7-13-2010

Energy-Efficient Data Management in Wireless Sensor Networks

Chunyu Ai
Georgia State University

Follow this and additional works at: http://scholarworks.gsu.edu/cs_diss

ENERGY-EFFICIENT DATA MANAGEMENT IN WIRELESS SENSOR NETWORKS

by

CHUNYU AI

Under the Direction of Dr. Yingshu Li

ABSTRACT

Wireless Sensor Networks (WSNs) are deployed widely for various applications. A variety of useful data are generated by these deployments. Since WSNs have limited resources and unreliable communication links, traditional data management techniques are not suitable. Therefore, designing effective data management techniques for WSNs becomes important. In this dissertation, we address three key issues of data management in WSNs.

For data collection, a scheme of making some nodes sleep and estimating their values according to the other active nodes' readings has been proved energy-efficient. For the purpose of improving the precision of estimation, we propose two powerful estimation models,

Data Estimation using a Physical Model (DEPM) and Data Estimation using a Statistical Model (DESM).

Most of existing data processing approaches of WSNs are real-time. However, historical data of WSNs are also significant for various applications. No previous study has specifically addressed distributed historical data query processing. We propose an Index based Historical Data Query Processing scheme which stores historical data locally and processes queries energy-efficiently by using a distributed index tree.

Area query processing is significant for various applications of WSNs. No previous study has specifically addressed this issue. We propose an energy-efficient in-network area query processing scheme. In our scheme, we use an intelligent method (Grid lists) to describe an area, thus reducing the communication cost and dropping useless data as early as possible. With a thorough simulation study, it is shown that our schemes are effective and energy-efficient. Based on the area query processing algorithm, an Intelligent Monitoring System is designed to detect various events and provide real-time and accurate information for escaping, rescuing, and evacuation when a dangerous event happened.

INDEX WORDS: Data management, Data estimation, Historical data query, Area query, Wireless sensor networks

ENERGY-EFFICIENT DATA MANAGEMENT IN WIRELESS SENSOR NETWORKS

by

CHUNYU AI

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2010

ENERGY-EFFICIENT DATA MANAGEMENT IN WIRELESS SENSOR NETWORKS

by

CHUNYU AI

Committee Chair:	Dr. Yingshu Li
Committee:	Dr. Yi Pan
	Dr. Raheem Beyah
	Dr. Xu Zhang

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2010

DEDICATION

This dissertation is dedicated to my parents.

ACKNOWLEDGEMENTS

I would like to thank all of those people who helped make this dissertation possible.

First of all, thank my advisor, Dr. Yingshu Li, for all her guidance, encouragement, support, and patience. Dr. Li was always there to listen and to provide constructive advice. She taught me how to express my ideas and showed me different ways to approach a research problem. During difficulty times of my life, she understood my situation and totally supported me. Without her guidance and persistent help this dissertation would not have been possible.

I would like to thank my committee members, Dr. Yi Pan , Dr. Raheem Beyah, and Dr. Xu Zhang, for their very helpful insights, comments, and suggestions. Dr. Pan proposed thoughtful suggestions for improving my dissertation. Dr. Beyah taught me how to write an academic paper, guided me to solve the research issues, and provided me useful materials and information. Many thanks also goes to Dr. Raj Sunderraman who instructed me Ph.D. study and teaching.

A special thanks goes to Prof. Jianzong Li, my advisor of M.S., who brought me in the research field in the first place and encouraged me to pursue the Ph.D. degree.

In addition, I would like to thank my fellow graduate students and postdoc, Dr. Longjiang Guo, Yiwei Wu, Shan Gao, Chinh T. Vu, Jing He, Souling Ji, and Yueming Duan, who provided invaluable support and suggestions throughout this process. Thank my friends for sharing happiness.

Last, but not least, I thank my family: my parents, Shuqin Tian and Guozhen Ai, for giving me life, for educating me, for unconditional supporting me. Thank my brother, aunties, and uncles for support and understanding.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- WSN - Wireless Sensor Network

- BS - Base Station

- 2-D - Two Dimensional

Chapter 1

INTRODUCTION

A Wireless Sensor Network (WSN) is a wireless network composing of spatially distributed sensor nodes. Each sensor node is normally small, lightweight, and portable. A sensor node in a WSN is typically equipped with a transducer, a radio transceiver, a small microcontroller, and a power source (usually batteries). The transducer can convert sensed physical effects and phenomena into electrical signals. Consequently, sensor nodes can cooperatively monitor physical or environmental conditions to collect, record, and report data and information of monitored objects. Generally, monitored parameters are temperature, humidity, wind direction, speed, illumination intensity, sound intensity, vibration intensity, pressure, power-line voltage, chemical concentrations, motion, pollutant levels, and vital body functions. The microcontroller processes and stores the sensed data. The radio transceiver is responsible for receiving and sending data packets from/to other sensor nodes and BS. Also, each sensor supports a multi-hop routing algorithm, so a packet can be received by the destination node through multi-hop forwarding. The size of a single sensor node can vary from the size of a shoebox down to grain. Moreover, a sensor node costs from hundreds of dollars to a few pennies, depending on the size and the complexity requirements. Due to size and cost constraints of sensor nodes, there are corresponding constraints on resources including computational capability, memory capacity, energy, and bandwidth [1].

Initially, WSNs were designed for military applications such as battlefield surveillance. Nowadays WSNs are widely used in many civilian and industrial application areas, such as industrial process monitoring and control, machine health monitoring, robot control, environmental monitoring, habitat monitoring, healthcare applications, home automation, object tracking, detecting disasters, and traffic control. In the recent years, WSNs have entered a wide variety of systems and applications with various characteristics and requirements [1]

[2]. WSNs has attracted many researchers' attention in the field of computer science and telecommunications, and there are numerous workshops and conferences arranged each year.

1.1 Data Management in Wireless Sensor Networks

Nowadays large-scale sensor networks are widely deployed around the world for various applications. Tremendous volumes of various useful data are generated by these deployments. An interface between WSNs and users, which can extract the data users want and represent the data as the format users required, is significant. To develop this kind of interface, it is indispensable of supporting of data management techniques. However, compared to traditional networks or databases, a WSN has limited storage capacity, processing, analysis, computation abilities, bandwidth, and power supply. Obviously, traditional data management and processing methods are not suitable for WSNs. Thus, how to collect, store, process, and analyze data generated by WSNs is a challenging issue. In the past years, researchers proposed a lot of systems, algorithms, and methods to address data management problems of WSNs such as data collection, various aggregations, queries processing, event detection, and data mining. When we design algorithms for WSNs, the following are several characteristics and constraints of sensors and WSNs we have to consider.

1. Unstable and unreliable. The network topology of WSNs change frequently due to the following reasons:

   (a) Some sensor nodes have mobility.

   (b) New sensor nodes join in.

   (c) Some sensor nodes leave because of using up energy.

   Moreover, wireless communication is unreliable in practice since wireless channel is not reliable especially compared to wired channel. Due to practical issues such as link failures, high retransmission rate, and high packet missing rate, 100% communication reliability is not achievable.

2. Limited power supply. Due to the constraint of deployment environment, sensor nodes generally use batteries as power supply instead of wired power supply. For instance, the popular motes, Mica2, Micaz, and TelobB, use 2 AA batteries [3]. Normally, the lifetime of batteries can be several weeks or months [4]. Unfortunately, for some sensor nodes which are deployed at dangerous environment or unreachable spots, replacing the batteries is not possible. Therefore, prolonging the network lifetime is an important optimization goal for designing an algorithm of WSNs. Also, the network lifetime or energy consumption is usually used to judge the performance of an algorithm. Importantly, the communication operations, such as sending, receiving, and forwarding packets, are the most energy consumed operations compared to sensing, computing, etc. The energy consumption for transmitting 1 Kb a distance of 100 m is approximately the same as that for the executing 3 million instructions by 100 million instructions per second/W processor [5]. Thus, reducing unnecessary communication cost is an efficient way to save energy and prolong the network lifetime. Therefore, we only consider the communications cost (network traffic) when we evaluate the energy efficiency of the designed algorithms.

3. Low bandwidth. The bandwidth of WSNs is low because of the constraints of wireless radio,power supply, and other charactoristcs of WSNs. The transmit data rate of Micaz and TelobB's wireless transceiver is 250 kbps [3]. However, the bandwidth of WSNs cannot reach a sensor's maximum bandwidth since low qualify links, collisions, and high retransmission rate due to packets missing affect the transmission performance of WSNs. Moreover, for large-scale WSNs, the transmission latency is high since WSNs must use multi-hop routing mechanism due to the limited transmission range of a wireless transceiver.

4. Limited storage capacity and computational ability. The storage capacity and computation ability of a sensor node is restricted by the requirements of size and cost. For instance, the processor performance of a TelosB mote is 16-bit RISC. TelosB has

48K bytes program flash memory, 10K bytes RAM, and 1024K bytes measurement serial flash memory [3]. Therefore, the processing and storing ability of a single sensor node is limited. However, the ability of a WSN is powerful by making all sensor nodes cooperate with each other.

Aforementioned constraints of WSNs make designing data management algorithms more challenging.

1.2 Some Research Issues of Data Management in WSNs

In this section several research issues of data management in WSNs, which are addressed in my dissertation, are introduced.

1.2.1 Data Estimation Using Physical and Statistical Methodologies

Sensor-based applications extract different kinds of data via collecting data, processing in-network aggregations, and detecting complicated events. Since sensor nodes have limited resources, novel data management techniques are needed to satisfy application requirements, especially, the limitation of resources. To date, data acquisition using WSNs has became an essential ingredient of current technologies. Due to the energy limitations faced by sensor nodes, most techniques focus on *in-network aggregating data* so as to improve energy efficiency. Although this is a good way to conserve energy, for some applications it is necessary that all sensor readings are collected by BS. In such a scenario, *in-network aggregation* cannot be permitted as it reduces the data-set communicated to BS. Consequently, in such situations, *data-estimation* can prove to be a very fruitful alternative to *data aggregation*. An example is the determination of light intensity distribution in a region, such as photosensitive horticulture under artificial illumination [6]. Commercial gadgets are now available [7] for such purposes, the HortiSpec [7] was developed to measure light intensity and spectral distribution in the visible and NIR range inside greenhouses. The light intensity can be measured by these sensors, and data estimation techniques employed in such environments

would help conserve energy and prolong network lifetime. Commercial light sensors which use photodiodes that produce a voltage proportional to the light intensity are now available. Similarly, light intensity sensors are required to monitor maintenance of optimum lighting in animal husbandry related business [8], and also in cell-culture experiments [9] under artificial light, where also data estimation techniques would help enhance the longevity of WSNs.

Usually, users can accept approximate data. As a result, a method of data estimation is a perfect option for conserving energy. In fact, the key challenge is how to provide estimated data with high precision while consuming as little energy as possible. To address this problem, two novel data estimation methods are proposed in Chapter 3. In our scheme, a minimal number of nodes are set to be active and the other nodes are set to sleep. All nodes serve as active working nodes by turns. BS estimates values of sleeping nodes based on two estimation models, Data Estimation using Physical Model (DEPM) and Data Estimation using Statistical Model (DESM). The scheme not only prolongs network lifetime, but also fulfills users' expectations of data precision. Notably, the DEPM model is the first one to take advantages of *physical characteristics* of monitored attributes for estimation. Experiments on a real sensor network consisting of twenty TelosB nodes [10] were conducted to evaluate the proposed models. The experimental results show that the proposed methods are energy efficient.

1.2.2 In-network Historical Data Storage and Query Processing

Most existing applications just process real-time data generated by WSNs (e.g., [11, 12]). However, historical data of sensor networks are also significant to us, especially statistical meaning of historical data. For instance, maximum, minimum, and average temperatures of the past two months in a specific area are concerned in the weather monitoring application. By capturing rush hours and the bottleneck of traffic according to historical data, a large quantity of useful information can be provided to improve traffic conditions. Through analysis of historical data, some knowledge, principles, and characteristics can be discovered.

One simple method to process historical data is that BS collects all data and processes in

a centralized manner. Nevertheless, sensor nodes will deplete energy rapidly for continually reporting and forwarding data. Another method is storing data locally, that is, data are stored at a sensor node where they are generated. Intuitively, a sensor node cannot store a large quantity of historical data since its memory capacity is low. Popular motes such as Mica2 and Micaz [3] are equipped with a 512K bytes measurement flash memory, it can store more than 100,000 measurements. Another popular mote, TelosB [3], has a 1024K bytes flash memory. 512K or 1024K bytes are really small memory capacity. However, it is enough to store most of sensing data, sampling data, or statistical data during a sensor node's entire lifetime since the lifetime of a sensor node is short due to the limitation of batteries. For a Mica mote powered by 2 AA batteries, the lifetime is 5-6 days if continuously running, and it can be extended to over 20 years if staying sleep state without any activity [13]. For most applications, a sensor can live for several weeks or months [4]. Assume a Mica mote with a 512K bytes flash memory can live 3 months. $(100,000/90) = 1111$ measurements can be saved locally every day. Suppose we have 4 sensing attributes need to be saved, frequency of sampling can be 1 per 5 minutes which is normal in wireless sensor network applications. If proper compressing techniques are applied according to data's characteristics, much more data can be stored locally. Consequently, storing data locally at a sensor is feasible. The historical data can be downloaded when the battery is replaced or recharged if possible.

The motivation of storing historical data is to support historical data queries. However, locally storing data might cause a query to be flooded in the entire network to probe data when it is processed. We propose a scheme in Chapter 4 named Historical Data Query Processing based on Distributed Indexing (HDQP) which stores historical data in network and processes queries energy-efficiently by using index techniques. Historical data is stored at each sensor. For saving memory capacity, compressing and sampling techniques can be used according to data characteristics and users' requirements. To process queries quickly and energy-efficiently, in-network distributed tree-based indexes are constructed. Using indexes to process queries can reduce the number of involved sensors as many as possible, thus conserving energy consumption. To avoid load skew, index data are partitioned on different

nodes according to their time stamp. That is, there exist multiple index trees in the network. Which index trees are used depends on query conditions.

1.2.3   Area Query Processing
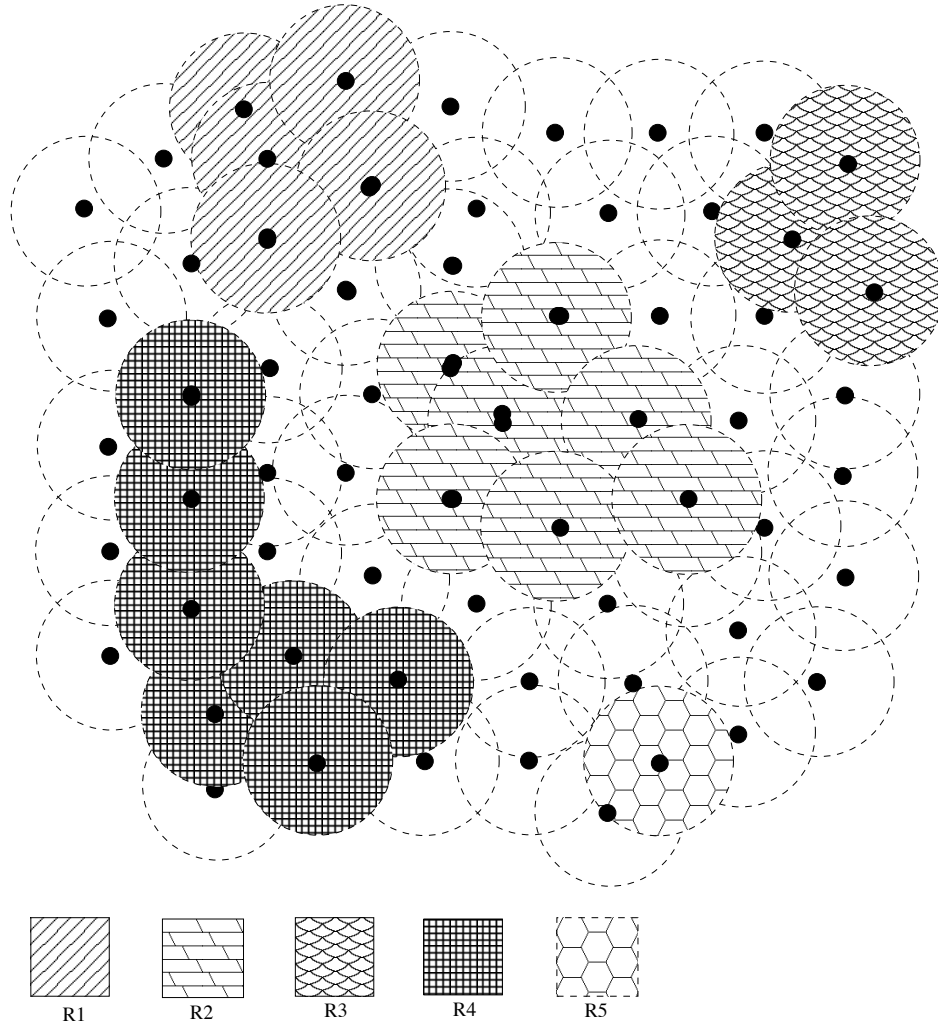
Figure 1.1. Area Queries

Most existing data collection systems are query-based ones. Traditional query processing techniques of WSNs mainly deal with retrieving sensor node locations, sensing values, and aggregating the sensed values. However, in a lot of applications, users expect information about areas of their interests. For instance, workers in a coal mine want to find an area with

a high oxygen density to take a break, and this area must be big enough to accommodate several workers. In Figure 1.1, all sensing values of the sensors in regions R1 through R5 reach the expected oxygen density. However, regions R3 and R5 are not big enough for several workers, so R3 and R5 are not the areas users expect. For this application, previous methods may only return the locations and sensing values of the sensors, which is meaningless and helpless. Here, users want to find areas instead of several locations since the size of an area should also be a filtering condition.

Another interesting scenario is an air pollution monitoring application within a city. Users expect this monitoring system to detect regions whose pollution levels reach thresholds. In reality, the polluted thresholds might be different for different areas. For example, the threshold of an industrial area is 80, and that of a residential area is 50. In Figure 1.1, R1 and R4 are in the industrial area, and R2, R3, and R5 are in the residential area. The polluted level of R1 through R5 are 70, 60, 40, 90, and 40 respectively. Usually, traditional methods apply the same filtering condition for the whole network. In this example, if the threshold 80 is used, polluted area R2 is missed since R2's polluted level actually exceeds the threshold for residential areas while the filtering condition does not indicate this. If the threshold 50 is used, R1 is identified as a polluted area which triggers a false alarm. So the existing methods cannot be used for this application. This application concerns specific areas, and different filtering conditions should be applied for each area.

We define *Area Query* as requests for area information including area locations, sizes of areas, and collected and aggregated data of areas. Sensor nodes with expected readings and adjacent sensing coverage are divided into the same group. The total coverage area of sensor nodes in the same group is a possible result area. Area Query can retrieve not only sensing values but also specific geographical information compared to traditional queries of wireless sensor networks. Area Query is more practicable and useful than traditional queries for some applications, which require geographical information. A common property of Area Query applications is that results of these queries are areas and aggregation values of sensors in these areas. Size of areas also can be query conditions. Furthermore, queries might be run

for either the whole network or sub-areas. For different areas, it is possible to use different querying conditions. Area Query is a new kind of query in wireless sensor networks.

Intuitively, BS can collect data from all sensor nodes and answer area queries. However, sensor nodes will drain energy quickly due to frequently reporting. Existing techniques of in-network query processing suppress and aggregate sensing values to save energy. However, these methods do not consider sensor coverage area as part of results. Consequently, a new in-network area query processing technique is necessary.

