2010

Building efficient wireless infrastructures for pervasive computing environments

Bo Sheng
College of William & Mary - Arts & Sciences

BUILDING EFFICIENT WIRELESS INFRASTRUCTURES FOR PERVASIVE COMPUTING ENVIRONMENTS

Bo Sheng

Nanjing, Jiangsu, China

Bachelor of Science, Nanjing University, 2000

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Computer Science

The College of William and Mary
January, 2010

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

Bo Sheng

Approved by the Committee, September 2009

Committee Chair
Associate Professor Qun Li, Computer Science
The College of William & Mary

Associate Professor Weizhen Mao, Computer Science
The College of William & Mary

Associate Professor Haining Wang, Computer Science
The College of William & Mary

Professor Evgenia Smirni, Computer Science
The College of William & Mary

Assistant Professor Gexin Yu, Mathematics
The College of William & Mary

ABSTRACT PAGE

Pervasive computing is an emerging concept that thoroughly brings computing devices and the consequent technology into people's daily life and activities. Most of these computing devices are very small, sometimes even "invisible", and often embedded into the objects surrounding people. In addition, these devices usually are not isolated, but networked with each other through wireless channels so that people can easily control and access them. In the architecture of pervasive computing systems, these small and networked computing devices form a wireless infrastructure layer to support various functionalities in the upper application layer.

In practical applications, the wireless infrastructure often plays a role of data provider in a query / reply model, i.e., applications issue a query requesting certain data and the underlying wireless infrastructure is responsible for replying to the query. This dissertation has focused on the most critical issue of efficiency in designing such a wireless infrastructure. In particular, our problem resides in two domains depending on different definitions of efficiency. The first definition is time efficiency, i.e., how quickly a query can be replied. Many applications, especially real-time applications, require prompt response to a query as the consequent operations may be affected by the prior delay. The second definition is energy efficiency which is extremely important for the pervasive computing devices powered by batteries. Above all, our design goal is to reply to a query from applications quickly and with low energy cost.

This dissertation has investigated two representative wireless infrastructures, sensor networks and RFID systems, both of which can serve applications with useful information about the environments. We have comprehensively explored various important and representative problems from both algorithmic and experimental perspectives including efficient network architecture design and efficient protocols for basic queries and complicated data mining queries. The major design challenges of achieving efficiency are the massive amount of data involved in a query and the extremely limited resources and capability each small device possesses. We have proposed novel and efficient solutions with intensive evaluation. Compared to the prior work, this dissertation has identified a few important new problems and the proposed solutions significantly improve the performance in terms of time efficiency and energy efficiency. Our work also provides referrable insights and appropriate methodology to other similar problems in the research community.

Table of Contents

To my parents, my wife Ningfang, and my son Anray.

ACKNOWLEDGMENTS

When pursuing my Ph.D. at the College of William and Mary, I am fortunate to work with a group of great people who have also contributed to this dissertation. The collaboration with them is an enjoyable experience that I will cherish in my life.

My deepest gratitude goes to my adviser Dr. Qun Li. I still remember how unsure I felt about research five years ago. It is Qun's supervising that has helped me mature towards a successful researcher. I have learned a lot from his incredible understanding about how to do good quality research. He has taught me to be confidant and persistent. In my view, Qun is the role model of a knowledgeable and enthusiastic researcher. I have always been encouraged by his passion and inspired by his keen wit.

I would also like to thank my dissertation committee including Dr. Qun Li, Dr. Weizhen Mao, Dr. Haining Wang, Dr. Evgenia Smirni, and Dr. Gexin Yu. I really appreciate their time and efforts reviewing my proposal, pre-defense and this dissertation. They have given a lot of thoughtful comments and suggestions which have certainly helped improve this dissertation.

I would like to give special thanks to Dr. Weizhen Mao who I have worked with for a long time. During our collaboration, I was greatly impressed by her solid theoretical knowledge, her meticulous research style, her patience and kindness. I have learned a lot from her on algorithm design and analysis. She has also given me many suggestions to help improve my writing skill.

In addition, I have collaborated with several excellent Ph.D. students, Dr. Haodong Wang, Chiu C. Tan, Hao Han, and Dr. Heng Yin. I am grateful to them for their contributions to my research work that makes this dissertation possible. I would also like to acknowledge Mengjun Xie, Chuan Yue, Adam Wu, Lei Xie, and Fengyuan Xu for the discussions with them and their feedback about my work.

Furthermore, I am thankful to the former and current staff in Computer Science Department, Vanessa Godwin, Jacqulyn Johnson, Judy Fiorello, Lynn Miller, and the Techie support group, for their various forms of support during my graduate studies.

Finally, none of this would be possible without the love and support from my family. Thanks to my parents, Luqu and Weiguo, for their patience, trust, and understanding during all these years. Thanks to my lovely wife Ningfang who has experienced every moment with me over the past five years. Her warm support and encouragement always comfort me through this journey. Thanks to my adorable son Anray who has made the last days of writing this dissertation full of sunshine.

List of Tables

List of Figures

BUILDING EFFICIENT WIRELESS INFRASTRUCTURES FOR PERVASIVE COMPUTING ENVIRONMENTS

Chapter 1

Introduction

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Mark Weiser (1952 - 1999)

Pervasive computing was first introduced by Mark Weiser in early 1990's described as "the next stage of computing" after the mainframes and the evolution of personal computers. It represents an emerging concept that thoroughly integrates computing devices into everyday objects and activities, in short, "computers everywhere". Beyond that, pervasive computing also emphasizes networks, applications, data, and services everywhere. A pervasive computing environment essentially consists of various connected devices that conduct diverse information processing tasks on the behalf of users. IBM Chairman Lou Gerstneras once said the following about pervasive computing environments, "Picture a day a billion people interacting with a million e-businesses through a trillion interconnected intelligent devices."

Unlike desktop computers, pervasive computing devices are usually very tiny, sometimes even "invisible", and can be carried by people or embedded into the surrounding objects in our daily

life. These small devices are networked through wireless channels so that they can easily interact with people or other devices. Nowadays, technology development has substantially brought pervasive computing into the mainstream. A lot of small pervasive computing devices have already been manufactured. Computer chips are embedded into appliances, mobile phones, automobiles, digital pens/desks, industrial machines, health care devices, or even on walls and floors. The presence of these non-desktop computers and the associated applications or services has greatly changed people's life styles from many aspects. In addition, a wide variety of wireless network protocols have been developed providing strong connectivity to each device, e.g., cellular networks, 802.11 family, bluetooth, zigbee, WiMax, DSRC (Dedicated Short-Range Communications), IrDA (Infrared Data Association) and NFC (Near Field Communication). With these protocols, embedded devices are seamlessly connected and data on any device is available to other entities across the network. At this point, pervasive computing has never been closer to a reality with all available resources. Although the development of pervasive computing is promising, some fundamental issues still remain unsolved.

In this dissertation, we focus on the efficiency of the wireless infrastructure which is the underlying layer in the architecture of a pervasive computing system. This wireless infrastructure is formed by the networked pervasive computing devices and plays the role of data provider to support upper layer applications. In a typical pervasive computing system, data are provided in a query / reply model, where applications generate queries to request certain data and the wireless infrastructure transparently finds the desired data and returns them as the reply to the queries. As the wireless infrastructure handles the data transmission, it becomes the key component for achieving the efficiency. Designing an efficient wireless infrastructure has been one of the most fundamental challenges for pervasive computing applications and the objective can be further defined in two

domains. The first is *time efficiency* which refers to a quick response to a query. Pervasive computing applications, especially when involved in real time systems, require the wireless infrastructure to reply to a query in a timely fashion. A long delay further postpones other consequent operations and may make the application infeasible or dysfunctional. The second definition of efficiency considers energy consumption in the wireless infrastructure. This *energy efficiency* is more specific to pervasive computing environments since a lot of small devices are powered by batteries and they are often deployed for long-term tasks. For example, a sensor network is usually expected to work solely on battery power for several months or even years. Therefore, conserving energy to prolong the lifetime of each device and the whole system becomes mandatory for practical implementation. With the above two definitions, the objective in this dissertation is to design efficient wireless infrastructures that reply to queries quickly and with low energy consumption.

Developing such an efficient wireless infrastructure is challenging because of the unique characteristics of pervasive computing systems. First, pervasive computing devices are weak hardwares with extremely limited resources and ability. These hardware constraints often demand efficient designs, but they are also serious obstacles for achieving the efficiency. Second, as computing devices become tiny and ubiquitous, pervasive computing usually refers to a large scale system with many devices. It implies a large amount of data in the system and numerous data transmissions for a query leading to difficulties in efficiently delivering the reply. Some efficiency problems encountered in this area have been studied in regular computer systems, but most of the existing approaches cannot be directly applied because of the limited ability of pervasive computing devices. Additionally, pervasive computing has opened a broad field of applications which yield new efficiency problems that people have never experienced with regular computers. Recent research work has spent much effort in addressing the efficiency issues in the wireless infrastruc-

ture, which include a list of fundamental questions, such as how to deploy pervasive computing devices to help improve the efficiency, how to efficiently transfer data from one device to another, what is the role of each device in data processing and delivery, and how to make devices collaborate for a task. More important, what are the proper methodologies that can be adopted to answer these questions.

In this dissertation, we concretely investigate two important and representative instances of wireless infrastructures, sensor networks and RFID systems. By exploring the efficiency problems in both systems, we demonstrate how to design the infrastructure with general pervasive computing devices. In the following sections, we first briefly introduce some background information about theses two types of systems, and then we detail our target problems.

1.1 Research Scope and Background

Typical characteristics of pervasive computing environments include small-sized devices, wireless connection, computing ability, and a large scale system. In the current stage of the development, sensors and RFID tags are widely-used and representative hardwares for pervasive computing applications as they possess all the above characteristics. Therefore, sensor networks and RFID systems are selected to be the platforms for the research work in this dissertation. This section presents basic information about sensor networks and RFID systems and their features are summarized at the end.

1.1.1 Sensor Networks

A sensor network consists of spatially distributed devices, called sensor nodes. Fig. 1.1 illustrates a typical sensor hardware *mote*. Each sensor node contains one or more sensing devices, and is

able to monitor the physical or environmental conditions such as temperature, humidity, vibration, pressure, and motion. In addition, sensors are powered by batteries and equipped with a microcontroller, memory, and external storage. Furthermore, each sensor has a radio transceiver for wireless communication. The following Table 1.1 shows the detailed specification of a typical mote hardware.

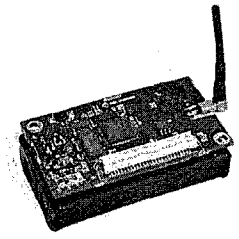


Figure 1.1: A typical sensor ('Mote' manufactured by Crossbow)

Microcontroller	ATMega128L, Up to 8 MIPS throughput at 8 Mhz
Memory	128-Kbyte Program Memory, 4-Kbyte SRAM, 4-Kbyte EEPROM
External Memory	512-Kbyte flash memory
Radio Transceiver	2.4 GHz IEEE 802.15.4, 250 kbps

Table 1.1: Specification

The major task of a sensor network is to collect data measured by every sensor. Usually, one or more powerful computers, called sink or base station, are deployed as a gateway between the sensor network and the end users, i.e., sensors will deliver their data to the sink while the end users also send data requests to the sink. Because of the limited radio transmission range, it is not practical for each sensor to reach the sink directly. Instead, the wireless communication in a sensor network follows a hop-by-hop fashion, i.e., data from a sensor are sent to the sink through multiple relay sensors. In practical applications, sensors often form a tree routing structure rooted

at the sink which enables data aggregation and filtering at non-leaf nodes further reducing the energy consumption. We explain with more details in Chapter 2.

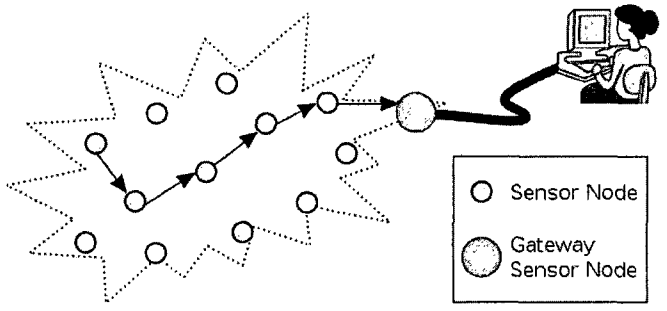


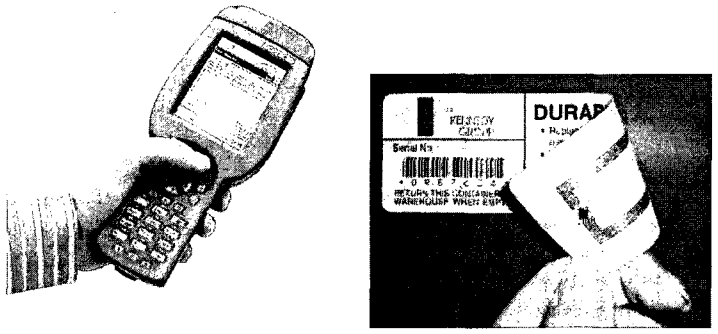
Figure 1.2: Wireless Communication in a Sensor Network

Sensor networks have been increasingly deployed in various civil and military applications, and also highly desirable in many scientific research areas. Their major tasks are monitoring environments and detecting special events. Typical applications include monitoring habitat of animals, monitoring structures of buildings or bridges, monitoring agriculture growing conditions, detecting or forecasting disasters such as forest fire and earth quack, and detecting vehicles or enemy in a battlefield.

1.1.2 RFID Systems

The second infrastructure studied in this dissertation is RFID system. RFID, short for Radio Frequency Identification, is a new technology and has attracted a lot of attention recently. An RFID system consists of RFID readers and tags, where an RFID reader is a typical handheld device such as PDA. The most common RFID tags are passive tags with no power supply. They have certain memory space for storing data, usually tens of bytes. Each RFID tag contains an integrated circuit that can process the data and the commands from RFID readers. In addition, RFID tags have an antenna for communicating with RFID readers via wireless channels. The

latest generation of RFID tags can work at ultra high frequency range (860-960 MHz).



(a) An RFID reader (b) An RFID tag

Figure 1.3: Typical RFID Hardware

In an RFID system, there is no communication between different RFID tags. Data are transferred only between an RFID reader and an RFID tag through wireless channels. The data transmission is conducted in a unique way due to the fact that an RFID tag has no power to transmit anything. The RFID reader initializes the communication by broadcasting signals with certain commands. The integrated circuit on the RFID tag is triggered by the incoming signals from the RFID reader. Then, the RFID tag will process the commands and transmit data back to the RFID reader by backscattering the received signals. The RFID reader is able to decode the useful information from the reflected signals.

In RFID applications, every product or item is attached with an RFID tag, which stores some data describing the product. This data on each tag is called tag ID which contains various useful information about the affiliated product. People can use an RFID reader to remotely scan these tags and fetch the ID information stored on them. Some important applications include inventory control, supply chain management and product tracking. RFID tags are also used in other applications such as electric passport, transportation payment, item management in a library or museum,

and animal identification.

1.1.3 Summary

Both sensor networks and RFID systems are representative wireless infrastructures in pervasive computing applications as they provide useful information about the environment. Here we summarize their common features and differences.

First, both systems consist of typical weak hardwares. Compared to a regular computer, sensor devices have very limited resources in terms of power supply, CPU speed, storage capacity and network bandwidth. RFID tags are even weaker devices than sensors. They can hardly handle any computation and they have no power to communicate with each other. Therefore, studying these two systems helps understand the design principles for weak hardwares.

Second, both systems represent large scale pervasive computing environments. Sensors are usually densely deployed over a large field. A typical monitoring application involves hundreds or thousands of sensors. RFID applications often target at large warehouse or shipping port, where we may encounter a huge volume of items each attached with an RFID tag.

Third, the data carried by these two systems are different while they both provide information about the environment. Sensors collect physical environmental data in an active means. These data are dynamically changing over the time and locations. They are highly correlated in many scenarios. For example, sensors in the same room may have similar temperature readings. Thus, redundancy exists among the data collected by a sensor network. In an RFID system, people preload data to each tag to annotate the objects. The data on each RFID tag is fixed and supposed to uniquely represent an item.

