propose a protocol to solve them in a distributed way.

3.5.1.2 Dynamically Adjust the Frequency of Index Updating

We consider a certain target, and assume that:

(1) The source of the target changes when its distance to the current detecting node of the target is larger than d_c hops.

(2) During a certain time period, the source changes at an average rate of r_c per second, and the average cost of updating the location at the index nodes is d_u hops.

(3) During the same period, the source is queried at an average rate of r_q.

Therefore, if the original ARI scheme is used, the total cost of querying the source and updating its location at the index nodes is:

$$C = r_q \cdot (d_{i,i} + d_{i,o}) + r_c \cdot d_u \text{ hops per second,} \tag{3.1}$$

where $d_{i,i}$ is the average distance (in hop) between a sink and the closest index node, and $d_{i,o}$ denotes the average distance between the index node and the source.

When LIU is used, we assume that a source updates its location at the index nodes after every α ($\alpha \geq 1$) location changes. Thus, the total cost of querying the source and updating the index nodes becomes:

$$C' = r_q \cdot (d_{i,i} + d_{i,o} + \frac{\alpha}{2} \cdot d_c) + \frac{r_c}{\alpha} \cdot d_u \text{ hops per second,}$$

and

$$C - C' = r_c \cdot d_u - (\frac{\alpha}{2} \cdot r_q \cdot d_c + \frac{r_c}{\alpha} \cdot d_u). \tag{3.2}$$

To maximize $C - C'$,

$$\alpha = max(1, \sqrt{\frac{2r_c \cdot d_u}{r_q \cdot d_c}}). \tag{3.3}$$

Accordingly, an old storing node should keep a pointer to the next storing node for at least $\alpha \cdot \frac{1}{r_c}$. To deal with the case that the query rate (i.e., the source location change rate) suddenly changes a lot, the threshold may be changed to

$$\beta = \alpha \cdot \frac{1}{r_c} \cdot (1 + \rho), \text{ where } \rho > 0. \tag{3.4}$$

3.5.1.3 The Protocol

A storing node can be in one of the following states: *new*, *registered*, *old*, and *terminated*. The transitions between these states are illustrated in Figure 3.7, and is further explained as follows.

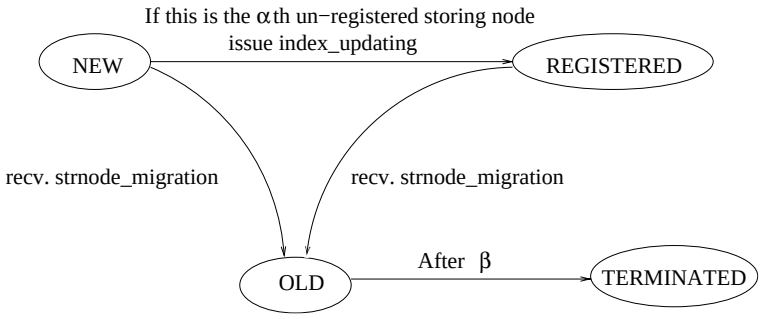


Fig. 3.7. The state transition of a storing node

When a detecting node selects node N_s as a new storing node, N_s sets its state as *new*, and sends a message *strnode_migration* to the previous storing node. N_s will also receive following information from the previous storing node: (1) the empirical query rate: $\hat{r_q}$; (2) the empirical rate of changing storing node: $\hat{r_c}$; (3) the empirical overhead of updating the storing node location at the index nodes: $\hat{d_u}$; and (4) the number of un-registered storing nodes (i.e., whose locations are not registered at the index nodes) since the last registered one: n_u. Based on the above information, N_s computes the empirical value of α, i.e.,

$$\hat{\alpha} = max(1, \sqrt{\frac{2\hat{r_c} \cdot \hat{d_u}}{\hat{r_q} \cdot d_c}}).$$

If $n_u = \hat{\alpha} - 1$, N_s should send out a message *index_updating* to the index nodes to register its location and change its state to *registered*. Otherwise, it will not change its state. When a new storing node is selected, as shown in Figure 3.7, N_s will receive a message *strnode_migration*, and changes its state to *old*. An old storing node keeps a

pointer to the next storing node for a time period of length

$$\hat{\beta} = \hat{\alpha} \cdot \frac{1}{r_c} \cdot (1 + \rho).$$

During the time period that N_s is a storing node, N_s continuously monitors the queries and the storing node changes, and the empirical values accordingly. For example, after receiving a query, N_s adapts $\hat{r_q}$ as follows:

$$\hat{r_q} = \frac{1}{2} \cdot \hat{r_q} + \frac{1}{2} \cdot \frac{1}{\tau},$$

where τ is the time interval between the current query and the last query. Similarly, $\hat{r_c}$ is adapted when receiving a $strnode\_migrate$, and $\hat{d_u}$ is adapted when an index updating finishes.

3.5.2 Reduce the Query Overhead

A sink may continuously monitor a moving target by frequently querying the current storing node of the target. With ARI, the sink needs to query the index nodes before querying the source. Therefore, the overhead for query can be large, particularly when the sink and the source are far away from the index nodes. To reduce the overhead, we propose a lazy index query (LIQ) mechanism, which enhances the lazy index updating mechanism as follows:

- An old storing node keeps a pointer to the next storing node for at least $max(\beta, \theta)$, where θ is system parameter.

- When a source replies a data message to a sink, it attaches its location to the message. On receiving the message, the sink caches the location.

- When a sink wants to query the source of a target, it first checks if it has cached the location of the source. If the location is cached and the caching time is less than θ, it will send query directly to the source. Otherwise, the query is sent to the index nodes.

3.6 Performance Evaluations

3.6.1 Simulation Model

We develop a simulator based on ns2 (version 2.1b8a) [38], to evaluate and compare the index-based data dissemination scheme (with ARI) to three data dissemination schemes: ES, DCS [39] and LS (with source-initiated sink-source matching mechanism) [53]. In this simulator, the MAC protocol is based on IEEE 802.11[1], and the transmission range of each node is $40m$ [27]. 2500 sensor nodes are distributed over a $850 \times 850m^2$ flat field, which is divided into $17 \times 17m^2$ GAF grids, such that there is one sensor node in each grid.

Several targets (the number of target is denoted as N_t) are deployed in the detecting region. Each target may move in any direction and its average velocity is denoted as v. When a target enters a grid, the sensor node located in that grid can detect it. If the index-based data dissemination scheme is simulated, the sensor node that first detects

[1] In the simulation, considering the fact that a packet in sensor networks is typically very small, we modify the format of a packet, such that the packet size is much smaller than that specified by the IEEE 802.11 standard.

a target becomes the initial source (storing node) of the target. When the distance between the current detecting node of a target and the source is larger than a threshold, the source is changed to be the current detecting node. Each target has a name, which is an integer between 0 and $N_t - 1$. We use a simple hash function to map the name of a target to one of N_i index centers, which are uniformly distributed in the detecting field. The mapping rule is as follows: $H(k) = L_k \bmod N_i$, where k is the name of a target and L_j is the location of the j^{th} index center. Any sensor node can be a sink which issues a query for the data of a certain target. Table 3.2 lists most of the simulation parameters.

3.6.2 Simulation Results

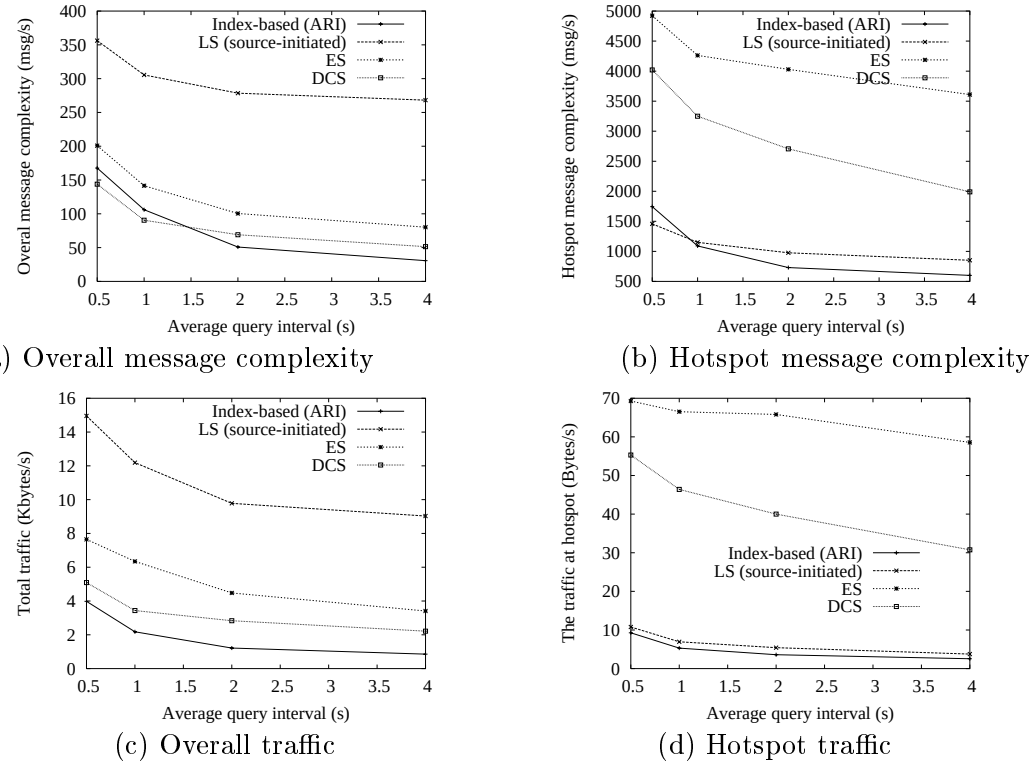
The simulation includes three parts: The first part (Section 3.6.2.1) evaluates and compares the overhead of four data dissemination schemes to verify the effectiveness of the analytical results presented in Section 3.3.2. The second part (Section 3.6.2.2-3.6.2.4) evaluates the ARI scheme to show that this scheme does not have large overhead and is resilient to clustering failures and can achieve load balance. The third part (Section 3.6.2.5-3.6.2.6) evaluates the proposed optimization mechanisms to the ARI scheme.

3.6.2.1 Comparing the performance of data dissemination schemes

We first evaluate and compare the overall message complexity of the data dissemination schemes, and the results are shown in Figure 3.8 (a). LS is shown to have the highest message complexity, since it needs to flood control messages to the whole network. ES and DCS have the same order of message complexity in theory. However, ES pushes data to a base station outside of the network, and hence has larger overhead

Table 3.2. Simulation Parameters

Parameter	Value
field size (m^2)	850×850
number of nodes	2500
communication range (m)	40.0
grid side (m)	17.0
number of targets: N_t	10
data update rate: r_d (per target per second)	0.25
number of index centers: N_i	4
the migration threshold for a source (m):	34.0
initial radius of an index ring: r (m)	34.0
initial number of index nodes on a ring: m	4
simulation time for each experiment (s)	1000.0
average velocity of a mobile target: v (m/s)	1.0-6.0
size of an update message ($byte$)	10
size of a query message ($byte$)	10
size of a data message ($byte$)	50



(a) Overall message complexity (b) Hotspot message complexity

(c) Overall traffic (d) Hotspot traffic

Fig. 3.8. Evaluating data dissemination schemes ($v = 1.0m/s$)

for pushing and retrieving data than DCS, which pushes data to nodes within the network. Figure 3.8 (a) also shows that the index-based scheme and DCS has similar overall message complexity. When the query interval is small (i.e., the query rate is high), the index-based scheme has a little bit higher message complexity than DCS. The trend is reversed when the query interval is large (i.e., the query rate is low). This phenomenon is consistent with *Observation 1* presented in Section 3.3.2.

Figure 3.8 (b) shows the hotspot message complexity of the data dissemination schemes. ES is shown to have the highest hotspot message complexity, since the hotspot nodes (i.e., the edge nodes close to the base station) need to forward all the messages related to storing and retrieving data, which include the data pushed from the detecting nodes, the queries from the sinks, and the responses sent from the base station. DCS has much smaller hotspot message complexity, because a hotspot node (i.e. a nodes surrounding a index center) only processes messages related to one type of targets whose data are stored there. LS is shown to have smaller hotspot message complexity than DCS. This is due to the fact that, in our simulations, the target does not move quickly and the number of index centers is small (i.e., 4). According to the theoretic estimation results (4.2) and (3.2) presented in Section 3.3.2, the index-based scheme has a smaller number of messages at hotspots than DCS. From Figure 3.8 (b), we can also see that, the index-based scheme has lower hotspot message complexity than LS only when the query interval is large (i.e., the query rate is small), which is also consistent with *Observation 2* presented in Section 3.3.2.

The overall traffic of the data dissemination schemes are shown in Figure 3.8 (c). From the figure, we can see that LS has very large traffic due to the fact that it floods

control messages over the whole network. ES and DCS also cause large overall traffic, since they both push data from detecting nodes to storing nodes regardless of queries. ES has larger traffic than DCS, because it pushes data to nodes outside of the network, and hence brings more communication overhead. The index-based scheme has the smallest traffic among all the schemes. This is due to the fact that, in this scheme, data are sent from a source to a sink only when the data is queried, and the control messages (i.e., queries and index updates) are never flooded.

Figure 3.8 (d) shows the hotspot traffic of the data dissemination schemes. From this figure, we can see that ES has the highest hotspot traffic, which is followed by DCS. The index-based scheme has the smallest hotspot traffic when the query interval is large (i.e., the query rate is high). The phenomenon is similar to that shown in Figure 3.8 (b) due to the same reasons as explained before.

3.6.2.2 Impact of Parameter r on The Performance of ARI

Figure 3.9 (a) shows that the index updating overhead increases as r increases. This is because, each index updating message needs to travel the whole index ring. As r increases, the index ring also increases, and an index updating message has to travel a longer path. Figure 3.9 (b) shows that increasing r can reduce the overhead of querying. This is due to the fact that most sinks are outside of the index ring. As r increases (i.e., the ring expands), the query message can be intercepted by an index node soon. Since the reduction in querying overhead is smaller than the increase in updating overhead, as shown in Figure 3.9 (c), the overall message complexity increases as r increases. However, the total message complexity does not increase very much if r is not large.