The first challenge of in-network area query processing is that it is hard to suppress useless data as early as possible. For instance, an area satisfies all the conditions except size of requirement. Obviously, this area is not the one expected by users. However, we cannot drop this area since it is difficult to decide the boundary of a result area locally. Moreover, how to describe an area in-network is another challenge. Because areas are also part of query results, ideal area description can reduce the size of transmitted data and the complexity of area size computation, which are two primary consideration factors for energy conservation.

In Chapter 5, we propose an energy-efficient in-network area query processing scheme. The whole sensor network is partitioned into grids, and a gray code is used to represent each grid. We construct a reporting tree to process merging areas and aggregations. For conserving energy, incremental update techniques are used to process continuous queries. Our contributions are summarized as follows:

1. A new area query is studied.

2. A smart area description, GID lists, is used to reduce the size of query results.

3. An energy-efficient in-network area query processing scheme is proposed. By using GIDs and merging strategies, partial results can be merged to reduce the size of results and useless data can be dropped as early as possible to reduce the number of messages.

4. An incremental update method is addressed to reduce the size of reports for continuous queries.

<div align="center">

Chapter 2

RELATED WORK

</div>

In this chapter, we begin with a brief introduction of related work to data management in WSNs. Then, related literature of research issues which are addressed in the dissertation is discussed.

2.1 Basic Approaches to Data Management in WSNs

First, we introduce some early data management approaches. In [14], a scalable and robust communication paradigm, *directed diffusion*, is proposed. Attribute-value pairs are used to name data generated by sensor nodes. A request node sends its interests of named data to destination sensor nodes or regions. Then, data satisfying the interest are returned along the reverse path of interest dissemination to the request node. To improve the performance and save energy, intermediate nodes can cache data and might aggregate the data. [15] extends and improves the directed diffusion method especially on experiments. TinyOS [16] is a free and open source operating system designing for WSNs. TinyOS is an embedded operating system which is written in the nesC. The component library of TinyOS consists of sensor drivers, network protocols, distributed services, and data acquisition tools. Therefore, TinyOS can support basic data requests. Moreover, users can develop their own applications based on TinyOS.

TinyDB [4] is a data management system for WSNs based on TinyOS. It can extract information from WSNs by sending queries. Importantly, TinyDB allows users describe the data they want to acquire by writing a simple, SQL-like query. For answering a query, TinyDB requests the data from sensor nodes in the network and routes it back to a PC. In the phase of processing queries, filtering and aggregation algorithms might be used. TinyDB uses intelligent in-network energy-efficient processing algorithms to prolong the network lifetime.

For instance, tree-based routing is used for query delivery, data collection, and in-network aggregation, and [17] is used to processing aggregation in in-network manner. Also, TinyDB supports metadata management, in-network persistent storage, multiple concurrent queries. REED [18] extends TinyDB with the ability to process joins operations between sensing data and static tables which is built outside the WSN. Filter conditions is stored in static tables, and then those tables are distributed throughout the network. Join operation is executed in in-network manner. REED is also suitable for various event detection applications which TinyDB and data collection systems cannot handle.

Cougar [19] is another distributed database system to sensor networks. It considered query languages, aggregation processing, query optimization, catalog management, and multi-query optimization. When a new query is coming, query optimizer of Cougar either merge it with an existing query or generate a new query plan. A leader is chosen to control the execution of the query plan.

Recently, researchers focus on specific issues which were found in some applications such as data estimation, data aggregation, event detection, and data centric storage. In the following sections we will introduce literature which is related to the issues we focus on.

2.2   Data Collection and Estimation

Common techniques to save energy in WSNs employ various alternative strategies to minimize data acquisition and/or transmission by using a variety of approaches. For instance, in-network aggregation (e.g., MIN [20] and AVG [17]) can reduce communication cost significantly. Also, using synopsis diffusion [21] and wavelets compression [22] to precess aggregation can decrease network traffic. However, aforementioned methods are not suitable for some applications which require all sensed data to be sent to BS since these methods do not provide detailed data desired by users. Apparently, reporting all sensed data will consume energy quickly. Therefore, researchers proposed some energy-efficient approximate algorithms to collect all sensed data.

Data collection is widely used for applications, which collects all sensed data contin-

uously ($SELECT * $ query). In [23], Chu *et al.* have proposed a mechanism, Ken, using conditional data transmission to conserve energy by reporting *only* if the difference between the sensed value and the predicted value is beyond certain bounds. The replicated dynamic probabilistic model for a sensor node is installed on both the sink and that sensor node. A sensor node does not need to report sensed value normally if the predicted error is within the threshold user specified, thus saving reporting communication cost. Once the predication error is greater than the error bound, the sensed data is reported to the sink, and the parameters of the model are adjusted to match current data changing intend. The effectiveness of this scheme in reducing the communication cost is relied on the predication model chosen. As shown in [24], the prediction models based on the temporal and spatial correlations of data work extremely well in WSNs. A novel disjoint-Cliques Model is also proposed to use spacial correlation to reduce data transmission and improve prediction accuracy. In [25], a data-driven approach was presented. Model-based suppression is used to provide continuous data without continuous reporting. In addition, a key problem for data suppression, link failure, is addressed. A mobile filtering approach for error-bounded data collection was proposed in [26]. By migrating filters wisely the number of data reporting is reduced significantly. Jain *et al.* have built dynamic procedures that employ maximum filtering of data using a technique called stochastic recursive data filtering, to conserve resources subject to meeting precision standards [27]. Primarily, these methods are aimed at reducing energy consumption by substituting data acquisition using data estimation. Another method of data estimation is based on collection of data samples for a relatively long time and calculating the autocorrelation of the vector of samples [28]. This approach aims at enabling nodes to identify patterns in the behavior of sensed processes and report only uncommon observable data to conserve energy.

Dash *et al.* have studied using physical laws for data estimation, approximate values of land surface temperature and emissivity from passive sensor data [29]. Essentially, this work makes use of well-known laws of Physics to estimate the Land Surface Temperature realistically. While physical laws have been used in some of technologies mentioned above to

govern sensor mechanisms, in the DEPM technique proposed by us in [30] it seems for the first time that physical laws are employed in a direct energy saving strategy aimed at data estimation. Most other methods rely on some forms of temporal and/or spatial correlations and/or data filters for data estimation whereas in the present work well-established physical laws that are known to be correct with extremely high accuracy are used directly in the network plan to achieve minimization of energy consumption which is the primary challenge in WSN technology. The data estimation scheme we propose can conserve more energy than the methods mentioned above obviously, because in the methods, such as [23, 27, 28], all nodes should be always in working status; however, in our scheme nodes serve as working nodes by turns, and others go to sleep, so our scheme can prolong network lifetime significantly.

2.3   Data Storage

Many approaches have been proposed to describe how to store data generated by WSNs. One category of such storage solutions is that BS collects and stores all data. Apparently, it is easy to store and access the data at BS. However, such methods (such as [31] [17] [32]) might be more applicable to answer *continuous queries*. Obviously, the mortal drawback of collecting all data is shortening the network lifetime for WSNs with limited power supply since sending packets costs large amount of energy. Also, shipping all data out of network may be not necessary because users are not interested in all data. Sensor nodes near BS become the bottleneck as a result of forwarding packets. Moreover, the WSNs may not able to transfer all data continuously generated by sensor nodes due to the limitation of bandwidth. Since centralized storage is not practical, naturally distributed in-network data storage is considered. Nevertheless, in-network data storage and data retrieve of WSNs are challenging issues for WSNs because each sensor node just has limited memory space and there are a number of sensor nodes in a WSN normally.

For improving network lifetime, in-network storage techniques have been addressed to solve ad-hoc queries. These frameworks are primarily based on the Data-Centric Storage (DCS) concept [33]. In DCS, relevant data are categorized and named according to its

meanings (e.g., tiger sightings). All data with the same general name will be stored at the same sensor node. Then, when users query the data with a particular name, it can be sent directly to the sensor node who stores those named data. The major difference among in-network DCS schemes is using different events-to-sensors mapping methods. The mapping was designed using hash tables in DHT [33] and GHT [34], or using k-d trees in DIM [35], KDDCS [36], and STDCS [11]. STDCS uses sensor location as data indexing instead of the sensed values. Hence, STDCS addresses a sensors-to-sensors mapping instead of the readings/events-to-sensors mapping. Moreover, a switching-time is defined to be the time duration after which the mapping function changes. STDCS uses a spatio-temporal indexing to balance query load among sensors.

As we know, indexing techniques can significantly improve the data acquiring/query performance. For WSNs, another benefit of using index is reducing cost of data request dissemination since the destination of data request can be obtained from the index. The works in [37], [35], and [36] use a spatially distributed hashing index technique to solve range queries in a multi-dimensional space. The work in [12] proposed a distributed spatial-temporal index structure to track moving objects. The work in [38] addressed a time-based index management for event query processing. For saving energy, the index is adjusted according to the event frequency.

Specially, most of these approaches just store partial data, which satisfy conditions or present events and moving objects, generated by sensors [34], [36], [35], [37], [12], and [38]. To our best knowledge, no in-network distributed historical data storage, indexing, and query processing schemes have been presented in the literature.

Since sensors have limited memory space, storing all historical data on sensors might not be applicable for some applications with a large quantity of sensing data. However, most applications desire statistical results, events, the trend of value changing of historical data rather than specific values. This characteristic provides us a chance to store sketch information to answer queries based on historical data of sensor networks. For enhancing query processing performance and saving energy, a distributed index is necessary to guide query

forwarding. In [39], we proposed a in-network historical data storage and query processing scheme based on distributed index. This scheme stores historical data locally and processes queries energy-efficiently by using a distributed index tree. Regression techniques can be used to save memory capacity according to data characteristics and users' requirements. In order to process queries quickly and energy-efficiently, in-network distributed tree-based indexes are constructed and maintained. Using indexes to process queries can significantly reduce the number of involved sensors, thus conserving energy consumption. Furthermore, for avoiding load skew, index data are partitioned on different nodes according to their time stamp.

2.4 Area Query

Area query processing is a special kind of data processing in wireless sensor networks. As we know, data processing of wireless sensor networks was well-studied. It mainly involves three aspects, data collection, data aggregation, and query processing. Resource-limitation and unreliable communication links are the main concerns.

Data collection techniques achieve ideal data reduction when bounded error is acceptable for users as discussed in Section 2.2. Data collection may be used to process area queries. That is, BS computes results after collecting data from all sensor nodes. However, since errors are not tolerable for area queries, then the methods which provide results with errors are not suitable for area query processing. If the methods which can provide accurate results are employed, energy consumption will be large since most of data cannot be suppressed and must be sent to BS.

In-network data aggregation is an efficient way to reduce energy consumption since data items are compressed on their way to the destination. TAG [17] and TiNA [40] aggregate sensed data as early as possible through tree-based routing. Aggregation of values from multiple sensor nodes are routed to one destination (BS). In [41], a many to many aggregation scheme was proposed. In this scheme, there are multiple destination nodes. Each destination may also have multiple source nodes. In [42], a continuous Spatial Aggregation was studied.

Users specify a set of spatial regions, data aggregation is then performed in each region. Existing data aggregation methods either consider the sensor nodes in the whole network as belong to one group or divide sensor nodes into static groups such as the works in [41] and [42]. Area query processing requires dynamic sensor grouping based on readings from each sensor. Therefore, the above mentioned methods cannot be used.

Query processing technique involves query optimization, decomposition, distribution, and result retrieval. A query processing system must have the ability to execute multiple continuous queries simultaneously. Cougar [19] system reduces communication cost by pushing operations such as selections and aggregations into the network. TinyDB [4] is a robust query processing system. A SQL-like interface is used to specify the data expected by users. It also supports event detection. Power-aware optimization, dissemination, routing, and execution techniques are used in TinyDB to prolong network lifetime. Query processing can be utilized for event detection. In an event detection system, when an event is detected, a warning should be delivered to users. In [18] and [4], thresholds are set for sensor readings in a query to detect events. In [43], the top-k monitoring issue over WSNs is addressed. Usually, sensor readings are correlated with location, so top-k nodes are clustered at some areas. A tree structure, partial ordered tree(POT), is used to maintain clusters with the highest sensed values, thus only potential top-k result are evaluated for query processing. In [44], events are abstracted into spatial-temporal patterns. In this scheme, event detection is effectively carried out through matching contour maps of sensed data to event patterns. Contour map is constructed in distributive manner via merging submaps. In [45], an algorithm for detection the boundary of events (such as detection the region in forest fire) is proposed. Since the detection algorithm is based on a simple clustering technique, the computation cost is low. In [46], robust median estimator based approaches for detection of the reach of events are proposed. In [47] [48], map-based world model (MWM) is proposed to model the physical world as a stack of maps by presenting temporal and sptial distributions of sensed data. MWM supports efficient event detection, prediction, and queries.

To the best of our knowledge, our work [49] is the first work to address area queries

specifically. We propose an area query processing scheme, which can handle not only normal queries like selection, aggregation, and threshold-based event detection but also area queries. More importantly, this scheme is energy-efficient since it processes area queries in in-network manner. Compared with existing techniques, the remarkable difference is that this scheme supports dynamic sensor grouping which is an indispensable step for area query processing. For dynamic sensor grouping, we consider not only geographical correlations of sensors' sensing coverage but also sensed data correlations of sensors to derive expected results.

Chapter 3

DATA ESTIMATION USING PHYSICAL AND STATISTICAL METHODOLOGIES

In this chapter, we introduce data estimation models using physical and statistical methodologies. In our scheme nodes serve as working nodes by turns, and others go to sleep, and sleeping nodes' readings are estimated by our data estimation models, so our scheme can prolong network lifetime significantly.

3.1 Problem Definition and Working Scenario

In this section, we formally define the SEAQA problem, and describe the working scenario of our estimation scheme.

In many applications, it is not necessary for users to obtain completely accurate values. A scheme that provides approximate answers might offer an opportunity to prolong network lifetime efficiently. We formally define this problem as the Sensor Energy-efficient Approximate Query Answer (SEAQA) problem.

Definition 1 (SEAQA Problem): Given a three dimensional area A and a set of N sensors $S = \{s_1, s_2, ..., s_N\}$, derive a working scheme *ws* for S such that:

1. For each sensor s_i, its returned value V_e deviates its real sensing value V as little as possible, that is, $|V_e - V|$ is minimized.

2. The energy consumptions among all the sensors are balanced.

3. The network lifetime is maximized.

The lifetime of a WSN depends on the node with the minimal remaining energy, so maintaining energy consumption balance among sensors is also a significant problem to

prolong network lifetime.

To conserve energy, some nodes in a network can be in sleep mode, and only a subset of the nodes are responsible for sensing and communication. Nodes work in *rounds* as shown in Figure 3.1. There are three phases in each round. In the first phase, BS gathers the remained energy information of all the nodes and selects a subset of nodes with much more remained energy to serve as *active working* nodes. In the second phase, BS informs each node its role and working duration. Then, in the rest of this round, the selected working nodes are in charge of sensing and communication, and the other nodes go to sleep in order to conserve energy. The sensing values of the sleeping nodes are estimated by the proposed DEPM and DESM methods at BS.

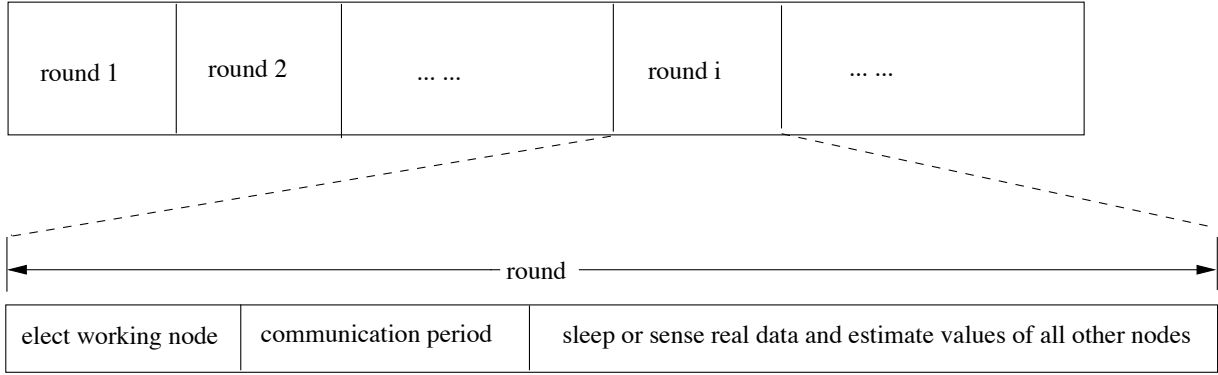Figure 3.1. Working rounds.

3.2 Estimation Models

To address the data estimation problem, we propose two efficient methods DEPM and DESM. DEPM estimates data using physical laws and DESM accomplishes data estimation using a statistical model.

3.2.1   Data Estimation using Physical Model (DEPM)

In this section, we introduce DEPM which employs basic laws of Physics. This is achieved by exploiting the principle of superposition of the physical sensed parameters. Essentially, the problem is to determine a physical quantity at a *field point* when its value is the result of linear superposition produced by *multiple sources*. DEPM does not need to deploy any more sensors than a constant number, that is the number of sources, and all the *extra* sensors go to sleep. The sensing values of sleeping nodes can be predicted by DEPM model according to sensing values of the active nodes. By activating one of the sleeping nodes, one can carry out an actual sensory measurement at that location and verify the prediction of the DEPM model. After such verification, DEPM enables. As long as source properties remain invariant, the prediction of the physical quantities at an infinite number of locations within the region R and makes it redundant to deploy sensors at these new locations. DEPM thus achieves enormous energy conservation by exploiting the principle of linear superposition and solving a set of algebraic linear inhomogenous equations. DEPM provides accurate physical estimates of the physical observable at an infinite number of field points in the region without having to consume energy in activating additional sensors.

To define and illustrate the functioning of DEPM, we use light intensity as an example monitored attribute. We consider a total of N nodes and M sources of light, where $M \ll N$. Each of the light sensors registers an amount of light intensity I_k at the node S_k ($k \in [1, N]$) in the system (Figure 3.2). BS needs to determine the *power radiated by each source of light* from the measurements of light intensities measured by nodes.