Finally, these two systems represent different communication models. In a sensor network,

data are transferred from a sensor to the sink or another sensor in a multi-hop fashion. Data aggregation, processing, and filtering are possible at the relay sensor nodes. In an RFID system, however, there is only one-hop communication between the RFID reader and an RFID tag. RFID tags can not communicate with each other.

In summary, sensor networks and RFID systems represent the most popular scenarios for wireless infrastructures in a pervasive computing environment. Investigating these two systems provides us with full understanding of the efficiency problems and the results in this dissertation have a broad impact on this research community. The next section presents the research problems examined in this dissertation.

1.2 Research Directions and Problems

Implementations of pervasive computing applications include layered components as shown in Fig. 1.4. Each application task from users is first translated to one or more queries. The corresponding query protocols determine what information is needed for the reply and how to obtain it. These query protocols are built on the physical network architecture which provides low-level general functions. For example, a query protocol may need device A to transfer data to device B, and the network architecture layer is responsible for a routing path from A to B according to the topology.

In this dissertation, with the objective of achieving the efficiency, we focus on two research directions of designing network architectures and query protocols. The first direction represents underlying frameworks that can efficiently support different applications. The second direction explores application-aware optimizations for the efficiency. We investigate representative efficiency problems including some known problems with more practical settings and a few new challenging

problems. With the novel and substantial solutions, we demonstrate several useful techniques for pervasive computing environments. In the rest of this section, we present the details about the problems this dissertation addresses.

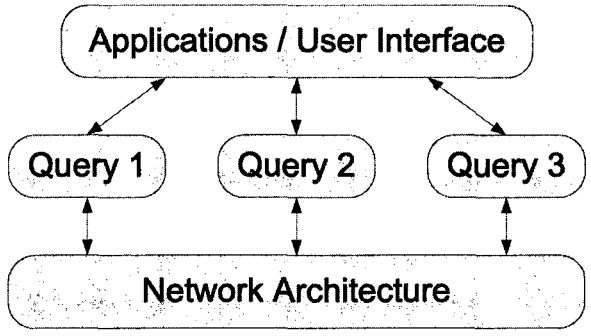


Figure 1.4: Design Layers in Pervasive Computing Applications

1.2.1 Network Architecture

In many applications, the performance of replying to queries is related to the network architecture of the infrastructure referring to structural factors such as network topology, routing protocol, and data flow. With general characteristics of queries, the underlying network architecture can be optimized for the efficiency. A sensor network supports flexible configurations of the network architecture leaving us the opportunity to improve the efficiency in this direction.

A major research trend in the literature is to deploy powerful sensors to build a hybrid network with regular sensors to achieve more efficient performance. Powerful nodes have more enriched resources, such as faster CPU, higher bandwidth, and more battery capacity. Different from the prior work, we explore a novel idea of deploying powerful sensors with large storage capacity, called *storage nodes*. Since flash memory becomes inexpensive, deploying storage nodes is more practical compared to upgrading the CPU or antenna. We utilize storage nodes to form a two-tiered sensor network and support in-network storage mechanism. Our research focuses on finding the

best locations to deploy storage nodes with the goal of improving the efficiency.

1.2.2 Query Protocol

The major research direction in this dissertation is to design efficient query protocols. Real applications may issue various queries to extract different data information. There is no universal solution that can achieve the efficiency in all queries. The most efficient solution is to specifically optimize the query protocol for each particular query. In this dissertation, therefore, we investigate some representative queries in sensor networks and RFID systems. By exploring them, we demonstrate the appropriate methodology for designing efficient query protocols and our solutions can be easily extended to solve other similar queries.

1.2.2.1 Basic Query

Our work starts with the most basic and fundamental queries. These queries are frequently used and they are often the building blocks of other complicated queries. Therefore, the efficiency of the basic queries has been well studied in the literature. The prior work, however, mostly considers an ideal problem setting ignoring practical requirements and constraints. The solutions for simplified settings may not solve the efficiency problems in real implementations. Our work re-visits the efficiency problems in these basic queries, and goes one step further with more realistic settings. The details of the problems are presented in the next.

Range Query in Sensor Networks We first considers range query in sensor networks, which requests the data in a specified value range $[a, b]$. Range query is a basic operation and widely used in sensor network applications. Different from the prior work, we investigate range query with security and privacy requirements which are the common concerns in real applications because

sensor networks are often deployed in an untrusted or even hostile environment. A compromised sensor network may leak sensitive information to an unauthorized party, which leads to a privacy breaching. In addition, it may also give false information about the collected data to a valid query, misleading the application. In deploying such a realistic sensor network, a fundamental question is how much one should trust a sensor network and how to prevent, or at least, detect the misbehavior of the sensor network. Unfortunately, little research work has focused on these privacy and security issues.

We consider a network model consisting of storage nodes and regular sensors, where storage nodes are responsible of hosting raw data from nearby regular sensors and replying to the queries from the sink. The role of a storage node implies that it has to gain some understanding about the stored data for an energy-efficient data reply by avoiding sending all the collected data back. The practice would not be a problem if the storage nodes are assumed to be trusted. It is not valid, however, if the storage nodes are susceptible to compromise and the disclosure of the information may endanger the crucial assignment for the users in the network. With more sensor networks deployed for pervasive computing applications, this issue becomes even more serious if the user information is leaked through the storage nodes, which breaches the privacy requirement.

Generally, an adversary is not able to compromise numerous sensors. The limited number of compromised regular sensors do not affect the query reply seriously because of redundant sensor deployment and limited coverage of the compromised sensors. The storage nodes, which hold much data collected from many sensors, however, easily become the targets for compromise and have to be a great concern when privacy related information is collected and query is imposed to the collected data.

Two threats arise when storage nodes are compromised by the adversary. First, the compro-

mised storage nodes may disclose the data stored on them to the adversary. Thus, we would like the storage nodes to reply to the range query without gaining too much information. Second, the compromised storage nodes may send false information as the reply. It is difficult to prevent the malicious storage nodes from cheating on data reply. But at least the user is entitled to know whether the reply is intact. Therefore, our goal is to design a range query protocol that preserves the data privacy and enables the sink to verify the reply.

Continuous Scans in RFID systems The second basic query examined in this dissertation is to scan RFID tags and collect all tag IDs. It is the most fundamental query in an RFID system and the major focus of all previous work. In this operation, time efficiency is of crucial importance for many RFID applications, especially when they deal with a large volume of RFID tags. For example, inventory management often requires an RFID reader to scan all the products in a warehouse. This task usually involves tens of thousands of RFID tags, assuming every product is attached with one tag, and mandates a quick scanning process.

The previous literature, however, has only focused on developing efficient protocols for a single scanning process. In fact, many tasks cannot be accomplished by a single scan. Instead, multiple scanning processes have to be continuously launched. For example, to scan all the products in a large warehouse for inventory management, it is impossible for an RFID reader to read all the tags at one location due to the limited reading range. Usually a single mobile reader (or multiple readers) has to launch multiple scanning processes at different locations to cover all the tags in the entire warehouse (illustrated in Fig. 1.5). In this dissertation, we study this common practice of continuous scanning, and aim at designing efficient protocols for it.

Continuous scanning is generally defined as a series of multiple scanning processes. This work investigates continuous scanning in both spatial and temporal domains. The above example

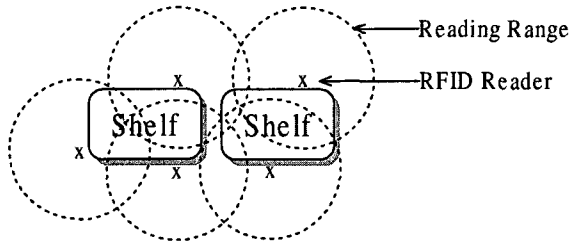


Figure 1.5: An example of spatial continuous scanning

illustrates a *spatial continuous scanning*, where a series of scans are executed at different locations. Namely, *temporal continuous scanning* represents a series of scans occurring at different time points. It is often used for monitoring inventory update. For example, some applications may want to keep track of the products stored at a certain location in a dynamic environment where new products may be put on shelf and some existing products may be moved out. An RFID reader, in this case, has to periodically scan all the present RFID tags to keep a fresh record. Therefore, our goal is to develop efficient continuous scanning protocols that can quick detect the inventory changes, i.e., collecting the tags newly added and remove the tags that are no longer present.

1.2.2.2 Data Mining Query

After exploring basic queries in pervasive computing systems, the natural next step is to investigate some complicated queries. The most representative ones are data mining queries which have been well studied in the database community. In a pervasive computing environment, the wireless infrastructure can be treated as a large database since each device contains useful data. As data mining queries have been shown important for applications in the database literature, little research on data mining has been conducted in pervasive computing environments. Different from the problems in the database area, data mining in pervasive computing is more challenging because the target data set is distributed on each small device rather than collected at a central server which

is a common assumption in the database literature. In addition, energy efficiency, one of our major goals, is not a concern for regular data mining in the prior work.

In this dissertation, we investigate two classic data mining queries. First, we study the problem of finding outlier data in a sensor network. The definition of outlier data is determined not by a single data value, but by the distribution of all data values. However, it is extremely hard for a sensor network to efficiently obtain the global information about all data values. Our work presents the first efficient solution to this outlier detection query. The second data mining query we consider is to find popular categories of the items in an RFID system. The challenge is that little computation and communication ability seriously impedes RFID tags to collaborate and efficiently supply the requested information. Our work proposes efficient protocols that work with the off-the-shelf RFID hardwares requiring no extra functionality. The details are introduced as follows.

Detect Outliers in Sensor Networks As we mentioned earlier, sensor networks are usually large scaled and expected to work for a long time, thus accumulate a large amount of data. Mining this large data repository for useful information is crucial in many applications. In a simple solution, data collected by sensors can be transmitted to the sink for data mining analysis. This method, however, consumes too much energy because the data volume transferred can be extremely large. Thus, the batteries of the sensors will be quickly depleted, leading to network partition and dysfunction. Therefore, a desired method must be energy-efficient, while still be able to extract information from the large amount of data distributed over the network.

In this work, we consider one of the most important data mining problems: outlier detection, which defines a process to identify data points that are very different from the rest of the data based on a certain measure. Outlier detection in a central database has been a hot topic in the data

mining community, but little work has been done in the context of a sensor network in which data are distributed over thousands or tens of thousands of sensors.

Essentially, outliers represent a complex form of abnormal data which are critical in many sensor network applications. Abnormal data is often a warning of suspicious events, and effective detection schemes can trigger an alarm in the early stage and prevent serious consequences. Some abnormal data can be easily defined and detected. For example, if a sensor network is deployed to detect fire by monitoring temperature, then the very high temperature readings will be the abnormal data which can be distinguished by a threshold for normal temperature. This kind of abnormal data can be detected by just looking at the data values. Outliers, however, are defined based on a global view of all the data and hard to be identified without knowledge of other data values. For example, assume sensors are embedded on a bridge to monitor structural integrity by measuring the vibration generated by the passing vehicles. Heavier cars will cause larger vibrations and structural weakness may also yield larger than normal vibrations. When a sensor measures a large vibration, it is difficult to determine whether it is abnormal by only considering the data value. The large reading might be normal if it is caused by a heavy truck, but it also could be an abnormal value if the passing car is a sedan. The only way to find out is to compare with the data measured by other sensors. Thus, outlier detection represents a category of complicated data mining queries that require the knowledge of all the data generated by every sensor. In practice, however, gathering global information in a sensor network is extremely costly due to the network-wide transmission. Therefore, our goal here is to efficiently identify outlier data without collecting all data at a central point.

Find Popular Items in RFID Systems The last query we examine is to find popular categories in an RFID system. Many RFID applications involve a large amount of tags to be read, e.g., in a

shipping portal or warehouse, the items in pallets and cases are read together (i.e., in bulk). An open question in RFID area is how to efficiently extract useful information from a large scale RFID system considering the limited ability of each tag. Our work investigates a particular problem of finding the popular categories among these numerous items. The categories have flexible definitions on different domains and granularity and we consider a user specific threshold to define 'popular', i.e., if the number of tags in a category exceeds the threshold, the category is called popular. Finding popular items is important for tracking the most popular categories shipped in a day, or the least consumed types of goods in a warehouse, or the most frequent values sensed by RFID sensors when the values can be classified into categories.

In the prior work, all queries are answered by first collecting all tag IDs. However, when the collection of tags is large, reading data from every tag is very time consuming. In addition, getting all raw data is not necessary for this particular query. Therefore, our goal is to design an efficient protocol to find the popular categories without scanning all RFID tags.

1.2.3 Summary

As a quick summary, the following Fig. 1.6 lists the research work in this dissertation. In the direction of network architectures, we investigate data storage placement in sensor networks. Our major work in this dissertation resides in designing efficient query protocols and we examine two basic queries and two data mining queries in sensor networks and RFID systems.

	Network Architectures	Query Protocols	
		Basic Query	Data Mining Query
Sensor Networks	Data Storage Placement	Range Query	Detect Outliers
RFID Systems		Continuous Scans	Find Popular Items

Figure 1.6: Summary of Research Work

1.3 Contributions

This dissertation focuses on the fundamental efficiency problems in wireless infrastructure design for pervasive computing environments. The contributions of this dissertation are:

- We have introduced a two-tiered hybrid network architecture composed of regular sensors and special storage nodes. We have developed the optimal algorithms for deploying storage nodes to reduce energy consumption.
- We have examined two basic queries, range query in sensor networks and continuous scans in RFID systems. Our work has developed efficient query protocols for them with practical problem settings.
- We have investigated the efficiency issues with complicated data mining queries. We have developed efficient protocols for two representative queries of detecting outliers in sensor networks and finding popular items in RFID systems.

1.3.1 Network Architecture

In the direction of network architecture, we have focused on the data storage placement in sensor networks. Two typical network models are studied for achieving the efficiency. The first is the fixed tree model, where we assume the sensor network has organized into a tree rooted at the sink. The communication routes from sensors towards the sink are predefined by the tree. Our goal is to select some of the nodes in the tree as storage nodes. The second model is called the dynamic tree model, where the (optimal) communication tree is constructed after the storage nodes are deployed. Specifically, each sensor selects a storage node in its proximity for its data storage with the goal to minimize the energy cost of the resulting communication tree.

- For the fixed tree model, we have examined deterministic placement of storage nodes and present the optimal algorithms based on dynamic programming. Our simulation results show that our algorithms significantly reduce the total energy consumption in the sensor network.

- The problem in the dynamic tree model is proven to be NP hard. We have first presented a stochastic analysis for random deployment in the dynamic tree model which is a common practice in sensor network applications. Our analysis provides a useful guideline and accurate estimation of the performance.

- We have further developed an approximation algorithm for the dynamic tree model. From the theoretical aspect, our problem is similar to the facility location problem, but in a more complex form as defined with multiple levels and a in non-metric domain. Our algorithm is the first constant approximation algorithm for this problem. Our evaluation shows the proposed approximation algorithm is much more efficient than random deployment in terms of energy consumption.

1.3.2 Basic Query

The representative basic queries investigated in this dissertation includes range query in sensor networks and continuous scans in RFID systems. For the range query, we have focused on the efficiency issue with security and privacy requirements, especially when a storage node is compromised. Our goals are to prevent the compromised storage nodes from disclosing data stored there and to enable the sink to verify the reply from storage nodes. For the continuous scans in RFID systems, our design focus is how to avoid collecting all IDs at each scan and quickly finish the whole process.

- To the best of our knowledge, our work is the first to consider privacy issue in sensor network data range query. Our work explores the privacy concerns in sensor networks in a very general setting. We have developed a privacy-preserving storage scheme which uses bucketization to obscure the view of the storage node to the data stored on it. Our scheme satisfies the privacy requirements while storage nodes can still efficiently process the raw data and reply to queries.

- We have also developed a scheme that enables the sink to verify the reply to a range query. The major challenge here is how to prevent compromised storage nodes from dropping data. Our solution is based on a novel encoding number scheme. We have developed the optimal algorithm to derive the best parameters for the protocol considering both efficiency and security. With our parameter setting, the proposed protocol is able to detect the false replies with low energy overhead.