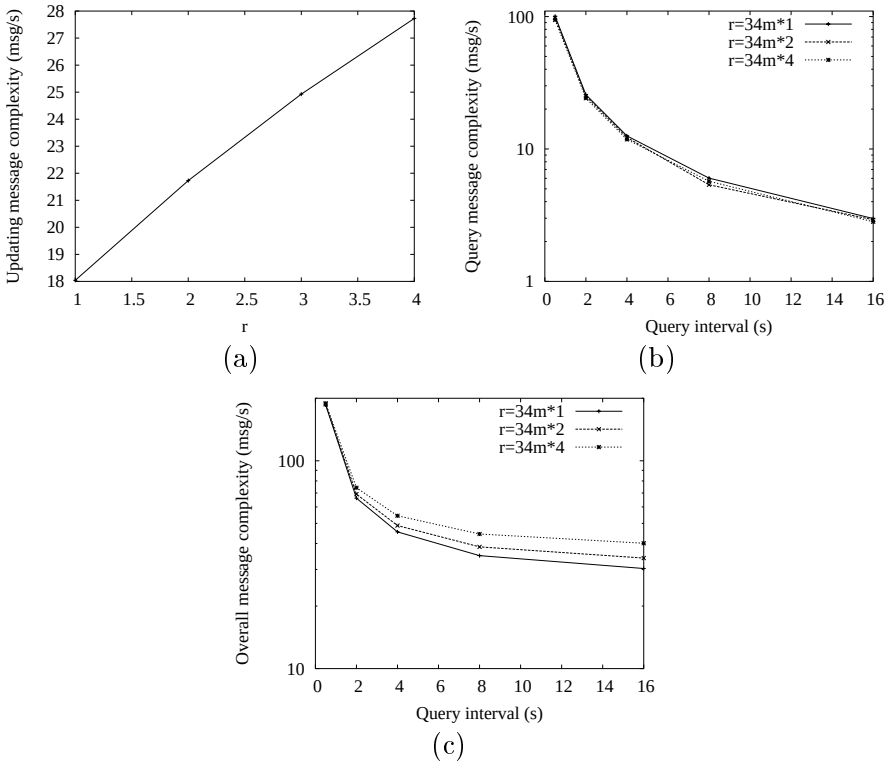


Fig. 3.9. Impact of r $(v = 3.0m/s, m = 8)$

3.6.2.3 Tolerating Clustering Failures

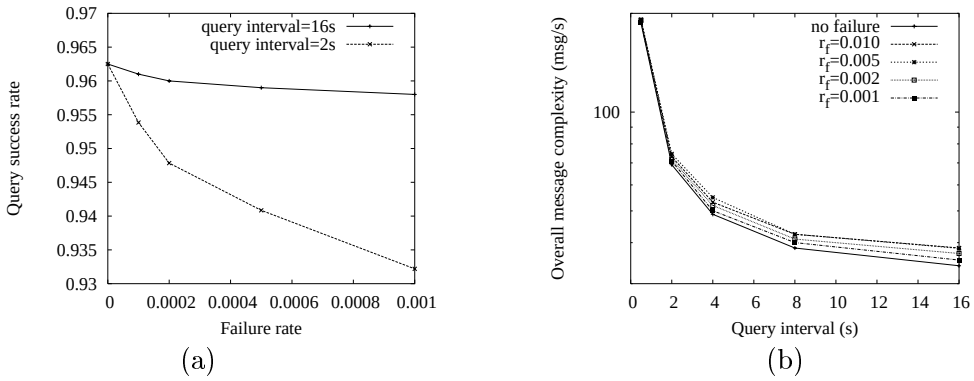


Fig. 3.10. Evaluating the fault tolerance feature of ARI ($r = 34m*2, m = 8, v = 3.0m/s$)

We now show how the ARI scheme reacts to clustering failures. Similar to [39], the *up/dowm* model is adopted in the simulations: each $17 * 17m^2$ grid may be isolated or fail at an average rate denoted as r_f, and an isolated or failed grid may recover at an average rate denoted as r_c (We assume $r_c = 3r_f$). The success rate of the queries and the overall message complexity are measured as r_f (r_c) varies, and the results are shown in Figure 3.10.

From Figure 3.10 (a), we can see that the query success rate decreases slightly when r_f increases. This is due to the fact that, when the index ring is broken due to clustering failure, the ARI scheme guarantees that a query can still be routed to a normal node on the index ring, and routed to one functioning index node. Only when all index nodes are failed, or a ring node fails after it has received a query and before

it has not forwarded the query, the query is dropped. Figure 3.10 (a) also shows that, when query interval decreases (i.e., the query rate increases), the query success rate drops more quickly as r_f increases. This is because, the frequency that a ring node fails after receiving a query increases as the query rate increases, which results in more query messages being dropped. Note that, the success rate is smaller than 1.0 even when the failure rate is 0 since a query message may be dropped due to traffic congestions.

Figure 3.10 (b) shows that the overall message complexity is slightly increased as the failure rate increases. This can be explained as follows: When an index ring is broken due to clustering failures, the ring should be repaired, and hence some communication overhead is introduced. Also, the repaired ring becomes longer, which increases the overhead for index updates.

3.6.2.4 Load Balance

Next, we evaluate the ARI scheme's ability to achieve load balance. In this simulation, each node is initially equipped with an energy of $0.2J$, and other simulation parameters are the same as the previous simulations. We measure the overall message complexity and the hotspot message complexity, when the load balance module is turned on (i.e., the load balance parameter is larger than 0) or off (i.e., the load balance parameter is set to 0).

Figure 3.11 (a) shows the overall message complexity. We can see that, the overall message complexity is increased by about 10% as α changes from 0 to 0.75. This is due to the reason that, the index ring has to be dynamically reconfigured when the load

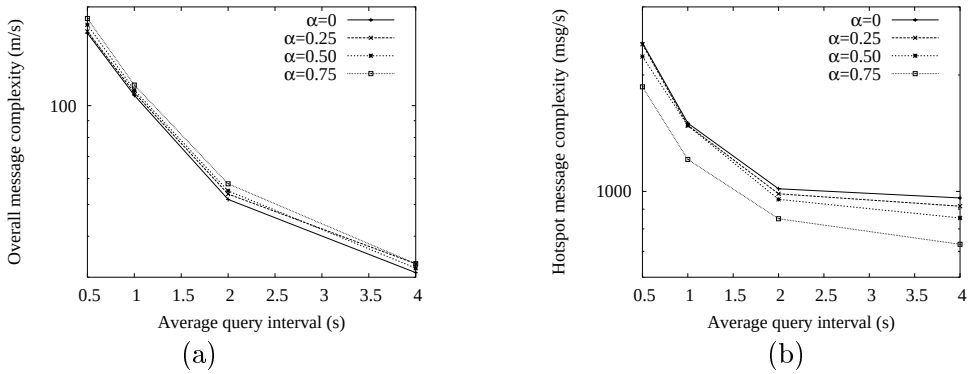
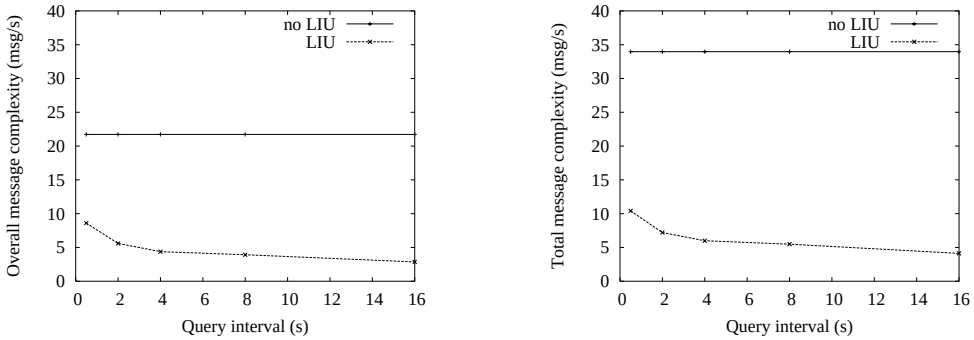


Fig. 3.11. Evaluating the load balance feature of ARI ($r = 34m * 2, m = 8, v = 1.0m/s$)

balance module works. With the dynamic reconfigurations, as shown in Figure 3.11 (b), the hotspot message complexity is decreased by around 25%.

3.6.2.5 Evaluating LIU

Figure 3.12 shows that LIU can significantly reduce the overhead of index updating. This is due to the following reasons: In the original ARI scheme (without LIU), a source needs to update its new location at the index nodes whenever its location changes. This may introduce large overhead. However, when LIU is used, a source can dynamically adjust the index updating frequency based on several factors (e.g., the query rate, the frequency of changing its source location, *etc.*) to minimize the index updating overhead. The figure also shows that the overhead decreases as the query interval increases. This is because, as shown in Eq. (3.3), α is proportional to $\sqrt{\frac{1}{r_q}}$ (r_q is the query rate). So, α increases as the query interval increases (i.e., r_q decreases). As a result, the frequency and the overhead of index updating are reduced. Comparing Figure 3.12 (a), (b)

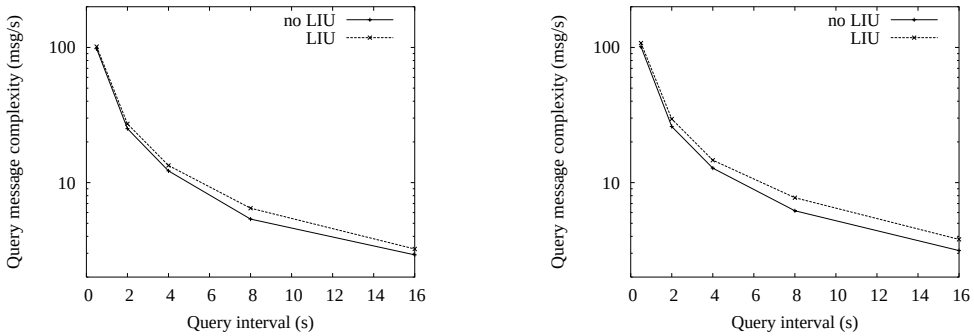


(a) v=3.0m/s (b) v=6.0m/s

Fig. 3.12. The index updating message complexity with/without LIU ($r = 34m*2, m = 8$)

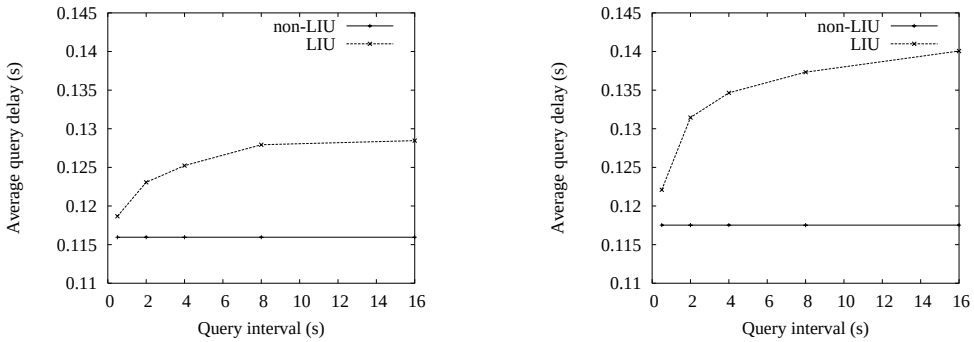
and (c), we can further find that LIU reduces more overhead as the target velocity (v) increases. This is because, when ARI is used, the frequency and the overhead of index updating increase rapidly as v increases. However, LIU can slow down the increase by adjusting the index updating frequency (i.e., increasing α) according to Eq. (3.3).

Figure 3.13 and 3.14 show that LIU increases the query overhead in terms of query message complexity and query delay. This phenomenon can be explained as follows. When LIU is not used, the index nodes know the current source locations, and a query message travels a path: $sink \rightarrow index\_node \rightarrow current\_source$. However, when LIU is used, the index nodes may not know the current source locations, and a query message may travel a longer path: $sink \rightarrow index\_node \rightarrow old\_source_1 \rightarrow \cdots old\_source_m \rightarrow current\_source$. A longer querying path results in larger querying overhead. The figures also show that the overhead increases as the query interval increases. This is because, as shown in Eq. (3.3), the index updating frequency (α) increases as the query rate r_q decreases (i.e. the query interval increases). Consequently, the source location recorded



(a) v=3.0m/s (b) v=6.0m/s

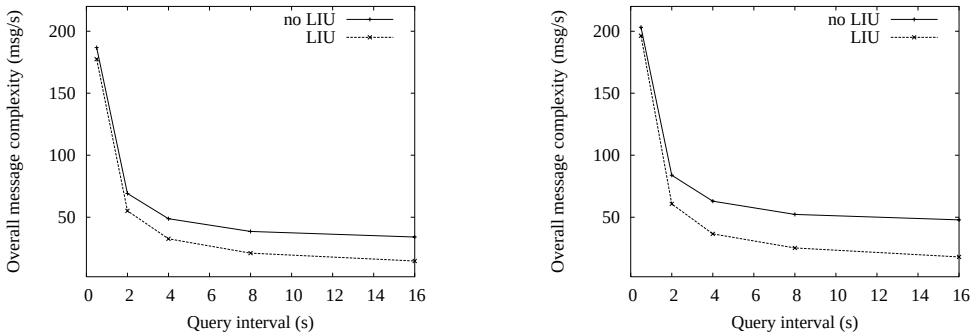
Fig. 3.13. The query message complexity with/without LIU ($r = 34m * 2, m = 8$)



(a) v=3.0m/s (b) v=6.0m/s

Fig. 3.14. The average query delay with/without LIU ($r = 34m * 2, m = 8$)

by the index nodes becomes further older, and a query message may travel a longer path as the query interval increases. Comparing Figure 3.13 (3.14) (a), (b) and (c), we can find that the query overhead increases as the target velocity (v) increases, since the index updating frequency increases more slowly than the frequency of changing source location, which increases the length of querying paths and the querying overhead.



(a) v=3.0m/s (b) v=6.0m/s

Fig. 3.15. The total message complexity with/without LIU ($r = 34m * 2, m = 8$)

Figure 3.15 shows that LIU reduces the total message complexity. The phenomenon can be explained based on Figure 3.12 and Figure 3.13. LIU significantly reduces the overhead of index updating, and at the same time, it slightly increases the querying overhead. As a result, the total message complexity is reduced. Furthermore, as the target velocity (v) increases, the savings of index updating increase more rapidly than the increase of query overhead. Consequently, more total message complexity is reduced as v increases.