DEPM operates in two modes: *dynamic* and *static*. The dynamic mode of DEPM is employed when the power radiated by the M sources is not known and needs to be measured by active sensors. DEPM provides energy conservation by accurately estimating the power of the M sources by employing only M active sources. Toward this goal, DEPM exploits the principle of linear superposition and provides solutions by addressing a set of linear algebraic inhomogeneous equations. Energy minimization is achieved by requiring only a constant minimal number M out of the available N nodes to be activated to determine
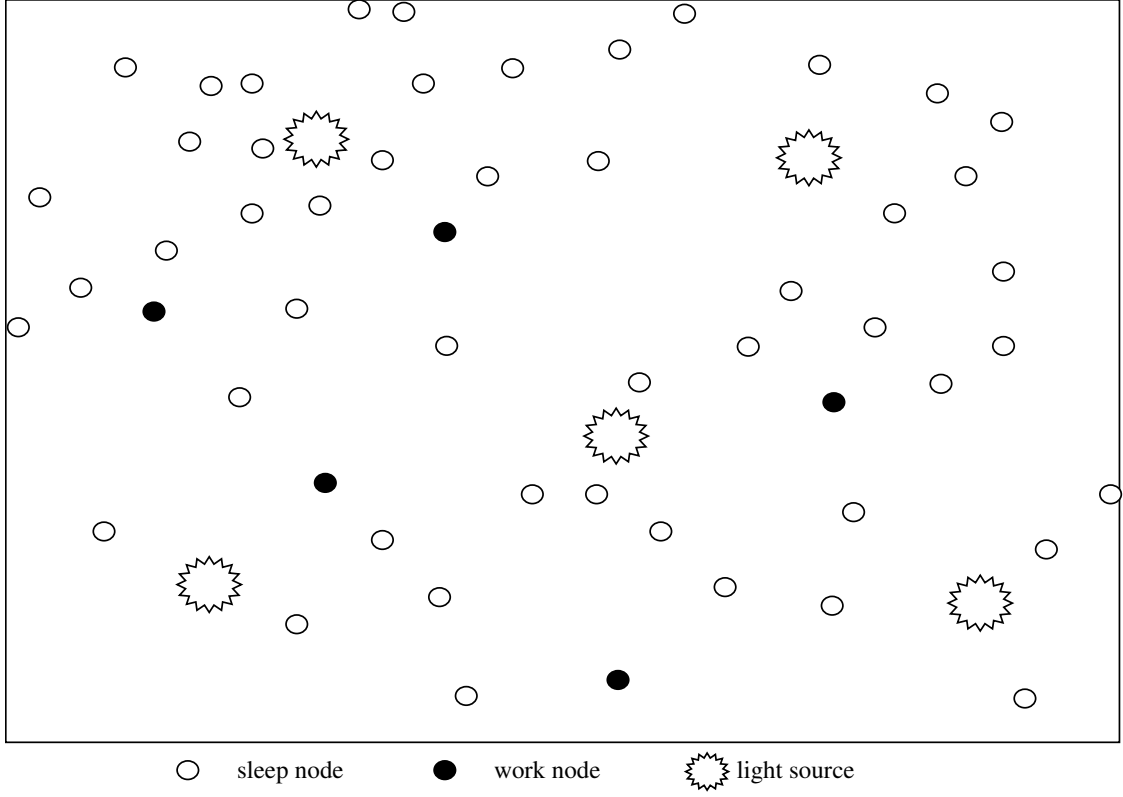
Figure 3.2. A light intensity monitoring WSN.

the power radiated by the M sources. These constant number of active nodes are shown as dark nodes in Figure 3.2. In each round, the DEPM algorithm solves the set of linear inhomogeneous equations, thus providing the accurate reliable values of the *light power P radiated by the M* light sources.

The static mode of DEPM is employed when (a) the power radiated by the sources is first determined by carrying out sensor measurements in the dynamic mode, and (b) when it is known that the power radiated by the light sources is time-invariant. In the static problem, no node at all is required to be activated, except to verify the DEPM prediction, even as the DEPM algorithm solves the inverse problem and estimates with complete accuracy the values of light intensity distribution at an infinite set of point locations in the region R. No energy is consumed in the activation of any node at all, both data acquisition energy and data transmission energy are thus fully conserved while the DEPM Static Al-

gorithm provides light intensity solutions at an infinite number of field points as shown below.

DEPM DYNAMIC Algorithm

In each execution round, three tasks are processed:

1. At the beginning, M out of N nodes are elected to work as active nodes by using a random algorithm which ensures that energy is conserved in all of the N nodes optimally.

2. The *light intensity* I_k ($k \in [1, M]$) sensed by the M active nodes are sent to BS.

3. BS computes the values of *Power P_l radiated by each of the lth* light source using the DEPM dynamic algorithm explained below.

Table 3.1 lists the symbols used by DEPM.

Table 3.1. Symbol Table

Symbol	Meaning
N	Number of sensor nodes.
M	Number of light sources.
P_i	Power radiated by the light source l_i.
$\{s_1, s_2, ..., s_M\}$	Set of active nodes.
I_i	Value of light intensity measured by an active node s_i.
$D(l_i, s_j)$	Distance between light source l_i and location j where the active sensor s_j is located.

If the jth light source alone is switched on, the rest of the $(M - 1)$ light sources being switched off, then the *light intensity* I_k measured by the kth active sensor is related to the power P_j of the jth light source by the well-known *inverse square law*:

$$I_k = \frac{P_j}{4\pi d^2(l_j, s_k)}. \tag{3.1}$$

Now, if all the M light sources are switched on, one requires a *minimum* of M nodes to uniquely determine the light powers of each of the M light sources. DEPM requires only a constant number M of nodes to be activated for this purpose and permits all of the remaining nodes to sleep, thereby conserving their energy. Thus, at the beginning of each round, DEPM elects M nodes as active working nodes. For the sake of balancing the energy consumptions, the active nodes are always selected randomly from the subset of sensors that have residual energy that is higher than the average *energy per sensor* in the entire network. Then, the selected active nodes sense the light intensity I_k of light at each kth node and communicate the recorded values to BS.

Since the *light intensity I_k* at the kth node obeys the principle of superposition with respect to light reaching that node from each of the M number of light sources, we have:

$$I_k = \frac{P_1}{4\pi d^2(l_1, s_k)} + \frac{P_2}{4\pi d^2(l_2, s_k)} + ... + \frac{P_M}{4\pi d^2(l_M, s_k)}, where k = 1 \ldots M \qquad (3.2)$$

$$I_k = \sum_{j=1}^{M} \frac{P_j}{4\pi d^2(l_j, s_k)} = \sum_{j=1}^{M} a_{jk} \; P_j, \; a_{jk} = \frac{1}{4\pi d^2(l_j, s_k)} \qquad (3.3)$$

Equation 3.3 represents a family of linear algebraic equations in which the coefficients a_{jk} are known from the geometrical arrangements of the WSN and the locations of the M light sources. The inhomogeneous linear equations can be solved using well-known techniques in [50]. These equations can be written in a matrix form:

$$\alpha \; \pi = \iota,$$

$$where \; \alpha = \begin{bmatrix} a_{1,2} & a_{1,2} & ..... & a_{1,l} & ..... & a_{1,M} \\ ..... & ..... & ..... & ..... & ..... & ..... \\ a_{l,1} & a_{l,2} & ..... & a_{l,l} & ..... & a_{l,M} \\ ..... & ..... & ..... & ..... & ..... & ..... \\ a_{M,1} & a_{M,2} & ..... & a_{M,l} & ..... & a_{M,M} \end{bmatrix},$$

$$\pi = [P_1, P_2, ..., P_l, ..., P_M]^T, \; \iota = [I_1, I_2, ..., I_l, ..., I_M]^T.$$

We note that the matrix α is determined entirely by geometrical arrangement and is

completely independent of the light sources and the sensor properties.

The solution to this system of equations is: $\pi = \alpha^{-1}\iota$, where α^{-1} is the inverse of the matrix α. A condition is easily incorporated in the algorithm that when the determinant of the matrix α, $\left|\alpha\right|$, is small, a different set of active nodes must be chosen such that $\left|\alpha\right| \succ \delta$, where δ is a user-defined small quantity.

Thus, from the values of the light intensities sensed by the set of M active nodes, the DEPM dynamic algorithm determines the power radiated by each of the M light sources. So far, we can use Equation 3.3 to calculate the light intensity of each sleeping node easily.

3.2.2 Data Estimation using Statistical Model (DESM)

Another estimation mode, DESM, is introduced in this section. A WSN is logically divided into cells using a grid (as shown in Figure 3.3) of which each cell is called an *observing region*. In each observing region and in each round, only the node that has the maximal remained energy is chosen to be the active working node.

Usually, the collected data has strong temporal and spatial locality property. The temporal locality means that values sensed by the same sensor over a continuous time domain have strong relationships. Spatial locality means that values sensed by the sensors whose locations are nearby often are also similar. For instance, HortiSpec mentioned above shows the strong temporal and spatial locality. Aware of these two locality properties, DESM studies the temporal relationship of a sleeping node s from its historical data set and the spatial relationship between s and the current working node that is in the same grid with s, and gives an estimation value for s.

DESM needs a real historical sensing data set of each node to start. In the first several rounds, all the nodes are in working mode. The sensed values are collected and stored at BS as a historical data set. Thereafter, only one node in each cell is active and the other nodes in the same cell go to sleep. We use the following method to estimate the sensing values of the sleeping nodes.

Formally, let $S_R = \{S_1, S_2, \ldots, S_n\}$ be the nodes in an observing region R. For each
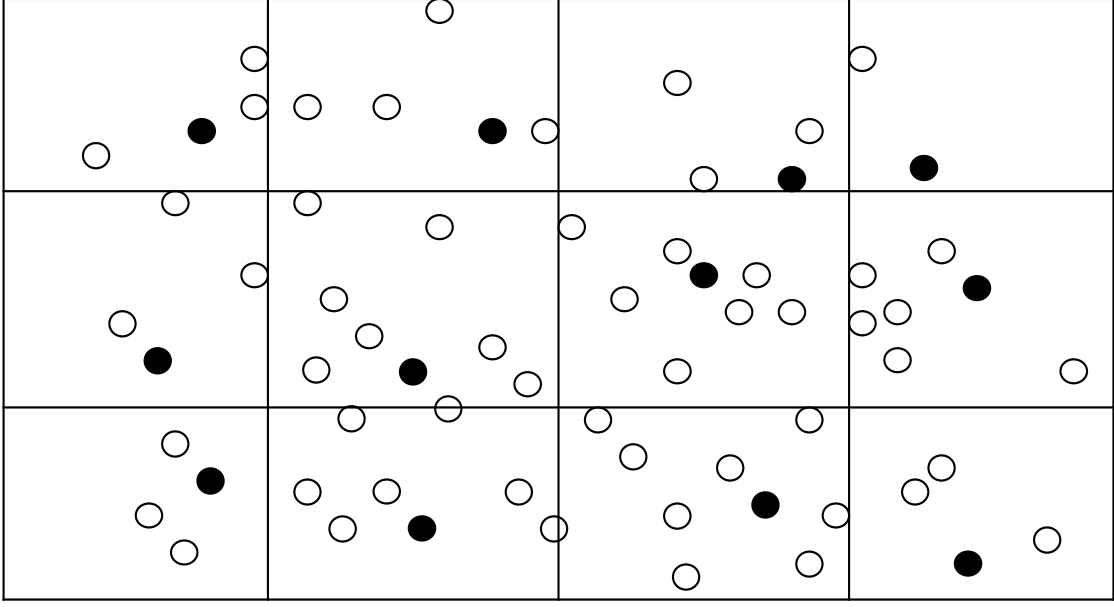
Figure 3.3. A Grid.

node $S_i \in S_R$, $X_{i1}, X_{i2}, ..., X_{im}$ represent its corresponding historical value sequence. The historical value sequences of the node set S_R are stored in a matrix A_R. Suppose that S_i is the active node in S_R. Each time S_i observes a new value $X_{i(m+1)}$, the values of the other $n-1$ nodes can be estimated according to $X_{i(m+1)}$ and A_R.

$$A_R = \begin{bmatrix} X_{11} & X_{12} & ... & X_{1m} \\ X_{21} & X_{22} & ... & X_{2m} \\ ... & ... & ... & ... \\ X_{n1} & X_{n2} & ... & X_{nm} \end{bmatrix}$$

Being aware of the locality of nodes, we can use the following equation to estimate the value $\hat{X}_{j(m+1)}$ of S_j.

$$\hat{X}_{j(m+1)} = (1 - \alpha)\hat{Y} + (\alpha)\hat{Z} \tag{3.4}$$

where $\hat{Y}$ is the last estimated value $\hat{X}_{j(m)}$, which measures the influence of the historical sensing data on the current value at node S_j. $\hat{Z}$ is computed by $X_{i(m+1)}$, which measures the influence of the data sensed by the active node S_i on the data of node S_j. α is a weight

parameter that evaluates the effects of $\hat{Y}$ and $\hat{Z}$ on the estimated value $\hat{X}_{j(m+1)}$, whose value is in $[0, 1]$.

We use the following method to compute $\hat{Z}$. Given two random variables $X_i = \{X_{i1}, X_{i2}, \ldots, X_{im}\}$ and $X_j = \{X_{j1}, X_{j2}, \ldots, X_{jm}\}$ with $X_{i(m+1)}$, the estimated value $\hat{Z}$ for $X_{j(m+1)}$ is computed as follows:

$$\hat{Z} = X_{j(m)}(1 + \frac{X_{i(m+1)} - X_{i(m)}}{X_{i(m)}}) \tag{3.5}$$

Equation 3.5 is based on an assumption that X_i and X_j have the similar data fluctuation trend, which is usually true for the two nodes in the same cell according to the space locality of a WSN.

Intuitively, if X_i is more related with X_j, then $\hat{Z}$ is expected to impact more influence on the value of $\hat{X}_{j(m+1)}$, that is, α should be a larger value. According to the above analysis, correlation coefficient $\varphi(X_i, X_j)$ is an exact way to define α.

Given two random variables X_i and X_j, whose first m values are $X_{i1}, \ldots, X_{im}$ and $X_{j1}, \ldots, X_{jm}$ respectively. We can use the following formula to compute the correlation coefficient between X_i and X_j:

$$\varphi(X_i, X_j) = \frac{Cov(X_i, X_j)}{\sigma_{X_i}\sigma_{X_j}} \tag{3.6}$$

In Equation 3, $Cov(X_i, X_j)$ is covariance between X_i and X_j. $Cov(X_i, X_j) = E[(X_i - EX_i)(X_j - EX_j)] = E(X_iX_j) - E(X_i)E(X_j)$.

3.2.3 Discussion

The Dynamic DEPM technique offers by itself as a very powerful algorithm to solve the inverse problem *when it is known that the power radiated by each of the light sources is not changing*, as is often the case, and it is nevertheless of importance to know the distribution of intensity of light at a large number of point-locations in the region illuminated by the M light sources. Such problems are of interest in very many different areas, such as in

the determination of light intensity distribution in a sports arena where games are played under artificial light sources of constant power, and in horticulture experiments [7] in which cultivation of some vegetation is sought under artificial light from a set of constant-power sources. Likewise, cell-culture experiments [9] under controlled light intensity environment is another example of a situation where the Static DEPM model introduced below can be extremely valuable.

To solve such problems, the DEPM *inverse* algorithm is to be used, treating the matrix π as known and determining the intensity matrix ι for any set of M point-locations whose coordinates alone determine the corresponding matrix α. Since in this model the matrix π is considered to be known (*i.e.* pre-determined), the inverse algorithm is referred to as STATIC DEPM. Thus for a predetermined matrix π, and for a set of arbitrary M number of locations whose coordinates alone determine the required matrix, the STATIC DEPM algorithm solves the matrix relation: $\alpha\pi = \iota$, thereby giving the values of the light intensities at a set of M arbitrary locations whose coordinates alone need to be provided as input for the STATIC DEPM algorithm. Not a single node needs to be activated at this set of M locations, thus providing reliable data estimation that substitutes data acquisition conserving energy of the WSN. Of course, some of the sensors can be activated to verify the prediction of the static DEPM algorithm, and this is a tremendous advantage that lends itself as a tool to check reliability of the model.

Light intensity distribution at arbitrary locations (x_k, y_k, z_k) can be achieved by data estimation using a method completely based on physical laws instead of using actual nodes being activated at these locations. The physical laws that are employed are very rigorous. Hence, as long as the physical conditions of the algorithm are satisfied, the DEPM is expected to provide very accurate predictions. To the best of our knowledge, most of the energy saving techniques employed in WSNs employ various combinations of spatial and temporal coherence and/or data filters, whereas the present method employs well-established laws of physics as direct energy conservation strategy and thus provides a very novel approach to develop energy saving mechanisms in WSNs based on physical laws. Based on the temporal

and spatial correlation among adjacent nodes, DESM method can estimate the intensity of light, temperature, humidity and *etc.* We merely need to know the position information of sensor nodes, and do not require any information about the monitored objects.

How to set up a grid is a primary factor in the accuracy of estimation. In order to conserve energy as much as possible, it is desirable that the number of nodes in sleep mode is maximized. However, the accuracy of DESM would decrease in such a case. There is thus a trade-off between the estimation accuracy and energy conservation. In some applications, using clusters to separate the observation regions is more preferred. For instance, in order to monitor the temperature of a big building, in each room must be placed several sensors. Consequently, a room can be considered as an observing region, and it is no doubt that the trend of temperature fluctuation in a room is rather similar.

3.3 Experiment Results

In the prototype experiments that were performed, twenty TelosB [10] sensors were deployed in a laboratory of size 25'x50' as shown in Figure 3.4. The laboratory was insulated from natural light so that the sensors would sense and record only light intensities from two artificial electrical light sources L_1 and L_2. The twenty sensors and the two light sources were geometrically distributed randomly in the room, such that there is no obstacle between sensors and the light sources.

Each sensor sensed 10 values of light intensity every 2 seconds, and sent the average value to BS. The data that were received by BS were stored as the tested data set. Three different experimental sets were employed to test the methodology developed. In the first set, intensity data recorded by the sensors were analyzed with both the light sources turned on. In the second data set, intensity data were collected from the sensors and the experiment was performed while the two light sources were turned on and off every 5 minutes. In the third set, data were obtained with the two light sources turned on and off randomly and frequently. Detailed experiments settings can be found at [51].

We have implemented our two methods on TelosB motes and have conducted extensive

Figure 3.4. Experimental environment.
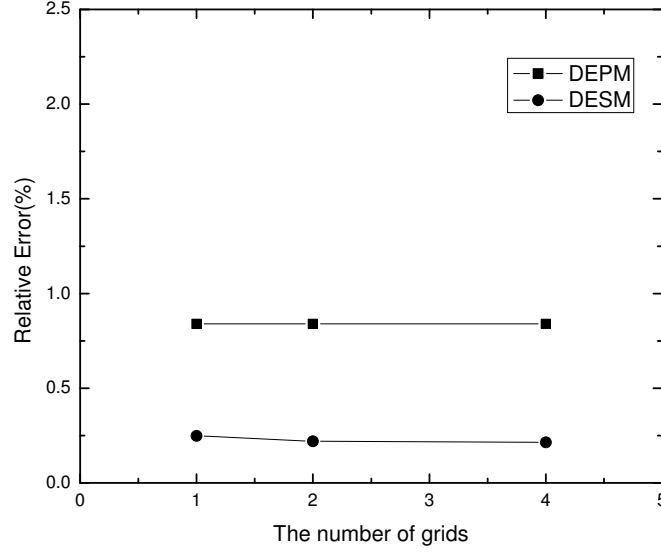
Figure 3.5. The relative error of estimations reported by DEPM and DESM using data set 1.

experiments. Our experiments are designed with two objectives in mind. First, we verify the proposed methods, DEPM and DESM, are able to report highly accurate answers. Second, we assess the two methods DEPM and DESM for their energy consumption.

To evaluate the two algorithms we proposed, we need the real data set to validate the accuracy of DESM and DEPM. We use 20 TelosB sensors to sense and collect the intensity of light as the testing data set. To prolong the lifetime of sensor network is our motivation to propose these two algorithms, so we also test and analysis the energy consumption of each algorithm.