- We have designed two efficient algorithms for continuous scanning in RFID systems, one to collect the 'new' tag IDs and the other to detect those 'old' RFID tags that have been moved out. We have presented in-depth analysis to derive the proper parameters for the proposed algorithms to achieve the efficiency and satisfy the accuracy requirements. In addition, we have proposed an improvement for temporal continuous scanning based on a pre-computation before scanning the RFID tags.

1.3.3 Data Mining Query

In this dissertation, we have worked on two classic data mining queries, detecting outliers in sensor networks and finding popular categories in RFID systems. The prior work uses a simple but costly

solution that collects ALL data to reply to these queries. In this dissertation, we have proposed more efficient solutions.

- We have developed an efficient outlier detection scheme based on histogram information for sensor networks. To the best of our knowledge, this is the first histogram-based detection approach to solving this problem. The proposed scheme uses small-sized histogram information to approximate the sensor data distribution and reduce the communication cost under two different detection schemes. We have presented the theoretical analysis for the communication cost incurred in the network.

- Additionally, we have proposed a multi-round histogram refinement technique for some critical portion in the data distribution to gain more information about outliers. The finer histogram information helps filter out more non-outliers, hence further reduce more communication cost. Our trace-driven simulation has demonstrated that our approaches decrease the communication cost dramatically compared to the prior work.

- We are the first to propose efficient solutions to data mining queries in RFID systems without collecting all tag IDs. We have proposed a simple and fast threshold checking scheme (*TCS*), which accurately answers whether the number of involved tags exceeds a threshold with high probability. Furthermore, we have designed two randomized algorithms based on group testing and *TCS* to efficiently find popular categories. We have comprehensively evaluated the proposed schemes and compare them against existing solutions. Our simulation results show that our schemes significantly reduce the total scanning time.

- For both data mining queries, we have demonstrated appropriate methodologies and design principals for such complicated queries in a pervasive computing environment. Particularly,

for a sensor network to handle complicated queries, protocols with multiple rounds of information collection are suitable, where each round obtains small-sized information, such as histogram in our problem, followed by rigorous analysis and estimation. In an RFID system, we have shown randomized algorithm is an useful technique for solving complicated queries because compatible schemes are built on the slotted ALOHA protocol which inherently cope with randomized algorithm. Our work can be easily extended to solve other similar problems and has inspired other work in the research community.

1.4 Organization

The rest of this dissertation is organized as follows. In Chapter 2, we present our work on the network architecture design with the focus on data storage placement schemes. Chapter 3 and Chapter 4 are our work on representative basic queries in sensor networks and RFID systems. In Chapter 5 and Chapter 6, we propose efficient solution to the data mining queries. Finally, we conclude in Chapter 7.

Chapter 2

Network Architecture in Sensor Networks: Data Storage Placement

In this chapter, we present our work in the direction of network architecture, particularly, data storage placement in sensor network. We propose to deploy special storage nodes with large storage capacity to support in-network storage and help reduce energy consumption. Structural optimization in a sensor network often surprisingly improves the efficiency for upper layer application queries. In this work, we take general query characteristics as input and develop algorithms to derive the best locations for storage nodes.

A sensor network deployed for pervasive computing applications, e.g., sensing environmental conditions and monitoring people's behaviors, generates a large amount of data continuously over a long period of time. This large volume of data has to be stored somewhere for future retrieval and data analysis. One of the biggest challenges in these applications is how to store and search the collected data.

The collected data can either be stored in the network sensors, or transmitted to the sink. Sev-

eral problems arise when data are stored in sensors. First, a sensor is equipped with only limited memory or storage space, which prohibits the storage of a large amount of data accumulated for months or years. Second, since sensors are battery operated, the stored data will be lost after the sensors are depleted of power. Third, searching for the data of interest in a widely scattered network field is a hard problem. The communication generated in a network-wide search is prohibitive. Alternatively, data can be transmitted back to the sink and stored there for future retrieval. This scheme is ideal since data are stored in a central place for permanent access. However, the sensor network's per-node communication capability (defined as the number of packets a sensor can transmit to the sink per time unit) is very limited [43, 62]. A large amount of data cannot be transmitted from the sensor network to the sink efficiently. Furthermore, the data communication from the sensors to the sink may take long routes consuming much energy and depleting of the sensor battery power quickly. In particular, the sensors around the sink are generally highly used and exhausted easily, thus the network may be partitioned rapidly.

It is possible that, with marginal increase in cost, some special nodes with much larger permanent storage (e.g., flash memory) and more battery power can be deployed in sensor networks. These nodes back up the data for nearby sensors and reply the queries. The data accumulated on each storage node can be transported periodically to a data warehouse by robots or traversing vehicles using physical mobility as Data Mule [111]. Since the storage nodes only collect data from the sensors in their proximity and the data are transmitted through physical transportation instead of hop-by-hop relay of other sensor nodes, the problem of limited storage, communication capacity, and battery power is diminished.

In this storage model, where to deploy storage nodes is a critical issue for the performance in terms of energy cost. The optimal solution depends on the characteristics of raw data genera-

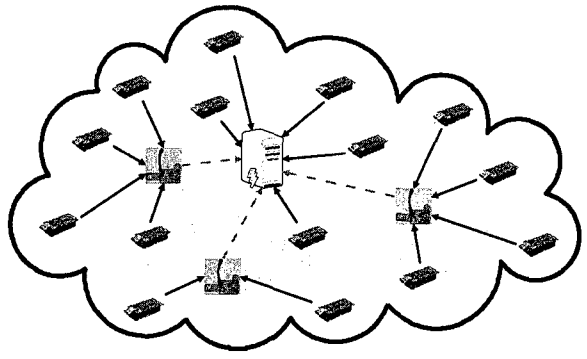


Figure 2.1: Access Model with Storage Nodes

tion and user query as well as the network topology. As already mentioned, the most important operation for sensor networks is the query because the purpose of a sensor network is to provide information of the environment to the end users. A user query may take various forms; for example, a user query may be how many nodes detect vehicle traversing events, and the average temperature of the sensing field. In this scenario, each sensor, in addition to sensing, is also involved in routing data for two network services: the raw data transmission to storage nodes and the transmission for query diffusion and query reply. Two extreme approaches are to transmit all the data to the sink and to store them on each sensor node locally. On one hand, data solely stored in the sink is beneficial to the query reply incurring no transmission cost, but the data accumulation to the sink is very costly. On the other hand, storing data locally incurs zero cost for data accumulation, whereas the query cost becomes large because a query has to be diffused to the whole network and each sensor has to respond to the query by transmitting data to the sink. The storage nodes not only provide permanent storage as described previously, but also serve as a buffer between the sink and the sensor nodes. The placement of the storage nodes can strike a balance between these two schemes characterizing a tradeoff between data accumulation and data query. Therefore, our work aims to achieve the energy efficiency in data accumulation and data query by judiciously placing the storage nodes in the network.

2.1 Related Work

There has been a lot of prior research work on data querying models in sensor networks. In early models [70, 71, 92, 93], query is spread to every sensor by flooding messages. Sensors return data back to the sink in the reverse direction of query messages. Those methods, however, do not consider the storage concern in sensor networks.

On the other hand, some new models introduce an intermediate tier between the sink and sensors. LEACH [65] is a clustering-based routing protocol, in which cluster heads can fuse the data collected from its neighbors to reduce communication cost to the sink. LEACH has a similar structure to our scheme, having cluster heads aggregate and forward data to the sink. However, LEACH aims to reduce data transmission by aggregating data; it does not address storage problem in sensor networks. Data-centric storage schemes [44, 79, 102, 108, 113], as another category of the related work, store data to different places in sensor networks according to different data types. In [79, 108, 113], the authors propose a data-centric storage scheme for sensor networks based on Geographic Hash Table, which inherits ideas from distributed hash table. The home site of data is obtained by applying a hash function on the data type. Thus, queries for the same type of data can be satisfied by contacting a small number of nodes. Another practical improvement is proposed in [44] by removing the requirement of point-to-point routing. In [13], Ahn et al. analyze the scaling behavior of data-centric query for both unstructured and structured (e.g., GHT) networks and derive some key scaling conditions. GEM [102] is another approach that supports data-centric storage. GEM applies graph embedding technique to map data to sensor nodes. In general, the data-centric storage schemes assume some understanding about the collected data and store them remotely for easy data access. Extra cost is needed to forward data to the corresponding keeper nodes. Our work, however, does not assume any prior knowledge about the data: indeed in many

applications, raw data may not be easily categorized into different types. To transmit the collected data to a remote location is also considered expensive in our work because the total collected data may be in a very large quantity. To facilitate data query, Ganesan et al. [50–52] propose a multi-resolution data storage system, DIMENSIONS, where data are stored in a degrading lossy model, i.e., fresh data are stored completely while long-term data are stored lossily. DIMENSIONS uses wavelets to obtain temporal-spatial summarizations in a hierarchical structure, which helps locate a subset of sensor nodes for range queries. Its performance is heavily dependent on the data correlation because of the data summarization scheme. In comparison, our scheme is more general in making no assumption about the data correlation. PRESTO [41, 90] is a research work on storage architecture for sensor networks. A proxy tier is introduced between sensor nodes and user terminals and proxy nodes can cache previous query responses. When a query arrives in a proxy node, it first checks if the cached data can satisfy the query before forwarding the query to other nodes. Compared with the storage nodes in our work, proxy nodes in PRESTO have no resource constraints in term of power, computation, storage and communication. It is a more general storage architecture that does not take the characteristics of data generation or query into consideration. In the Cougar project [22, 40, 53], a data dissemination tree is built with data sources as leaves. *View nodes* introduced in Cougar have similar functionality as storage nodes in our work. Our scheme focuses more on how to optimize the placement of storage nodes, while Cougar mainly focuses on how to implement data query in more details in a sensor network.

In [21, 119], operator placement for query processing is investigated. Srivastava et al. proposed an optimal algorithm to place a set of operators on a query tree. They considered both the computational cost of executing filters and the transmission cost of data tuples, and the proposed algorithm can find the best nodes to execute the operators so that the total cost is minimized.

In [21], Bonfils et al. presented a decentralized operator placement algorithm to adaptively adjust the positions of the operators to improve the performance. Their solution suites for the scenario where no global information is available and network conditions vary over the time. Our work considers a large storage space as a requirement for holding data and processing data for queries, while an operator/filter can be executed at any node. Additionally, placing storage nodes incurs extra hardware cost and there is only a limited number of storage nodes available in our problem formulation. Therefore, our problem is quite different from operator placement.

Data placement schemes in sensor networks are studied in [20, 78]. The authors consider a scenario where multiple observers are interested in some data sources. Data are disseminated by a multicast tree and may be cached to reduce the power consumption. Even though their scheme is close to data storage, they are mainly concerned with data replication, which is quite different from the scope of our work.

In addition, other research work has shown the feasibility of manufacturing storage nodes. In [96], Mathur et al. investigate hardware supports to attach large capacity flash memory to sensors. Their measurement study shows that access to large local storage is practical for sensors in term of energy cost. On the other hand, storage nodes are also supported by the research on software system. Zeinalipour-Yazti et al. propose Micra's [128] indexing structure to manage external flash memory of sensors in order to efficiently look up the stored data. In [95], Mathur et al. design Capsule system as a intermediate layer between flash memory and sensor applications. Object-based primitives are implemented to enable applications to flexibly utilize flash storage.

In [16, 17], the authors introduce an approach to analyzing communication networks based on stochastic geometry. They consider models built on Poisson processes and obtain formulas to express the average cost in function of the intensity parameters of Poisson processes. Baek et al.

extend this work specifically to sensor networks in [18]. They consider a hierarchical architecture with a *compressor* layer between sensor nodes and sinks. Data are aggregated at compressor nodes before further relay. One part of our work also analyzes random placement of storage nodes. We will use similar means with [16–18] to derive analytical formulas for the performance.

One part of storage placement problem with a general model is quite similar to the k-median problem ([14, 15, 31, 32, 35, 73]) and the facility location problem ([47, 61, 72, 84, 85, 114]). We design an approximation algorithm to solve it following the ideas in [32], which give an approximation factor of $6\frac{2}{3}$ to the k-median problem. In our problem, however, the sink is a special facility as the final destination of all data. From another aspect, our problem is similar to the two-level facility location problem ([3, 5, 27, 130]) with the sink as the only one level-2 facility. However, in our problem, the cost triangle inequality does not always hold, which makes the problem more complicated, as a special case of the non-metric two-level facility location problem. No prior work guarantees a constant approximation factor for the general non-metric two-level facility location problem. The best known solution has an approximation factor of $O(ln(C))$ ([130]), where C is the number of clients.

2.2 Problem Formulation

We consider an application in which sensor networks provide real-time data services to users. A sensor network is given with one special sensor identified as the sink (or base station) and many normal sensors, each of which generates (or collects) data from its environment. Users specify the data they need by submitting queries to the sink and they are usually interested in the latest

readings generated by the sensors[1]. To reply to queries, one typical solution, shown in Fig. 2.2, is to let the sink have all the data. Then any query can be satisfied directly by the sink. This requires each sensor to send its readings back to the sink immediately every time it generates new data. Generally, transferring all raw data could be very costly and is not always necessary. Alternatively, we allow sensors to hold their data and to be aware of the queries, then raw data can be processed to contain only the readings that users are interested in and the reduced-size reply, instead of the whole raw readings, can be transferred back to the sink. This scheme is illustrated in Fig. 2.3, where the black nodes, called *storage nodes*, are allowed to hold data. The sink diffuses queries to the storage nodes by broadcasting to the sensor network and these storage sensors reply to the queries by sending the processed data back. Compared with the previous solution, this approach reduces the raw data transfer cost (as indicated by the thick arrows in the figures), because some raw data transmissions are replaced by query reply (as indicated by the thin arrows). On the other hand, this scheme incurs an extra query diffusion cost (as indicated by the dashed arrows). In this work, we are interested in strategically designing a data access model to minimize energy costs associated with raw data transfers, query diffusion, and query replies.

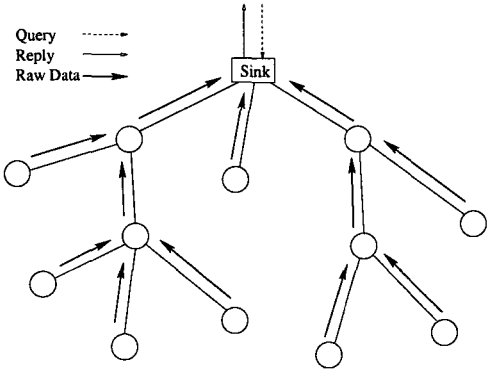


Figure 2.2: Data Access Model (All data are forwarded to the sink)

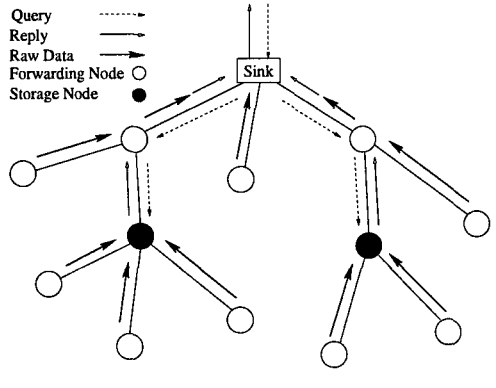


Figure 2.3: Data Access Model with Storage Nodes

[1]Our algorithms also apply to the queries to the historic data. For the ease of presentation, we assume all queries are corresponding to the latest generated data.

We now give formal definitions of two types of sensors (or nodes):

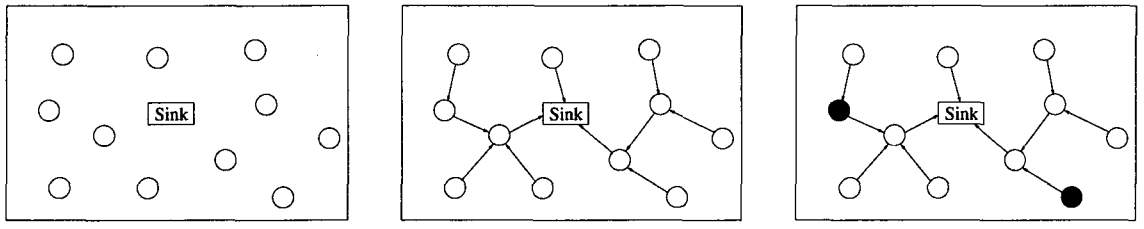
Storage nodes: This type of nodes store all the data it has received from other nodes or generated by themselves. They do not send out anything until queries arrive. According to the query description, they obtain the results needed from the raw data they are holding and then return the results back to the sink. The sink itself is considered as a storage node.