3.6.2.6 Evaluating LIQ

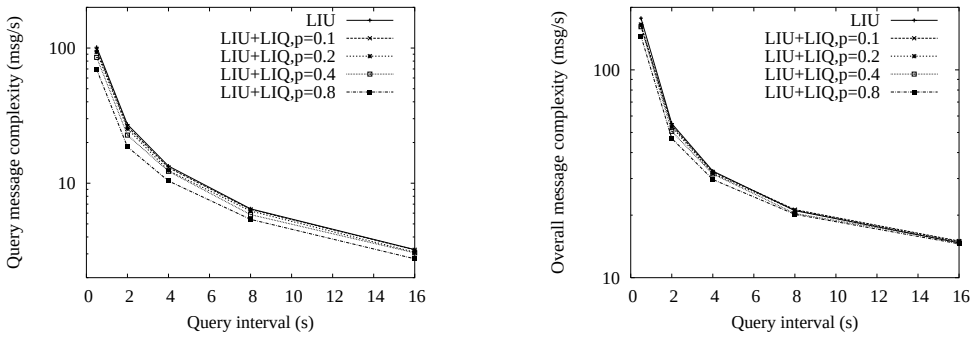


Fig. 3.16. The message complexity with/without LIQ ($r = 34m * 2, m = 8, v = 3.0m/s$)

To evaluate LIQ, we randomly select a set of sinks, each continuously query the same source. The total number of sink is 50, and the number of the selected sinks is $50p$ (p varies from 0.1 to 0.8). Also, a cached source location is valid if and only if it has been cached for less than $100s$.

Figure 3.16 shows that LIQ can reduce the query message complexity and the overall message complexity. This is due to the following reasons: With LIU, each query message should travel a path: $sink \rightarrow index\_node \rightarrow old\_source_1 \cdots \rightarrow old\_source_m \rightarrow current\_source$. When LIQ is used, a sink may use the source location information cached locally to locate the source and avoid accessing the index nodes. Consequently, the querying path becomes: $sink \rightarrow old\_source_1 \cdots \rightarrow old\_source_m \rightarrow current\_source$. The reduction in querying overhead become more significant if the sink and the source

is close to each other, and they are both far away from the index nodes. Figure 3.16 also shows that the benefit of using LIQ becomes larger as p increases or the query interval reduces. This phenomena can be explained as follows: As p increases, a larger portion of queries are issued from the same sink. Once a sink has queried a source, the source location information is cached locally, and can be used for the next query as long as the information is still valid. Also, as the query interval reduces, the cached source locations can be updated more frequently. Consequently, the cache hit ratio is reduced and hence the querying overhead decreases.

3.7 Summary of the Data Dissemination Scheme with Ring-Based Index

In this chapter, we proposed an index-based data dissemination scheme with adaptive ring-based index (ARI). This scheme is based on the idea that sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes. The index nodes for the targets of one type forms a ring surrounding the location which is determined based on the event type, and the ring can be dynamically reconfigured for fault tolerance and load balance. We also proposed several mechanisms to optimize the ARI scheme. Analysis and simulations were conducted to evaluate the performance of the proposed index-based scheme. The results show that the index-based scheme outperforms the ES scheme, the DCS scheme, and the LS scheme. The results also show that using the ARI scheme can tolerate clustering failures and achieve load balance, and the proposed optimization mechanisms can further improve the system performance.

Chapter 4

Distributed Group Key Management for Data Authenticity and Confidentiality

4.1 Introduction

When a sensor network [2] is deployed in unattended and hostile environments such as battlefields, adversaries can easily eavesdrop the communications in the network. Even worse, the adversary may capture and reprogram nodes, or inject their own nodes into the network and induce the network to accept them as legitimate nodes [22]. Once in control of a few nodes, the adversary can mount various attacks from inside the network. For example, a compromised node (intruder) may inject false sensing reports or maliciously modify reports that go through it. Under such attacks, the sink may accept wrong sensing data and take inappropriate responses, which may be dangerous in the scenarios such as battlefield surveillance and environmental monitoring.

To defend against the eavesdropping attacks by outsiders, all legitimate sensor nodes may share a group key to encrypt/decrypt messages. To defend against the message injection attacks, messages should be authenticated. The digital signature-based technique can be used to authenticate and filter false messages. However, this technique has high overhead both in terms of computation and bandwidth [25], which makes it unsuitable for sensor networks [35]. Therefore, researchers proposed to adopt symmetric cryptographic techniques such as the *statistical en-route filtering (SEF)* scheme [54] and

the *interleaved hop-by-hop authentication (I-LHAP)* scheme [64], to address the problem. The basic idea of the SEF scheme can be generalized as follows: Sensor nodes are randomly divided into multiple groups. Nodes in the same group share a symmetric group key, and the final receiver (sink) knows all the group keys. Each message is authenticated using multiple keyed MACs, each is generated using one group key. When such a message is forwarded along a path to the receiver, an en-route node may use its group key to verify the MACs carried in the message. Normally, an en-route node only knows one group key, so it cannot modify a passing message or inject a false message without being detected by other en-route nodes who know different group keys.

The group key-based techniques will become ineffective if some nodes are compromised since the adversary may obtain group keys from these nodes. To deal with node compromise, the compromised nodes should be identified and the innocent nodes should update their group keys to prevent the adversary from utilizing the captured keys.

To identify compromised nodes, each node can use the *watchdog* mechanism [34] to monitor its neighbors and identify the compromised nodes when observing misbehaviors, and the collaborative intruder identification scheme proposed by Wang *et al.* [50] can be used to improve the accuracy.

The group key updating problem has been extensively studied in the context of secure multicast in wired or wireless networks. Many centralized solutions [26, 49, 51, 4] and a few distributed solutions [6] have been proposed. However, most of them are not suitable for sensor networks. For example, in SKDC [26], each key updating requires N key encryptions and N transmissions (N is the number of nodes in the networks) of keys from the central controller to each individual node, which results in very high

communication overhead and rekeying delay. The logic tree-based schemes proposed by Wallner *et al.* [49], Wong *et al.* [51], and Balenson *et al.* [4] can achieve logarithmic broadcast size, storage, and computational cost. However, the communication cost and the rekeying delay are still high when applied to a large scale network. Also, the central controller has to trace the status of all nodes, and maintain a large logic tree connecting all the trusted nodes, which incurs high management overhead. Furthermore, if the network is partitioned, the sensor nodes that are connective to the central controller cannot update their group keys. A distributed solution, e.g., Blundo's scheme [6], allows a set of nodes to set up a group key in a distributed way. However, it is still not scalable since the storage cost of each node increases rapidly as the group size increases; each node must know and communicate with other trusted members in the same group.

To address the drawbacks of the existing group rekeying schemes, we propose a family of distributed and localized group rekeying schemes, called the *predistribution and local collaboration-based group rekeying (PCGR)* schemes. The design of these schemes are motivated by the following ideas: (1) Future keys can be preloaded to individual nodes before deployment to avoid the high overhead of securely disseminating new keys at the key updating time. (2) Neighbors can collaborate with each other to effectively protect and appropriately use the preloaded keys; the local collaboration also relieves the high cost of the centralized management. We first propose a *basic PCGR (B-PCGR)* scheme. To address some security limitations of B-PCGR, we propose two enhanced PCGR schemes, i.e., a *cascading PCGR (C-PCGR)* scheme and a *random variance-based PCGR (RV-PCGR)*. Extensive analysis is conducted to evaluate the security level and the performance of the proposed schemes, as well as comparing the performance

of the proposed schemes with some existing group rekeying schemes. Simulations are used to evaluate the effectiveness of the proposed group rekeying scheme in filtering false data. The analysis and simulation results show that the proposed schemes can achieve a good level of security, outperform most previously proposed schemes, and significantly improve the effectiveness of filtering false data with low overhead.

The rest of the paper is organized as follows: The next section presents the system model. In Section 4.3, we describe and analyze the basic PCGR scheme. The enhanced PCGR schemes are presented in Section 4.4. Section 4.5 reports the performance evaluation results. Section 4.6 discusses some issues related to the proposed schemes. Section 4.7 concludes the paper.

4.2 System Model

We consider a large scale wireless sensor network which is deployed in a hostile environment (for example, it may be deployed in a battlefield for tracking enemy tanks [57, 56]). The network is composed of low-complexity sensor nodes, e.g. the Berkeley MICA mote [14], which has a processor running at 4 MHz and a 4KB RAM for data storage. 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 enough space for storing a few kilobytes of keying information.

Node deployment is managed by a central controller (setup server), which is responsible for picking group keys and preloading some keying information to a node before it is deployed. We assume that each node is innocent before deployment, and cannot be

compromised at least during the first several minutes after deployment [63] since compromising a node takes some time. Also, each pair of neighboring nodes can establish a pairwise key using some existing techniques [17, 11, 33, 16]. When two neighbors exchange some messages for key updating, the messages must be encrypted using their pairwise key to prevent eavesdropping.

We assume that a compromised node can eventually be detected by most of its neighbors within a certain time period. To achieve this, the watchdog mechanism [34] and and some collaborative intruder detection and identification scheme [50] can be used. To defend against compromised nodes (or outside intruders) from injecting false reports or modifying the reports generated by other innocent nodes, the statistical en-route filtering (SEF) mechanism [54] is used to detect and drop false messages. With this mechanism, sensor nodes are randomly assigned to multiple groups, and the nodes in the same group share a unique key. We also assume that nodes are loosely time-synchronized, and group rekeying is launched by each node every certain time interval [60].

4.3 Basic Predistribution and Local Collaboration-Based Group Rekeying (B-PCGR) Scheme

In this section, we first present the basic idea of the B-PCGR scheme, and then give a detailed description. Finally, we analyze its security property.

4.3.1 Basic Idea

The B-PCGR scheme includes the following three steps.

4.3.1.1 Group Key Predistribution

Before a node is deployed, it is randomly assigned to a group, and is preloaded with the current and all future keys of the group. The keys are represented by a polynomial called *group key polynomial (g-polynomial)*. Compared to most existing group rekeying protocols, in which new group keys are generated and distributed at the key updating time, the B-PCGR scheme can significantly reduce the communication overhead and the key updating delay, since the keys are already preloaded.

4.3.1.2 Local Collaboration-Based Key Protection

Since all group keys have been preloaded, it is important to protect the keys from being exposed to intruders. For this purpose, even a node that is currently trusted should not explicitly keep the future group keys, because the keys can be captured by an adversary when the node is compromised. Based on the assumption that every node is innocent at least during the first few minutes after deployment, we propose a local collaboration-based group key protection technique, which is briefly described as follows:

- Each node randomly picks a polynomial, called *encryption polynomial (e-polynomial)*, to encrypt its g-polynomial. We call the encrypted g-polynomial *g'-polynomial*.
- Some shares of the e-polynomials, are distributed to its neighbors.
- The node removes its g-polynomial and e-polynomial, but keeps its current key and its g'-polynomial.

After the above steps, a node can not access its future group keys without collaborating with a certain number of neighbors, each of which has a share of its e-polynomial.

4.3.1.3 Local Collaboration-Based Group Key Updating

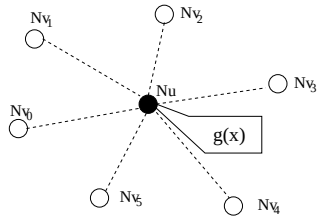
At the time for group key updating, every innocent node needs to receive a certain number of e-polynomial shares from its trusted neighbors. Also, the received shares can only be used to calculate one instance of the e-polynomial which is necessary for computing the new group key. This group key updating mechanism guarantees that a node can compute its new group key as long as it is trusted by a certain number of neighbors; meanwhile, the node can not derive any group keys that should not be disclosed at this time.

4.3.2 Detailed Description of B-PCGR

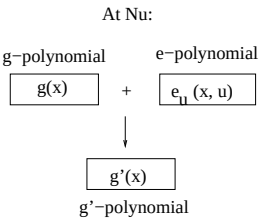
4.3.2.1 Predistributing g-Polynomials

Initially, the setup server decides the total number of groups. For each group i, a unique t-degree (t is a system parameter) univariate g-polynomial $g_i(x)$ is constructed over a prime finite field $F(q)$ to represent the keys of the group, where $g_i(0)$ is the initial group key, $g_i(j)$ $(j \geq 1)$ is the group key of version j, and q is a large prime whose size can accommodate a group key.

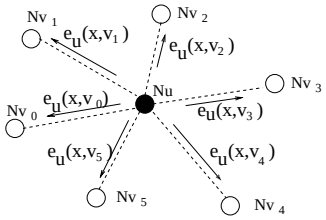
Before a node N_u is deployed, the setup server randomly assigns it to a group, and preloads the g-polynomial of the group, denoted as $g(x)$, to it.



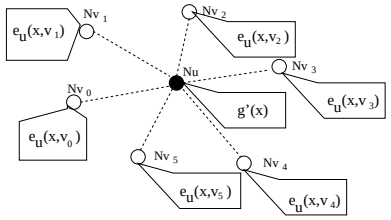
(a) after group key predistribution



(b) encrypting the g-polynomial



(c) distributing shares of e-polynomial



(d) after encryption and share distribution

Fig. 4.1. B-PCGR: Polynomial encryption and share distribution

4.3.2.2 Encrypting g-Polynomials and Distributing the Shares of e-Polynomials

After N_u has been deployed and has discovered its neighbors, it randomly picks a bivariate e-polynomial

$$e_u(x, y) = \sum_{0 \le i \le t, 0 \le j \le \mu} A_{i,j} x^i y^j, \tag{4.1}$$

where μ is a system parameter.

Using the e-polynomial (i.e., $e_u(x, y)$), as shown in Figure 4.1 (b), N_u encrypts its g-polynomial (i.e., $g(x)$) to get its g'-polynomial (denoted as $g'(x)$). The encryption is conducted as follows:

$$g'(x) = g(x) + e_u(x, u). \tag{4.2}$$

After that, as shown in Figure 4.1 (c), N_u distributes the shares of $e_u(x, y)$ to its n neighbors N_{v_i} $(i = 0, \cdots, n-1)$. Specifically, each neighbor N_{v_i} receives share $e_u(x, v_i)$. At the same time, N_u removes $e_u(x, y)$ and $g(x)$, but keeps $g'(x)$. The final distribution of $g'(x)$ and $e_u(x, v_i)$ is illustrated in Figure 4.1 (d).