3.3.1 Estimation accuracy

In this section, we report the results of DEPM and DESM and assess the accuracy of the data estimates reported by the proposed methods. The average relative error is used as the metric to measure the accuracy. We conducted the experiments with the three data sets that mentioned above.
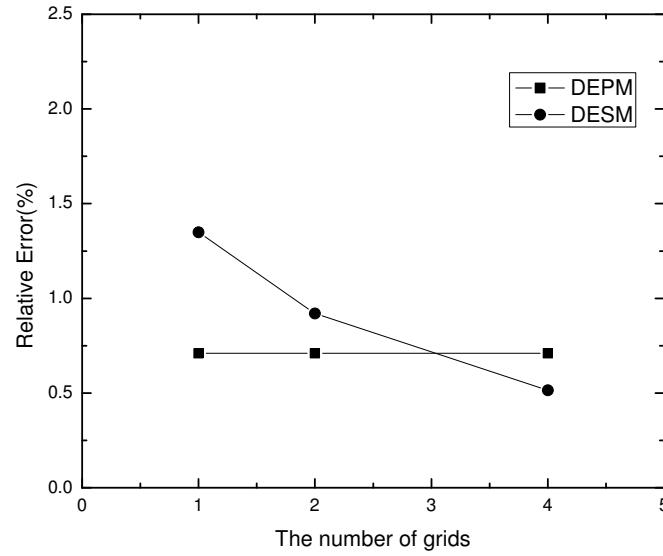
Figure 3.6. The relative error of estimations reported by DEPM and DESM using data set 2.
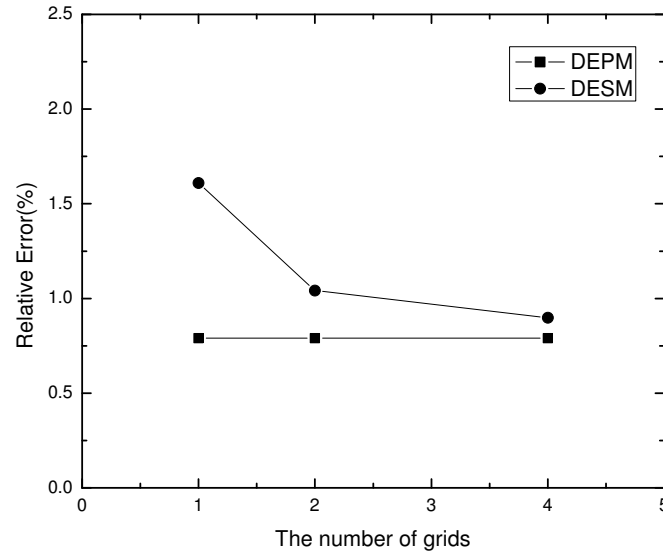
Figure 3.7. The relative error of estimations reported by DEPM and DESM using data set 3.

As is shown in Figures 4.4, 4.5, and 4.6, with the number of grids increases, the relative error of answers reported by DESM goes down. This is because the more the number of grids is, the less the number of sensors in each grid, while the stronger the space locality of sensors in one grid has. We could also see that DESM is more accurate for data set 1 but gets worse for the data sets 2 and 3. The reason is that the light intensity values of data set 1 are stable while vary in the data set 2 and 3.

The accuracy of DEPM method has no relationship with the number of grids as shown in Figures 4.4, 4.5 and 4.6. DEPM outperforms DESM on data set 2 and 3, but performs worse on data set 1. Furthermore, obviously DEPM is not affected by different data sets. It can keep good and stable quality on all three data sets.

3.3.2   Energy consumption

Table 3.2. System Parameters and Setting.

Parameter	Setting
Number of sensor nodes	20
Message size	8 bytes
Transmission distance	50m
Energy Cost for Sending a Message	19.2uJ
Energy Cost for Receiving a Message	3.2uJ
Energy Cost for Sensing a Light Intensity	100nJ
Energy Cost in Sleeping Mode	0.016mW
Initial Energy Budget at Each Sensor Node	1J

In this section, we assess the energy consumption. We use the energy model in [52] as presented in Table 3.2. For the DESM and DEPM methods, one execution round is set to 10 minutes. For DESM, all the sensors collect data in the first two rounds of every two hours considered as the history data. After the history data collecting phase, in each round one node with the most remained energy works as an active node in every grid, and others go to sleep in the rest time. For DEPM, in each round two nodes with the most remained energy work as active nodes, and others go to sleep.

Table 3.3. Model Selection Guideline.

Which model should be used?	DESM	DEPM (large δ)	DEPM (small δ)
Lowest total energy consumption	No	Yes	Yes
Accuracy	Yes	Yes	No
Accuracy and balanced energy consumptions	Yes	No	No
Sensing variable light intensity sources	No	Yes	Yes
Accuracy at the cost of balanced energy consumptions	N/A	Yes	No
Balanced energy consumptions at the cost of accuracy	N/A	No	Yes

We compare the proposed models with a naive model where all the sensors are set active. As shown in Figure 4.8, the average energy consumption of both DEPM and DESM is much less than that of the naive model. After running about 10 rounds, the accumulative average energy consumption of DEPM is about 50% of that of the naive model. So DESM and DEPM can prolong network lifetime effectively. DEPM is more effective than DESM, because DESM has the collecting history data phase, consuming more energy. DESM-1 (DESM-2, DESM-4) represents the DESM method with one grid (2 grids, 4 grids correspondingly) in the WSN. From Figure 4.8, we can see that the more grids for DESM, the more energy consumption. The reason is that each grid has an active sensor in DESM. The more number of grids means the more sensors are set to active in the same time, hence the amount of energy consumed is more.

We show the remained energy of DEPM with different constraints in Figure 3.9. It is defined as the coefficient of variation of all the nodes' remained energy. The average energy consumption of all the nodes is almost half of initial energy. When we use larger δ, the estimation accuracy is better, but it makes some nodes impossible to be active, so the balance of energy consumptions is worse.

Neither DESM nor DEPM is a perfect method in all situations. Table 3.3 provides a guide for choosing the proper estimation model in different situations. For instance, if users prefer high accuracy, it is better to choose DESM or DEPM with large δ. If users expect long network lifetime, DEPM is a good choice.
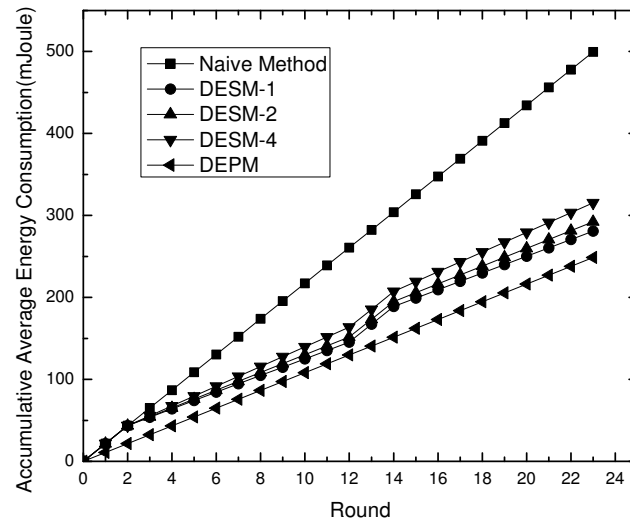
Figure 3.8. The energy consumptions of different models.
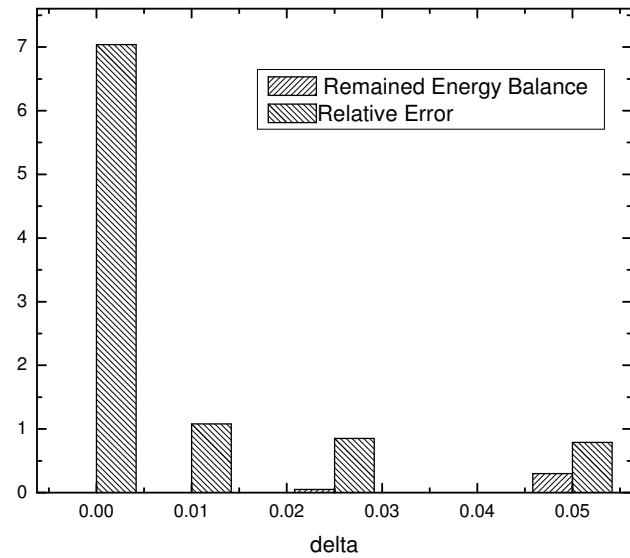
Figure 3.9. The remained energy of DEPM.

Chapter 4

IN-NETWORK HISTORICAL DATA STORAGE AND QUERY PROCESSING

In this chapter, we propose a scheme, HDQP, to store historical data locally on sensor nodes and process historical data queries efficiently by using distributed index.

4.1 Historical Data Storage

Since a sensor node's memory capacity is limited, we have to fully utilize it to store as many data as possible. The administrator of the sensor network can specify the attributes which need to be stored locally according to users' requirements. Suppose there are n attributes $(A_1, A_2, \cdots, A_n)$ need to be saved, and the sensor node senses the values $V_1, V_2, \cdots, V_n$ of $A_1, A_2, \cdots, A_n$ respectively at sensing intervals. For each sensing interval, a record with the format $(T, V_1, V_2, \cdots, V_n)$ (where T is a time-stamp) is written to the flash memory. However, if the sensing interval is very short such as 5 seconds, the flash memory will be full filled quickly. Then the rest of coming sensing data cannot be stored. To avoid this happen, whenever the flash memory is occupied more than 80%, a *weight-reducing* process is trigged. One record of every two consecutive records is erased. This weight-reducing process can make at least half of memory space available again. Applying the weight-reducing process repeatedly causes partial historical data lost. However, as we mentioned in Chapter 1, even though the flash memory capacity is small, it still can store 5 to 10 values per hour for each attribute. Consequently, the historical data stored in the flash memory can reflect the changing trend of each sensing attribute.

Each sensor node needs to calculate the maximum, minimum, and average for each attribute periodically. The administrator specifies an *update interval* which is much greater than the sensing interval. For instance, if the sensing interval is 5 minutes, the update
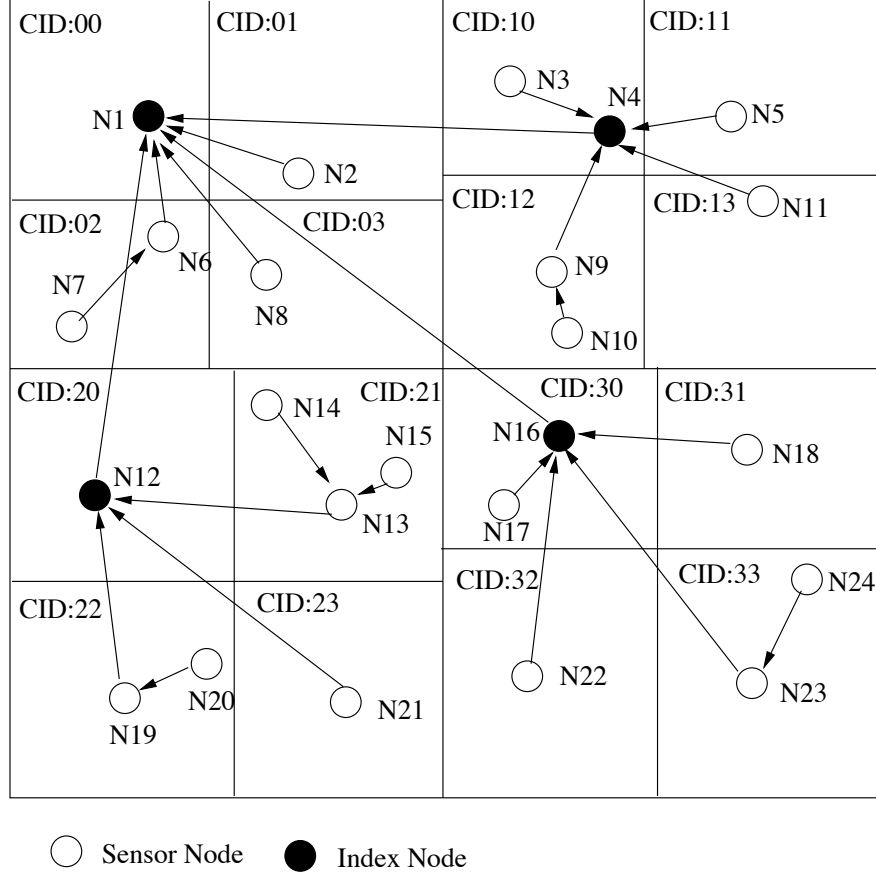
Figure 4.1. Network Partition

interval can be 2 hours. At the end of each update interval, a sensor node sends the *index update message* including the maximum, minimum, and average values of that interval to its parent node of the index tree (we will discuss it in the next section).

4.2 Construct and Maintain Distributed Index Tree

Constructing effective distributed index structures can help process queries efficiently. In this section, we introduce how to construct and maintain distributed index trees.

4.2.1 Construct a Hierarchical Index Tree

We assume that any point in the monitored area is covered by at least one sensor node. Moreover, the sensor network is static, and absolute or relative locations (2-D cartesian co-
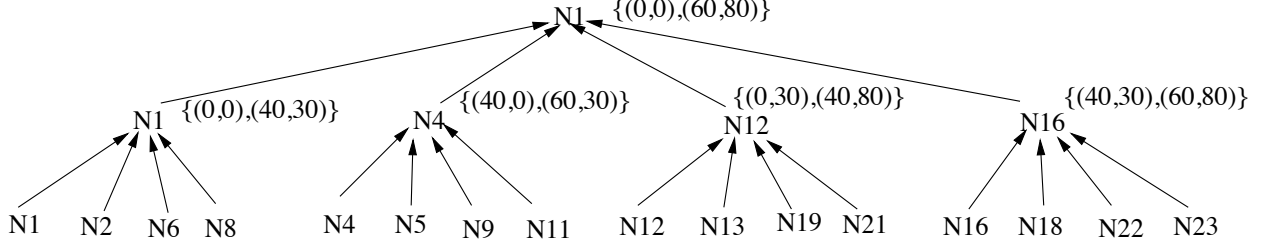
Figure 4.2. Index Tree

ordinates) of sensor nodes are known via using GPS or location algorithms. This is necessary since users use the location as query conditions or results sometimes. At BS, the entire network is divided into four subregions in vertical and horizontal directions, and each subregion has almost the same number of sensor nodes. Then, for each generated subregion, it is continuously divided into four subregions with almost the same number of sensor nodes. This partition process keeps going until there is no enough sensor nodes to promise that every generated subregion has at least one sensor node. A partition example is shown in Figure 4.1. The entire network is divided into a lot of rectangles called cells in this paper. Each cell has at leat one sensor node. In each iteration of the partition, a generated subregion is assigned an ID from 0,1,2, and 3 according to its relative position, top-left, top-right, bottom-left, and bottom-right, respectively. Finally, the cell id (CID) is the combination of IDs generated by all partition steps (as shown in Figure 4.1). For example, the node N18's CID is 31 since in the first partition iteration, the ID is 3 (bottom-right), and in the second partition iteration, the ID is 1 (top-right). After partition, the location and CID of each cell are recorded and sent to the sensor nodes within that cell. We use 2-D coordinates of the cell's top-left and bottom-right vertices, $\{(x_{tl}, y_{tl}), (x_{br}, y_{br})\}$, to describe the cell's geographical position.

For processing queries efficiently, a hierarchical index tree is constructed. An index tree example is shown in Figure 4.1 and 4.2. Firstly, each cell randomly chooses a sensor node as the leader of this cell. Other nodes in that cell set the leader as their parent. Since only the leader has a chance to be the index node, to maintain energy balance, sensor nodes in the same cell serve as the leader in turn. In the example, node $N13$, $N14$, and $N15$ are in the same cell, and node $N13$ is chosen to be the leader of $N14$ and $N15$. If a cell only has

one sensor node, the only node is chosen to be the leader naturally. Then, the index tree is established in a reverse manner of network partition. In other words, the index tree is constructed from bottom to top. For each group of four cells which are generated from the same subregion, the leader of the top-left cell is chosen to be the parent of these four cells' leaders including itself. Then, for each group of four generated parent nodes which belong to the same subregion in the partition process, the top-left one is chosen to be the parent. The process continues until the root is generated. In the example shown in Figure 4.1, Node $N1$ is chosen to be the parent of Node $N1$, $N2$, $N6$, and $N8$. Node $N4$, $N12$, and $N16$ are chosen to be the parents of other three groups respectively. Next step, Node $N1$ is chosen to be the parent of $N1$, $N4$, $N12$, and $N16$, and the root $N1$ is generated. The structure of the index tree is shown in Figure 4.2. Obviously, the index tree is a quad-tree, and all inter nodes have four children.

If we use the same index tree during the entire network lifetime of networks, the accumulative index data will occupy most memory space quickly, especially the top layer nodes. Furthermore, index nodes will use up energy much earlier than others because the query processing frequently accesses these nodes. To balance index data distribution and energy consumption among sensor nodes, a set of index trees is generated by switching nodes' positions in the index tree periodically.

The administrator needs to specify a *tree switching period* P_S. The ideal value of P_S is the estimated network lifetime divided by the number of sensor nodes. Thus, every node has a chance to sever as an index node. Four children of the root serve as the root node in turn. For the root node (first layer), the switching period is P_S. Each node in the second layer also makes its four children serve as the parent in turn. However, the switching period of nodes in the second layer is $4 * P_S$. The switching period for nodes in the layer L is $4^{L-1} * P_S$. The lower the layer, the longer the switching period. In fact, in our index tree structure, a higher layer index node contains more data than a lower layer index node since an internal node appears at each layer of the subtree rooted at itself. For example, the root node $N1$ in Figure 4.2 appears in every layer of the index tree. Our schedule replaces the

higher layer nodes more frequently than lower layer nodes, thus evenly distributing index data to all sensor nodes.

Through switching nodes in the index tree, a large number of index trees are available. However, a sensor node does not need to store all these index trees. Since we follow the rules to switch nodes, a sensor node can calculate its current parent according to the current time-stamp T. For a node in the (L)th layer of an index tree, its parent's CID at time T can be calculated by replacing its own CID's $(L-1)$th bit with $(\lfloor T/(4^{L-1} * P_S) \rfloor \%4)$. Geographic routing is used to generate routes among nodes of the index tree since the CID of a sensor node also indicates its relative position.

4.2.2 Index Maintaining

Table 4.1. Index Structure

Child0:		Child1:		Child2:	Child3:
Location:	$\{(x_{tl}, y_{tl}), (x_{br}, y_{br})\}$	Location:	$\{(x_{tl}, y_{tl}), (x_{br}, y_{br})\}$	. . .	. . .
Attribute1:	T_i, max, min, avg	Attribute1:	T_i, max, min, avg		
	T_{i+1}, max, min, avg		T_{i+1}, max, min, avg		
	$\vdots$		$\vdots$		
Attribute2:	T_i, max, min, avg	Attribute2:	T_i, max, min, avg		
	T_{i+1}, max, min, avg		T_{i+1}, max, min, avg		
	$\vdots$		$\vdots$		
Attribute3:	T_i, max, min, avg	Attribute3:	T_i, max, min, avg		
	T_{i+1}, max, min, avg		T_{i+1}, max, min, avg		
	$\vdots$		$\vdots$		
$\vdots$		$\vdots$			

The leader of each cell is responsible for calculating the maximum, minimum, and average values of each attribute for the cell it belongs to periodically. If there are more than one sensor node in a cell, the maximum, minimum, and average values are calculated among all sensing values of the current interval from these sensor nodes. At the end of each update interval, the leader sends the update message, (T, max, min, avg) (where T is the beginning time stamp of the current time interval), to its parent in the index tree. The update message

also includes the time stamp and node ID of the max and min values. The internal node in the index tree maintains the received index data with the structure as shown in Table 4.1. When an update message is received, it is inserted into the index structure. The internal node also merges four update messages for the current time interval T by calculating max, min, and avg, and sends this update message to its parent. For instance, at time T, a node receives four update messages (with the format (max, min, avg)), (78, 56, 68), (70, 56, 65), (75, 54, 65), and (72, 55, 64) from its children respectively. The merge result for the attribute temperature at time T of N_i is (78, 54, 65.5).