Forwarding nodes: This type of nodes always forward the data received from other nodes or generated by themselves along a path towards the sink. The outgoing data are kept intact and the forwarding operation continues until the data reach a storage node. The forwarding operation is independent of queries and there is no data processing at forwarding nodes.

Therefore, our goal is to design a centralized algorithm that can derive the best locations of the storage nodes to guide the deployment of such a hybrid sensor network. We make the following assumptions about the characteristics of data generation, query diffusion, and query replies. First, for data generation, we assume that each node generates r_d readings per time unit and the data size of each reading is s_d. Second, for query diffusion, we assume that r_q queries of the same type are submitted from users per time unit and the size of the query messages is s_q. Third, for query reply, we assume that the size of data needed to reply a query is a fraction α of that of the raw data. Specifically, we define a data reduction function f for query reply. For input x, which is the size of raw data generated by a set of nodes, function $f(x) = \alpha x$ for $\alpha \in (0, 1]$ returns the size of the processed data, which is needed to reply the query. We do not restrict the types of queries we impose on the sensor network in this work, but we assume that α can be obtained through examining the historic queries to get an empirical value for this parameter. The parameter α characterizes many queries satisfied by a certain fraction of all the sensing data, e.g., a range query may be "return all the nodes that sense a temperature higher than 100 degree" and α can be

estimated based on the data distribution information.

The communication among all n nodes is based on a tree topology with the sink as the root. Data are transferred along the edges in this communication tree. The communication tree can be formed before or after storage node deployment. Accordingly, we will consider two models in the rest of this chapter. In the fixed tree model, as illustrated in Fig. 2.4, we first deploy regular sensors and construct a communication tree as usual. Based on the topology information, we select some of the regular sensors to be storage nodes. We can attach large flash memory to these selected sensors or replace them by more powerful storage nodes at the same locations. The other model, the dynamic tree model, is illustrated in Fig. 2.5. In this model, storage nodes are deployed before the communication tree is formed and their location information is broadcast to nearby regular sensors. After that, all sensors organize themselves into a communication tree according to the locations of the storage nodes. In both models, after tree construction and storage node deployment, each storage node needs to send a notification towards the sink. In this way, every sensor is aware of the existence of storage nodes among its descendants and when a query arrives, it is able to determine whether to continue the diffusion or not. In both the fixed tree and dynamic tree models, we aim to find the optimal locations for storage nodes in a deterministic way. In reality, however, the storage nodes may not be deployed in a precise way. Instead, their deployment may be random with a certain density λ, e.g., the storage nodes are dispersed from an airplane. We also evaluate the performance of random deployment of storage nodes in this chapter.

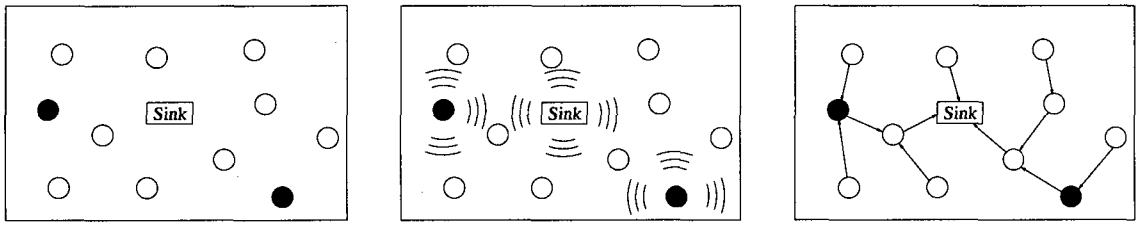


(a) Step 1: Regular sensors are deployed in a field.

(b) Step 2: A communication tree is constructed to relay data.

(c) Step 3: Some regular sensors are upgraded to or replaced by storage nodes.

Figure 2.4: Deploy Storage Nodes in the Fixed Tree Model (storage nodes are black)



(a) Step 1: Regular sensors and storage nodes are deployed in a field.

(b) Step 2: Each storage node (including the sink) broadcasts its location information.

(c) Step 3: A communication tree is constructed based on the locations of storage nodes.

Figure 2.5: Deploy Storage Nodes in the Dynamic Tree Model (storage nodes are black)

2.3 Methodology

2.3.1 Fixed Tree Model

We first introduce the communication model in the fixed tree model as follows. To transmit one data units, the energy costs of the sender and receiver are e_{tr} and e_{re} respectively, and e_{tr} is also relevant to the distance between the sender and receiver. To simplify the problem, we set the length of each tree edge to one unit, which means that sensor nodes have a fixed transmission range and the energy cost of transferring data is only proportional to the data size. Our algorithms can be easily extended to non-uniform transmission ranges as long as topology information is available.

In our energy model, for simplicity of presentation, the receiving energy cost is assigned to the sender without changing the total energy cost. When sensor i sends one data unit to j, the energy cost of j is 0, and the energy consumed by i is

$$\begin{cases} e_{tr} + e_{re} & \text{if } j \text{ is } i\text{'s parent;} \\ e_{tr} + e_{re} \cdot c_i & \text{if } j \text{ is one of } i\text{'s children,} \end{cases}$$

where c_i is the number of i's children. In the following discussion, we normalize the energy costs by $(e_{tr} + e_{re})$ for easy presentation. Thus, transferring one data unit from i to its parent consumes one energy unit and to broadcast one data unit to its children, sensor i will consume b_i energy units, where

$$b_i = \frac{e_{tr} + e_{re} \cdot c_i}{e_{tr} + e_{re}}.$$

Let i be any node in the communication tree and T_i be the subtree rooted at i. We use $|T_i|$ to denote the number of nodes in T_i. We define $e(i)$ to be the energy cost incurred at i per time unit, which includes, the cost for raw data transfer from i to its parent if i is a forwarding node, the cost for query diffusion if i has storage nodes as its descendants, and cost for query reply if i is a storage node or has a storage descendant. To define $e(i)$ mathematically we need to consider several possible cases.

Case A. i is a forwarding node and there are no storage nodes in T_i. All raw data generated by the nodes in T_i have to be forwarded to the parent of i and there is no query diffusion cost. So $e(i) = |T_i| r_d s_d$.

Case B. i is a storage node and there are no other storage nodes in T_i. The latest readings of all raw data generated by the nodes in T_i are processed at node i and the reduced reply size will be $\alpha |T_i| s_d$. Node i sends the reply to its parent when queries arrive. So $e(i) = r_q \alpha |T_i| s_d$.

Case C. i is a storage node and there is at least one other storage node in T_i. In addition

to the cost for query reply as defined in Case B, i also incurs a cost for query diffusion that is implemented by broadcasting to its children. So $e(i) = r_q\alpha|T_i|s_d + b_i r_q s_q$.

Case D. i is a forwarding node and there is at least one storage node in T_i. This is the case where all three types of cost (for raw data transfer, query diffusion, and query reply) are present. Among the $|T_i| - 1$ descendants of i, let d_1 be the number of forwarding descendants without any storage nodes on their paths to i (the raw data generated at these d_1 nodes and at i itself will be forwarded from i to its parent without reduction) and d_2 be the number of storage descendant's or forwarding descendants with at least one storage node on their paths to i (the last readings of the raw data generated at these d_2 nodes will have been processed and reduced before reaching i). Obviously, $d_1 + d_2 = |T_i| - 1$. So $e(i) = (d_1 + 1)r_d s_d + b_i r_q s_q + r_q \alpha d_2 s_d$.

Within the fixed tree model, we will consider two problems of storage node placement. Given an undirected tree T with nodes labeled with $1, 2, \ldots, n$. The length of each edge is 1. Let $e(i)$ be the energy cost of node i in one time unit as defined above. The objective is to place storage sensors (and hence forwarding sensors) on nodes in T such that the total energy cost $\sum_{i \in T} e(i)$ is minimized. In the case when there is no limit on the number of storage nodes that can be used to minimize the energy cost, the problem is denoted with UNLIMITED. In the case when there is a limited number of storage nodes, say k, to use, the problem is denoted with LIMITED.

2.3.1.1 Unlimited Number of Storage Nodes

We will present a linear-time algorithm for the problem UNLIMITED, where an unlimited number of storage nodes are available to use to minimize the energy cost of a communication tree. Recall that $e(i)$ is the energy cost at node i. Let T_i be the subtree rooted at i. Then $E(i)$ is the energy cost of nodes in T_i, defined to be $E(i) = \sum_{i \in T_i} e(i)$.

Our algorithm relies on the following lemma.

Lemma 2.1 *Given a node i and its subtree T_i. If $\alpha r_q \geq r_d$, then i must be a forwarding node to minimize $E(i)$. If $\alpha r_q < r_d$, then i must be a storage node to minimize $E(i)$.*

Proof: We compare the energy cost of two trees, which are identical in every aspect except that the first tree's root is a forwarding node and the second tree's root is a storage node. Let E_1 and E_2 be the energy cost of these two trees, respectively. Comparing the energy cost of individual nodes, one by one, in the two trees, we observe that any two non-root nodes in the same position of the trees must have the same energy cost. The only difference is the energy cost of the roots. Let e_1 and e_2 be the energy cost of the roots in the two trees, respectively. Therefore, $E_1 - E_2 = e_1 - e_2$. To prove the lemma, it suffices to prove that

$$e_1 - e_2 \begin{cases} \leq 0 & \text{if } \alpha r_q \geq r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases}$$

We consider two cases. First, if both roots have no storage descendants, then according to the four-case definition of energy cost given in the previous section (Cases A and B, specifically), we have

$$\begin{aligned} e_1 - e_2 &= |T_i| r_d s_d - r_q \alpha |T_i| s_d \\ &= |T_i| s_d (r_d - \alpha r_q) \begin{cases} \leq 0 & \text{if } \alpha r_q \geq r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases} \end{aligned}$$

Second, if both roots have at least one storage descendent, then according to the four-case defini-

tion of energy cost given in the previous section (Cases D and C, specifically), we have

$$\begin{aligned} e_1 - e_2 &= ((d_1+1)r_d s_d + b_i r_q s_q + r_q \alpha d_2 s_d) \\ &\quad -(r_q \alpha |T_i| s_d + b_i r_q s_q) \\ &= (d_1+1)s_d(r_d - \alpha r_q) \begin{cases} \leq 0 & \text{if } \alpha r_q \geq r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases} \end{aligned}$$

In the first tree with a forwarding root, recall that d_1 is the number of forwarding descendants of the root without any storage nodes on their paths to the root and that d_2 is the number of storage descendants plus the number of forwarding descendants with at least one storage node on their paths to the root. Also recall that $d_1 + d_2 = |T_i| - 1$. ■

From the above lemma, we can conclude that if $\alpha r_q \geq r_d$ then every node (except for the root/sink, which is always a storage node) in the sensor network must be a forwarding node to minimize the energy cost. However, if $\alpha r_q < r_d$, things are a little tricky. Although the root of the tree, say i, must be a storage node, it may not be true that every node in the sensor network must be a storage node to minimize the energy cost. One would think that in order for the tree to incur a minimum energy cost, all of its subtrees should incur a minimum energy cost. However, since $\alpha r_q < r_d$, these optimal subtrees all have storage nodes as their roots. This means that the energy cost of root i will have to include the cost for query diffusion $b_i r_q s_q$ since it has storage children i.e., $e(i) = r_q \alpha |T_i| s_d + b_i r_q s_q$. The cost for query diffusion, however, can be eliminated if all subtrees of i has only forwarding nodes, i.e., $e(i) = r_q \alpha |T_i| s_d$. (See Cases C and B in the four-case definition of $e(i)$ in the previous section.) Thus, the minimum energy cost of the tree rooted at i should be derived from the better of these two scenarios.

For a tree T_i rooted at i, let C_i be the set of children of i. Let $E^*(i)$ be the minimum (optimal) energy cost of T_i. If C_i is empty, i.e., i is a leaf, then i must be a storage node to achieve its

minimum energy cost. So $E^*(i) = r_q \alpha s_d$. If C_i is not empty, then for any $j \in C_i$, let $E_f(j)$ be the energy cost of T_j when all nodes in T_j are forwarding nodes. So

$$E^*(i) = \min \{ \ r_q \alpha |T_i| s_d + b_i r_q s_q + \sum_{j \in C_i} E^*(j), \ \ r_q \alpha |T_i| s_d + \sum_{j \in C_i} E_f(j) \ \}.$$

Algorithm 1 given in pseudo-code finds the optimal placement of storage nodes in two cases: (1) $\alpha r_q \geq r_d$, (2) $\alpha r_q < r_d$, where the first case is trivial and the second case is solved by dynamic programming that works from the bottom to the top of the tree. We now explain how the dynamic programming algorithm for the second case is set up. Assume that the n nodes in the tree T are labeled using the post-order[2]. A table $E^*[1..n]$ is used to hold the minimum energy cost of all subtrees rooted at node $i = 1, \ldots, n$. So at the end of the computation, $E^*[n]$ will hold the minimum energy cost of T (which is rooted at n according to the post-order labeling). We also maintain a second table $E_f[1..n]$ which records the energy cost of all subtrees when all nodes in each subtree are forwarding nodes. In the algorithm, lines 5-9 compute the E^* and E_f entries for all leaves and lines 10-19 compute the E^* and E_f entries for the remaining nodes following our post-order.

[2]The post-order used in this work is slightly different from the textbook definition of post-order in that our post-order requires all leaves to be listed first.

Algorithm 1 Place Unlimited Storage Nodes

```
make the root a storage node
if αr_q ≥ r_d then
    make all non-root nodes forwarding nodes and return
end if
for all leaves i do
    make i a storage node
    E*[i] = r_q α s_d
    E_f[i] = r_d s_d
end for
for all remaining nodes i, in post-order, do
    make i a storage node
    cost1 = r_q α |T_i| s_d + b_i r_q s_q + Σ_{j∈C_i} E*[j]
    cost2 = r_q α |T_i| s_d + Σ_{j∈C_i} E_f[j]
    E*[i] = min{cost1, cost2}
    E_f[i] = |T_i| r_d s_d + Σ_{j∈C_i} E_f[j]
    if cost1 ≥ cost2 then
        change each descendent of i that is a storage node to a forwarding node
    end if
end for
```

There are only $O(n)$ entries to compute in tables E^* and E_f and to compute each entry that corresponds to a node, only its children will have to be considered. Furthermore, each node starts as a storage node. Once it is changed to a forwarding node by the algorithm, it will never be changed back. Therefore, the time complexity of Algorithm 1 is $O(n)$, where n is the number of nodes.

Summarizing the discussion above, we have the following theorem.

Theorem 2.1 *If $\alpha r_q \geq r_d$, then the optimal tree with the minimum energy cost contains only forwarding nodes (except for the root). If $\alpha r_q < r_d$, then the optimal tree can be constructed by a dynamic programming algorithm in $O(n)$ time.*

From the design of the algorithm, we also observe that every node starts as a storage node and that once it is changed to a forwarding node, all of its descendants are changed to forwarding nodes as well. Thus, it is impossible for a forwarding node to have a storage descendent. Likewise, it is impossible for a storage node to have a forwarding ancestor. We then have the following corollary.

Corollary 1 In the optimal tree, if i is a forwarding node, all of its descendants are forwarding nodes as well. If i is a storage node, all its ancestors are storage nodes as well.