4.3.2.3 Key Updating

Each node maintains a *rekeying timer*, which is used to periodically notify the node to update its group key, and a variable called *current group key version* (denoted as c). Note that c is initialized to 0 when the node is deployed.

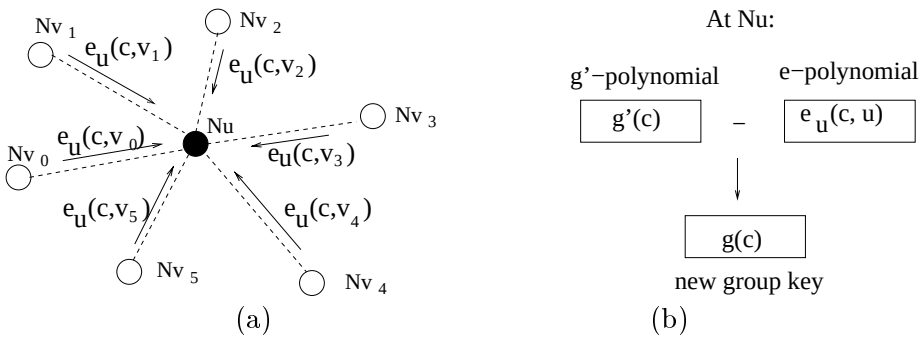


Fig. 4.2. B-PCGR: Key updating

To update group keys, each innocent node N_u increases its c by one, and returns share $e_{v_i}(c, u)$ to each trusted neighbor N_{v_i}. Meanwhile, as shown in Figure 4.2 (a), N_u receives a share $e_u(c, v_i)$ from each trusted neighbor N_{v_i}. Having received $\mu + 1$ shares, which are denoted as $\{\langle v_i, e_u(c, v_i)\rangle, i = 0, \cdots, \mu\}$, N_u can reconstruct a unique μ-degree

polynomial

$$e_u(c, y) = \sum_{j=0}^{\mu} B_j y^j, \tag{4.3}$$

by solving the following $\mu + 1$ $(\mu + 1)$-variable linear equations:

$$\sum_{j=0}^{\mu} (v_i)^j B_j = e_u(c, v_i), \quad i = 0, \cdots, \mu. \tag{4.4}$$

Having known $e_u(c, x)$, as shown in Figure 4.2 (b), N_u can compute its new group key $g(c) = g'(c) - e_u(c, u)$.

4.3.3 Security Analysis

The security property of the B-PCGR scheme can be stated by Theorem 2.

THEOREM 2. *For a certain group, its g-polynomial $g(x)$ can be compromised if and only if:*

(1.1) a node (N_u) of the group is compromised, and

(1.2) at least $\mu + 1$ neighbors of N_u (In the encryption polynomial picked by N_u, i.e., $e_u(x, y)$, the degree of x is t and the degree of y is μ.) are compromised;

or

(2) at least $t + 1$ past keys of the group are compromised.

Proof (sketch) First, we prove that: if condition (1.1) and (1.2) or (2) is satisfied, $g(x)$ can be compromised.

Assume that N_u and its $\mu + 1$ neighbors, i.e., N_{v_i} $(i = 0, \cdots, \mu)$, have been compromised. The adversary can obtain $g'(x)$ and $e_u(x, v_i)$ $(i = 0, \cdots, \mu)$. For an arbitrary x', a unique polynomial

$$e_u(x', y) = \sum_{j=0}^{\mu} B'_j y^j \tag{4.5}$$

can be reconstructed by solving the following $\mu + 1$ $(\mu + 1)$-variable linear equations:

$$\sum_{j=0}^{\mu} (v_i)^j B'_j = e_u(x', v_i), \text{where } i = 0, \cdots, \mu. \tag{4.6}$$

Knowing $e_u(x', y)$, $g(x')$ can be computed as follows:

$$g(x') = g'(x') - e_u(x', u) \tag{4.7}$$

Similarly, we can prove that: if condition (2) is satisfied, $g(x)$ can be compromised.

Second, we prove that: if condition (1.1) is satisfied, but condition (1.2) and (2) are not satisfied, $g(x)$ can not be compromised. Assume that N_u and its $m \leq \mu$ neighbors, i.e., N_{v_i} $(i = 0, \cdots, m-1)$, are compromised. Thus, for an arbitrary x', the adversary can obtain $g'(x')$ and $e_u(x', v_i)$ $(i = 0, \cdots, m-1)$. Since $m \leq \mu$ and $e_u(x', y)$ is a μ degree polynomial of variable y, similar to the proof in [6], we prove in the following that the adversary can not find out $e_u(x', u)$:

We consider the worst case that $m = \mu$. Suppose the adversary guesses $e_u(x', u) = a$. A unique polynomial $e_u(x', y)$ (shown in Eq. (4.5)), which is a μ-degree polynomial

of y, can be constructed by solving the following linear equations

$$\begin{cases} \sum_{j=0}^{\mu} (v_i)^j B'^j = e_u(x', v_i), \ i = 0, \cdots, \mu - 1 \\ \sum_{j=0}^{\mu} u^j B'^j = a. \end{cases} \tag{4.8}$$

However, since a can be an arbitrary value in $F(q)$, the adversary can construct q different polynomials $e_u(x', y)$. Also, since $e_u(x, y)$ is randomly chosen and hence $e_u(x', y)$ can be an arbitrary polynomial; i.e., the q polynomials the adversary can construct are equally likely to be the actual $e_u(x', y)$. Formally,

$$\begin{aligned} &\forall a, Pr(e_u(x', u) = a \mid \{e_u(x', v_i), 0 \leq i \leq \mu - 1\}) \\ &\equiv Pr(e_u(x', u) = a). \end{aligned} \tag{4.9}$$

Therefore, the adversary can not derive $e_u(x', u)$.

Without knowing $e_u(x', u)$, the adversary can not find out $g(x)$, which is equal to $g'(x) - e_u(x, u)$. On the other hand, since condition (2) is not satisfied, we assume that the adversary also knows $l \leq t$ keys of the considered group. Without loss of generality, we further assume that the compromised keys are $g(0), g(1), \cdots, g(l-1)$. Because $l \leq t$ and $g(x)$ is a t-degree polynomial of variable x, the adversary can not find out $g(x')$, either.

Similar to the above proof, we can also prove that: if condition (1.2) is satisfied, but condition (1.1) and (2) are not satisfied, $g(x)$ can not be compromised.

4.3.4 Parameter Selection

When applying the B-PCGR scheme in a sensor network, each node in the network must select parameter μ appropriately to achieve a desired level of system security and reliability. In this paper, the level of system security and reliability is measured by two metrics:

- $P_s(\mu, t)$: the probability that an arbitrary node, if not compromised or failed, can update its group key successfully at time t; i.e., it has $\mu+1$ or more neighbors that are not compromised or failed at time t.

- $P_c(\mu, t)$: the probability that at least one group key polynomial is compromised; i.e., there exists a node that is compromised along with its $\mu+1$ or more neighbors.

In the following, we develop a theoretic model for analyzing the relationship between parameter μ, the number of neighbors (denoted as n), $P_s(\mu, t)$, and $P_c(\mu, t)$. Based on the model, we then show how each node determines its value of μ.

4.3.4.1 A Theoretic Model

To develop the model, we assume that a node fails (or is compromised) following a Poisson process. Specifically, a node fails before time t with the probability of $1-e^{-\lambda_f t}$, where λ_f denotes the failure rate. For example, if the expected life time of a node is 30 days (i.e., 2592000 seconds) and t is represented in the unit of second, $\lambda_f = 1/2592000 = 3.9 \times 10^{-7}$. Similarly, a node, if not fails, is compromised before time t with the probability of $1-e^{-\lambda_c t}$, where λ_c denotes the compromising rate. Considering

a scenario where the number of nodes in the network, denoted as N, is 2000, and the adversary can compromise one node per hour in average, $\lambda_c = \frac{1}{1hour \times 2000} = 1.4 \times 10^{-7}$.

Based on the above assumption, we can derive the probability that a node is not compromised or failed before t, denoted as $p_n(t)$, to be

$$p_n(t) = e^{-(\lambda_f + \lambda_c)t}. \tag{4.10}$$

Therefore,

$$P_n(\mu, t) = \sum_{i=\mu+1}^{n} \binom{n}{i} p_n(t)^i [1 - p_n(t)]^{n-i}. \tag{4.11}$$

We can also derive the probability that a node is compromised before t, denoted as $p_c(t)$, to be

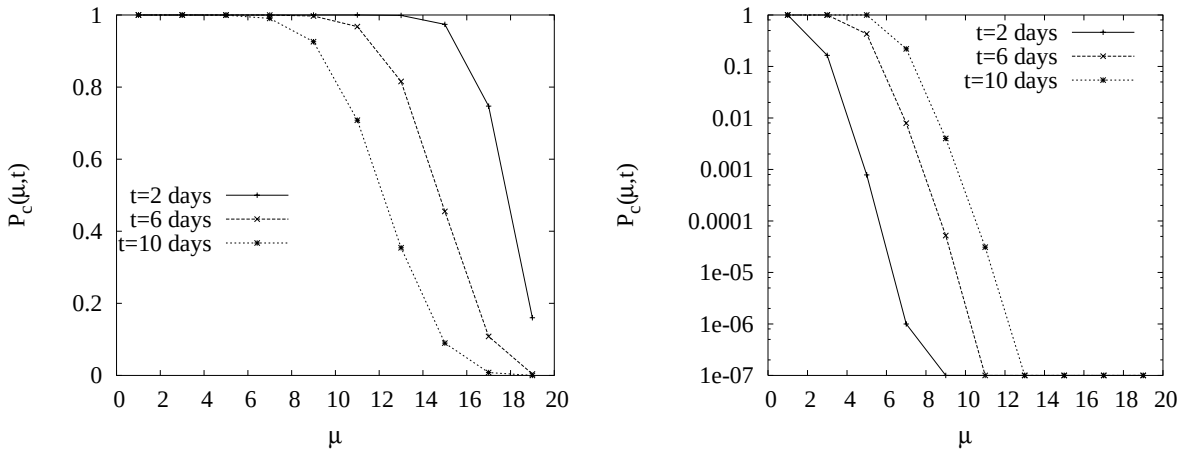
$$p_c(t) = \frac{\lambda_c}{\lambda_c + \lambda_f}(1 - e^{-(\lambda_c + \lambda_f)t}). \tag{4.12}$$

So the probability that a node and its $\mu + 1$ or more neighbors are compromised is

$$[1 - p_n(t)] \times \sum_{i=\mu+1}^{n} \binom{n}{i} p_c(t)^i [1 - p_c(t)]^{n-i}. \tag{4.13}$$

Further, the probability for the adversary to compromise at least one group key polynomial, i.e., $P_c(\mu, t)$, is no larger than

$$1 - \{1 - [1 - p_n(t)] \times \sum_{i=\mu+1}^{n} \binom{n}{i} p_c(t)^i [1 - p_c(t)]^{n-i}\}^N. \tag{4.14}$$



(a) (b)

$N = 2000$, $n = 20$, $r_f = 3.9e - 7$ (expected life time of a node: 30 days)
$r_c = 1.4e - 7$ (compromise rate: 1 node/hour)

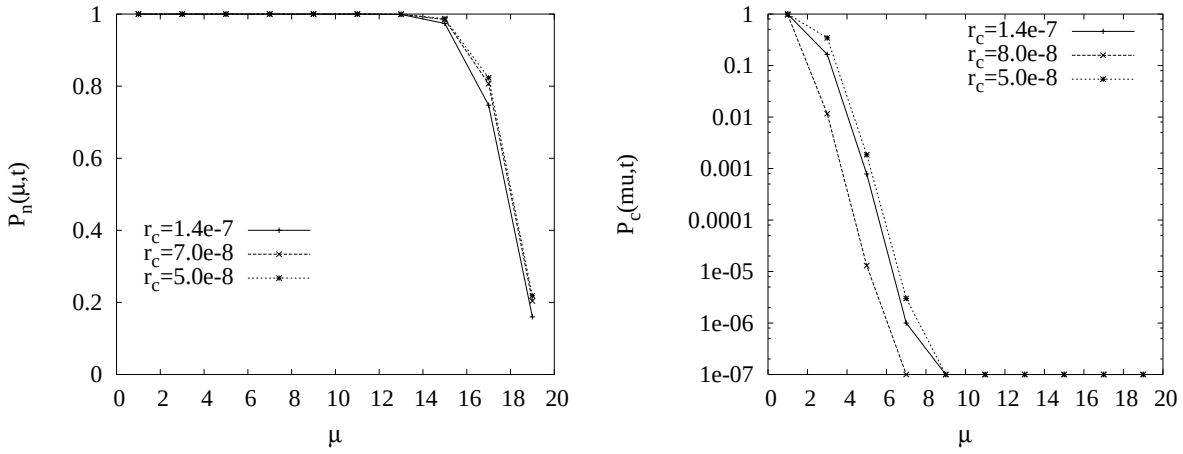
Fig. 4.3. An Example

4.3.4.2 Some Numeric Results

Figure 4.3 illustrates how $P_n(\mu, t)$ and $P_c(\mu, t)$ change as μ and t vary, under the scenario where the number of nodes is 2000, each node has 20 neighbors, the expected life time of a node is 30 days, and the compromise rate is 1 node per hour in average. As demonstrated by Figure 4.3 (a), when t is fixed, $P_n(\mu, t)$ decreases as μ increases. This is because, as shown in Equation (4.11), the larger is μ, the more normal (i.e., not compromised or failed) neighbor are required by a node to update its group key. On the other hand, when μ is fixed, $P_n(\mu, t)$ decreases as t increases. This can be explained based on Equation (4.10), which shows that more node will be compromised or failed as time elapses. Thus, the probability for finding $\mu + 1$ or more normal nodes is decreased accordingly.