4.3 Historical Data Query Processing

Historical data queries can inquire max, min, and avg values for a past period of time in a specified geographic area. Also, the sensing values of specific sensor nodes or locations in the past can be retrieved. A historical query can be issued at BS or a sensor node in the network which is named as *query node*.

Query Example1:

SELECT S.temperature, S.humidity, S.location

FROM sensor S

WHERE S.location

WITHIN {(10, 10), (30, 30)} AND S.time

BETWEEN now()-1 days AND now()

Query Example2:

SELECT MAX(S.temperature), S.location

FROM sensor S

WHERE S.location

WITHIN {(0, 0), (50, 30)} AND S.time

BETWEEN 02/01/2009 and /02/04/2009

If the query node receives a query like Query Example1, the query is forwarded to the query location {(10, 10), (30, 30)}. When a sensor node within {(10, 10), (30, 30)} receives

this query, it forwards this query to the closest ancestor node (node $N1$ at the second layer in Figure 4.1) of the index tree, whose location contains the query location. Then, $N1$ accesses its indexes and sends the query to all its children which have intersection with the query location. Thus, the query is sent to sensor nodes which possibly satisfy the query conditions through the index tree. When a sensor node receives a query, it verifies itself. If the conditions are satisfied, it sends corresponding results back to the query node. In this query example, the index tree can bound the number of involved sensor nodes efficiently. Only cells which have intersection with location $\{(10, 10), (30, 30)\}$ are probed.

For Query Example2, the query node forwards the query to the lowest layer index node (the root $N1$ in Figure 4.1), which contains the query location. Then, $N1$ probes MAX(S.temperature) within $\{(0, 0), (40, 30)\}$ from $N1$ and MAX(S.temperautre) within $\{(40, 0), (50, 30)\}$ from $N4$ between 02/01/2009 and /02/04/2009 separately. For location $\{(0, 0), (40, 30)\}$, we can calculate the index nodes, which store the MAX value between 02/01/2009 and /02/04/2009, and merge results from these index nodes to get MAX(S.temperautre) within $\{(40, 0), (50, 30)\}$. However, since $\{(40, 0), (50, 30)\}$ does not perfectly match $N4$'s location $\{(40, 0), (60, 30)\}$, $N4$ has to further probe some of its children, then merges results returned by its children to get MAX(S.temperautre). Finally, the root $N1$ merges results returned by $N1$ and $N4$ to get the MAX(S.temperature) within $\{(0, 0), (50, 30)\}$ and sends the results back to the query node. Usually, users send queries like Query Example2 to inquire statistical information from wireless sensor networks.

The distributed index tree can guide queries to be processed efficiently and restrict the number of involved sensor nodes. For some queries, the results are already stored in the index structure, so accessing one or several index nodes instead of a large number of sensor nodes can acquire the results. The query processing method is shown in Algorithm 4.3 and 4.3. Algorithm 4.3 illustrates how a query node sends requests to corresponding sensor nodes and merges results returned by sensor nodes. Algorithm 4.3 shows how a sensor node processes a query request. For queries with aggregation functions, if a sensor node in the index tree maintained results already, we do not need to probe its children. If not, usually

---

Algorithm 1 : A query node processes queries.

Input: A query Q.

Output: Requests to index nodes or sensor nodes.

1: **if** There is no aggregation function in the query Q **then**
2:    Send *Type1* query requests, including the query conditions to all sensor nodes within the queried area;
3: **else**
4:    Find the smallest subtree T which can totally covers the queried area;
5:    Calculate the root nodes of the subtree T which served as the roots within the queried time span. Insert these root nodes into set R;
6:    Send *Type2* query requests with query conditions to nodes in R;
7: **end if**
8: Wait for results from queried sensor nodes;
9: **if** Query type is *Type1* **then**
10:    return results to users directly;
11: **else**
12:    Merge results according to query aggregation functions and return final results to users.
13: **end if**

---

this is because the queried area and time span do not match corresponding information of the index node, we need to send query requests to its children.

4.4 Multi Query Optimization

For most applications, users concern raw data or statistical data of some hot areas and/or some specific time slices. Therefore, there is a big probability several queries acquire results based on similar data. In the literature [53], [54], [55], and [56], authors designed query optimization strategies to process multi queries energy-efficiently by sharing common data requests among multi queries. This strategy also works for historical queries. However, if there are not a lot of query requests simultaneously, this strategy does not work well. For improving the performance of our scheme, we propose an optimization method which caches results at BS to avoid accessing the same data repeatedly, thus conversing unnecessary energy consumption.

On BS, we construct an index tree as shown in Figure 4.3 which has similar architecture

---

Algorithm 2 : A sensor node S processes a query request.

Input: A query request.
Output: Query results.

1: **if** Query request is *Type1* **then**
2: Search data which satisfy the conditions in the memory or flash memory and send results to the query node.
3: return;
4: **end if**
5: **if** Query request is *Type2* **then**
6: **if** S is a leaf node in the tree **then**
7:   Look for results from the index and return results to the sensor node which sent the request. If results are not maintained by the index, then calculate the results based on the raw data;
8: **else**
9:   **for** each child C of S which has overlap with the queried area **do**
10:    **if** Results are maintained by the index **then**
11:     Return results to the sensor node which sent the request;
12:     return;
13:    **else**
14:     Send a *Type2* query request with query conditions to the child C;
15:    **end if**
16:   **end for**
17:   Wait for results from queried sensor nodes;
18:   Merge results according to query aggregation functions and return merged results to the sensor node which sent the request;
19: **end if**
20: **end if**

---

with the distributed index tree. However, in the index tree of BS, only leaf nodes represent the sensor nodes. Each non-leaf node uses coordinates of top-left and bottom-right vertices to represent the area it covers. For saving unnecessary energy cost of accessing same data repeated from sensor nodes, we insert results of queries into corresponding nodes of the index tree on BS. When a new query is submitted, we will search results from the index tree first. The optimization algorithm is shown in Algorithm 4.4. If the index tree does not include any useful data, we will process the query normally. If the index tree has complete results, we don't need to send query requests to the sensor network. The complicated case is that the index tree just maintains partial results, then we need to rewrite the query to request data which the index tree cannot provide. Then merge the partial results from the index

Algorithm 3 : Multi Query Optimization.

Input: A query request q and the index tree T of BS.
Output: Query results.
1: **if** T contains the results of q **then**
2:     Return results to users;
3:     return;
4: **else**
5:     **if** T contains partial results of q **then**
6:         Rewrite WITHIN and BETWEEN subclauses of the query by subtracting the part which already in the index tree T.
7:         Call Algorithm 4.3 to process the rewroten query.
8:         Merge partial results from the index tree T and in-network, and send final results to users.
9:     **else**
10:         Call Algorithm 4.3 to process the original query q.
11:     **end if**
12: **end if**
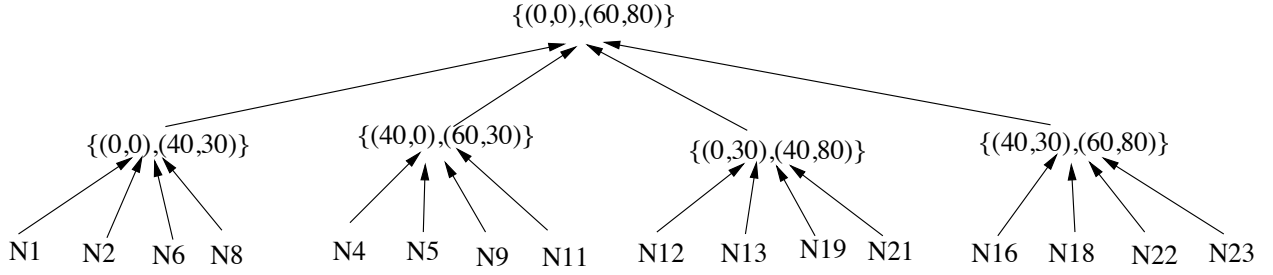
Figure 4.3. Index tree on BS.

tree and the network to answer the query. When we rewrite the query to reduce the size of queried area or the time span, the energy cost will be reduced.

4.5   Simulation

In this section, we evaluate the performance of our proposed HDQP. The sensing interval of a sensor is 5 minutes, and the update interval P_S of the index tree is 2 hours. Since network traffic greatly affects energy efficiency, we use it as the metric for performance evaluation.

The main cost of maintaining the index tree is sending and forwarding update messages along the index tree periodically. Figure 4.4 shows the number of average accumulative messages of sensor nodes with 100, 500, and 1000 sensor nodes in the network respectively.
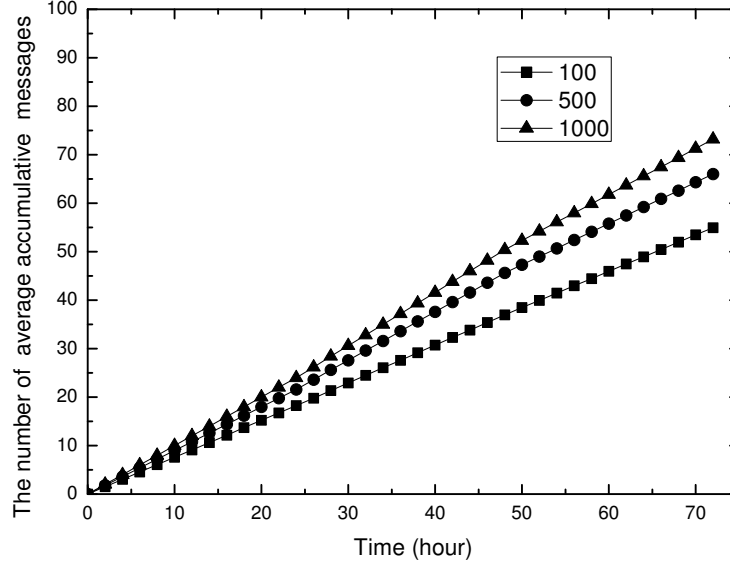
Figure 4.4. Cost of maintaining the index tree with various network size.

The more sensor nodes the network has, the more cost of maintaining the index tree. The reason is that a bigger network must generate a bigger index tree structure, thus increasing the maintaining load. Since for a sensor node, one time index updating just causes about 2 messages averagely, and the index tree is not updated frequently, so the cost of maintaining the index tree is acceptable for wireless sensor networks with limited energy. Furthermore, as can be seen in Figure 4.4, the maintaining cost does not rapidly increase with the increasing of the network size. Therefore, the distributed index structure is also suitable for large-scale wireless sensor networks.

The performance of our HDQP is evaluated on two aspects, delay of query processing and cost of network traffic. The delay of query processing is query responding time since the query is issued until the user receives results. In our simulation, computation delay of sensor nodes is ignored, so this delay is measured by the number of hops of the longest path for sending a query and returning results. Network traffic is defined as the average number of messages sent and forwarded by all sensor nodes. Figure 4.5, 4.6, 4.8, and 4.9 show the
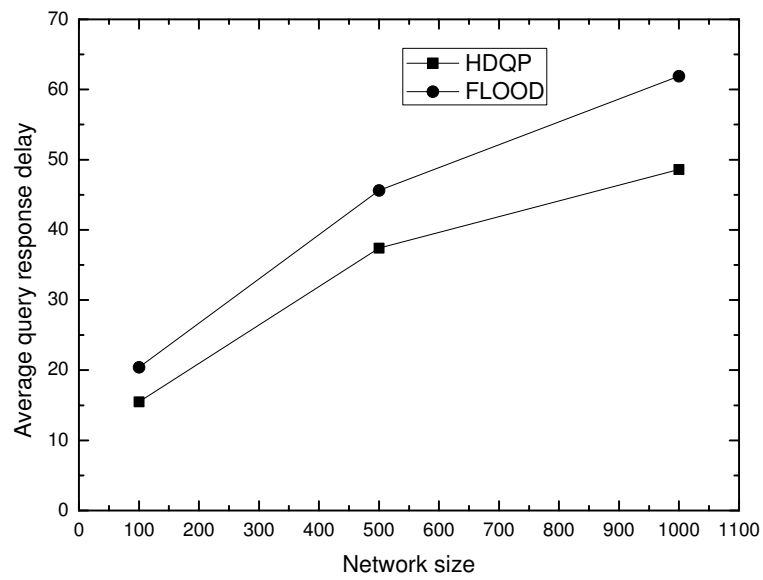
Figure 4.5. Query responding delay of variant network size.

Figure 4.6. Average network traffic of variant network size.

Figure 4.7. Total network traffic of variant network size.

Figure 4.8. Average network traffic of variant average involved nodes percentage of queries.

Figure 4.9. Average network traffic of variant average length of query time window.
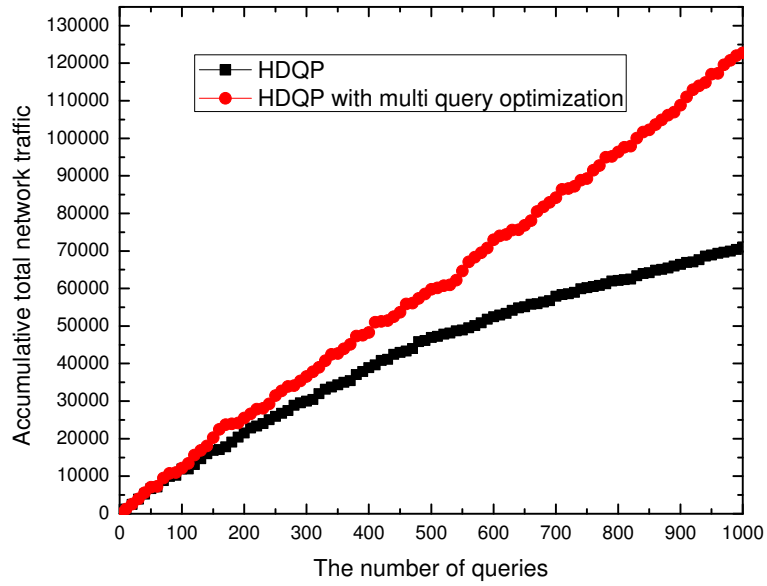
Figure 4.10. Accumulative network traffic of
HDQP with multi query optimization strategy.

average query responding delay and network traffic (including cost of maintaining the index tree) of processing 1000 queries during 72 hours. Figure 4.7 shows the total network traf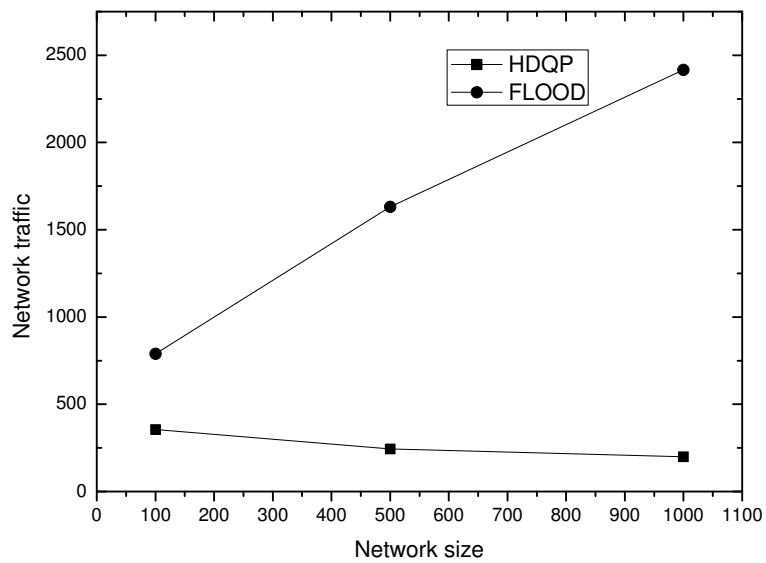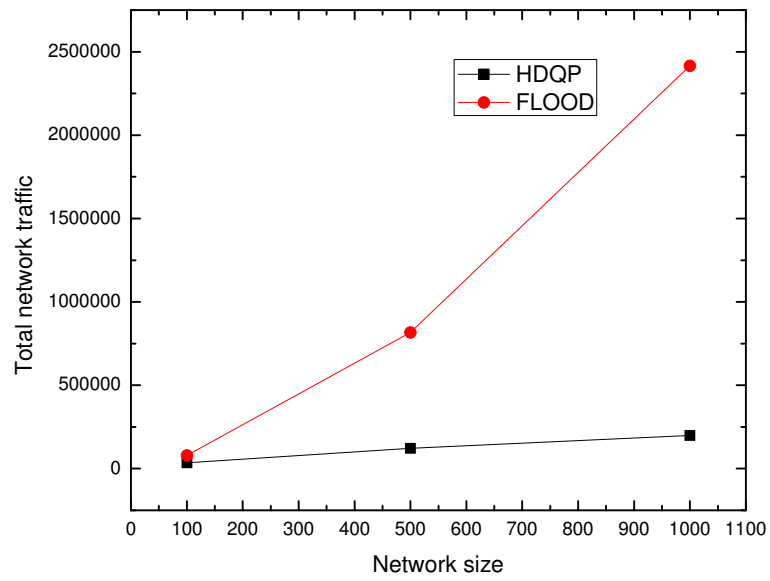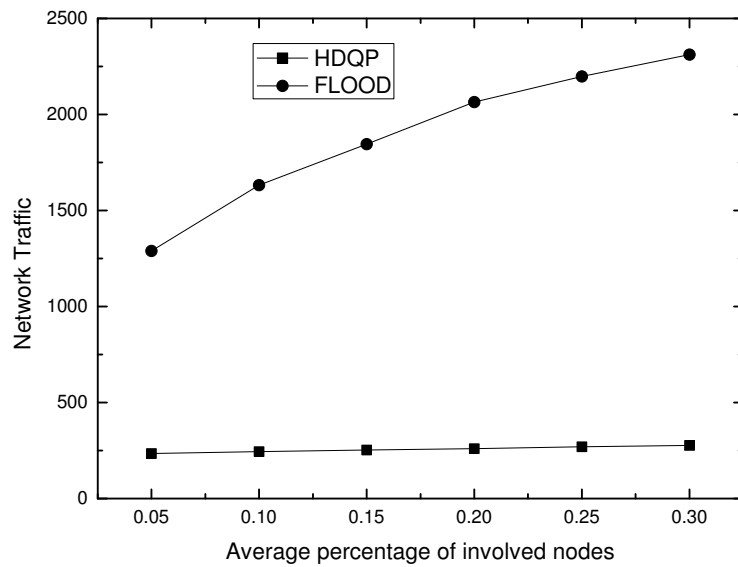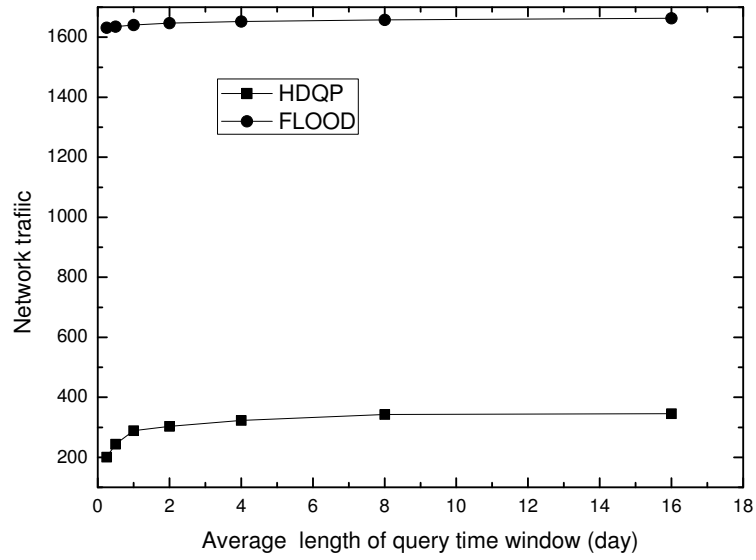fic of processing 1000 queries during 72 hours. We compare our HDQP with Flood method where the query node floods the query to entire network and all sensor nodes satisfied query conditions send corresponding data back to the query node. As seen in Figure 4.5, with the increasing of network size, query responding delay increases too since the length of pathes for sending queries and returning results increases with the network size. HDQP achieves shorter delay than Flood because it does not need to probe all qualified sensor nodes, and partial or all results can be acquired from the index tree. Figure 4.6 and 4.7 show that HDQP is much better than Flood on network traffic especially for large-scale networks. This is because the index tree can help HDQP to answer queries by accessing index nodes and a small number of qualified sensor nodes if necessary. However, Flood method lets all qualified sensor nodes involve in the query processing. Network traffic of HDQP decreases with the network size increasing since for processing the same number of queries, a large-scale network has much more nodes to share work load. Figure 4.8 shows the average network traffic with variant involved sensor nodes percentage of queries. This parameter is decided by the location condition in a query. As shown in the figure, with the increasing of involved node percentage, the network traffic of Flood increases obviously since all involved nodes need to report results. However, HDQP is not affected by this parameter obviously, since HDQP acquire most of data from the index trees. The average network traffic with variant length of query time window is shown in Figure 4.9. With the increasing of time window length, the network traffic of HDQP increases too. Since index data are partitioned by time stamp, more index nodes are accessed for processing queries with a longer time window. When the length of time window reaches a certain value, the network traffic does not evidently increase any more. The reason is that when the time window is long than 4 times of partition period of the accessed layer, the number of accessed index nodes does not increase any more. However, an index node may include index data of several time slices.