In summary, this UNLIMITED problem refers to the scenario that the deployment budget is sufficient to upgrade every sensor to be a storage node. However, simply making all sensors storage nodes may not be the best strategy. The appropriate deployment still depends on the characteristics of query and data generation. Intuitively, if there are a large volume of queries for a certain set of data and the reduction function yields a large α, it would be better to transfer these data to the sink. On the other hand, if queries are infrequent and the reply size is much less than the raw data, it would be more efficient to hold the raw data locally. According to Theorem 2.1

and Corollary 1, the optimal deployment of storage nodes has a special property. For any path from a leave to the root, there is a clear boundary that distinguish forwarding nodes from storage nodes. The nodes below the boundary layer to leaves are forwarding nodes and the nodes above the boundary towards the sink are storage nodes.

2.3.1.2 Limited Number of Storage Nodes

In the problem UNLIMITED discussed in the previous section, we assume that we have enough storage nodes for the need to minimize the energy cost of the network. In reality, however, storage nodes may come with a hardware cost. Considering a limited budget for deploying a sensor network, there might be only a small portion of sensors as storage nodes. This is why we have also defined the problem LIMITED, which is similar to UNLIMITED except that we have only k storage nodes to deploy. Since the root (sink) is always a storage node, we assume that $k \geq 1$ and that $k-1$ is the maximum number of storage nodes that may appear as descendants of the root. Furthermore, from the discussion in the previous section, if $\alpha r_q \geq r_d$, the optimal tree has no storage nodes at all except the root. In this case, we just do not deploy any of the $k-1$ storage nodes and we get an optimal tree. Our discussion in this section on LIMITED is for the case of $\alpha r_q < r_d$. Since the number of storage nodes is limited, where to place them becomes a crucial problem. A bad placement strategy may hardly improve the performance. Basically, there is a tradeoff between two trends. On the one hand, if storage nodes are close to the sink, i.e., at a high level in the tree structure, they can process more raw data, thus reduce the reply size from storage nodes to the sink. However, the sensor network spends much energy in transferring the raw data from low level forwarding nodes to the storage nodes. On the other hand, if the storage nodes are far away from the sink, the raw data from their descendants can be processed earlier along

the path towards the sink. However, storage nodes may cover only a few regular sensors, which leads to much raw data transferred to the sink without being processed. Besides this tradeoff, the benefits from a storage node also depend on the locations of other storage nodes. Therefore, in this section, we propose the optimal placement strategy in order to maximize the benefits from deploying k storage nodes.

Assume that a communication tree T is given with up to k storage nodes already optimally deployed. By definition, the energy cost of T is $\sum_{i \in T} e(i)$. However, we are going to use a different and unique method to calculate this cost, which works from the bottom of the tree towards the root. Starting from the leaf nodes and following the post-order until the root is eventually reached, for each node i, we compute the energy cost already incurred within the subtree T_i rooted at i, which is $E(i)$ by our notation, plus the energy cost contributed by the nodes in T_i to their ancestors, which includes both raw data transmission cost and query reply cost according to the four-case definition of the energy cost of an individual node. Specifically, if i is a forwarding node, it contributes a raw data transmission cost of $r_d s_d$ to each of its forwarding ancestors that lie between i and i's closest storage ancestor (due to Cases A and D) and a query reply cost of $r_q \alpha s_d$ to each of the other ancestors (due to Cases B and D). If i is a storage node, however, it contributes a query reply cost of $r_q \alpha s_d$ to each of its ancestors (due to Cases C and D). Fig. 3 depicts the two scenarios. The top path from node i to the root (sink) is when i is a forwarding node and the bottom path from i to the root is when i is a storage node. Above each node (except i) is the contribution from i to the energy cost incurred at the node.

Let l be the number of forwarding nodes between i and its closest storage ancestor, not including i. Let m be the upper bound on the number of storage nodes in T_i. Then, we use $E_i(m, l)$ for the energy cost that includes $E(i)$ and the amount contributed by the nodes in T_i to the energy cost

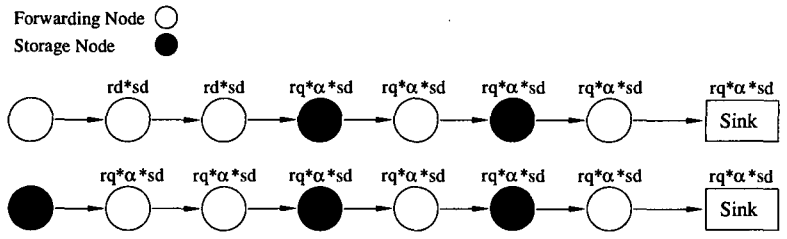


Figure 2.6: Computing the contribution to the energy cost of all ancestors

of their ancestors. Note that $0 \leq m \leq k$ and $0 \leq l \leq n-2$. In the case that i is a storage node or i's parent is a storage node, l becomes 0. Furthermore, if $m = 0$, no storage node is used in T_i and if $m \geq 1$, at least one but no more than m storage nodes are used in T_i. Therefore, $E_n(k,0)$ is the minimum energy cost of T with up to k storage nodes to deploy, assuming n is the label for the root.

When traversing the nodes in post-order in the tree starting from the leaves, let i be the current node being traversed. Let d_i be the depth of i in the tree, which is the number of edges on the path from i to the root n. We can define $E_i(m,l)$ recursively. For notational simplicity, we first define $Q_0(m)$ and $Q_1(m)$ as follows.

$$
\begin{aligned}
Q_0^i(m) &= \begin{cases} 0 & \text{if } m = 0; \\ b_i r_q s_q & \text{if } m \geq 1. \end{cases} \\
Q_1^i(m) &= \begin{cases} 0 & \text{if } m = 1; \\ b_i r_q s_q & \text{if } m \geq 2. \end{cases}
\end{aligned}
$$

If i is a leaf node, $E_i(m,l)$ includes the energy cost of i and the pre-calculated amount contributed by i to all of its d_i ancestors. Specifically, if i is a forwarding node, its own energy cost is $r_d s_d$ and its contribution to the energy cost of its ancestors is $l r_d s_d + (d_i - l) r_q \alpha s_d$. If i is a storage node, its own energy cost is $r_q \alpha s_d$ and its contribution to the energy cost of its ancestors

is $d_i r_q \alpha s_d$. Therefore,

$$E_i(m,l) = \begin{cases} (l+1)r_d s_d + (d_i - l)r_q \alpha s_d & \text{if } m = 0; \\ (d_i + 1)r_q \alpha s_d & \text{if } m \geq 1. \end{cases}$$

If i is a forwarding non-leaf node with a child set of C_i, the upper bound m must be divided among all of its children. Let $P(m)$ be the set of all permutations $p = (m_j^p | j \in C_i)$, where $\sum_{j \in C_i} m_j^p = m$ and m_j^p denotes the upper bound on the number of storage nodes for subtree T_j in permutation p. Then $E_i(m,l)$ is defined to be the sum of the amount from all of its subtrees,

$$\min_{\forall p \in P(m)} \{ \sum_{j \in C_i} E_j(m_j^p, l+1) \},$$

the energy cost of i, $r_d s_d + Q_0^i(m)$, and the pre-calculated amount of energy cost contributed by i to its ancestors,

$$l r_d s_d + (d_i - l) r_q \alpha s_d.$$

So,

$$E_i(m,l) = \min_{\forall p \in P(m)} \{ \sum_{j \in C_i} E_j(m_j^p, l+1) \} + (l+1) r_d s_d + (d_i - l) r_q \alpha s_d + Q_0^i(m).$$

If i is a storage non-leaf and non-root node, the upper bound $m-1$ must be divided among all of its children. Let $P(m-1)$ be the set of all permutations $p = (m_j^p | j \in C_i)$, where $\sum_{j \in C_i} m_j^p = m-1$ and m_j^p denotes the upper bound on the number of storage nodes for subtree T_j in permutation p. Then $E_i(m,l)$ is defined to be the sum of the amount from all of its subtrees,

$$\min_{\forall p \in P(m-1)} \{ \sum_{j \in C_i} E_j(m_j^p, 0 \},$$

the energy cost of i, $r_q \alpha s_d + Q_1^i(m)$, and the pre-calculated amount of energy cost contributed by

i to its ancestors, $d_i r_q \alpha s_d$. So,

$$E_i(m,l) = \min_{\forall p \in P(m-1)} \{ \sum_{j \in C_i} E_j(m_j^p, 0) \} + (d_i + 1) r_q \alpha s_d + Q_1^i(m).$$

Algorithm 2 given here maintains a two-dimensional $(k+1) \times (n-1)$ table, $E_i[m,l]$, at each node i, where $0 \leq m \leq k$ and $0 \leq l \leq n-2$. Assume that a post-order traversal is done beforehand and that the depth of each node is computed beforehand. Both the post-order and the depths can be obtained in $O(n)$ time. In the algorithm, lines 1-12 computes the E_i tables for all leaves i, lines 13-21 compute the E_i tables for the remaining non-root nodes i, and line 22-23 compute the entry $E_n[k,0]$ for the root n. After all tables are constructed, the minimum energy cost of the tree with up to k storage nodes can be found in the entry $E_n[k,0]$. Note that instead of constructing a table for the storage root n, we compute only the needed entry for n.

Algorithm 2 Place Limited Storage Nodes

```
for all leaves i do
    for m = 0 to k do
        for l = 0 to n − 2 do
            if m = 0 then
                E_i[m, l] = (l + 1) r_d s_d + (d_i − l) r_q α s_d
            end if
            if m ≥ 1 then
                E_i[m, l] = (d_i + 1) r_q α s_d
            end if
        end for
    end for
end for
for all remaining non-root nodes i, in post-order, do
    for m = 0 to k do
        for l = 0 to n − 2 do
            min1 = min_{∀p∈P(m)} {Σ_{j∈C_i} E_j[m_j^p, l + 1]} + (l + 1) r_d s_d + (d_i − l) r_q α s_d + Q_0^i(m)
            min2 = min_{∀p∈P(m−1)} {Σ_{j∈C_i} E_j[m_j^p, 0]} + (d_i + 1) r_q α s_d + Q_1^i(m)
            E_j[m, l] = min{min1, min2}
        end for
    end for
end for
E_n[k, 0] = min_{∀p∈P(k−1)} {Σ_{j∈C_n} E_j[m_j^p, 0]} + r_q α s_d + Q_1^i(m)
return E_n[k, 0]
```

Assume that every node in the tree has at most c children. To partition an upper bound m into up to c upper bounds with the sum equal to m, there are at most $\binom{m+c-1}{m} = \binom{m+c-1}{c-1} \leq \binom{k+c-1}{c-1}$ permutations. The algorithm constructs $O(n)$ tables and each table consists of $O(kn)$ entries. To compute each entry, the time is

$$O(\binom{k+c-1}{c-1}c) = O((k+c-1)^{c-1}c/(c-1)!) = O((\max\{k,c\})^{c-1}).$$

Thus, the time complexity of the algorithm is $O(kn^2(\max\{k,c\})^{c-1})$.

We summarize the discussion above in the following theorem.

Theorem 2.2 *Given a communication tree with n nodes and at most c children for each parent. Let k be the maximum number of storage nodes that may be deployed in the tree. Then the optimal tree with the minimum energy cost can be constructed by a dynamic programming algorithm in* $O(kn^2(\max\{k,c\})^{c-1})$ *time.*

In the next, we consider a special case of LIMITED, where the given network is a regular tree with exactly c children for each non-leaf node and all leaves at the same level. For such a c-ary regular tree, we can modify Algorithm 2 to achieve a faster time complexity by making use of the regularity of the tree structure.

Obviously, any subtree in a regular tree is also a regular tree and nodes at the same level have the subtrees with the same topology. This suggests that instead of keeping a table for each node as in Algorithm 2, we may keep just one table for each level. For easy discussion, we name the levels from bottom to top, with all leaves at level 0, all parents of the leaves at level 1, and finally the root at level $\lfloor \log_c n \rfloor$. For each level h, we define a two-dimensional table $E_h[m,l]$ for $0 \leq m \leq k$ and $0 \leq l \leq \lfloor \log_c n \rfloor - 1$, which returns the energy cost incurred within the subtree rooted at level h plus the contribution from the nodes in the subtree to their ancestors. As used previously, m is

still the maximum number of storage nodes to use in the subtree and l is the number of forwarding nodes between the root of the subtree and the storage ancestor closest to the root of the subtree.

We first define $Q_0(m)$ and $Q_1(m)$ as follows.

$$Q_0(m) = \begin{cases} 0 & \text{if } m = 0; \\ \frac{e_{tr}+c\cdot e_{re}}{e_{tr}+e_{re}} r_q s_q & \text{if } m \geq 1. \end{cases}$$

$$Q_1(m) = \begin{cases} 0 & \text{if } m = 1; \\ \frac{e_{tr}+c\cdot e_{re}}{e_{tr}+e_{re}} r_q s_q & \text{if } m \geq 2. \end{cases}$$

Let $H = \lfloor \log_c n \rfloor$. Algorithm 3 first computes the table $E_0[m,l]$ for all leaves at level 0, for $0 \leq m \leq k$ and $0 \leq l \leq H-1$ in lines 2-11. Then it works its way up, level by level, until level $H-1$ in lines 12-20. The root, which is at level H, is treated in lines 21-22, differently from the other nodes since it must be a storage node. By using one table for each level, our algorithm will construct $\lfloor \log_c n \rfloor$ tables. This will result in savings in both space and time, compared with our algorithm for arbitrary trees, which needs to construct n tables. The modified Algorithm 3 for regular trees has a time complexity of $O(k(\log n)^2(\max\{k,c\})^{c-1})$.

The result of the algorithm can be summarized in the following theorem.

Theorem 2.3 *Given a c-ary regular tree with n nodes. Let k be the maximum number of storage nodes that may be deployed in the tree. Then the optimal tree with the minimum energy cost can be constructed by a dynamic programming algorithm in* $O(k(\log n)^2(\max\{k,c\})^{c-1})$ *time.*

Algorithm 3 c-ary Regular Tree

```
H = ⌊log_c n⌋
for m = 0 to k do
    for l = 0 to H − 1 do
        if m = 0 then
            E_0[m, l] = (l + 1) r_d s_d + (H − l) r_q α s_d
        end if
        if m ≥ 1 then
            E_0[m, l] = (H + 1) r_q α s_d
        end if
    end for
end for
for h = 1 to H − 1 do
    for m = 0 to k do
        for l = 0 to H − 1 do
            min1 = min_{∀p∈P(m)} {Σ_{j=1}^{c} E_h[m_j^p, l + 1]} + (l + 1) r_d s_d + (H − h − l) r_q α s_d + Q_0(m)
            min2 = min_{∀p∈P(m−1)} {Σ_{j=1}^{c} E_h[m_j^p, 0]} + (H − h + 1) r_q α s_d + Q_1(m)
            E_h[m, l] = min{min1, min2}
        end for
    end for
end for
E_H[k, 0] = min_{∀p∈P(k−1)} {Σ_{j=1}^{c} E_{H−1}[m_j^p, 0]} + r_q α s_d + Q_1(m)
return E_H[k, 0]
```

2.3.2 Dynamic Tree Model

For the dynamic tree model, the storage node placement problem becomes how to place k given storage nodes to form a communication tree with the minimum total energy cost. In the deployment, we first deploy normal forwarding nodes. After collecting their location information, we select at most k of them to be storage nodes. We can attach large flash memory to these selected forwarding nodes or replace them by deploying more powerful storage nodes at the same locations. We also associate each forwarding node with a storage node which will hold the raw data from the forwarding node. We broadcast the association information to the network in the initial phase. This problem is NP-hard since it is a general case of the minimum k-median problem. We present a 10-approximation algorithm for the dynamic tree model in this subsection.

We consider multi-hop communication for relaying data. We assume the data routing between a pair of sensors, e.g., a forwarding node and a storage node, or a storage node and the sink, follows the geographic routing algorithm [76], which looks for the shortest path connecting them. Thus, the energy cost model is simplified by the assumption that the transmission cost is proportional to the data size and the hop distance between the sender and the receiver. In a densely deployed sensor network, the hop distance between two sensors is proportional to the Euclidean distance ([38, 46, 122]). Therefore, in this work, we use

$$\text{Euclidean distance} \times \text{Data size}$$

to measure the energy consumed to send data.

Therefore, the problem in this work is to find the optimal placement of the storage nodes such that the energy cost associated with raw data transfer and query reply is minimized. This problem

is a general case of the k-median problem[3]. Especially when there is no data transfer between storage nodes and the sink, i.e. $r_q = 0$, the problem becomes the classic k-median problem, which has been proven to be NP-hard. In the following, we give an approximate algorithm for our optimal storage node placement problem.