Figure 4.3 (b) shows that, when t is fixed, $P_c(\mu, t)$ decreases as μ increases. This is due to the fact that the probability for compromising an arbitrary node and its $\mu + 1$ or more neighbors decreases as μ becomes large, and the probability of compromising a group key polynomial decreases accordingly. The figure also shows that, when μ is fixed, $P_c(\mu, t)$ decreases as t increases. This is obvious because more nodes will be compromised as time passes.

Figure 4.4 and Figure 4.5 illustrate the impact of r_c and r_f on $P_n(\mu, t)$ and $P_c(\mu, t)$, respectively. As shown in Figure 4.4 (a) and 4.5 (a), when μ is fixed, $P_n(\mu, t)$ increases as r_c or r_f decreases. This is due to the fact demonstrated by Equation (4.10); i.e., the probability that a node is not failed or compromised increases as r_f or r_c



(a) (b)

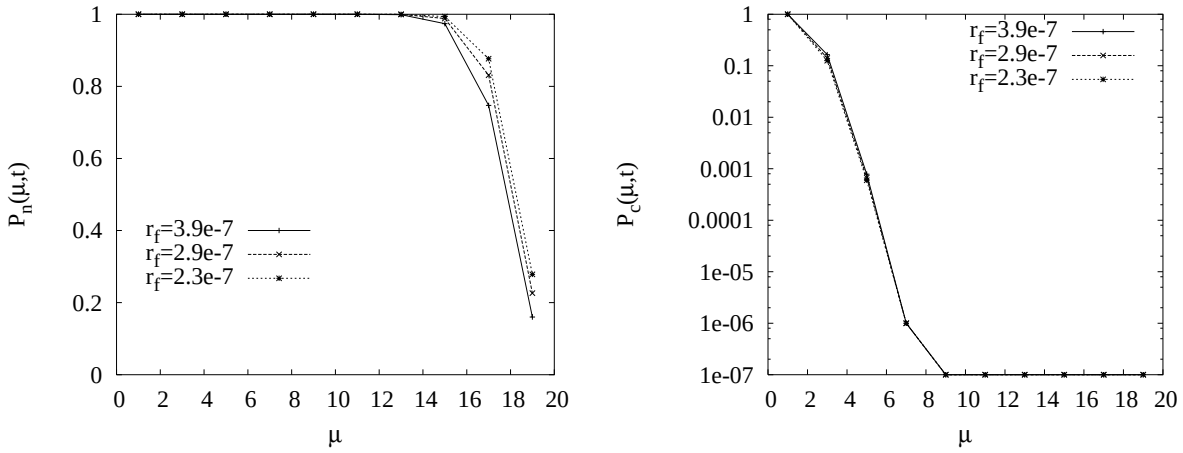
$N = 2000$, $n = 20$, $r_f = 3.9e-7$ (expected life time of a node: 30 days), $t = 2$ days

Fig. 4.4. Impact of r_c

decreases. Consequently, $P_n(\mu, t)$ also increases according to equation (4.11). Similarly, as shown in Figure 4.5 (b), $P_c(\mu, t)$ decreases as r_f decreases according to formula (4.14)

Figure 4.4 (b) shows an interesting phenomenon that, when other parameters are fixed, $P_c(\mu, t)$ increases as r_f decreases. This can be explained as follows: As r_f decreases, nodes will live longer. On the other hand, since the compromise rate is unchanged, more nodes alive will be compromised. As a result, the adversary can capture more shares and thus has higher probability to break a group key polynomial.

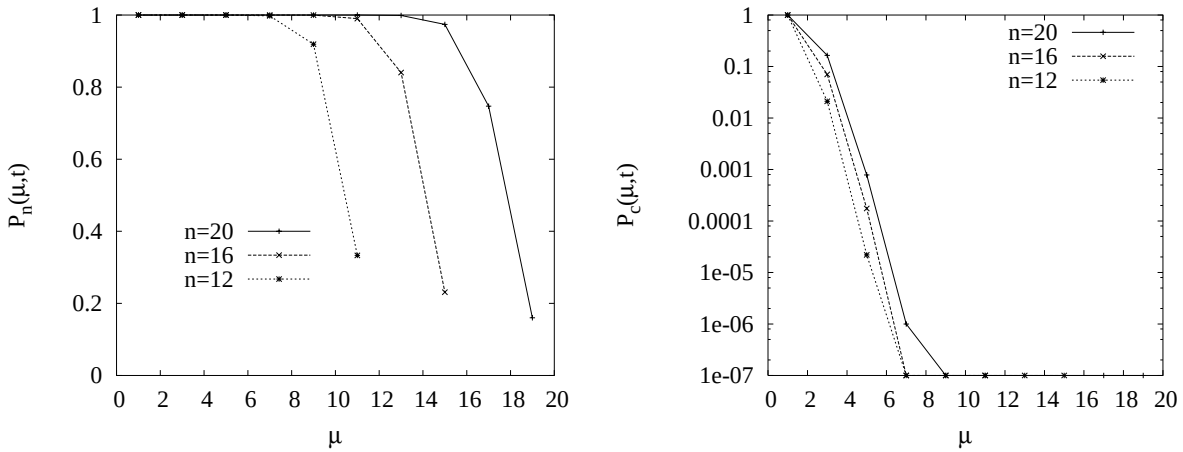
Figure 4.6 shows the impact of the number of neighbors (n) on $P_n(\mu, t)$ and $P_c(\mu, t)$. As illustrated by Figure 4.6 (a), when other parameters are fixed, $P_n(\mu, t)$ increases as n becomes large. This is because, a node will have more neighbors that are not failed or compromised, as it has more neighbors. As a result, the node will have higher probability to update its group key successfully. On the other hand, as n



(a) (b)

$N = 2000$, $n = 20$, $r_c = 1.4e-7$ (expected life time of a node: 30 days), $t = 2$ days

Fig. 4.5. Impact of r_f



(a) (b)

$N = 2000$, $r_f = 3.9e-7$ (expected life time of a node: 30 days)
$r_c = 1.4e-7$ (compromise rate: 1 node per hour), $t = 1$ day

Fig. 4.6. Impact of n

increases, the adversary can compromise more shares of a e-polynomial. Consequently, as shown in Figure 4.6 (b), $P_c(\mu, t)$ also increases accordingly.

4.3.4.3 The Approach for Parameter Selection

Having studied how $P_n(\mu, t)$ and $P_c(\mu, t)$ are affected by μ, t, r_f, r_c and n, we now describe how a node determines its value of μ to achieve a desired level of system security. The approach will be explained through an example.

Suppose we want to deploy a sensor network that is composed of 2000 nodes. The expected life time of each node is 30 days; i.e., $r_f = 3.9e-7$. The rate for the adversary to compromise the sensor node is no larger than 1 node/hour; i.e., $r_c \leq 1.4e-7$. We aim to achieve the goal that, during the 6 days after deployment, at least 90% nodes, if alive, can update their group keys successfully, and the probability for the adversary to compromise one group key is lower than 0.01. That is, $P_n(\mu, t) \geq 0.9$ and $P_c(\mu, t) \leq 0.01$ for $t = 518400s$.

Table 4.1. Lookup table for selecting μ ($N = 2000$,$r_f = 3.9e-7$,$r_c = 1.4e-7$,$t = 6$ days; $P_n(\mu, t) \geq 0.9$, $P_c(\mu, t) \leq 0.01$)

Number of neighbors	12	14	16	18	20	22	24
μ	6	6-8	7-9	7-10	7-12	8-13	8-14

For each node to select its value of μ appropriately, the following steps are executed:

(1) The setup server computes $P_n(\mu, t)$ and $P_c(\mu, t)$ based on the known values of t, r_f and r_c.

(2) The server construct a lookup table as shown in Table 4.3.4.3. This table shows the range from which the value of μ is selected if the number of neighbors (i.e., n) is known. For example, if a node has 16 neighbors, μ must be 7, 8 or 9 such that $P_n(\mu, t) \geq 0.9$ and $P_c(\mu, t) \leq 0.01$ for $t = 518400s$.

(3) Before a node is deployed, the server preloaded the lookup table to it.

(4) After a node has discovered its neighbors, it select a value for μ according to the preloaded lookup table.

4.3.5 Detecting False Shares

In addition to compromising nodes and breaking group key polynomials, the adversary may prevent a normal node from updating its group key by injecting false shares to the network or letting an undetected compromised return false shares to its neighbors. To thwart this type of attacks, a node must be able to authenticate its shares. Specifically, a node (say N_u) can keep some *signature*s of the correct shares, and use them to detect and filter false shares. A naive approach for generating signatures is to use hash functions. That is, for each share s, N_u keeps a hashed value $h(s)$ for authentication. This method, however, is not scalable. For example, if a node has 20 neighbors and it is expected to update its group keys for 100 times, it has to store 2000 hashed values.

Note that a Mote of current generation has only $4KB$ for storage. So this poses a high storage requirement.

To achieve scalability, we propose a polynomial-based technique for signature generation and authentication. Specifically, the key polynomial of N_u, i.e., $e_u(x, y)$, can be constructed as $e_u(x, y) = \alpha(x, y) \times d_u(x, y) + q_u(x, y)$, where $\alpha(x, y)$, $d_u(x, y)$ and $q_u(x, y)$ are all polynomials constructed over finite field $F(q)$. When distributing the shares of $e_u(x, y)$, N_u sends $e_u(x, v)$ and $\alpha(x, v)$ to each neighbor N_v. Then, N_u removes $e_u(x, y)$ and $\alpha(x, y)$, but keeps $d_u(x, y)$ and $q_u(x, y)$. Note that, N_u can deliberately select $d_u(x, y)$ and $q_u(x, y)$ such that most of the coefficients of these polynomials are zero and hence the required storage space is small. At the time for updating group keys from version $c - 1$ to c, N_v returns both $e_u(c, v)$ and $\alpha(c, v)$ to N_u. Based on $d_u(x, y)$ and $q_u(x, y)$, N_u accepts a received share (say $\hat{e}_u(c, v)$ and $\hat{\alpha}(c, v)$ from a neighbor (N_v) only if $\hat{e}_u(c, v) = \hat{\alpha}(c, v) * d_u(c, v) + q_u(c, v)$.

4.4 Enhancements

The B-PCGR scheme is effective on the condition that: (1) no node will be compromised together with $\mu + 1$ or more neighbors, and (2) the adversary can not obtain $t + 1$ or more keys from the same group. In some hostile scenarios, the above conditions can be violated. To deal with these limitations, we propose two enhanced PCGR schemes in this section.

4.4.1 Cascading PCGR (C-PCGR) Scheme

The C-PCGR scheme is proposed to address the first limitation of B-PCGR. In this scheme, the e-polynomial shares of N_u are distributed to its multi-hop neighbors, instead of only to its one-hop neighbors; at the same time, the e-polynomial shares are distributed/collected in a cascading way, and hence does not introduce much communication/storage overhead.

4.4.1.1 The Scheme

The C-PCGR scheme is designed based on the B-PCGR scheme, and it also includes three steps. However, it differs from B-PCGR in the second and the third steps, which are described in the following. To simplify the presentation, we only describe the case where the e-polynomial shares are distributed to its 1- and 2-hop neighbors, while the scheme can be extended to more general cases.

Polynomial Encryption and Share Distribution

After each node (N_u) has been deployed and has discovered its neighbors, it randomly picks two e-polynomials: one is called *0-level e-polynomial* (denoted as $e_{u,0}(x,y)$), and the other is called *1-level e-polynomial* (denoted as $e_{u,1}(x,y)$). In both e-polynomials, the degree of x and y are t and μ, respectively.

Using the 0-level e-polynomial (i.e., $e_{u,0}(x,y)$), each node N_u can encrypt its g-polynomial (i.e., $g(x)$) to get its g'-polynomial (i.e., $g'(x)$). The encryption is conducted as follows:

$$g'(x) = g(x) + e_{u,0}(x,u). \tag{4.15}$$

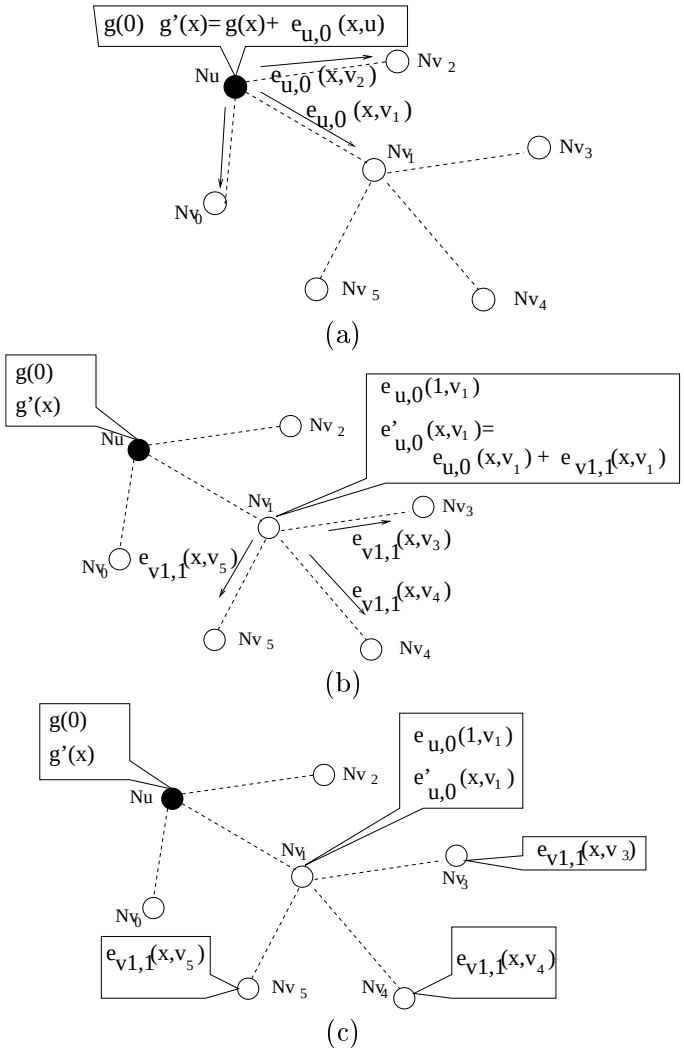


Fig. 4.7. C-PCGR: Polynomial Decryption and Share Distribution

After that, as shown in Figure 4.7 (a), N_u keeps $g(0)$ (i.e., the current group key), removes $g(x)$ and $e_{u,0}(x,u)$, and distributes the shares of $e_{u,0}(x,y)$ to its neighbors. Specifically, each neighbor N_v is given $e_{u,0}(x,v)$.