For evaluate the multi query optimization strategy we proposed, we execute 1000 queries

using HDQP without multi query optimization and HDQP with multi query optimization respectively. As shown in Figure 4.10, the accumulative total network traffic of HDQP with multi query optimization is less than HDQP without multi query optimization since query optimization avoids accessing the same data shared by multi queries repeatedly. With the number of queries increasing, the more energy multi query optimization can save since more results already are saved by the index tree.

In summary, our HDQP scheme can process historical data query quickly and energy-efficiently, and it is also suitable for large-scale networks.

Chapter 5

AREA QUERY PROCESSING

In this chapter, we propose an energy-efficient in-network area query processing scheme. We define the area queries and introduce our area query processing scheme in detail in the following sections.

5.1 Area Query in Wireless Sensor Networks

In monitored areas of a sensor network, sensor nodes might be equipped with several sensing components to monitor environment or detect events. Users can specify a query, which describes an event or specific areas of users' interests.

An area query is defined as followings:

> SELECT areas and/or aggregation functions
>
> FROM entire sensor network or subareas
>
> [WHERE predicates]
>
> [GROUP BY Adjacency]
>
> [HAVING predicates]
>
> [DURATION time − span
>
> EVERY time − interval]

SELECT presents query results. It can be an area, some aggregation function of sensing values of an area. *FROM* specifies queried areas. Users can write complicated query conditions on sensing attributes by *WHERE*. *GROUP BY* is used to divide sensor nodes with expected readings and adjacent sensing coverage into groups. *HAVING* presents conditions on aggregation functions. *DURATION* specifies lifetime of a continuous query. *EVERY* defines an execution interval, which means the query is continuously executed and results are returned every *time − interval* seconds. If a query does not specify *DURATION* and

EVERY, it is not a continuous query, and it will be executed just once.

The coal mine example mentioned in Chapter 1 can be presented as the following area query.

SELECT Area, Area.avg(sensors.oxygen)

FROM sensors

WHERE sensors.oxygen > 80

GROUP BY Adjacency

HAVING Area.size > 50m^2

DURATION 24 Hours

EVERY 600 Seconds

It depicts that an area is qualified when oxygen density of sensors is greater than 80, and the acreage of this area is greater than 50m^2. This query will continuously run for 24 hours. Results will be reported to users every 600 seconds. To promise the safety of workers, this query continuously reports results to users. This query also reports the average oxygen density of each qualified area to users. Most previous aggregation methods provide the aggregated values of some particular properties, but not for areas.

Another example, air pollution monitoring, is described as the following query.

SELECT Area, Area.avg(sensors.pollution − level)

FROM industrial area as R1, residential area as R2

WHERE R1.sensors.pollution − level > 80,

R2.sensors.pollution − level > 50

GROUP BY Adjacency

HAVING R1.Area.size > 1000m^2, R2.Area.size > 100m^2

DURATION 72 Hours

EVERY 20 minutes

A scheme which efficiently processes area queries is needed. Due to the fact of limited computation and energy resources of sensor networks, energy-efficiency is the main optimization goal here, and reducing in-network computation complexity of in-network should also be a

concern.

5.2 In-network Area Query Processing Scheme

In this section, an in-network area query processing scheme is presented. Our approach is to construct an initial query results and incrementally update query results along a reporting tree in a bottom-up manner, rather than collecting all sensor reports and transmitting them to BS to process queries centrally. As we know, energy is the most limited resource for current battery-powered sensor nodes. Moreover, communication cost is the dominating factor of energy consumption. Consequently, in-network area query processing is more energy-efficient than a simple, centralized approach.

We first need to partition the whole monitored area into grids and assign a unique ID for each grid, which is presented in Section 5.2.1. Then a reporting tree is constructed for the whole network, which is described in Section 5.2.2. Based on the reporting tree, we show in Section 5.2.3 how to obtain an initial set of query results. The procedure to incrementally update the previous results through the whole query lifetime is presented in Section 5.2.4.

5.2.1 Area Partition

We assume that any point in the monitored area is covered by at least one sensor node. In addition, the sensor network is static, and the locations (2-D Cartesian coordinates) of sensor nodes are known to BS. This location information can be either acquired by GPS or measured manually. Localization algorithms also can be used to calculate locations of sensor nodes. For easy to describe areas, the whole monitored area is divided into small grids. First, the whole monitored area is vertically partitioned into two even subregions. Then, these two subregions are further horizontally partitioned into four subregions. For each subregion, we recursively partition it vertically and horizontally. The partition process is stopped until either there is only one sensor node in that subregion or the size of this subregion is not greater than partitioning threshold P_t. P_t is the minimum size of a grid defined by users. If there are enough sensor nodes, we suggest that the diagonal length of
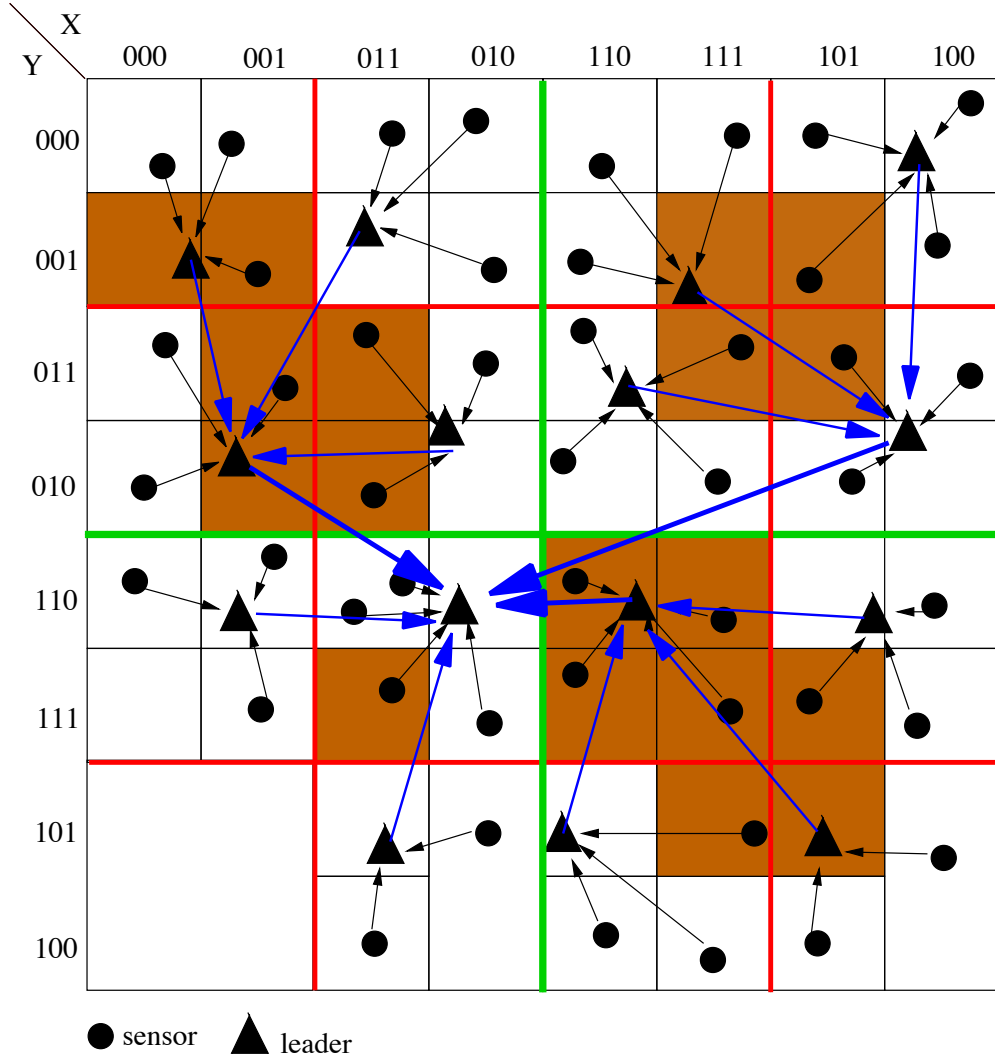
Figure 5.1. Working Scenario.

a grid is no larger than a sensor's sensing range, so that each sensor can cover its resident grid. The accuracy is therefore guaranteed. In this way we divide the whole network into grids as shown in Figure 5.1. The benefit of this partition is that it is convenient to merge the qualified subareas and to calculate the aggregated acreage.

The grid at top-right corner has two sensor nodes. Since the size of this grid is less than P_t, we can stop the partition even though there are more than one sensor node in that grid. The grid at bottom-right corner cannot be further divided because there is only one

---

Algorithm 4 : Partition($X_l, X_r, Y_l, Y_r, X_b, Y_b, D, C_X, C_Y$)

Input: Partition area. {Initially, Partition($X_{left}, X_{right}, Y_{left}$,
$Y_{right}, null, null, X, 0, 0$) is called.}
Output: Grids with GIDs.
 1: **if** area$((X_l, X_r), (Y_l, Y_r))$ has no intersection with the monitored area **then**
 2:    return;
 3: **end if**
 4: **if** $\sqrt{(X_r - X_l)^2 + (Y_r - Y_l)^2} \leq P_t$ **then**
 5:    Output the Grid $((X_l, X_l), (X_r, Y_r))$with GID:(X_b, Y_b).
 6:    return.
 7: **end if**
   {The condition, at least one sensor node in each grid, cannot be guaranteed.}
 8: **if** it cannot be partitioned in direction D **then**
 9:    **if** $D == X(i.e.$ verticality) **then**
10:       Partition$(X_l, X_r, Y_l, Y_r, X_b, Y_b, Y, C_X, C_Y)$ {If it cannot be partitioned in vertical direction, try to partition it in horizontal direction.}
11:    **else**
12:       Output the Grid $((X_l, X_l), (X_r, Y_r))$ with GID:(X_b, Y_b).
13:    **end if**
14: **end if**
15: **if** $D == X$ **then**
16:    Partition$(X_l, \frac{(X_l+X_r)}{2}, Y_l, Y_r, Shift_{left}(X_b) + C_X, Y_b, Y, 0, C_Y)$.
17:    Partition$(\frac{(X_l+X_r)}{2}, X_r, Y_l, Y_r, Shift_{left}(X_b) + \overline{C_X}, Y_b, Y, 1, C_Y)$.
18: **else**
19:    Partition$(X_l, X_r, Y_l, \frac{(Y_l+Y_r)}{2}, X_b, Shift_{left}(Y_b) + C_Y, X, C_X, 0)$.
20:    Partition$(X_l, X_r, \frac{(Y_l+Y_r)}{2}, Y_r, X_b, Shift_{left}(Y_b) + \overline{C_Y}, X, C_X, 1)$.
21: **end if**

---

sensor node in it. After partition, there is at least one sensor node in each grid. The average sensing value of all sensor nodes in each grid is treated as the sensing value of that grid. For prolonging the lifetime of sensor networks, sensor nodes in a grid can be on duty alternately.

As BS is in charge of partitioning, it stores geographical location information of all grids. We can use a unique Grid ID (GID) to represent each grid so that BS can easily acquire location information according to a GID. Gray code [57] is a binary numeral system where two successive numbers differ in only one bit. We adopt gray codes as GIDs. We use X to present a gray code in horizontal direction, and Y to present a gray code in vertical direction. A GID is formatted as (X, Y). As shown in Figure 5.1, the grid at the top-right

corner is $(100, 000)$. The grid at the bottom-right corner is $(100, 10x)$, where x means either 0 or 1 since it is the union of $(100, 101)$ and $(100, 100)$. Obviously, the length of a GID is the number of partition iterations in both vertical and horizontal directions. Furthermore, we can use one GID to express the union of multiple adjacent grids by replacing the union of 0 and 1 with x. For instance, $(10x, 00x)$ is the union of the four grids at top-right corner. This scheme can significantly reduce the amount of the maintained information.

The recursive partition algorithm is described in Algorithm 1. First, the whole monitored area is enclosed with a rectangle. We use two vertices on the same diagonal to decide a rectangle's location. BS needs to record what is the location information of a particular GID. BS runs Algorithm 1 to divide this rectangle into grids. Since the monitored area might be in an irregular shape, it is possible that there exists non-monitored area in this rectangle as shown by the bottom-left corner gris in Figure 5.1. In the process of partitioning, once a non-monitored subregion is detected, it is ignored (Algorithm 1, Line 1-3). The rectangle is recursively partitioned in vertical and horizontal directions alternatively until *return* condition is satisfied. For each generated grid, the location information, GID (X_b, Y_b), and sensor nodes in this grid are recorded. In Algorithm 1, $(X_l, X_r), (Y_l, Y_r)$ are coordinates of the two vertices representing the current partition area. D is partition direction, and it can be either X (vertical) or Y (horizontal). C_X and C_Y are parameters to generate gray codes. Function $Shift_{left}$ is used to shift a binary number one bit left. After partition, BS sends a unique GID to every sensor node.

5.2.2 Reporting Tree Construction

For in-network query processing, a reporting tree is constructed by BS. In Figure 5.1, a reporting tree example is shown. This tree is established from leaf nodes to the root. After deploying sensor nodes, BS knows the initial energy and GID of all sensor nodes. We group the grids whose GIDs differ on the last bits of X and Y codes (*i.e.* $(\%x, \%x)$, where $\%$ can represents any binary string of any length) together. In each group, a sensor node is chosen to be the leader (triangular nodes shown in Figure 5.1) of this group . Other sensor nodes in

the group report data to its leader. In the other words, the leader is the parent node of other sensor nodes in the same group. Then, among the leaders in the groups of grids $(\%xx, \%xx)$, one sensor node is chosen to be the parent node of other leaders generated from the last step. Next step, we consider groups with grids $(\%xxx, \%xxx)$ and so on. The process is repeated until the root of this tree is identified. That is, finally we consider all the grids as an entire group. The height of this reporting tree is $Max(|X|, |Y|) + 1$, where $|X|$ and $|Y|$ are the length of gray code X and Y respectively. BS sends children and parent information to each sensor node. In our scheme, the reporting tree does not serve as the routing tree. The routing can be generated by any routing protocol such as DSDV in [58].

Since leaders, non-leaf nodes, have to process the reports from their children and report to their parent, they consume much more energy than leaf nodes. To prolong the network lifetime, we should reconstruct the reporting tree periodically, and let sensor nodes with more residual energy serve as leaders. However, to construct a new tree, BS has to collect energy information from all sensor nodes, and after generating the new tree, BS has to broadcast the tree structure to the whole network. This consumes a large amount of energy. Therefore we generate a new reporting tree through switching child and parent roles in the tree in distributed way. This switching process starts from the bottom of the tree and ends at the root. The child node with the maximum remaining energy accepts its parent role to sever as new parent, the old parent becomes a child of this new parent, and new parent's siblings become its children. Switching process is accomplished in a bottom-up manner which is similar to the way we construct the initial reporting tree.

5.2.3   In-network Area Query Processing

The motivation of in-network processing is that the number of transmission messages can be reduced by merging multiple reports as one and filtering useless data as early as possible, thereby prolonging network lifetime.

When BS receives an area query request, it registers this query in the system. Then, the query is decomposed into query conditions, merging conditions, query lifetime, and executing

interval. Query conditions, lifetime, and executing interval are sent to every sensor node in the queried area. Merging conditions are sent to non-leaf nodes of the reporting tree. If a query is a continuous query, it is processed at interval.

In essence, query conditions are these filtering conditions on sensing attributes defined by *WHERE* clause such as temperature, moisture, and oxygen density. Query lifetime and executing interval are specified in *DURATION* and *EVERY* clauses respectively. A noteworthy point is that different sensor nodes might accept different query conditions if they are in different query areas since users might define different conditions for different areas such as air pollution monitoring example mentioned in Section 1. Within the lifetime of a query, a sensor node sends a report to its parent every interval if its readings satisfy the query conditions. The reporting message is formatted as $(SID, GID, RID, V_1, V_2, ..., V_M)$, where SID is a sensor node ID, GID is the grid ID, RID is the area ID to indicate different query areas such as industrial area and residential area in the example mentioned in Chapter 1, and V_i is the sensing value of the attribute i.

Merging conditions include aggregation operations and filtering conditions specified by *HAVING* clause. The number of merging steps S is $Max(|X|, |Y|)$. We start from step 1 involving grids $(\%x - \%x)$ and stop at step S involving all grids. In the other words, merging process follows the reporting tree in bottom-up order. In the reporting tree, every non-leaf node is responsible for at least one merging step. If a non-leaf node is the root for grids $(\%x \cdots x, \%x \cdots x)$, it is the merging node for these grids. The merging step i is the number of x in X or Y. Some non-leaf nodes might participate in several merging steps, and the root node participates all merging steps. For instance, in Figure 5.1 the root node (in the grid $(010, 110)$) processes step 1, 2, and 3 merging for grids $(01x, 11x)$, $(0xx, 1xx)$, and (xxx, xxx) one by one. In the process of merging, a leader merges adjacent subareas from its children into one and filters areas which cannot be merged by the up-level leader. Moreover, aggregations are calculated after merging. Then, the leader generates a report and sends it to its parent. Specially, the root sends a report to BS. The report is formatted as $(RID, Subarea_1, GIDlist, datalist, RID, Subarea_2, GIDlist, datalist, \cdots,$

Figure 5.2. Merge subareas.