More specifically, given L as a set of locations of sensor nodes including the sink, the problem is to select at most k sensors to be storage nodes such that the total energy cost is minimized. Assume different nodes are placed at distinct locations, L can be also regarded as the set of sensor nodes. All nodes/locations are labeled from 0 to n and node 0 is the sink. We define y_i as the type flag of node i,

$$\forall i \in L, y_i = \begin{cases} 1 & \text{if } i \text{ is a storage node;} \\ 0 & \text{if } i \text{ is a forwarding node.} \end{cases}$$

Let c_{ij} be the Euclidean distance[4] between node i and j and l_i be the Euclidean distance between node i and the sink, i.e. $l_i = c_{i0}$. We use x_{ij} as an indicator denoting if the raw data generated by node j are sent to storage node i and stored there,

$$x_{ij} = \begin{cases} 1 & \text{if } y_i = 1 \text{ and node } j \text{ forwards its raw data to } i; \\ 0 & \text{otherwise.} \end{cases}$$

[3]Definition of k-median problem ([32]): Given n points, we must select k of them to be cluster centers, and then assign each point j to the selected center that is closest to it. The goal is to minimize the sum of the distance between each node and its associated center.

[4]We use the Euclidean distance to approximate the minimal number of communication hops between two nodes, which translates to the total optimal power consumption of the nodes on the communication path between those two nodes. This approximation is valid when a large number of nodes are deployed ([38, 46, 122]).

Thus, our problem can be formulated as an integer program,

$$
\begin{aligned}
\text{IP:} \quad \min \quad & \sum_{i,j \in L} x_{ij}(c_1 c_{ij} + c_2 l_i) \\
s.t. \quad & \forall j \in L, \sum_{i \in L} x_{ij} = 1, && (2.1) \\
& \sum_{i \in L} y_i \leq k, && (2.2) \\
& \forall i, j \in L, y_i \geq x_{ij} \geq 0, && (2.3) \\
& \forall i, j \in L, x_{ij} = \{0, 1\}, \\
& \forall i \in L, y_i = \{0, 1\}, y_0 = 1.
\end{aligned}
$$

where $c_1 = r_d s_d$ and $c_2 = r_q \alpha s_d$. In the objective, the cost incurred by a node j includes two parts. The first part ($c_1 c_{ij}$) is the cost for raw data transfer from node j to the associated storage node i. The second part ($c_2 l_i$) is the cost of sending the query reply, which is derived from the raw data generated by j, from the storage node i to the sink. The first constraint requires every sensor to send its data through a storage node. Since we treat the sink as a storage node, it includes the case that sensors send data directly to the sink. The second constraint is for the number of storage nodes, where k is given as a parameter of this problem. In the third constraint, if node j forwards data to node i, node i must be a storage node. It shows the connection between variables x and y.

Since c_1 and c_2 are constants, the objective function is equivalent to

$$\min \sum_{i,j \in L} p_{ij} x_{ij},$$

where $p_{ij} = c_{ij} + \beta l_i$ with $\beta = \frac{c_2}{c_1} = \frac{r_q \alpha}{r_d}$. In the rest of this section, we will use the above objective

function for the IP problem. Its LP-relaxation is

$$
\begin{aligned}
\text{LP:} \quad \min \quad & \sum_{i,j \in L} p_{ij} x_{ij} \\
s.t. \quad & \forall j \in L, \sum_{i \in L} x_{ij} = 1, \\
& \sum_{i \in L} y_i \leq k, \\
& \forall i, j \in L, y_i \geq x_{ij} \geq 0, \\
& \forall i, j \in L, x_{ij} \in [0,1], \\
& \forall i \in L, y_i \in [0,1], y_0 = 1.
\end{aligned}
$$

Note that the difference between this LP and the k-median problem is that p_{ij} is neither symmetric nor proportional to the Euclidean distance between i and j, i.e., $p_{ij} \neq p_{ji}$ and $c_{ij} > c_{uv}$ does not imply $p_{ij} > p_{uv}$.

Theorem 2.4 *If $\beta \geq 1$, there is no need to place storage nodes.*

Proof: Assume node i is a storage node, and a node j (j may be equal to i) sends data via node i. Recall that the cost incurred by node j is $p_{ij} = c_{ij} + \beta l_i$. If j sends data directly to the sink, the cost will be l_j. According to the triangle inequality

$$l_j \leq c_{ij} + l_i \leq c_{ij} + \beta l_i = p_{ij}.$$

It shows that when $\beta \geq 1$, there is no benefit from transmitting data through a storage node. Thus, there is no need to deploy storage nodes. ■

In the rest of this section, we only consider the scenario with $\beta < 1$.

2.3.2.1 Outline of the Algorithm

In the next, we propose an approximation algorithm to resolve the IP problem. We first express the LP problem in a different but equivalent form by introducing a demand d_j to every node. Intuitively, d_j can be regarded as the amount of the raw data generated by node j. We set d_j set to 1 for any node j and keep the same constraints of the LP problem. But the objective function is rewritten as

$$\text{LP: min} \sum_{i,j \in L} d_j p_{ij} x_{ij}.$$

Initially, we obtain an optimal solution $(\bar{x}, \bar{y})$ to the LP problem. For any node $j \in L$, we use $\bar{C}_j$ to represent the cost of raw data transfer and query reply incurred by a data unit from node j in solution $(\bar{x}, \bar{y})$:

$$\bar{C}_j = \sum_{i \in L} p_{ij} \bar{x}_{ij}. \tag{2.4}$$

Let C_{LP} be the value of the objective of the LP problem, which represents the total cost.

$$C_{LP}(\bar{x}, \bar{y}) = \sum_{j \in L} d_j \bar{C}_j. \tag{2.5}$$

Based on $(\bar{x}, \bar{y})$, we use the following three steps to obtain an approximate solution to the IP problem. Here we only summarize the basic intuitions. More details will be presented in the next section.

Step 1: We modify the demand of every node by moving the demands of some nodes to the others. We call this process *consolidating demands.* After this step, only some nodes hold positive demands while the other nodes' demands become 0. We call the problem with new demand values the *new demand problem.* Since we keep the same constraints in this step, $(\bar{x}, \bar{y})$ is also feasible to the new demand problem. In addition, our modification follows some rules such that an integer

solution to the new demand problem can be converted to an integer solution to the LP problem with no more than $4C_{LP}(\bar{x},\bar{y})$ extra cost.

Step 2: In solution $(\bar{x},\bar{y})$, the values of the variables are not necessarily integers. We call node i a *fractional storage node* if $\bar{y}_i \in (0,1)$. In this step, we simplify the problem by *consolidating fractional storage nodes*, i.e., moving $\bar{y}_i$ of fractional storage nodes to other nodes. We modify $(\bar{x},\bar{y})$ to another solution (x',y'), such that non-zero values in x',y' reside in $[\frac{1}{2},1]$ and the cost of (x',y') is at most three times of the cost of $(\bar{x},\bar{y})$. We then further modify (x',y') to another $\{\frac{1}{2},1\}$-integral solution (x'',y'') (non-zero values in x'',y'' are either $\frac{1}{2}$ or 1) to the new demand problem, which yields no more cost than (x',y').

Step 3: Finally, we apply a rounding algorithm to convert (x'',y'') to a $\{0,1\}$-integral solution to the new demand problem with at most twice the cost of (x'',y''). As we mentioned in Step 1, this integer solution can be further converted to an integer solution to the LP problem with a bounded cost. After all, we obtain an approximate solution to the IP problem.

2.3.2.2 Details of the Algorithm

Step 1: Consolidating Demands Originally, every node has demand of 1. In this step, we try to reallocate demands from all nodes to fewer number of nodes such that for any pair of nodes i and j with positive demands after this reallocation, their Euclidean distance $c_{ij} > 4\max\{\bar{C}_i,\bar{C}_j\}$. The following procedure is applied to consolidate the demands.

1. We re-index the nodes in an increasing order of $\bar{C}_j$, i.e., $\bar{C}_1 \leq \bar{C}_2 \leq \ldots \leq \bar{C}_n$.

2. We modify the demands of nodes in the new order. Let d'_j be the new demands. Initially, $d'_j = d_j$. For a node j, we check if there is another node i satisfying $i < j$, $d_i > 0$ and $c_{ij} \leq 4\max\{\bar{C}_i,\bar{C}_j\} = 4\bar{C}_j$. If there exists such a node i, we move the demand of j to node

i, i.e.,

$$\begin{aligned} d'_i &\leftarrow d'_i + d'_j; \\ d'_j &\leftarrow 0. \end{aligned}$$

After this process, a node still with a positive demand is called a *demand node.* Now we get a new problem, called *new demand problem*, which has the same constraints as the LP problem, but the objective becomes:

$$\text{New Demand: } \min \sum_{i,j \in L} d'_j p_{ij} x_{ij}.$$

In the process above, we only modify the demands, but nothing on the constraints. Thus, the feasible solution $(\bar{x}, \bar{y})$ to the LP problem is also feasible to the new demand problem. Let C_{ND} be the cost in the new demand problem,

$$C_{ND}(\bar{x}, \bar{y}) = \sum_{j \in L} d'_j \bar{C}_j.$$

Theorem 2.5 *After consolidating the demands, the cost of* $(\bar{x}, \bar{y})$ *in the new demand problem is less than that in the original problem, i.e.,* $C_{ND}(\bar{x}, \bar{y}) < C_{LP}(\bar{x}, \bar{y})$.

Proof: To see this, assume that during the consolidation, we move demands from j to i, i.e., $d'_i \leftarrow d_i + d_j, d'_j \leftarrow 0$ and $\bar{C}_j > \bar{C}_i$. Thus, the change of the total costs is:

$$\begin{aligned} & C_{ND}(\bar{x}, \bar{y}) - C_{LP}(\bar{x}, \bar{y}) \\ = \; & (d'_i \bar{C}_i + d'_j \bar{C}_j) - (d_i \bar{C}_i + d_j \bar{C}_j) \\ = \; & ((d_i + d_j)\bar{C}_i + 0 \cdot \bar{C}_j) - (d_i \bar{C}_i + d_j \bar{C}_j) \\ = \; & d_j(\bar{C}_i - \bar{C}_j) < 0. \end{aligned}$$

■

Theorem 2.6 *For any feasible integer solution* $(x1, y1)$ *to the new demand problem, there is a feasible integer solution* $(x2, y2)$ *to the LP problem such that,*

$$C_{LP}(x2, y2) \leq C_{ND}(x1, y1) + 4C_{LP}(\bar{x}, \bar{y}).$$

Proof: Let $(x1, y1)$ be an integer solution to the new demand problem. We will convert it to an integer solution $(x2, y2)$ to the LP problem. Initially, we set $y2 = y1$, i.e., the solution to the LP problem has the same set of storage nodes as the solution to the new demand problem. Then we derive $x2$ according to the consolidating process for the new demand problem considering the following two cases. If node j has positive demand after consolidation in the new demand problem, we set $\forall i, x2_{ij} = x1_{ij}$. In this case, no cost difference is introduced between $C_{LP}(x1, y1)$ and $C_{ND}(x2, y2)$. In the other case, node j moves its demand to another node j' during the consolidating process. Here we look at which storage node j' is associated with in solution $(x1, y1)$, assume it is node i, i.e., $x1_{ij'} = 1$. Then, we also assign node j to the same storage node i in the LP problem, i.e., $x2_{ij} = x1_{ij'} = 1$. The following analysis illustrates the cost difference between $C_{LP}(x1, y1)$ and $C_{ND}(x2, y2)$ incurred by the second case.

In the LP problem, with solution $(x2, y2)$, the cost of sending demand $d_j = 1$ to the sink via i becomes

$$p_{ij}x2_{ij} = p_{ij} = c_{ij} + \beta l_i.$$

Similarly, in the new demand problem, the cost of sending the same demand $d_j = 1$, which is actually a part of $d'_{j'}$, is

$$p_{ij'}x1_{ij'} = p_{ij'} = c_{ij'} + \beta l_i.$$

The difference is $p_{ij} - p_{ij'} = c_{ij} - c_{ij'}$. According to the triangle inequality, $c_{ij'} - c_{ij} < c_{jj'}$. Recall the consolidating process, j moves its demand to j' only when $j' < j$ and $c_{jj'} < 4\max\{\bar{C}_j, \bar{C}_{j'}\} = 4\bar{C}_j$. Therefore, in the second case, $p_{ij} - p_{ij'} \leq 4\bar{C}_j$.

Therefore, summing up the cost difference for all $j \in L$, the largest total difference is reached when every node $j \in L$ falls in the second case. Thus, we have

$$\begin{aligned} C_{ND}(x2, y2) - C_{LP}(x1, y1) &\leq \sum_{j \in L}(p_{ij}x2_{ij} - p_{ij'}x1_{ij'}) = \sum_{j \in L}(p_{ij} - p_{ij'}) \\ &\leq \sum_{j \in L} 4\bar{C}_j = 4\sum_{j \in L} d_j\bar{C}_j = 4C_{LP}(\bar{x}, \bar{y}). \end{aligned}$$

■

Step 2: Consolidating Storage Nodes The goal of this step is to modify the values of $(\bar{x}, \bar{y})$ to obtain a new solution (x', y') to the new demand problem, such that

$$\begin{aligned} y'_i &= 0, \quad \text{if } d'_i = 0; \\ y'_i &\geq \tfrac{1}{2}, \quad \text{if } d'_i > 0. \end{aligned}$$

Then, we will further modify (x', y') to another $\{\frac{1}{2}, 1\}$ solution (x'', y'').

Starting with $x' = \bar{x}$ and $y' = \bar{y}$, we modify $(\bar{x}, \bar{y})$ to (x', y') using the following consolidation process: For each fractional storage node i, i.e., $1 > y'_i > 0$, if $d'_i = 0$, then

1. We will move the value of y'_i to the closest demand node $j(d'_j > 0)$,

$$\begin{aligned} y'_j &\leftarrow \min(1, y'_j + y'_i); \\ y'_i &\leftarrow 0. \end{aligned}$$

2. Also, we need move the forwarding nodes assignments, for each $j' \in L$

$$x'_{jj'} \leftarrow x'_{jj'} + x'_{ij'};$$
$$x'_{ij'} \leftarrow 0.$$

After these changes, we obtain a new solution (x', y') to the new demand problem. In the next, we prove the two properties of y' mentioned in the beginning of this subsection. Following the consolidation in this step, it is obvious to see the first property that if $d'_i = 0$ then $y'_i = 0$. We use the following Lemma 2.2 and Theorem 2.7 to prove the second property of y'. After that, Theorem 2.8 bounds the cost of the new solution after consolidation.

Lemma 2.2 *For a demand node j, any node i satisfying $c_{ij} \leq 2\bar{C}_j$ will move its value of $\bar{y}_i$ to y'_j after the consolidation step presented earlier in this subsection.*

Proof: First, $\forall i$, if $c_{ij} \leq 2\bar{C}_j$, the demand of i is 0, i.e., i is not a demand node. If node i was a demand node, then as a result of Step 1 discussed in Section 2.3.2.2, $c_{ij} > 4\max(\bar{C}_j, \bar{C}_i) > 2\bar{C}_j$, which is in contradiction to $c_{ij} \leq 2\bar{C}_j$.