Having received 0-level e-polynomial shares from its neighbors, as shown in Figure 4.7 (b), each node (say N_v) uses its 1-level e-polynomial (i.e., $e_{v,1}(x,y)$) to encrypt each received 0-level share (i.e., $e_{u,0}(x,v)$) to obtain

$$e'_{u,0}(x,v) = e_{u,0}(x,v) + e_{v,1}(x-1,v) \tag{4.16}$$

After that, N_v keeps $e'_{u,0}(x,v)$ and $e_{u,0}(c+1,v)$, which will be returned to N_u at the next key updating time. It also removes $e_{u,0}(x,v)$, and distributes the shares of its 1-level e-polynomial ($e_{v,1}(x,y)$) to its neighbors. Figure 4.7 illustrates how the e-polynomial shares of N_u are distributed to its 1-hop and 2-hop neighbors.

Key Updating

To update keys, as shown in Figure 4.8 (a), each innocent node N_u increases its c by one, and returns shares $e_{v,0}(c,u)$ and $e_{v,1}(c,u)$ to each trusted neighbor N_v (Here, we assume that N_u has received shares $e_{v,0}(x,u)$ and $e_{v,1}(x,u)$ from N_v before.). At the same time, N_u receives its own 0-level and 1-level e-polynomial shares from its neighbors (i.e., $e_{u,0}(c,v)$ and $e_{u,1}(c,v)$ from each trusted neighbor N_v).

Having received $\mu+1$ 0-level e-polynomial shares, as shown in Figure 4.8 (a), N_u reconstructs a unique polynomial $e_{u,0}(c,x)$. Knowing $e_{u,0}(c,x)$, N_u can compute its new group key $g(c) = g'(c) - e_{u,0}(c,u)$.

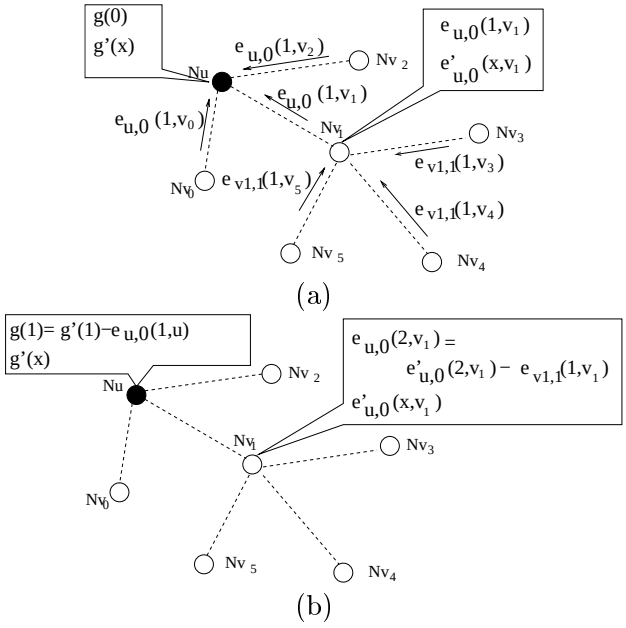


Fig. 4.8. C-PCGR: Key Updating (C is increased from 0 to 1)

Having received $\mu + 1$ 1-level e-polynomial shares, as shown in Figure 4.8 (a), N_v computes a unique polynomial $e_{v,1}(c, x)$, and then generates a share $e_{u,0}(c+1, v) = e'_{u,0}(c+1, v) - e_{v,1}(c, v)$, which will be returned to neighbor N_u at the next key updating time.

4.4.1.2 Security Analysis

The security property of the C-PCGR scheme can be expressed by Theorem 3.

THEOREM 3. *For a certain group, its g-polynomial $g(x)$ can be compromised if and only if:*

(1.1) a node N_u in the group is compromised, and

(1.2) the adversary has compromised at least $\mu + 1$ neighbors of N_u, each of which also has $\mu + 1$ neighbors compromised;

or

(2) at least $t + 1$ past keys of group i are compromised.

Proof Similar to the proof of Theorem 2.

4.4.2 Random Variance-Based PCGR (RV-PCGR) Scheme

The RV-PCGR scheme aims to address another limitation of B-PCGR. Specifically, as shown in Figure 4.9 (a), if the adversary has obtained $t + 1$ keys of a certain group, e.g., $g(0), g(1), \cdots, g(t)$, the adversary can break the g-polynomial of the group (i.e., $g(x)$) based on these keys.

4.4.2.1 Basic Idea

The basic idea of the RV-PCGR scheme is illustrated in Figure 4.9 (b). Let the length of $g(j)$ be $2L$ bits. We can add a L bit random number (called *random variance*) σ_j to $g(j)$ to obtain $g^r(j)$, such that the highest L bits of $g^r(j)$ are the same as those of $g(j)$, but the lowest L bits are different. This can be achieved by constructing the polynomials and conducting the addition/multiplication operations over an extended finite field [43] $F(2^{2L})$, in which the addition operation is defined as *modulo-2 addition*. Using some techniques (presented next), we can guarantee that the adversary can obtain only $g^r(j)$, not knowing the original $g(j)$. Based on $g^r(j)$ $(j = 0, \cdots, t)$, the adversary can construct a t-degree polynomial $g^r(x)$. But $g^r(x)$ is different from $g(x)$. That is, the adversary cannot break the future keys of the group.

Certainly, the adversary may guess each σ_j and attempts to find out each original $g(j)$ $(g(j) = g^r(j) + \sigma_j)$. However, since each σ_j can be an arbitrary number picked from $\{0, \cdots, 2^L - 1\}$, the probability of guessing correctly is $\frac{1}{2^{(t+1)L}}$. Consider an example, in which $t = 9$ and $L = 64$, the probability is as low as $\frac{1}{2^{640}}$. In the following, the scheme for implementing the above basic idea is described in detail.

4.4.2.2 The Scheme

The RV-PCGR scheme also has three steps.

Predistributing G-Polynomials

Similar to B-PCGR, the setup server decides the total number of groups, and picks a t-degree univariate g-polynomial for each group. Also, each node N_u is given a g-polynomial $g(x)$ when it is deployed.

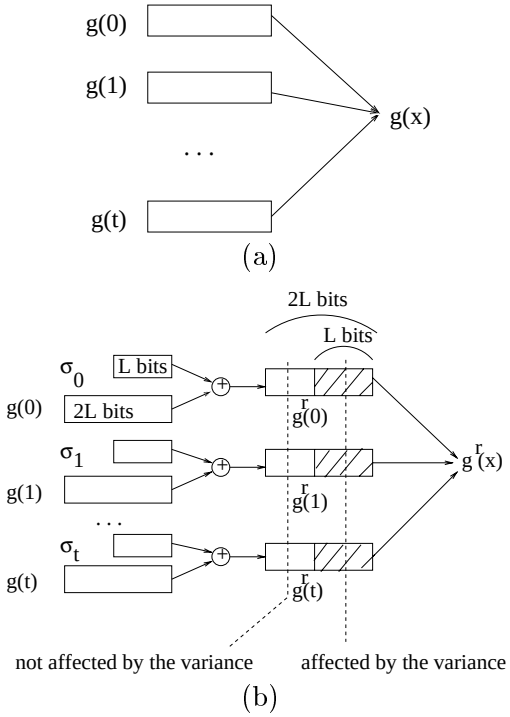


Fig. 4.9. Basic Idea of RV-PCGR

Different from B-PCGR, each $g(x)$ is constructed over an extended finite field $F(2^{2L})$, where L is the length of a group key (e.g., $64bits$). Also, the group key of any version j is defined as the highest L bits of $g(j)$, instead of $g(j)$ itself.

Encrypting G-Poly. and Distributing Components

Similar to B-PCGR, after N_u has been deployed and has discovered its neighbors, it randomly picks a t-degree e-polynomial $e_u(x)$[1]. Using the e-polynomial ($e_u(x)$), N_u encrypts its g-polynomial ($g(x)$) to get its g'-polynomial ($g'(x) = g(x) + e_u(x)$). After that, N_u randomly decomposes $e_u(x)$ into $\mu + 1$ components, denoted as $e_{u,i}(x)$ ($i = 0, \cdots, \mu$), such that

$$\sum_{i=0}^{\mu} e_{u,i}(x) = e_u(x). \tag{4.17}$$

These components are evenly distributed to the neighbors, and each neighbor gets only one components.

Key Updating

To update keys, each innocent node N_u increases its key version counter c by one, and returns

$$e^r_{v,j}(c) = e_{v,j}(c) + \sigma'_{c,v} \tag{4.18}$$

to each trusted neighbor N_v. Here, we assume that N_u has received share $e_{v,j}(x)$ from N_v before, and $\sigma'_{c,v}$ is randomly picked from $\{0, \cdots, 2^L - 1\}$. At the same time, N_u also receives $e^r_{u,i}(c)$ from N_v.

[1] In this subsection, the e-polynomial shares distributed to a neighbor is randomly constructed, irrelevant to the identity of the receiver. Thus, we can remove the second parameter y in the polynomial $e_u(x, y)$ as in B-PCGR.

Having received $\mu + 1$ distinct shares $\langle v_i, e^r_{u,i}(c)\rangle$ $(i = 0, \cdots, \mu)$, N_u computes $e^r_u(c) = \sum_{i=0}^{\mu} e^r_{u,i}(c)$. Knowing $e^r_u(c)$, N_u can compute $g^r(c) = g'(c) + e^r_u(c)$, and the highest L bits of $g^r(c)$ are used as the new group key.

4.4.2.3 Security Analysis

During the key updating process, the share returned by each node is added a random variance (refer to Eq. (4.18)). Therefore, the $g^r(c)$ calculated by node N_u has already included a random variance. Without loss of generality, we suppose the adversary has obtained $g^r(i)$, where $i = 0, \cdots, n-1$ and $n \geq t+1$, the adversary can use the process described in the following to guess the original $g(x)$; i.e.,

$$g(x) = \sum_{j=0}^{t} D_j x^j. \tag{4.19}$$

For every distinct $(t+1)$-tuple of variances $\langle \sigma_0, \cdots, \sigma_t \rangle$, where $\sigma_j \in \{0, 1, \cdots, 2^L - 1\}$ $(k = 0, \cdots, t)$:

(1) The guessed value of each D_j, denoted as $\hat{D}_j$, is calculated by solving the following $t+1$ $(t+1)$-variable linear equations:

$$\sum_{j=0}^{t} (i)^j \hat{D}_j = g^r(i) + \sigma_i, \text{ where } i = 0, \cdots, t. \tag{4.20}$$

(2) Let $g(x) = \sum_{j=0}^{t} \hat{D}_j(x)$. If the highest L bits of $g(i)$ are the same as those of $g^r(i)$, where $i = t+1, \cdots, n-1$, this $g(x)$ is recorded as a *candidate polynomial* of the original $g(x)$.

If the total number of the recorded candidate polynomials is 1, the candidate polynomial is the original $g(x)$; otherwise, one of the candidate polynomials is randomly picked as the guessed $g(x)$. Obviously, the original $g(x)$ is among the recorded candidate polynomials. However, the complexity to find out the candidates is as high as $o(2^{(t+1)L})$. For example, if $t = 9$ and $L = 64$, the complexity is 2^{640}.

4.5 Performance Evaluations

In this section, we first use the analytic approach to compare the performance of the proposed PCGR schemes and some previously proposed group rekeying schemes. Then, we conduct simulations to show that the proposed PCGR scheme can significantly improve the performance of filtering false messages.

4.5.1 Performance Analysis

Before analyzing the performance of different schemes, we list some notations that are used in this section as follows:

- N: the total number of nodes in the network.
- n: the average number of trusted neighbors that a node has.
- n_c: the number of (compromised) nodes that should be evicted.
- L: the length (in bits) of a group key.

4.5.1.1 Comparing the Performance of PCGR Schemes

We compare the performance of the proposed PCGR schemes in terms of communication cost, computation overhead, and storage requirement. The main results are shown in Table 4.2.

Communication Cost

In the B-PCGR scheme, each innocent node needs to send out n messages to its trusted neighbors during each key updating process. Each message includes one share, which has L bits. Therefore, nL bits are sent out by each node.

Similar to B-PCGR, the C-PCGR scheme also requires each innocent node to send out n messages for key updating. However, the size of each message includes two shares, which has $2L$ bits. Therefore, each node has to send out $2nL$ bits.

In RV-PCGR, each innocent node should send out n shares to its trusted neighbors, and each share has $2L$ bits. Therefore, each node needs to sent out $2nL$ bits for each key updating.

Computational Overhead

In all three schemes, each share sent/received by a node should be encrypted/decrypted using pairwise keys to prevent eavesdropping, and the total number of messages sent/received is $2n$ during each key updating process. Therefore, each node needs $2n$ encryptions/decryptions. Next, we discuss other computation overhead of each scheme.

In the B-PCGR scheme, each node N_u first needs to evaluate $n+1$ t-degree polynomials ($e_{v_i}(c)$ and $g'(x)$) to compute n shares and $g'(c)$, which needs $o((n+1)t^2)$

multiplications. After receiving $\mu + 1$ or more shares, it also needs to solve a $(\mu + 1)$-variable linear equation group to compute $e_u(c, u)$, and the computational complexity for solving such an equation group is $o(\mu^3)$ multiplications/divisions. The total computational complexity is $o((n + 1)t^2 + \mu^3)$ multiplications/divisions over $F(q)$.

In C-PCGR, each node needs to compute n more shares and solve one more $(\mu + 1)$-variable linear equation group, so the total computational complexity is $o((2n + 1)t^2 + 2\mu^3)$ multiplications/division over $F(q)$

In RV-PCGR, each node also needs to evaluate $n + 1$ t-degree polynomials to compute n shares and $g'(c)$. However, after receiving $\mu + 1$ or more shares, it only needs to add up these shares, instead of solving a equation group. Therefore, the total computational complexity is $o((n + 1)t^2)$ multiplications. However, the multiplications are conducted over a larger field $F(2^{2L})$.