RID, $Subarea_n$, $GIDlist$, $datalist$). A GID list is used to describe an area. For instance, the shaded areas at top-left corner in Figure 5.1 is denoted as $(00x, 001)$, $(0x1, 01x)$ instead of 6 GIDs. Consequently, GID lists can reduce the size of area description information since a GID can represent multiple grids.

There are two main difficulties for merging areas. Firstly, how to judge two subareas are adjacent since location information is not maintained by sensor nodes. Secondly, how to judge an isolated area is identified. For a non-leaf node, it cannot decide whether a subarea is adjacent to subareas of its siblings since it just has information of its children. However, using gray codes as GIDs provides an intelligent way to solve these two difficult problems.

For the example in Figure 5.1, suppose the shaded grids are eligible grids which satisfy the requirements on sensing attributes. Suppose the requirement of acreage is that the returned area should not be less than $2A$, where A is the acreage of a smallest grid. At step 1, grids in each group $(\%x, \%x)$ are merged. For example, the result of the group

$(00x, 00x)$ is $(00x, 001)$ which means the two lower grids are involved, and the result of the group (11x,11x) is (11x,11x) which means all the grids in this group are involved. At step 2, grids in each group $(\%xx, \%xx)$ are merge. For example, the result of group $(1xx, 0xx)$ is $(1x1, 0x1)$, and this is an isolated area since it is not adjacent to the border of the group $(1xx, 0xx)$. That is, only the grids adjacent to the border of a group have chances to be merged with other grids in other groups at the next step. We can infer the following rule. At step i, grids whose last $i-1$ bits are all 0's on X or Y, are on the border of a group. Once an isolated area is found, it is sent to BS instead of its parent if it is large enough, otherwise it is dropped. As a result, transmitting data size in the reporting tree can be reduced significantly. Thus, $(1x1, 0x1)$ is sent to BS since its size is $4A$, and $(011, 111)$ is dropped since its size is less than $2A$. Other two shaded subareas, $((00x, 001), (0x1, 01x))$ and $((11x, 11x), (101, 111), (1x1, 101))$, are sent to the leader in the grid $(010, 110)$ which is the root of the tree, because they have chances to be merged. Now we give another rule for merging. At step i, only GIDs, whose the last $(i-1)$th bit is 1 and last $i-2$ bits are all 0's on X (or Y), might be merged on X (or Y). For example, at step 2, grids in $(\%1, \%)$ and $(\%, \%1)$ might be merged; at step 3, grids in $(\%10, \%)$ and $(\%, \%10)$ might be merged. This rule is important for reducing the complexity of merging computation. The properties of gray codes and these two rules are crucial for efficiently solving these two difficult problems mentioned above.

The merging algorithm is described in Algorithm 2. An example is shown in Figure 5.2. After receiving reports from its children, a non-leaf node invokes this algorithm to do merging subareas with same RID and sends merged results to its parent. In this algorithm, if L_0 and L_1 are ordered by Y-GID in a gray code order when we try to merge on X, the complexity can be reduced since we try to find GIDs with intersection on Y. XNOR (the inverse of the exclusive OR operation) denoted as $\overline{\oplus}$ is used to judge whether two GIDs have intersection on Y or X. We define the result of $x\overline{\oplus}\omega = 1$, where ω is 0,1, or x. The acreage of a GID is $2^n * A$, where n is the number of x in the GID. The acreage of an area or subarea is the sum of acreage of all GIDs in its GID list. Acreage calculation is also

---

Algorithm 5 : Merging(i, GID lists from its children)

Input: Step i and subareas from children.

Output: Merged results.

 1: **for** each grid in the reporting messages **do**
 2:   **if** the last $(i-1)$th bit of GID on X is 1 and last $(i-2)$ bits are 0 **then**
 3:     Add this GID into merging list L_0 if the ith bit is 0, otherwise add it to L_1.
 4:   **end if**
 5: **end for**
 6: **for** all pairs of GID_1 in L_0 and GID_2 in $L1$ **do**
 7:   **if** $GID_1.Y \overline{\bigoplus} GID_2.Y == 1$ **then**
 8:     **if** $(GID_1.Y == GID_2.Y)$ AND $(GID_1.X == GID_2.X$ except the last ith bit) **then**
 9:       Merge GID_1 and GID_2 into one GID by replacing 0 and 1 on the last ith bit with x.
10:     **end if**
11:     Merge two GID lists (to which GID_1 and GID_2 belong) into one and Calculate aggregations.
12:   **end if**
13: **end for**
14: Merge subareas on Y coordinate using the similar method from line 1 to 13.
15: **for** each subarea A **do**
16:   **if** GIDs of this Area list have no intersection with $((\% \; \mathrm{x} \overbrace{0 \cdots 0}^{(i-1)bits}, \%) \cup (\%, \% \; \mathrm{x} \overbrace{0 \cdots 0}^{(i-1)bits}))$ **then**
17:     **if** Size of A does not satisfy the condition **then**
18:       Delete A.
19:     **else**
20:       Send this result A to BS and delete A.
21:     **end if**
22:   **end if**
23: **end for**
24: Send merged subareas to its parent.

---

simplified by using GIDs. The aggregation function *sum* and *avg* of an area are calculated by $\frac{\sum\limits_{i=1}^{n} V_i * A_i}{\sum\limits_{i=1}^{n} A_i}$ and $\frac{\sum\limits_{i=1}^{n} V_i * A_i}{n * \sum\limits_{i=1}^{n} A_i}$ respectively, where V_i and A_i are the sensing value and acreage of the grid i respectively.

5.2.4   Incremental Result Update

In most applications, sensing values only change slightly or mostly unchange over a long time. Therefore, consecutive merged results are extremely similar for a continuous query and incremental updates could conserve a large amount of energy. By comparison, regenerating query results at intervals using Algorithm 2 will quickly deplete energy because of the heavy communication cost.

We now present an incremental updating scheme to maintain merged results on non-leaf nodes of the reporting tree after the initial results are calculated. Each sensor node stores its previous report, and non-leaf nodes store previous merged results. We classify changed subareas into three classes, *only value*, *positive*, and *negative*. For a subarea in the previous report, if only sensing values or aggregation values are changed, it is a *only value* subarea; for a subarea not in the previous report but in the current report, it is a *positive* subarea; for a subarea in the previous but not in the current report, it is a *negative* subarea. The incremental updating process updates merged results along the reporting tree step by step. For a *negative* subarea, a merging node removes the dropped grids. A subarea might be split if some grids are dropped. For a *positive* subarea, it is merged into the previous results by using the similar method in Section 5.2.3. For *only value* subareas, aggregation values are updated.

A leaf node does not send a report if its readings are not changed or both previous and current readings are not satisfied the *WHERE* conditions. A non-leaf node generates an update report by comparing new merged results with the previous results and sends it to its parent. An update report includes *positive*, *negative* subareas, and new aggregation values for other subareas if these values are changed. If the size of an update report is greater than merged results, we still send merged results.

5.2.5   Advantages of Using Gray Codes

Communication cost is a significant guideline to evaluate the efficiency of an algorithm in wireless sensor networks. Gray codes are used to represent grids in our scheme. If we use 14-bit gray codes, *i.e.* 7 bits for X and 7 bits for Y, 16384 (that is $2^7 * 2^7$) grids can be represented. However, we cannot use binary representation since x (either 0 or 1) is also used in GIDs. Hence, 14-bit ternary (base 3) representation is used to represent GIDs. Due to the fact that 2^{24} is greater than 3^{14}, 3 bytes (*i.e.* 24 bits) is enough to express a GID after converting it to binary representation. Another effective method for representing a grid is using two vertices. Each vertex is stored as a pair of x-y coordinate. If we use 2 bytes for x and y coordinates respectively, 8 bytes are needed to represent a grid. 8 bytes are quite long compared to 3 bytes of a GID. Furthermore, GID description method does not lose location information, although GIDs do not maintain these information. Location information can be retrieved from BS. So, GID is the most effective approach to reduce the size of grid denotation.

In our merging algorithm, due to the properties of gray codes and two rules we found, it is easy to detect an isolate area as early as possible. Consequently, useless data can be dropped early, and results can be sent to BS without further merging. Moreover, adjacent subareas can be merged to reduce the size of GID list. In summary, all these strategies are effective methods to reduce the number of messages and the size of messages. However, it is very difficult to achieve these benefits by using other methods such as vertex denotation.

5.3   Intelligent Monitoring Systems

Sensor networks consisting of different kinds of sensors can help people gather a large amount of information at any time, any place, and under any circumstances. Monitoring systems of sensor networks are significant for various fields such as industrial, civilian, and military applications. In this section, we will introduce an Intelligent Monitoring System (IMS) we proposed, which can not only detect various events but also provide real-time
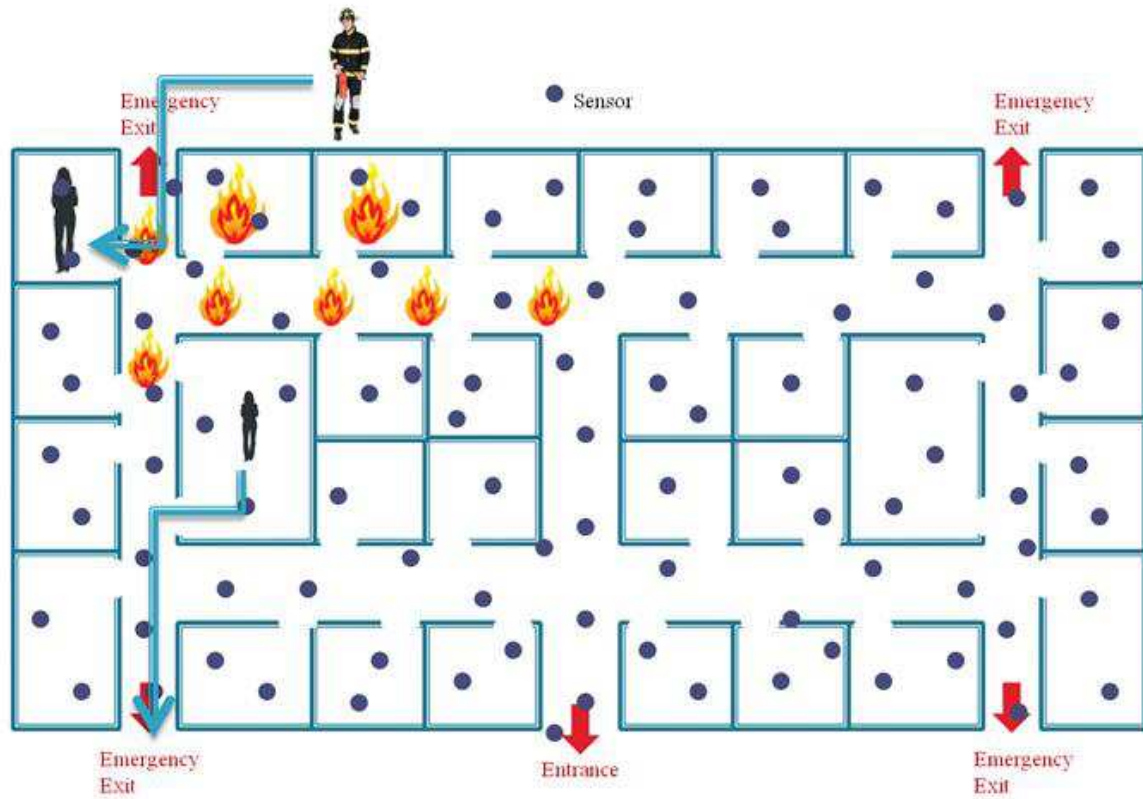
Figure 5.3. IMS for monitoring fire in a building.

and accurate information for escaping, rescuing, and evacuation when a dangerous event happened. To the best of our knowledge, this is the first study addressing the issues, event detection, escaping, rescuing, and evacuation, at the same time for a monitoring system in the literature. IMS can be widely used for various monitored environments and monitoring missions.

Area queries can find areas which users define. For monitoring applications, we usually try to find safe ares (e.g., high oxygen density areas for coal mine workers), dangerous areas (e.g., pollution areas for monitoring environment), or both. For a monitoring system of a building, we expect not only it can detect dangerous events such as fire and gas leak but also it can provide useful information for rescuing, finding a shortest safe route for trapped people and rescuers. For instance, if a building is on fire as shown in Figure 5.3, sensor nodes

can detect it and trigger an alarm since fire causes high temperature and smoke. Then, our area processing can find all safe areas (with sufficient oxygen density), and dangerous areas (with high temperature and smoke). Importantly, we need to detect the location of trapped people, and calculate a safe shortest escaping route for them. Specially, an isolate safe area (which is inside an dangerous area) needs to be payed more attention if there are people in that area, we should send rescuers to save them first. People inside the building can send query to the system and get the real-time information which can help them find a way out or find a safe place to stay. The challenging issues to develop this system are listed as follows:

1. How to find a safe escaping route for a person? This is not a simple routing problem in sensor networks since we try to find a route for people instead of a message. A safe route has to be a path people can get through and the entire area along the path should be safe.

2. How to detect an isolate safe area? This is the issue of how to identify a safe area which is contained by a dangerous area in sensor networks. This is very important for finding the trapped people.

3. How to update results in a real-time manner? This is important since any false information can result in severe loss. However, unstable and unreliable characteristics of sensor networks bring difficulties for implementing real-time updating.

4. How to improve system reliability? We need to consider how to make the system work properly under various bad cases like some sensor nodes are broken by accidents, thus guaranteeing the reliability.

Once an event which needs evacuation is detected, BS will send area queries to the whole network to identify dangerous and safe areas. Then, BS has the whole picture of the network. BS can identify an isolate safe area by judging if it is inside an dangerous area.

5.3.1 Escaping Routing Algorithm

An approach to calculate the best escaping route for a person in the monitored area. First, we mark sensors on the exit passageway as *routing sensors*. Then, designing a method to calculate all routes to exits. For each route, we assign a grade by considering various parameters such as the distance to the exit, safe factor on the path, and other possible factors.

After sensor nodes are deployed and routing sensor nodes are marked, each non-routing sensor broadcasts a *routing node finding* to find the closest routing sensor nodes in all directions. When a non-routing sensor node receives a routing node finding message, it will add its own id into the message (insert its id into the id list) and broadcast it. If a routing sensor node receives a routing node finding message, it will send a message with the id list back to the source sensor node via the reverse route (the id list) memorized by the message. Then, non-routing sensor nodes add this routing senor node into its routing sensor list L for finding escaping routes when an escaping route is requested.

Once a serious event happens, we force all sensor nodes work more actively than usual to enhance reliability and reduce respond delay. By increasing transmission range and sampling frequency, updating delay can be shortened obviously. System reliability can be improved through using multi-path routing and increasing transmission range.

The safe escaping algorithm is shown in Algorithm 5.3.1. When we try to find a safe escaping route, only safe routing sensor nodes respond the request message since only this kind of sensor nodes are in the safe exit passageway. Several escaping routes might be found. We only provide first 5 received safe routes to users since the early the route is found, the shorter the path is.

5.4 Simulation Results

In this section, we evaluate the performance of our proposed in-network area query processing scheme.

Algorithm 6 Escaping Routing Algorithm

Input: A sensor node S requests an escaping route.
Output: Escaping paths.
 1: Node S sends requests to all routing sensor nodes in its routing sensor list;
 2: **for** each routing sensor rs receives a request message **do**
 3:   **if** rs's sensing values do not satisfy safe area conditions **then**
 4:     Drop the message;
 5:     return;
 6:   **end if**
 7:   **if** rs is not a routing sensor located at an emergency exit **then**
 8:     Add rs into the id list of the message;
 9:     Broadcast the message;
10:   **else**
11:     A safe escaping path found. Add rs into the message, and send acknowledgment back to the source node vis the reverse path.
12:   **end if**
13: **end for**
14: Node S returns first 5 received safe routes to users.

5.4.1   Simulation Setup

We simulate the area query of coal mine mentioned in Section 5.1. The payload size limit of a packet is 29 bytes which are the standard payload size provided by the TinyOS [59]. Transmission range of sensor nodes is 30m, and sensing range is 15m. P_t is 15m. DSDV [58] is used as the routing protocol.

Since *network traffic* greatly affects energy efficiency, we use it as the metric for performance evaluation. The network traffic is defined as the total number of messages transmitted (sent and forwarded) by all nodes in the network during the execution of a query.

We compare the performance of our scheme, IPGID for short, with two other alternative approaches: (1) Centralized Processing (CP), and (2) In-network Processing based on Vertex description (IPV). For a fair comparison, both CP and IPV use conditions on attributes to filter useless readings. That is, a sensor node does not send a report if its readings cannot passs through the *WHERE* conditions. IPV is similar with our scheme except using an area's vertices to represent it instead of a GID list. In-network merging techniques are also used in IPV.

We varied several system parameters when comparing the performances of the three approaches. These parameters are listed as follows:

1. Length of a GID (L). This parameter affects network diameter and the number of sensor nodes deployed. The longer L means the bigger network and much more sensor nodes deployed.

2. Selective Rate on Attributes (S). It is the percentage of sensors' sensing values which can pass through the *WHERE* conditions.

3. Filtered Rate of Area (F). It is the percentage of areas which satisfy the *HAVING* conditions.

4. Changed Rate of sensing Values (C). This parameter is the percentage of changed results compared to the previous set of results.

5.4.2 Efficiency of Our Scheme

As shown in Figures 5.4, 5.5, 5.6, 5.7, and 5.8, our IPGID is always the best one, and IPV is better than CP. The reason is that since IPGID and IPV use incremental updating techniques to maintain results, only the difference between the current and previous set of results are reported. However, CP reports qualified data every interval. IPGID is better than IPV due to the fact that the size of a GID list is much smaller than that of vertex information for describing an area.

Figure 5.4 shows the network traffic of the first 20 intervals. At the first interval, there's only a marginal difference among the three approaches since both IPV and IPGID construct the initial query results at the first interval. Later on, a clear difference between CP and IPV (or IPGID) presents. The reason is that the number of the reported messages is reduced after the first interval since IPV and IPGID use incremental update techniques. At the 20th interval, IPGID has 83% less transmitted messages than CP. IPV has 71% less transmitted messages than IPV.
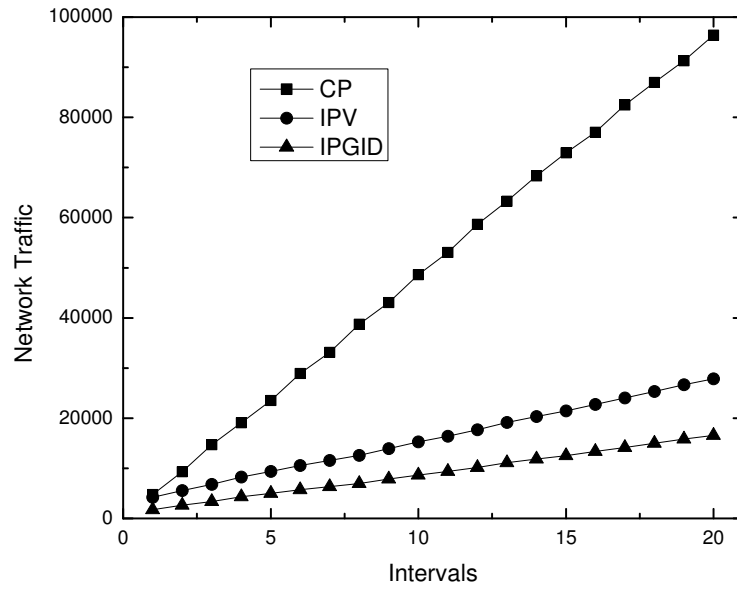
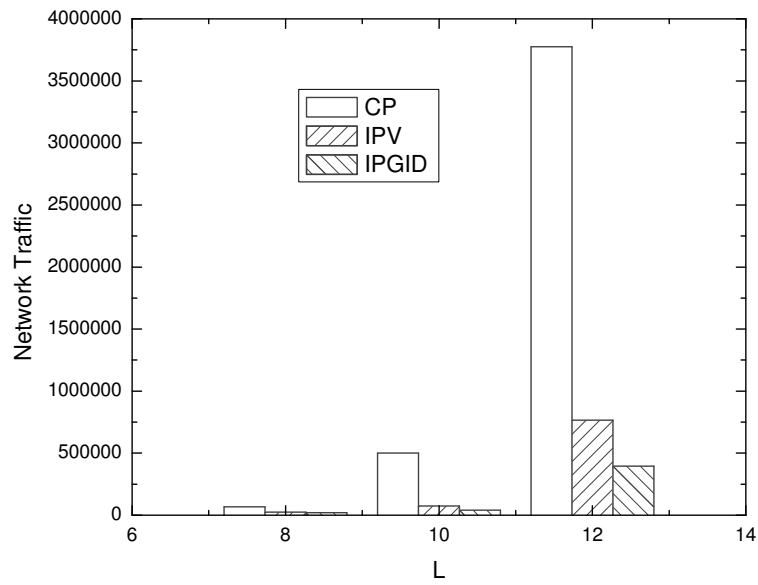Figure 5.4. Network Traffic of execution intervals (L=10 *bits*, S=50%, F=50%, C=20%).