Next, we prove that all these nodes will move their fractions to node j. Assume that there exists node i with $c_{ij} \leq 2\bar{C}_j$ that moves its $\bar{y}_i$ to another demand node j', which implies $c_{ij'} < c_{ij}$. According to the triangle inequality,

$$c_{jj'} < c_{ij'} + c_{ij} < 2c_{ij} \leq 4\bar{C}_j.$$

Since both node j and node j' are demand nodes, Step 1 in Section 2.3.2.2 would have guaranteed $c_{jj'} > 4\max(\bar{C}_j, \bar{C}_{j'}) \geq 4\bar{C}_j$, which is in contradiction to $c_{jj'} < 4\bar{C}_j$. Thus, after modifying $(\bar{x}, \bar{y})$ to (x', y'), all the nodes within distance of $2\bar{C}_j$ to node j will have moved their values of $\bar{y}$ to y'_j. ■

Theorem 2.7 *After consolidation in this step,*

$$d'_j > 0 \Rightarrow y'_j \geq \frac{1}{2}.$$

Proof: For each node j, recall $\bar{C}_j = \sum_{i \in L} p_{ij}\bar{x}_{ij}$. we have

$$\bar{C}_j \geq \sum_{p_{ij} > 2\bar{C}_j} p_{ij}\bar{x}_{ij} > \sum_{p_{ij} > 2\bar{C}_j} 2\bar{C}_j\bar{x}_{ij} \Rightarrow \sum_{p_{ij} > 2\bar{C}_j} \bar{x}_{ij} < \frac{1}{2}.$$

Since $\sum_i \bar{x}_{ij} = 1$ and $\bar{x}_{ij} \leq \bar{y}_i$,

$$\sum_{p_{ij} \leq 2\bar{C}_j} \bar{y}_i \geq \sum_{p_{ij} \leq 2\bar{C}_j} \bar{x}_{ij} = \sum_i \bar{x}_{ij} - \sum_{p_{ij} > 2\bar{C}_j} \bar{x}_{ij} = 1 - \sum_{p_{ij} > 2\bar{C}_j} \bar{x}_{ij} > \frac{1}{2}.$$

Additionally, because $p_{ij} > c_{ij}$, we have

$$\sum_{c_{ij} \leq 2\bar{C}_j} \bar{y}_i > \sum_{p_{ij} \leq 2\bar{C}_j} \bar{y}_i > \frac{1}{2}.$$

Since node j is a demand node ($d'_j > 0$), by Lemma 2.2, all the nodes within $2\bar{C}_j$ will have moved their values of $\bar{y}$ to j after the consolidation in this step. Therefore, we get

$$y'_j \geq \sum_{c_{ij} \leq 2\bar{C}_j} \bar{y}_i \geq \frac{1}{2}.$$

■

Theorem 2.8 $C_{ND}(x', y') \leq 3C_{ND}(\bar{x}, \bar{y})$.

Proof: Consider that a fractional storage node i has moved its $\bar{y}_i$ to node j during the consolidation. For any demand node j', the previous association $\bar{x}_{ij'}$ is also transferred to j. Since j is the closest demand node to i, we have $c_{ij} \leq c_{ij'}$. Recall $p_{jj'} = c_{jj'} + \beta l_j$, from the triangle inequality,

$$c_{jj'} < c_{ij} + c_{ij'} \leq 2c_{ij'}.$$

For the second term of $p_{jj'}$,

$$\beta l_j < \beta l_i + \beta c_{ij} \quad (l_j < l_i + c_{ij}) < \beta l_i + c_{ij} \quad (\beta < 1) \leq \beta l_i + c_{ij'} = p_{ij'}.$$

Therefore,

$$p_{jj'} = c_{jj'} + \beta l_j < 2c_{ij'} + p_{ij'} \leq 2p_{ij'} + p_{ij'} = 3p_{ij'}.$$

Considering all the consolidated fractional storage nodes, e.g., $\bar{y}_{i_1}, \bar{y}_{i_2}, \ldots$ are moved to y'_j,

$$\begin{aligned} C_{ND}(x',y') &= \sum_{j,j' \in L} d'_{j'} p_{jj'} x'_{jj'} \\ &= \sum_{j,j' \in L} d_{j'} p_{jj'} (\bar{x}_{jj'} + \bar{x}_{i_1 j'} + \bar{x}_{i_2 j'} + \cdots) \\ &< \sum_{j,j' \in L} d_{j'} (p_{jj'} \bar{x}_{jj'} + 3p_{i_1 j'} \bar{x}_{i_1 j'} + 3p_{i_2 j'} \bar{x}_{i_2 j'} + \cdots) \\ &< 3 \sum_{j,j' \in L} d_{j'} p_{jj'} \bar{x}_{jj'} = 3C_{ND}(\bar{x},\bar{y}). \end{aligned}$$

Therefore, the cost $C_{ND}(x',y')$ is at most triple of $C_{ND}(\bar{x},\bar{y})$. ■

Next, we will modify (x',y') to another feasible solution (x'',y'') to the new demand problem subject to $x'',y'' \in \{\frac{1}{2}, 1\}$, and the cost of (x'',y'') is no more than the cost of (x',y'). The condition that $y',y'' \geq \frac{1}{2}$ implies that there are at most $2k$ nodes with positive demands in both solutions (x',y') and (x'',y'') because of the second constraint in the new demand problem formulation. Initially, we assign $x'' = x'$ and $y'' = y'$. At this point, when calculating the objective function of cost, we only need consider demand nodes since the nodes with zero demand have no contribution to the cost. For each demand node i, in order to reduce the cost, the best choice is to send data through itself. However, considering the third constraint in the new demand problem, node i cannot send all data through itself. In our solution x'', we take the best case by setting

$$x''_{ii} = y''_i, \text{ if } d'_i > 0,$$

i.e., y_i'' portion of demand d_i' is sent through node i itself. In addition, we assign the remaining $(1-y_i'')$ portion of d_i' to another demand node i', where $p_{i'i}$ is the minimum among all demand nodes. Let $s(i)$ denote such node i'. The minimum cost associated with all demand nodes i is

$$\begin{aligned}\sum_{d_i'>0} d_i'(p_{ii}x_{ii}'' + p_{s(i)i}(1-x_{ii}'')) &= \sum_{d_i'>0} d_i'(p_{ii}y_i'' + p_{s(i)i}(1-y_i'')) \\ &= \sum_{d_i'>0} d_i'(\beta l_i y_i'' + p_{s(i)i} - p_{s(i)i}y_i'') \\ &= \sum_{d_i'>0} d_i' p_{s(i)i} - \sum_{d_i'>0} d_i' y_i''(p_{s(i)i} - \beta l_i), \qquad (2.6)\end{aligned}$$

where

$$p_{s(i)i} = c_{s(i)i} + \beta l_{s_i} > \beta(c_{s(i)i} + l_{s(i)}) \quad (\beta < 1) > \beta l_i.$$

So far, we only modify x'', but y'' is still equal to y'. Since formula (2.6) only depends on y'', we can use $f(y'')$ to represent it.

Next, we will show that under the constraint $\frac{1}{2} \leq y_i'' \leq 1$ for demand node i, we can obtain a $\{\frac{1}{2}, 1\}$-integral solution y'' such that $f(y'')$ is the minimum. The first term of Eq. (2.6) is a constant independent of y''. To minimize the cost, we should maximize y_i'' for the nodes with largest values of $d_i'(p_{s(i)i} - \beta l_i)$. Let n' be the number of demand nodes, as we know, $n' < 2k$. We reorder demand nodes according to $d_i'(p_{s(i)i} - c_2 l_i)$ decreasingly. We set $y_i'' = 1$ for the first $2k - n'$ nodes and $y_i'' = \frac{1}{2}$ for the remaining $2(n'-k)$. It is actually a greedy algorithm to maximum the second term of Eq. (2.6). Thus,

$$f(y'') \leq f(y') \leq C_{ND}(x', y').$$

Accordingly, x'' is also a $\{\frac{1}{2}, 1\}$-integral solution. For a demand node i, if $y_i'' = 1$,

$$x_{ji}'' = \begin{cases} y_i'' = 1 & \text{if } j = i; \\ 0 & \text{otherwise.} \end{cases}$$

Otherwise, if $y''_i = \frac{1}{2}$,

$$x''_{ji} = \begin{cases} y''_i = \frac{1}{2} & \text{if } j = i; \\ 1 - y''_i = \frac{1}{2} & \text{if } j = s(i); \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2.9 $C_{ND}(x'', y'') \leq C_{ND}(x', y')$.

Proof: It is obvious because (x'', y'') yields the minimum value of the cost function f. ■

Step 3: Rounding Finally, we apply a rounding algorithm to get a $\{0, 1\}$ integer solution. First, we place a storage node at node j if $y''_j = 1$. For the remaining nodes with $y''_i = \frac{1}{2}$, half data is sent via $s(i)$. Consider a directed graph G consisting of the remaining demand nodes, where each edge is from i to $s(i)$.

Lemma 2.3 *There is no loop of length more than 2 in G.*

Proof: Assume there is a loop in G involving nodes $n_1, n_2, \cdots, n_m$, where $m > 2$ and $\forall t \leq m$ there is a directed edge from n_t to $n_{(t \bmod m)+1}$. For each node n_t, $s(n_t) = n_{(t \bmod m)+1}$. According to the definition of $s(n_t)$ that $p_{s(n_t)n_t}$ is the minimum, we have

$$p_{n_2 n_1} < p_{n_m n_1}, \quad p_{n_3 n_2} < p_{n_1 n_2}, \quad \cdots \quad , \quad p_{n_1 n_m} < p_{n_{m-1} n_m}.$$

Recall $p_{ij} = c_{ij} + \beta l_i$, the conditions above become

$$\begin{aligned} c_{n_2 n_1} + \beta l_{n_2} &< c_{n_m n_1} + \beta l_{n_m} \\ c_{n_3 n_2} + \beta l_{n_3} &< c_{n_1 n_2} + \beta l_{n_1} \\ &\cdots \\ c_{n_1 n_m} + \beta l_{n_1} &< c_{n_{m-1} n_m} + \beta l_{n_{m-1}}. \end{aligned}$$

Thus, the summation of the left side should be less than the summation of the right side. We find by examining the inequalities, however, that the summation of both sides are equal. This contradiction means that the series of conditions can not be held at a same time. ■

Furthermore, if there are two edges between two nodes, i.e $s(i) = j$ and $s(j) = i$, we arbitrarily choose one of them as a root and eliminate the directed edge from the root to the other node. Finally, G becomes a forest graph, which consists of multiple rooted trees. Additionally, we assign every node a level value, which is the distance to the root of the tree that it belongs to. We can divide these nodes into two sets based on odd and even level values and select the smaller set of nodes to be storage nodes. Recall that $\{i|y''_i = \frac{1}{2}\}$ has $2(n' - k)$ nodes. Thus, we place at most $n' - k$ storage nodes at this step. Plus the storage nodes set earlier in $\{i|y''_i = 1\}$, which has $2k - n'$ nodes, the total number of storage nodes is at most $\sum y''_i \leq k$. In addition, each unselected node i in the tree will associate itself with $s(i)$, which must be a storage node, i.e., $(x''_{s(i)i} = 1)$. Finally, we get an integer solution to the new demand problem from feasible solution $(\bar{x}, \bar{y})$. Let $C_{ND}(INT)$ be the cost of this integer solution in the new demand problem (found after rounding). We can prove the following theorem.

Theorem 2.10 *After rounding, the cost of the integer solution $C_{ND}(INT)$ is no more than double the cost of (x'', y''), i.e., $C_{ND}(INT) \leq 2C_{ND}(x'', y'')$.*

Proof: In the routing process above, for j with $y''_j = \frac{1}{2}$, the previous cost is $\frac{1}{2}\beta l_j + \frac{1}{2}p_{s(j)j}$ and after rounding, it becomes $p_{s(j)j}$ or βl_j. Thus, the cost is at most doubled. ■

Based on the previous theorems, therefore,

$$\begin{aligned} C_{ND}(INT) &\leq 2C_{ND}(x'',y'')(\text{Theorem 2.10}) \\ &\leq 2C_{ND}(x',y')(\text{Theorem 2.9}) \\ &\leq 6C_{ND}(\bar{x},\bar{y})(\text{Theorem 2.8}). \end{aligned}$$

As we mentioned in Theorem 2.6, we can derive an integer solution to the original problem from an integer solution to the new demand problem. Let $C_{LP}(INT)$ denote the cost of this integer solution in the LP problem,

$$\begin{aligned} C_{LP}(INT) &\leq C_{ND}(INT) + 4C_{LP}(\bar{x},\bar{y})(\text{Theorem 2.6}) \\ &\leq 6C_{ND}(\bar{x},\bar{y}) + 4C_{LP}(\bar{x},\bar{y}) \\ &\leq 6C_{LP}(\bar{x},\bar{y}) + 4C_{LP}(\bar{x},\bar{y})(\text{Theorem 2.5}) \\ &= 10C_{LP}(\bar{x},\bar{y}). \end{aligned}$$

Since $(\bar{x},\bar{y})$ is the optimal fractional solution, the cost of $(\bar{x},\bar{y})$, $C_{LP}(\bar{x},\bar{y})$, must be no more than the cost of the optimal integer solution. Therefore, combining three steps together, we get a 10-approximation $(3 \times 1 \times 2 + 4)$ algorithm for this problem.

2.3.3 Stochastic Analysis for Random Deployment

The algorithms in the previous sections aim to find the optimal locations for storage nodes. In reality, however, the storage nodes may not be deployed in a precise way. Instead their deployment may be random, e.g., the storage nodes are dispersed from an airplane. In this section, we evaluate the performance of random deployment of storage nodes in fixed and dynamic trees in which every sensor node finds the best storage node for data storage.

2.3.3.1 Fixed Tree Model

Assume the forwarding nodes and storage nodes are randomly distributed to the field with density λ and λ_s respectively. For simplicity, we consider a disk network field, where the sink is placed at the center and R is the radius. In the fixed tree model, the network builds a communication tree in which each node finds the shortest path to the sink by following the tree edges. Each forwarding node sends its data to the first ancestor storage node on the path to the sink. As our simulation and other previous research show, the radius (r_i) of the area covered by the nodes that are i hops or less from the sink is proportional to i. Let this ratio be $c' = \frac{r_i}{i}$, Thus, we can estimate the number of nodes whose distances to the sink are between $(t-1)c'$ and tc', i.e., the nodes with depth t. Let $num(t)$ represent the total number of the nodes whose depth values are t,

$$num(t) = \lambda\pi(t^2c'^2 - (t-1)^2c'^2) = \lambda\pi(2t-1)c'^2.$$

For a node with depth t, let $s(t)$ be the expected hop distance to its closest storage ancestor. The cost caused by this node is

$$r_d s_d \cdot s(t) + r_q \alpha s_d \cdot (t - s(t)).$$

The probability that an individual node is a storage node is $p = \frac{\lambda_s}{\lambda}$. Therefore,

$$\begin{aligned} s(t) &= p\cdot 0 + p(1-p)\cdot 1 + p(1-p)^2\cdot 2 + \cdots + p(1-p)^{t-1}\cdot(t-1) + (1-p)^t\cdot t \\ &= (\frac{1}{p} - 1)(1-(1-p)^t). \end{aligned}$$

The total energy cost in the fixed tree model can be expressed as

$$E = \sum_{t=1}^{\frac{R}{c'}} num(t)(r_d s_d s(t) + r_q \alpha s_d (t - s(t))) + E_q,$$

where E_q is the cost of query diffusion. The value of c' is related to the communication range and node density. We can obtain the value from simulations.

Query messages are diffused from the sink to every storage node. For each storage node, it incurs an extra query diffusion cost along the path to its closest storage ancestor. If we assume there is no overlap among the paths connecting each storage node and its closest storage ancestor, the total query diffusion cost E_q can be formulated as

$$E_q = \sum_{t=1}^{\frac{R}{c'}} num'(t) r_q s_q s(t), \tag{2.7}$$

where $num'(t) = \lambda_s \pi (2t-1) c'^2$ is the number of storage nodes whose depth values are t and recall $s(t)$ is the expected distance to the closest storage ancestor.

2.3.3.2 Dynamic Tree Model

The fixed tree model assumes that the communication tree does not change according to the placement of the storage nodes. In the dynamic tree model, after the storage nodes have been positioned, each sensor node chooses the best storage node for storage with respect to the minimal communication cost for data forwarding and query diffusion and reply. The storage node placement in this model is more complicated than that in the fixed tree model because we need to consider the interplay between the storage node placement and the selection of the storage node for each sensor. These two steps affect each other dynamically.