To reduce rekeying delay caused by computations, most of the above computations (i.e., computing and encrypting shares, as well as computing $g'(c)$) can be performed beforehand, and only a few other computations (i.e., decrypting shares and computing $e_u(c, u)$) should be performed during the key updating time. To distinguish these two types of computational overhead, as shown in Table 4.2, we list both the computational overhead at key updating time and the total computational overhead for each key updating.

Storage Requirements

In the B-PCGR scheme, each node N_u needs to store the following information:

- the g'-polynomial $g'(x)$, which needs $(t + 1) \cdot L$ bits to store its coefficients.

- the shares of its neighbors' e-polynomials, i.e., $e_{v_i}(x,u)$ $(i = 0, \cdots, n-1)$, which need $n(t+1)L$ bits.

The total storage requirement is $(n+1)(t+1)L$ bits.

In C-PCGR, each node N_u needs to store the following information:

- the g'-polynomial $g'(x)$, which needs $(t+1) \cdot L$ bits to store its coefficients.
- the shares of its neighbors' 0-level e-polynomials, i.e., $e_{v_i,0}(x,u)$ $(i = 0, \cdots, n-1)$, which need $n \cdot (t+1) \cdot L$ bits.
- the shares of its neighbors' 1-level e-polynomials, i.e., $e_{v_i,1}(x,u)$ $(i = 0, \cdots, n-1)$, which need $n \cdot (t+1) \cdot L$ bits.

The total storage requirement is $(2n+1) \cdot (t+1) \cdot L$ bits.

The storage requirement of the RV-PCGR scheme is similar to B-PCGR, except that each coefficient of the polynomials has a length of $2L$. Thus, the overall storage requirement is $2(n+1)(t+1)L$ bits.

Summary

Table 4.2 compares the performance of B-PCGR, C-PCGR and RV-PCGR. From the table, we can see that C-PCGR and RV-PCGR have higher communication, computation, and storage overhead than B-PCGR, but achieves a higher level of security.

4.5.1.2 Comparison with Other Group Rekeying Schemes

Table 4.3 compares B-PCGR with some previous schemes, i.e., SKDC [26], LKH [49] and the Blundo's scheme [6]. We only compare the costs of these schemes related

Table 4.2. Comparing B-PCGR, C-PCGR and RV-PCGR

	B-PCGR	C-PCGR	RV-PCGR
Data sent/received by each node (bits)	nL	2nL	2nL
Rekeying delay	$o(1)$	$o(1)$	$o(1)$
Computational overhead at key updating time	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$ $(q > 2^L)$	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$	n decryptions
Total computational overhead per node	$2n$ encryptions/decryptions, $o(\mu^3 + (n+1)t^2)$ multiplications/divisions over $F(q)$	$2n$ encryptions/decryptions and $o(2\mu^3 + (2n+1)t^2)$ multiplications/divisions over $F(q)$	$2n$ encryptions/decryptions and $o((n+1)t^2)$ multiplications/divisions over $F(2^{2L})$
Storage requirement per node (bits)	$(n+1)(t+1)L$	$(2n+1)(t+1)L$	$2(n+1)(t+1)L$

to key updating, and n_c represents the number of (compromised) nodes that should be evicted.

In the SKDC scheme, the central controller sends a new key to each trusted node individually. We assume that such a message should go through $\sqrt{N}/2$ hops in average, and each new key has L bits. Therefore, the total traffic introduced by key updating is $\frac{(N-n_c)\cdot L\cdot\sqrt{N}}{2}$, and the average size of the data sent/received by a node is $\frac{(N-n_c)L}{2\sqrt{N}}$. To finish a key updating, each node should receive the new key, which takes $o((N-n_c)\sqrt{N}/2)$ units of time. The scheme is efficient in terms of computation and storage. Each node needs only one decryption and stores one key.

When analyzing the LKH scheme, we assume that a binary logic key hierarchy is used. Let s_c represent the size of the common ancestor tree (CAT) [15] of the evicted nodes. This scheme requires that each node on CAT should change its key encryption key (KEK) and notify the KEK to its two children (except the evicted nodes). Therefore, $2S_c - n_c$ keys should be transmitted. Each node should receive the keys, which results in a rekeying delay of $o(\sqrt{N})$ units of time. Also, each node in this scheme should keep $logN$ (the height of tree) number of KEKs, and hence the storage requirement is $LlogN$.

If the Blundo's scheme is used for distributedly generating a group key for up to N members, each node needs to store and compute a $(N-1)$-variable polynomial. Assume that the degree of the polynomial is t, the total storage requirement is as high as $N(t+1)L$ bits.

Comparing our B-PCGR scheme to the previous schemes, we can find that:

- B-PCGR has smaller rekeying delay than any other schemes.

Table 4.3. Comparing B-PCGR with previous group rekeying schemes

	SKDC	LKH	Blundo's	B-PCGR
Distributed	No	No	Yes	Yes
Data sent/received by each node (bits)	$\frac{(N-n_c)L}{2\sqrt{N}}$	$(2s_c - n_c)L$	$log(n_c)$	nL
Rekeying delay	$o(\frac{(N-n_c)\sqrt{N}}{2})$	$o(\sqrt{N})$	$o(\sqrt{N})$	$o(1)$
Maximum computational overhead per node (at key updating time)	1 decryption	$logN$ decryptions	evaluating a t-degree $(N-1)$-variable polynomial over $F(q)$ $(q > 2^L)$	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$
Maximum computational overhead per node (overall)	1 decryption	$logN$ decryptions	evaluating a t-degree $(N-1)$-variable polynomial over $F(q)$	$2n$ encryptions/decryptions, $o(\mu^3 + (n+1)t^2)$ multiplications/divisions over $F(q)$
Storage requirement per node (bits)	L	$LlogN$	$N(t+1)L$	$(n+1)(t+1)L$

- B-PCGR generates less traffic than the centralized schemes (SKDC and LHK).

- As a distributed scheme, B-PCGR requires each node to perform some computations that are performed solely by the central controller in a centralized scheme. Therefore, the computational cost (per node) of B-PCGR is larger than the centralized schemes (SKDC and LHK), but it is smaller than the Blundo's scheme. From the table, we can see that only a small fraction of the computations should be performed at key updating time, so the rekeying delay should not be increased too much. Furthermore, the key updating process may be initiated once every several hours or even a couple of days, so the computational cost is not significant in the long term.

- The storage requirement of B-PCGR is smaller than the Blundo's scheme, but larger than SKDC and LHK. However, the storage requirement is not very large in most cases. For example, if $n = 20$, $t = 30$ and $L = 64bits$, the required storage space is about $5KB$. Note that, if the network density is very high and each node has many neighbors, the node may select only a subset of the neighbors to distribute shares. Thus, the storage requirement of each node can be reduced.

4.5.2 Simulations

We use simulations to study how the group rekeying scheme (i.e., B-PCGR) can improve the performance of filtering false message.

4.5.2.1 Simulation Model

In the simulation, 2000 sensor nodes are uniformly distributed to a $1000 \times 800m^2$ field, and the communication range of each node is $40m$. The stationary sink (base station) sits at one corner of the field.

We simulate the behavior of the adversary as follows: The adversary keeps on capturing and compromising sensor nodes. Every certain time interval (denoted as τ_c), the adversary can compromise (reprogram) one node and obtain the keys held by the node. After that, the node is put back to the network (with all the keys already compromised by the adversary). Therefore, every τ_c, the number of compromised nodes is increased by 1, and the number of compromised keys may also be increased if the newly compromised node has keys previously unknown to the adversary. Each compromised node attacks the system by injecting a false report every 10 second (We assume that an intruder will not inject false reports with higher rate, since the intruder is easier to be detected in that case.)

The original SEF scheme [54], the SEF scheme with intruder isolation (*SEF-i*) and the SEF scheme with periodical key updating (called *SEF-u*) are simulated and evaluated in terms of:

- the *injected message load,* which is the total number of hops traversed by each injected message (before it is dropped) in one second;
- the *control message load,* which is the total number of hops traversed by each control message related to the intruder isolation (identification) or key updating in one second;

- the *total message load,* which is the sum of injected message load and control message load.

4.5.2.2 Simulation Results

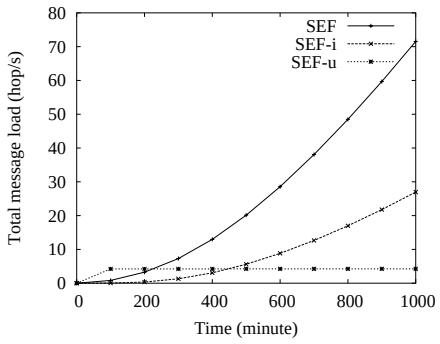


Fig. 4.10. Performance of the key updating scheme ($\tau_c = 10min$,$\tau_u = 100min$)

We first evaluate the key updating mechanism by comparing the performance of SEF-u to SEF and SEF-i. Figure 4.10 shows the result when the key updating interval (τ_u) is 100 minutes and the node compromise interval (τ_c) is 10 minutes. In the figure, a point corresponding to time t refers to the average message load during the 10 minute-phase ending at t.

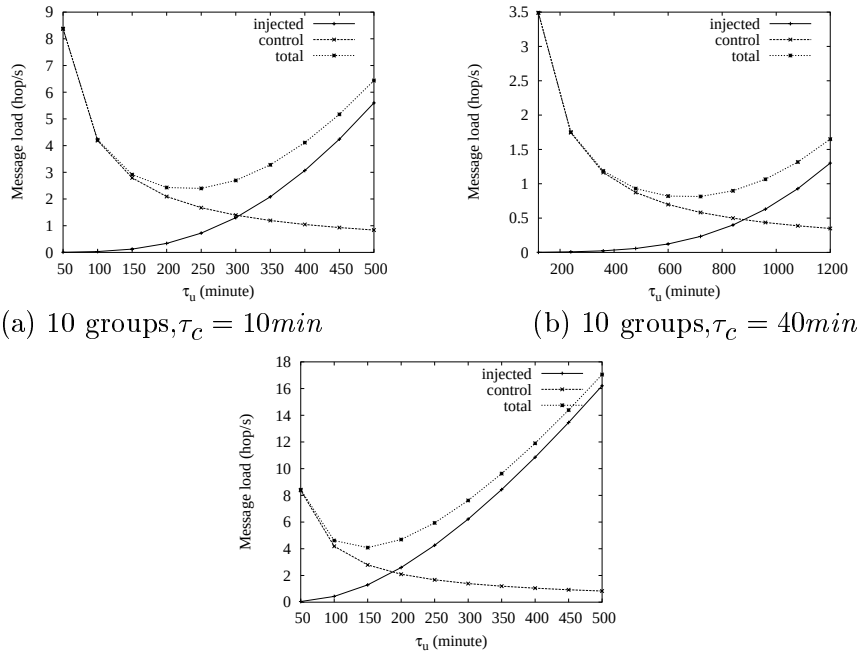
From the figure, we can see that SEF-i and SEF outperform SEF-u at the beginning of the network lifetime. This is due to the reason that there are very few intruders during that period. The message load injected by the intruders is small, and the intruder

isolation mechanism can effectively deal with the problem. In this case, if keys are periodically updated, it does not bring too much benefit, but increases the message load since each node needs to exchange information with its neighbors in order to update its keys.

As the attack continues, the trend is reversed and the SEF-u outperforms the other two. As shown in the figure, the message load increases rapidly in SEF and SEF-i. In SEF-u, since the keys are updated periodically, the keys compromised by the adversary become useless after the key updating. Therefore, the adversary can not continuously accumulate its knowledge about the keys to obtain a large portion or all keys. Certainly, some intruders may remain undetected and can renew its keys. However, these nodes have only one key after each key updating, and hence does not have significant impact.

When the periodical key updating mechanism is used, the total message load includes two components: the injected message load and the control message load (i.e., the messages exchanged between neighbors for key updating). Figure 4.11 shows the tradeoff between these two components. As key updating interval (τ_u) increases, the average control message overhead certainly decreases. At the same time, the average injected message load increases, since the adversary can compromise more keys to attack the network during each key phase (i.e., the period between two consecutive key updatings). Consequently, there exist an optimal τ_u, at which point the total message load is minimized.

From Figure 4.11 (a), (b) and (c), we can see the impact of the number of groups and parameter τ_c on selecting the optimal τ_u. When the number of groups is 10 and $\tau_c = 10min$, the optimal τ_u is between 150-250 minutes. As the number of groups



(a) 10 groups,$\tau_c = 10min$ (b) 10 groups,$\tau_c = 40min$

(c) 5 groups,$\tau_c = 10min$

Fig. 4.11. Tunning key updating interval (τ_u)

decreases (e.g., 5), it becomes easier for the adversary and compromise a larger portion of keys to cheat more innocent nodes. Therefore, the injected message load increases more quickly, and the optimal τ_u becomes smaller (i.e., 100-200 minutes). As τ_c increases, i.e., nodes are compromised more slowly, the injected message load also increases more slowly. Consequently, the optimal τ_u becomes larger (i.e., 500-700 minutes).

4.6 Discussions

4.6.1 Node Isolation and New Node Deployment

If a large fraction of neighbors are compromised, a node may not be able to update its group key and thus be isolated. To deal with this problem, some new nodes may be deployed to the isolated areas. After that, each new node distributes shares only to other new nodes to prevent compromised nodes from obtaining its shares. Also, an isolated innocent node can distribute its shares to these new nodes, which can help the node to update its group key and rejoin the network.

To enable the previous operations, each node should be given an initial master key K_0 before deployment. The key is used only during the first few minutes after deployment and must be removed after that. When a new node distributes a share to another new node, the share will be encrypted with K_0 to prevent old nodes from obtaining it. Also, a node (say N_u) should use K_0 to encrypt its $e_u(x, y)$ before removing $e_u(x, y)$ and K_0, and keep the encrypted polynomial. When N_u is isolated later, it will send the encrypted polynomial to newly deployed nodes. On receiving the encrypted

polynomial, a new node (say N_v) can decrypt it, obtain a share $e_u(x, v)$, and remove $e_u(x, y)$.

4.6.2 Other Issues

In the proposed schemes, group keys are updated periodically. While group keys are being updated in the network, nodes may not be able to send information to the sink since the group keys known by the nodes may not be consistent. To address this problem, information sent during this period should be authenticated using the old group keys.