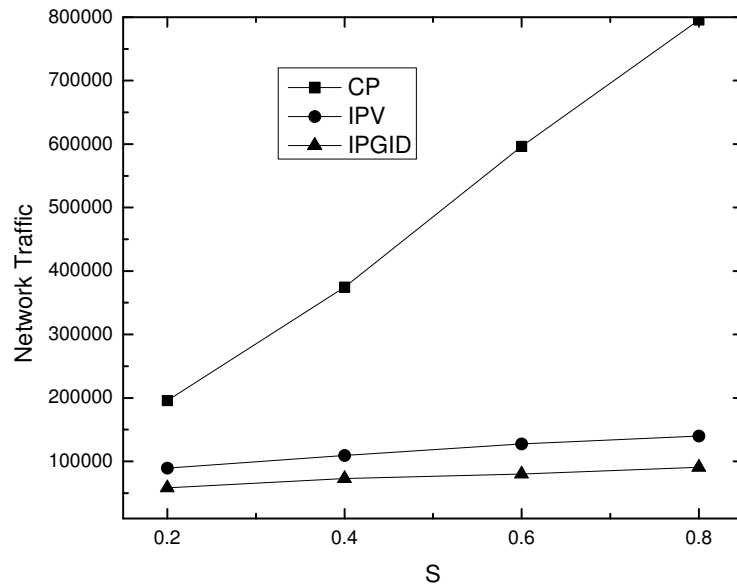Figure 5.5. Network Traffic of variant L (S=50%, F=50%, C=20%, 100 *intervals*).

Figure 5.6. Network Traffic of variant S (L=10 $bits$, F=50%, C=20%, 100 $intervals$).
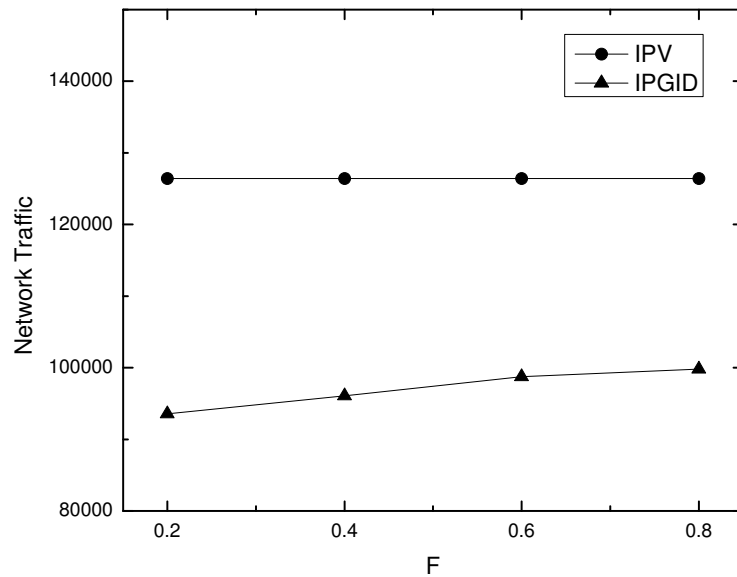

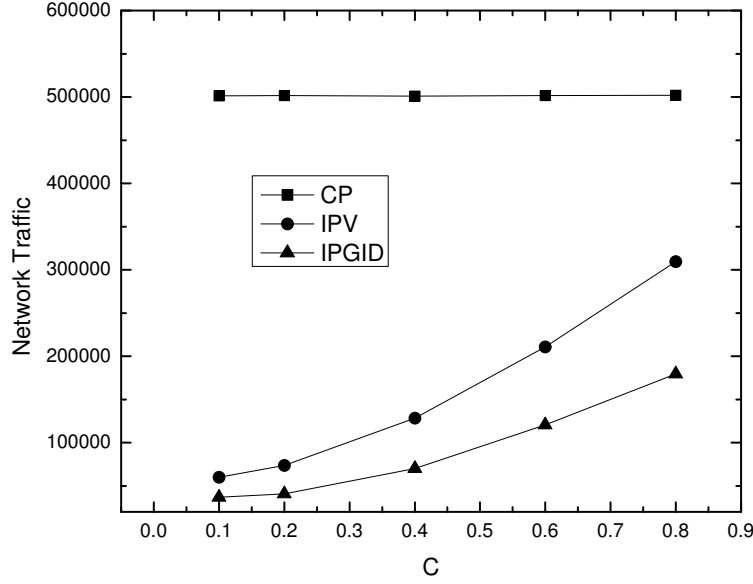Figure 5.7. Network Traffic of variant F (L=10 $bits$, S=50%, C=20%, 100 $intervals$).

Figure 5.8. Network Traffic of variant C (L=10 $bits$, S=50%, F=50%, 100 $intervals$).

As shown in Figure 5.5, with the increasing of L, the advantages of IPGID becomes more obvious. Although a longer GID incurs more messages, merging multiple reports and filtering useless data in the reporting tree as early as possible can reduce the number of messages, which is the method used by IPV and IPGID. However the reporting pathes of CP become longer as the network size becomes bigger. Thereby, on average, IPGID has 84% less transmitted messages than CP. IPGID has 37% less transmitted messages than IPV.

Figure 5.6 shows the network traffic for variant S. The network traffic increases with S because the bigger S causes more sensor nodes to report readings. CP is affected by S more obvious than IPV and IPGID. The reason is that for CP, one more sensor node involved causes a long path message transmitting to BS. However, for IPV and IPGID, it just causes a message to its parent. The network traffic for variant F is shown in Figure 5.7. Since smaller F implies that more areas can be dropped in the reporting tree according to $HAVING$ conditions, the smaller F, the less network traffic for IPGID. IPV and CP are not affected by this parameter since IPV and CP cannot filter the useless areas. The

network traffic for variant C is presented in Figure 5.8. With the increasing of C, the task of incremental updating becomes much heavier. As a result, network traffic increases with C. When C is close to 1, IPGID and IPV almost have to reconstruct merged results every interval. IPV is worse than IPGID since IPV use more messages to describe a big area than IPGID.

In summary, our in-network area query processing scheme is energy-efficient in various circumstances.

Chapter 6

CONCLUSIONS

In this dissertation, we addressed three key issues of data management in WSNs.

To conserve energy in WSNs for data collection, we propose a scheme where only a few nodes are chosen to work, and the other nodes are set to sleep. The data sensed by sleeping nodes are estimated by two efficient methods, DESM and DEPM. DESM is a statistical method that takes advantages of the time and space locality of a WSN. DEPM technique utilizes the physical laws to estimate data. Since the principle of superposition on which DEPM is based on is a very common principle in physics and is obeyed by a large number of physical quantities, the presented techniques have a very wide applicability. Furthermore, DEPM model enables us to deploy a rather small constant number of sensors out of a large number of available sensors, thereby conserving much energy. Experimental results show that the proposed methods are efficient and effective.

We have proposed a scheme, HDQP, to process historical data queries of wireless sensor networks. Our approach is the first work to study distributed historical data query processing by using an effective distributed index tree. The indexes can help process queries energy-efficiently and reduce the query responding delay. Index tree switching mechanism also can balance the load among sensor nodes to avoid data distribution and energy consumption skew.

An energy-efficient in-network area query processing scheme is proposed in this dissertation. Our approach is the first work to study area query processing in wireless sensor networks. We define the area queries and partition the network into grids. Gray Code is employed to express the ID of grids. Initial query results are generated by merging partial results along the reporting tree. For conserving energy, an incremental updating strategy is used to generate continuous query results. The size of results can be reduced by using

binary GIDs to describe areas. In-network processing can drop the useless data as early as possible. Furthermore, incremental updating avoids unnecessary reports. All these details fulfill the achievement of energy efficiency. Our simulation results confirm it.

Since energy cost for sensing and data processing is insignificant compared to data communication in a sensor node, when we design the algorithms we mainly focus on how to reduce network traffic, thus reducing data communication cost. The algorithms proposed in this dissertation can reduce the network traffic efficiently as shown in the experimental results. Therefore, our data management algorithms are energy-efficient.

REFERENCES

[1] K. Romer and F. Mattern, "The design space of wireless sensor networks," *Wireless Communications, IEEE*, vol. 11, no. 6, pp. 54–61, Dec. 2004.

[2] S. Hadim and N. Mohamed, "Middleware: middleware challenges and approaches for wireless sensor networks," *Distributed Systems Online, IEEE*, vol. 7, no. 3, pp. 1–1, March 2006.

[3] "Crossbow - wireless sensor networks - products - wireless modules," http://www.xbow.com/Products/productdetails.aspx?sid=156. [Accessed February 03, 2009].

[4] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.

[5] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks (Elsevier) Journal*, p. 393422, 2002.

[6] E. Runckle and R. Heins, "Manipulating the light environment to control flowering and morphogenesis or herbaceous plants," in *International Symposium on Artificial Lighting in Horticulture*, June 2006.

[7] "Horticulture measurements. [online]," available: http://www.avantes.com/Applications/HortiSpec.htm. [Accessed November 03, 2007].

[8] "Commonwealth of massachusetts. proposed regulations and public hearing notice," available: http://www.mass.gov/agr/animalhealth/petshops/. [Accessed November 03, 2007].

[9] G. Richter, "Blue light control of the level of two plastid mrnas in cultured plant cells," *Plant Molecular Biology*, vol. 3, pp. 271–276, 1984.

[10] "Crossbow wireless sensor networks. telosb," available: http://www.xbow.com/Products/productdetails.aspx?sid=252. [Accessed November 03, 2007].

[11] M. Aly, A. Gopalan, J. Zhao, and A. Youssef, "Stdcs: A spatio-temporal data-centric storage scheme for real-time sensornet applications," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on*, June 2008, pp. 377–385.

[12] A. Meka and A. Singh, "Dist: a distributed spatio-temporal index structure for sensor networks," in *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management.* New York, NY, USA: ACM, 2005, pp. 139–146.

[13] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton, "Codeblue: An ad hoc sensor network infrastructure for emergency medical care," in *International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[14] R. G. C. Intanagonwiwat and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *Proceedings of the sixth annual international conference on Mobile computing and networking*, Boston, MA USA, 2000, pp. 56–67.

[15] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles.* New York, NY, USA: ACM, 2001, pp. 146–159.

[16] "Tinyos website," available: http://www.tinyos.net/ [Accessed August 15, 2009].

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *SIGOPS Oper. Syst. Rev., vol. 36, no. SI*, June 2002, pp. 131–146.

[18] D. J. Abadi, S. Madden, and W. Lindner, "Reed: robust, efficient filtering and event detection in sensor networks," in *VLDB '05: Proceedings of the 31st international conference on Very large data bases.* VLDB Endowment, 2005, pp. 769–780.

[19] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.

[20] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden, "Distributed regression: an efficient framework for modeling sensor network data," in *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, April 2004, pp. 1–10.

[21] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM, 2004, pp. 250–262.

[22] J. M. Hellerstein and W. Wang, "Optimization of in-network data reduction," in *DMSN '04: Proceeedings of the 1st international workshop on Data management for sensor networks.* New York, NY, USA: ACM, 2004, pp. 40–47.

[23] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering.* Washington, DC, USA: IEEE Computer Society, 2006, p. 48.

[24] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases.* VLDB Endowment, 2004, pp. 588–599.

[25] A. Silberstein, R. Braynard, G. Filpus, G. Puggioni, A. Gelfand, K. Munagala, and J. Yang, "Data-driven processing in sensor networks," in *CIDR '07: Proceedings of the*

3rd Biennial Conference on Innovative Data Systems Research. Asilomar, California, USA: IEEE Computer Society, 2007.

[26] D. Wang, J. Xu, J. Liu, and F. Wang, "Mobile filter: Exploring migration of filters for error-bounded data collection in sensor networks," *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pp. 1483–1485, April 2008.

[27] A. Jain, E. Y. Chang, and Y. Wang, "Adaptive stream resource management using kalman filters," in *SIGMOD Conference*, 2004, pp. 11–22.

[28] D. de O. Cunha, R. P. Laufer, I. M. Moraes, M. D. Bicudo, P. B. Velloso, and O. C. M. B. Duarte, "Bio-inspired field estimation with wireless sensor networks," Universidade Federal do Rio de Janeiro, Tech. Rep., http://www.gta.ufrj.br/ftp/gta/TechReports/CLMB05a.pdf, accessed November 03, 2007.

[29] P. Dash, "Land surface temperature and emissivity estimation from passive sensor data: Theory and practice–current trends," *International Journal of Remote Sensing*, vol. 23, pp. 2563–2594, 2002.

[30] Y. Li, C. Ai, W. P. Deshmukh, and Y. Wu, "Data estimation in sensor networks using physical and statistical methodologies," in *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems.* Washington, DC, USA: IEEE Computer Society, 2008, pp. 538–545.

[31] P. B. Johannes, J. Gehrke, and P. Seshadri, "Towards sensor database systems," 2001, pp. 3–14.

[32] J. Gehrke and S. Madden, "Query processing in sensor networks," in *Proceedings of the First Biennial Conference on Innovative Data Systems Research. CIDR' 03*, 2003.

[33] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-centric storage in sensornets," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 137–142, 2003.

[34] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "Ght: a geographic hash table for data-centric storage," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002, pp. 78–87.

[35] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional range queries in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*.   New York, NY, USA: ACM, 2003, pp. 63–75.

[36] M. Aly, K. Pruhs, and P. K. Chrysanthis, "Kddcs: a load-balanced in-network data-centric storage scheme for sensor networks," in *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*.   New York, NY, USA: ACM, 2006, pp. 317–326.

[37] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker, "Difs: a distributed index for features in sensor networks," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, May 2003, pp. 163–173.

[38] G. Li, J. Li, and Y. Li, "Time: Time-based index management for event query processing in wireless sensor networks," in *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, Dec. 2008, pp. 1–8.

[39] C. Ai, R. Du, M. Zhang, and Y. Li, "In-network historical data storage and query processing based on distributed indexing techniques in wireless sensor networks," in *WASA '09: Proceedings of the International Conference on Wireless Algorithms, Systems and Applications*. Boston, MA, USA: Springer Berlin / Heidelberg, August 2009, pp. 264–273.

[40] A. Sharaf, J. Beaver, A. Labrinidis, and K. Chrysanthis, "Balancing energy efficiency and quality of aggregate data in sensor networks," *The VLDB Journal*, vol. 13, no. 4, pp. 384–403, 2004.

[41] A. Silberstein and J. Yang, "Many-to-many aggregation for sensor networks," *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 986–995, April 2007.

[42] D. Goldin, "Faster in-network evaluation of spatial aggregationin sensor networks," *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pp. 148–148, April 2006.

[43] Y. Cho, J. Son, and Y. D. Chung, "Pot: an efficient top-k monitoring method for spatially correlated sensor readings," in *DMSN '08: Proceedings of the 5th workshop on Data management for sensor networks.* New York, NY, USA: ACM, 2008, pp. 8–13.

[44] W. Xue, Q. Luo, L. Chen, and Y. Liu, "Contour map matching for event detection in sensor networks," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data.* New York, NY, USA: ACM, 2006, pp. 145–156.

[45] C.-R. Li and C.-K. Liang, "A fault-tolerant event boundary detection algorithm in sensor networks," pp. 406–414, 2008.

[46] M. Ding, "Fault tolerant event boundary detection and target tracking in sensor networks," Ph.D. dissertation, Washington, DC, USA, 2008, adviser-Cheng, Xiuzhen and Adviser-Choi, Hyeong-Ah.

[47] A. Khelil, F. K. Shaikh, B. Ayari, and N. Suri, "Mwm: a map-based world model for wireless sensor networks," in *Autonomics '08: Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems.* ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–10.

[48] P. Szczytowski, A. Khelil, and N. Suri, "Map++: support for map-based wsn modeling and design with omnet++," in *Simutools '09: Proceedings of the 2nd International*

Conference on Simulation Tools and Techniques. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–2.

[49] C. Ai, L. Guo, Z. Cai, and Y. Li, "Processing area queries in wireless sensor networks," in *Accepted by MSN '09: the Fifth International Conference on Mobile Ad-hoc and Sensor Networks*, Wuyi Mountain, China, December 2009.

[50] G.B.Arfken and H.J.Weber, *Mathematical Methods for Physicists.* Academic Press, 2003.

[51] "Computer science of georgia state university. experimental demonstration of the physical methodology," available: http://www.cs.gsu.edu/yli/datasets/sensor.html. [Accessed November 03, 2007].

[52] X.Tang and J.Xu, "Extending network lifetime for precision constrained data aggregation in wireless sensor networks," in *Proc. IEEE INFOCOM*, Apr. 2006.

[53] S. Xiang, H. B. Lim, and K.-L. Tan, "Multiple query optimization for wireless sensor networks," *Data Engineering, International Conference on*, vol. 0, pp. 1339–1341, 2007.

[54] S. Xiang, H. B. Lim, K.-L. Tan, and Y. Zhou, "Two-tier multiple query optimization for sensor networks," in *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems.* Washington, DC, USA: IEEE Computer Society, 2007, p. 39.

[55] Y. W. Lee, K. Y. Lee, and M. H. Kim, "Energy-efficient multiple query optimization for wireless sensor networks," in *SENSORCOMM '09: Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 531–538.

[56] M. Tang, J. Cao, and N. Chilamkurti, "Tampa: Tabu search-based multiple queries optimization for wireless sensor networks," in *Wireless Communications, Networking*

and Mobile Computing, 2007. WiCom 2007. International Conference on Digital Object Identifier, 2007, pp. 2731 – 2734.

[57] G. M., "The binary gray code," in *Ch. 2 of Knotted Doughnuts and Other Mathematical Entertainments*, New York: W. H., 1986.

[58] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 234–244, 1994.

[59] "Tinyos faq [online]," *Available: http://www.tinyos.net/faq.html. [Accessed Oct 28, 2008]*.