If a node has many neighbors, it is not necessary to send a share to each of them. How to determine the number of neighbors that should receive a share is an important issue needing further investigation. If the number is too small, the node may quickly become isolated, since some of its neighbors are compromised or failed and it cannot get enough number of shares to update its group key. On the other hand, the security level could be decreased if too large number of neighbors have received a share.

Certainly, we cannot discuss all related issues in this paper. In our future work, we will further investigate the problems discussed above, and identify and address other related issues.

4.7 Summary of Distributed Group Rekeying Schemes

In this chapter, we proposed a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes to address the node compromising problem and to improve the effectiveness of filtering false data in sensor networks. These schemes are based on the idea that future group keys can be preloaded to nodes before deployment,

and neighbors can collaborate to protect and appropriately use the preloaded keys. Extensive analysis and simulations are conducted to evaluate the proposed schemes, and the results show that the proposed schemes can achieve a good level of security, outperform most previously proposed schemes, and significantly improve the effectiveness of filtering false data.

In addition to filtering false data, the proposed PCGR schemes can also be applied to other group rekeying problems, especially for scenarios (e.g., pebblenets [5]) where a group has a large number of widely spread members, the membership changes frequently, or it is very expensive to maintain a central key manager.

Chapter 5

Conclusion and Future Work

This chapter concludes the thesis and discusses some future work.

5.1 Conclusion

In this thesis, we studied the problem of employing sensor networks for monitoring mobile targets in unattended environments, and disseminating the obtained sensing data from dynamically changed detecting nodes to mobile sinks. We proposed a data-centric framework to address the problem. The proposed solution facilitates collaborative detection, efficient and secure data collection/dissemination.

For collaborative detection and efficient data collection, we proposed a dynamic convoy tree-based collaboration (DCTC) framework. In DCTC, sensor nodes surrounding a mobile target form a tree structure, which can be dynamically adjusted, to facility the collaborations among them and collect their sensing data. One big challenge in implementing DCTC is how to reconfigure the convoy tree efficiently. We formalized the problem as an optimization problem of finding a convoy tree sequence with high tree coverage and low energy consumption. To solve the problem, we first proposed an optimal solution based on dynamic programming. Considering the real constraints of a sensor network, we then proposed several practical implementation techniques. The practical solution includes two schemes for tree expansion and pruning (the conservative

scheme and the prediction-based scheme) and two tree reconfiguration schemes for tree reconfiguration (the sequential reconfiguration scheme and the localized reconfiguration scheme). The simulation results showed that the prediction-based scheme outperforms the conservative scheme, and it can achieve a relatively high coverage and low energy consumption close to the optimal solution. When the same tree expansion and pruning scheme is used, the localized reconfiguration performs better when the node density is high, and the trend is reversed when the node density is low.

To efficiently disseminate sensing data, we proposed an index-based data dissemination scheme with adaptive ring-based index (ARI). This scheme is based on the idea that sensing data are collected, processed and stored at the nodes close to the detecting nodes, and the location information of these storing nodes is pushed to some index nodes. To achieve fault tolerance and load balance, the index nodes for each type of targets form a ring surrounding the location which is determined based on the target type. To further improve the performance, we proposed several mechanisms to optimize the ARI scheme. Specifically, we proposed a *lazy index updating (LIU)* mechanism and a *lazy index query (LIQ)* mechanism to reduce the overhead of index updating and index querying, and hence improve the overall system performance. The analysis and simulation results show that the index-based scheme outperforms the ES scheme, the DCS scheme, and the LS scheme. The results also show that using the ARI scheme can tolerate clustering failures and achieve load balance, and the proposed optimization mechanisms can further improve the system performance.

In data collection and dissemination, sensing data need to be forwarded from sensor nodes to sinks or between sensor nodes. To achieve authenticity and confidentiality

in sensing data forwarding, cryptographic techniques can be used. Specifically, innocent sensor nodes can share group keys for data encryption and authentication. However, the group key-based techniques will become ineffective if some nodes are compromised since the adversary may obtain group keys from these nodes. To deal with node compromise, the compromised nodes should be identified, and the innocent nodes should update their group keys to prevent the adversary from utilizing the captured keys. Because most previously proposed group rekeying schemes have high overhead and are not suitable for sensor networks, we proposed a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes. These schemes are based on the idea that future group keys can be preloaded to nodes before deployment, and neighbors can collaborate to protect and appropriately use the preloaded keys. Based on thorough analysis, we show that the proposed schemes can achieve a good level of security, outperform most previously proposed schemes. Based on simulation results, we show that they can significantly improve the effectiveness of filtering false data.

5.2 Future Work

The thesis work can be extended in the following directions:

- **Collaborative detection of multiple targets:** Our proposed DCTC framework did not consider how to monitor a group of mobile targets efficiently. In order to monitor multiple mobile targets that may move together or separately now and then over the time, the DCTC framework need to be extended. In the extended

framework, convoy trees must be merged or split as mobile targets move closely or apart.

- **Data storage in sensor networks:** The storage capacity of current sensor nodes is limited by the cost and the form factors. The problem of insufficient storage is exacerbated since sensor networks could be long-live, and more and more data are generated and stored in the network as time elapses. To address the problem, some techniques such as spatial-temporal summarization and graceful aging of summaries [19, 18] have been proposed. However, they are suitable for monitoring stationary phenomena, but can not be directly applied in sensor networks for mobile target monitoring. To be applicable for mobile target monitoring, some particular issues need to be studied. For example, sensing data should be summarized based on the associated target type, instead of based on the spatial relationship as in [19, 18]. In addition, due to the mobility of monitored targets, the storage load may not be balanced in different areas, and bring out load balance issues.

- **Other Security Issues:** In this thesis work, we studied the schemes for data authenticity and confidentiality in sensing data forwarding. We did not consider other security issues such as DoS attacks, authorization, revocation, intrusion detection and recovery (or tolerance), and intruder identification. To secure data-centric sensor networks, we plan to address these issues in the future.

References

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

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

[3] A. Aljadhai and T. Znati. Predictive Mobility Support for QoS Provisioning in Mobile Wireless Environments. *IEEE Journal on Selected Areas in Communications*, 19(10), October 2001.

[4] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. *IETF Internet draft*, August 2000.

[5] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure Pebblenets. *ACM MobiHoc '01*, 2001.

[6] 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.

[7] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM'99)*, August 1999.

[8] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low Cost Outdoor Location For Very Small Devices. *IEEE Personal Communication, Special Issue on "Smart Space and Environments"*, October 2000.

[9] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao. Application Driver for Wireless Communications Technology. *ACM SIGCOMM Workshop on Data Communication in Latin America and Caribben*, 2000.

[10] A. Cerpa, J. Elson, M. Hamilton, and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. *The Proceeding of the First ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

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

[12] M. Chu, H. Haussecker and F. Zhao. Scalable Information-driven Sensor Querying and Routing for Ad Hoc Heterogeneous Sensor Networks. *International Journal of High Performance Computing Applications*, 2002.

[13] T. Cormen, C. Leiserson and R. Rivest. Introduction to Algorithms. *The MIT Press*, pages 514–543, 1990.

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

[15] D. McGrew and A. Sherman. Key Establishment in Large Dynamic Groups using One-Way Function Trees. *TIS Report No. 0755, TIS Labs at Network Associates, Inc. Glenwood, MD*, May 1998.

[16] W. Du and J. Deng. A Pairwise Key Pre-distribution Schemes for Wireless Sensor Networks. *The 10th ACM Conference on Computer and Communications Security*, 2003.

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

[18] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin and J. Heidemann. An Evaluation of Multi-resolution Search and Storage in Resource-constrained Sensor Networks. *ACM Sensys'03*, 2003.

[19] D. Ganesan, D. Estrin and J. Heidemann. Dimensions: Why do we need a new data handling architecture for sensor networks? *First Workshop on Hot Topics in Networks (Hotnets-1)*, 1, 2002.

[20] A. Ghose, J. Grobklags and J. Chuang. Resilient data-centric storage in wireless ad-hoc sensor networks. *Proceedings the 4th International Conference on Mobile Data Management (MDM'03)*, pages 45–62, 2003.

[21] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy and S. Shenker. DIFS: A Distributed Index for Features in Sensor Networks. *First IEEE Ineternational Workshop on Sensor Network Protocols and Applications*, May 2003.

[22] H.Chan and A. Perrig. Security and Privacy in Sensor Networks. *IEEE Computer*, October 2003.

[23] 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.

[24] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Network. *Mobicom'99*, August 1999.

[25] Y. Hu, A. Perrig, and D. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. *IEEE Infocom*, April 2003.

[26] H. Hugh, C. Muckenhirn, and T. Rivers. Group Key Management Protocol Architecture. *Request for comments (RFC) 2093, Internet Engineering Task Force*, March 1997.

[27] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication. *MobiCOM '00*, August 2000.

[28] D. Jayasimha. Fault Tolerance in a Multisensor Environment. *Proc. of the 13th Symposium on Reliable Distributed Systems (SRDS'94)*, October 1994.

[29] B. Karp and H. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. *The Sixth Aunual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Aug. 2000.

[30] L. Klein. Sensor and Data Fusion Concepts and Applications. *SPIE Optical Engr Press, WA*, 1993.

[31] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling Data-Centric Routing in Wireless Sensor Networks. *IEEE INFOCOM'02*, June 2002.

[32] D. Levine, I. Akyildiz, and M. Naghshineh. Resource estimation and call admission algorithm for wireless multimedia using the shahow cluster concept. *IEEE/ACM Trans. Networking*, 5(1), Feburary 1997.

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

[34] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. *ACM MobiCom*, August 2000.

[35] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: Security Suite for Sensor networks. *Mobicom'01*, 2001.

[36] G. Pottie and W. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5), May 2000.

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

[38] The CMU Monarch Project. The CMU Monarch Projects Wireless and Mobility Extensions to ns. *http://www.monarch.cs.cmu.edu/cmu-ns.html*, October 1999.

[39] S. RatNasamy, B. karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. *ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[40] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *ACM SIGCOMM*, August 2001.

[41] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer-to-Peer Networks. *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.

[42] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. *The 18th IFIP/ACM Symposium on Operating Systems Principles*, August 2001.

[43] S. Lin and D. Costello. Error-Correcting Codes. *Prentice-Hall, Inc.*, 1983.

[44] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM*, August 2001.

[45] Ivan Stojmenovic and Xu Lin. Power-Aware Localized Routing in Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1122–1133, 2001.

[46] G. Tel. Introduction to distributed algorithm. *Cambridge University Press*, 2000.

[47] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman. A Taxonomy of Wireless Micro-Sensor Network Models. *ACM SIGMobile Computing and Communications Review (MC2R)*, 6, April 2002.

[48] W. Zhang and G. Cao. Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration Based Approach. *IEEE Infocom'05*, March 2005.

[49] D. Wallner, E. Harder, and R. Agee. Key Management for Multicast: Issues and Architectures.

[50] G. Wang, W. Zhang, G. Cao, and T. La Porta. On Supporting Distributed Collaboration in Sensor networks. *IEEE Military Communications Conference (MILCOM)*, October 2003.

[51] C. Wong, M. Goudaand, and S. Lam. Secure Group Communications Using Key Graphs. *ACM SIGCOMM'98*, 1998.

[52] Y. Xu, J. Heidemann and D. Estrin. Geography Informed Energy Conservation for Ad Hoc Routing. *ACM MOBICOM'01*, July 2001.

[53] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang. A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks. *ACM International Conference on Mobile Computing and Networking (MOBICOM'02)*, pages 148–159, September 2002.

[54] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical En-route Filtering of Injected False Data in Sensor Networks. *IEEE Infocom'04*, March 2004.

[55] W. Zhang and G. Cao. DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks. *IEEE Transactions on Wireless Communication, in press, also available at: http://www.cse.psu.edu/∼gcao.*

[56] W. Zhang and G. Cao. DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks. *IEEE Transactions on Wireless Communication*, September 2004.

[57] W. Zhang and G. Cao. Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks. *IEEE Infocom'04*, March 2004.

[58] W. Zhang, G. Cao and T. La Porta. Data Dissemination with Adaptive Ring-Based Index for Wireless Sensor Networks. *IEEE International Conference on Network Protocols*, November 2003.

[59] W. Zhang, G. Cao, and T. La Porta. Data Dissemination with Ring-Based Index for Sensor Networks. *IEEE International Conference on Network Protocol (ICNP), also available at: http://www.cse.psu.edu/∼gcao*, November 2003.

[60] X. Zhang, S. Lam, D. Lee, and Y. Yang. Protocol Design for Scalable and Reliable Group Rekeying. *IEEE/ACM Transactions on Networking*, 11(6):908–922, December 2003.

[61] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. *Technical Report UCB/CSD-01-1141*, 2001.

[62] F. Zhao, J. Shin and J. Reich. Information-driven Dynamic Sensor Collaboration for Tracking Applications. *IEEE Signal Processing Magazine*, pages 68–77, March 2002.

[63] 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.

[64] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An Interleaved Hop-by-Hop Authentication Scheme for Filtering False Data in Sensor Networks. *IEEE Symposium on Security and Privacy*, 2004.

Vita

Wensheng Zhang was born in Fujian, China. He received the B.S. degree in Computer Engineering from the Tongji University, Shanghai, China, in July 1997, and the M.S. degree in Computer Science and Engineering from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in July 2000. He enrolled in the Ph.D. program in Computer Science and Engineering at The Pennsylvania State University in August 2000. He is a student member of the IEEE.

SELECTIVE PUBLICATIONS

- Wensheng Zhang, Hui Song, Sencun Zhu, and Guohong Cao, "Least Privilege and Privilege Deprivation: Towards Tolerating Mobile Sink Compromises in Wireless Sensor Networks," *ACM MOBIHOC*, May 2005.
- Wensheng Zhang and Guohong Cao, "Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration-Based Approach," *IEEE INFOCOM*, March 2005.
- Wensheng Zhang and Guohong Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks", *IEEE Transactions on Wireless Communication*, September 2004.
- Wensheng Zhang, Guohong Cao, and Tom La Porta, "Data Dissemination with Ring-Based Index for Wireless Sensor Networks," *IEEE International Conference on Network Protocols (ICNP)*, November 2003.