In the optimal solution, a storage node should send query reply to the sink by following the shortest path because the data coming out of a storage node cannot be reduced any further according to the definition of data reduction function. A forwarding node has to choose a storage node for data storage to minimize the total communication cost for its data. Assume the sink is

located at the origin. Let $\vec{x}_i$ represent the location of sensor i. Additionally, we define $fd(\vec{x}_i)$ as the location of the forwarding destination (storage node) assigned to node i. If i is a storage nodes, then $fd(\vec{x}_i) = \vec{x}_i$. The energy cost of sending raw data from $\vec{x}_i$ to $fd(\vec{x}_i)$ is $r_d s_d |\vec{x}_i - fd(\vec{x}_i)|$. The query reply cost for the data from forwarding node i is $r_q \alpha s_d |fd(\vec{x}_i)|$. In total, the cost generated by a single node i in a time unit is:

$$r_d s_d |\vec{x}_i - fd(\vec{x}_i)| + r_q \alpha s_d |fd(\vec{x}_i)|.$$

To find the optimal solution, we need to minimize the cost for each sensor.

The total energy cost of the sensor network can be described as

$$E = \sum_i (r_d s_d |\vec{x}_i - fd(\vec{x}_i)| + r_q \alpha s_d |fd(\vec{x}_i)|) + E_q, \tag{2.8}$$

where E_q is the cost of query diffusion. We find that E_q in this dynamic tree model is the same as that in the fixed tree model. Because in both models, each storage node is connected to the sink by the shortest path. Therefore, we can also use Eq. (2.7) to estimate E_q. In the following of this section, we will analyze the rest part of Eq. (2.8), which is denoted by E'.

First, we define a function $F(\vec{x}, \vec{y})$ as the energy cost caused by the sensor at location $\vec{x}$ where $\vec{y}$ is the location of its forwarding destination.

$$F(\vec{x}, \vec{y}) = r_d s_d |\vec{x} - \vec{y}| + r_q \alpha s_d |\vec{y}|.$$

Moreover, we define an area

$$G(\vec{x}, U) = \{\vec{y} | F(\vec{x}, \vec{y}) \leq U\},$$

that is, if a sensor at $\vec{x}$ selects any storage node in that area, the energy cost for the data of that sensor would be no more than U. Theoretically, the minimum reply cost with Poisson deployment

is

$$E' = \lambda \iint P(\vec{y} \in S) F(\vec{x}, \vec{y}) P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi) dy dx,$$

where S is the set of all storage nodes including the sink. $P(\vec{y} \in S)$ is the probability that there is a storage node at location $\vec{y}$.

$$P(\vec{y} \in S) = \begin{cases} 1 & \text{if } \vec{y} \text{ is the origin;} \\ \lambda_s & \text{otherwise.} \end{cases}$$

$F(\vec{x}, \vec{y})$ is the energy cost if a forwarding node at $\vec{x}$ sends data through a storage node at $\vec{y}$. For a fixed $\vec{x}$,

$$P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi)$$

is the probability that its forwarding destination is at location $\vec{y}$, i.e., no other storage node would induce less energy cost than $F(\vec{x}, \vec{y})$. $G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi$ means that no other storage node is more eligible than the one at $\vec{y}$. According to Poisson processes,

$$P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi) = \begin{cases} e^{-\lambda_s |G(\vec{x}, F(\vec{x}, \vec{y}))|} & \text{if } F(\vec{x}, \vec{y}) \leq F(\vec{x}, 0); \\ 0 & \text{otherwise.} \end{cases}$$

Unlike other nodes, the sink is deterministically fixed in the network. So if area G covers the sink, there is no need to compute the probability. The forwarding node will definitely send data directly to the sink.

However, $|G|$ in the formula above, called the Cartesian Oval, cannot be expressed in a closed form. To approximate the energy cost, we make each forwarding node simply choose the closest storage node for data storage. The network field is then divided into Voronoi cells induced by storage nodes. The energy cost of this topology is very close to the optimal case, especially when $\lambda_s \ll \lambda$.

Assume there is a forwarding node i at location $\vec{x}$, the probability that i sends data through a storage node at location $\vec{y}$ becomes

$$P(\vec{x} \rightarrow \vec{y}) = \begin{cases} 0 & \text{if } |\vec{x}-\vec{y}| \geq |\vec{x}|; \\ e^{-\lambda_s \pi |\vec{x}|^2} & \text{if } \vec{y} \text{ is the origin;} \\ \lambda_s e^{-\lambda_s \pi |\vec{x}-\vec{y}|^2} & \text{Otherwise.} \end{cases}$$

Thus,

$$\begin{aligned} E' &= \lambda \int\int F(\vec{x},\vec{y}) P(\vec{x} \rightarrow \vec{y}) dy dx \\ &= \lambda (\int F(\vec{x},0) e^{-\lambda_s \pi |\vec{x}|^2} dx + \int\int_{|\vec{x}-\vec{y}|<|\vec{x}|} F(\vec{x},\vec{y}) \lambda_s e^{-\lambda_s \pi |\vec{x}-\vec{y}|^2} dy dx) \\ &= \lambda \int F(\vec{x},0) e^{-\lambda_s \pi |\vec{x}|^2} dx + \lambda \int\int_{|\vec{x}-\vec{y}|<|\vec{x}|} F(\vec{x},\vec{y}) \lambda_s e^{-\lambda_s \pi |\vec{x}-\vec{y}|^2} dy dx. \end{aligned} \tag{2.9}$$

In the first term, $F(\vec{x},0) = r_d s_d |\vec{x}|$. Therefore,

$$\lambda \int F(\vec{x},0) e^{-\lambda_s \pi |\vec{x}|^2} dx = \lambda r_d s_d \int |\vec{x}| e^{-\lambda_s \pi |\vec{x}|^2} dx \tag{2.10}$$

$$= \lambda r_d s_d \int_0^{2\pi} \int_0^R \rho e^{-\lambda_s \pi \rho^2} \rho d\rho d\theta$$

$$= 2\pi \lambda r_d s_d \int_0^R \rho^2 e^{-\lambda_s \pi \rho^2} d\rho. \tag{2.11}$$

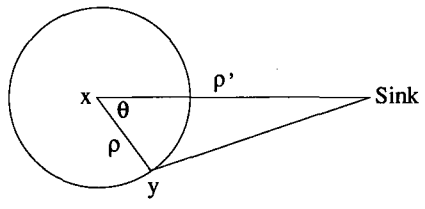


Figure 2.7: A forwarding node at location x sends data via a storage node at y.

For the second term in Eq. (2.9), Fig. 2.7 shows the variables after coordination conversions, where $\rho = |\vec{x}-\vec{y}|$ and $\rho' = |x|$. $F(\vec{x},\vec{y})$ can be expressed by

$$r_d s_d \rho + r_q \alpha s_d \sqrt{\rho'^2 + \rho^2 - 2\cos\theta \rho' \rho}.$$

Thus, the second term becomes:

$$
\begin{aligned}
& \lambda \int\int_{|\vec{x}-\vec{y}|<|\vec{x}|} F(\vec{x},\vec{y})\lambda_s e^{-\lambda_s \pi |\vec{x}-\vec{y}|^2} dydx \\
= \; & \lambda\lambda_s \int\int_0^{2\pi}\int_0^{|\vec{x}|} e^{-\lambda_s \pi \rho^2}(r_d s_d \rho + r_q \alpha s_d \sqrt{|\vec{x}|^2+\rho^2-2\cos\theta|\vec{x}|\rho})\rho d\rho d\theta dx \\
= \; & 4\pi^2\lambda\lambda_s r_d s_d \int_0^R\int_0^{\rho'} e^{-\lambda_s\pi\rho^2}\rho^2\rho' d\rho d\rho' \\
& +2\pi\lambda\lambda_s r_q \alpha s_d \int_0^R\int_0^{2\pi}\int_0^{\rho'} e^{-\lambda_s\pi\rho^2}\rho\rho'\sqrt{\rho'^2+\rho^2-2\cos\theta\rho'\rho} d\rho d\theta d\rho'. \qquad (2.12)
\end{aligned}
$$

Combining Eq. (2.10) and (2.12), the total energy cost except query diffusion is

$$
\begin{aligned}
E' \; = \; & 2\pi\lambda r_d s_d \int_0^R \rho^2 e^{-\lambda_s\pi\rho^2} d\rho + 4\pi^2\lambda\lambda_s r_d s_d \int_0^R\int_0^{\rho'} e^{-\lambda_s\pi\rho^2}\rho^2\rho' d\rho d\rho' \\
& +2\pi\lambda\lambda_s r_q \alpha s_d \int_0^R\int_0^{2\pi}\int_0^{\rho'} e^{-\lambda_s\pi\rho^2}\rho\rho'\sqrt{\rho'^2+\rho^2-2\cos\theta\rho'\rho} d\rho d\theta d\rho'.
\end{aligned}
$$

We can further approximate E' by examining its two components separately. Let E_{fs} be the cost of transferring raw data between forwarding nodes and their closest storage nodes. Let E_{ss} be the cost to relay reply from storage nodes to the sink. For a forwarding node i , the expected distance to the closest storage node is $\frac{1}{2\sqrt{\lambda_s}}$. Thus,

$$E_{fs} = \lambda\pi R^2 r_d s_d \frac{1}{2\sqrt{\lambda_s}}. \qquad (2.13)$$

$\lambda\pi R^2$ is the total number of forwarding nodes and $r_d s_d \frac{1}{2\sqrt{\lambda_s}}$ is the energy cost of transferring raw data from an individual forwarding node to its closest storage node. On the other hand,

$$E_{ss} = r_q \alpha s_d \sum_{i\in S}(D_i+1)L_i,$$

where D_i is the total number of descendants and L_i is the distance to the sink. Since each forwarding node chooses the closest storage node for data storage, the number of forwarding nodes that

each storage node is responsible for is approximately the same. If we replace L_i by the mean value L',

$$E_{ss} = r_q \alpha s_d L' \sum_{i \in S} (D_i + 1).$$

In this equation, $\sum_{i \in S}(D_i + 1)$ represents the number of nodes which send data via storage nodes to the sink. Let N' be the number of forwarding nodes that send data directly to the sink,

$$\sum_{i \in S} (D_i + 1) = \lambda \pi R^2 - N'.$$

N' can be derived as

$$\begin{aligned} N' = \lambda \int e^{-\lambda_s \pi |\vec{x}|^2} dx &= \lambda \int_0^{2\pi} \int_0^R e^{-\lambda_s \pi \rho^2} \rho d\rho d\theta \\ &= 2\pi\lambda \int_0^R \rho e^{-\lambda_s \pi \rho^2} d\rho = \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2}). \end{aligned}$$

And L' can be simply approximated as

$$L' = \frac{\lambda_s \int_0^R 2\pi r \cdot r dr}{\lambda_s \pi R^2} = \frac{2}{3} R.$$

Therefore,

$$E_{ss} = r_q \alpha s_d (\frac{2}{3} R(\lambda \pi R^2 - \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2}))). \tag{2.14}$$

Combining Eq.(2.13) and (2.14),

$$E' = \lambda \pi R^2 r_d s_d \frac{1}{2\sqrt{\lambda_s}} + r_q \alpha s_d (\frac{2}{3} R(\lambda \pi R^2 - \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2}))).$$

2.4 Performance Evaluation

2.4.1 Fixed Tree Model

We have implemented a simulator to simulate the deterministic storage node placement in the fixed tree model by using dynamic programming. and the random storage node deployment in the fixed tree and dynamic tree models. We evaluate the energy cost for various parameters.

2.4.1.1 Simulation Settings

In our simulation, we consider a network of sensors deployed on a disk of radius 5 with the sink placed at the center. One thousand sensor nodes ($n = 1000$) are deployed to the field randomly following 2-dimensional spatial Poisson process. Node transmission range is set to 0.65. After all nodes are deployed, a routing tree rooted at the sink is constructed by flooding a message from the sink to all the nodes in the network. The message carries the number of hops it travels at each node so that each node chooses among its neighbors the node that has the minimum number of hops to be its parent. This tree topology is needed in the simulation of the fixed tree model. This step, however, can be skipped for the dynamic tree model.

In the rest of this section, we will present and compare the following algorithms.

- **FT-DD**: It represents the fixed tree model with deterministic deployment. In this algorithm, the storage nodes are deployed by following the dynamic programming algorithm according to the known tree topology.

- **FT-RD**: It represents the fixed tree model with random deployment. In this algorithm, we randomly select a certain number of nodes in the network to be storage nodes.

- **DT-RD**: It represents the dynamic tree model with random deployment. In this algorithm, the storage nodes are randomly deployed. After that, each forwarding node selects the best storage node to deliver data and each storage node replies to query by following the shortest path to the sink.

- **ST-RD**: It represents semi-dynamic tree model with random deployment which is enhanced version of FT-RD with a local adjustment. When a sensor i is upgraded to storage node in a tree structure, its siblings' children will try to set i as their parents if i is within their communication range.

- **Greedy**: It represents a greedy algorithm where the most heavily loaded sensors will be upgraded to storage nodes. Usually, those sensors close to the sink will become storage nodes in this algorithm.

In addition, we use relative energy cost as performance metrics. We use the scenario that no storage node except the sink is deployed as the baseline. Let the energy cost in this no storage scenario be E_f. And let the energy cost after the storage nodes are deployed be E. The relative energy cost is defined as $\frac{E}{E_f}$, represented as a percentage. In the rest, we simply use "energy cost" for "relative energy cost".

Because of the randomness of our simulation environment, results from the same parameter setting might vary a lot. Therefore, for a certain set of parameters, we conduct 100 independent simulations and the average energy cost is used in the following analysis and comparison. Unless otherwise stated, we set the following parameters in our simulations: $r_d = 1$, $r_q = 1$, $s_d = 1$, $s_q = 1$ and $\alpha = 0.5$. We evaluate the energy cost by varying the number of storage nodes k and the data reduction rate α. The density of storage nodes λ_s can be derived by $\lambda_s = \frac{k}{\pi R^2}$. Note that the energy

cost is also related to r_d, s_d, r_q and s_q. However, for comparison purpose, changing r_q, s_q, r_d, s_d will be equivalent to changing α. To simplify the description, we fix $r_q = s_q = r_d = s_d = 1$ and only vary α to examine different characteristics of data and queries.

2.4.1.2 Random Deployment

Fig. 2.8 shows the energy cost of random deployment in the fixed tree model. We compare our theoretical estimation with simulation results. As we can see from the figure, the theoretical estimation and the simulation match well. We have examined the simulation carefully and found that many storage nodes are placed at the leaf nodes or have very few descendants. Therefore, the data reduction for those descendants is negligible and less energy is saved compared to the case that each node sends all the data to the sink.

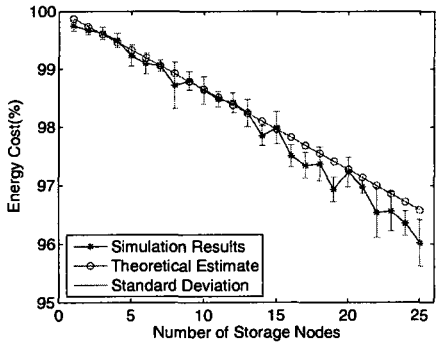


Figure 2.8: FT-RD: Energy cost with varying number of storage nodes (k), $n = 1000, r_d = r_q = s_d = s_q = 1, \alpha = 0.5$.

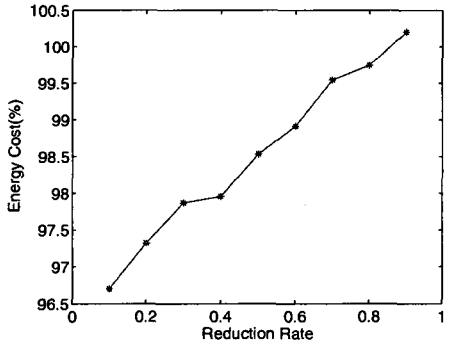


Figure 2.9: FT-RD: The impact of data reduction rate (α), $n = 1000, r_d = r_q = s_d = s_q = 1, k = 10$ and $\lambda_s = \frac{10}{\pi \cdot 5^2} = 0.127$.

In Fig. 2.9, we show the energy cost with respect to different data reduction rates α. We fix the number of storage nodes ($k = 10$) and change the data reduction rate α from 0.1 to 0.9. In this fixed tree model, decreasing data reduction rate cannot improve the performance too much. Even when α is set to 0.1, we still have more than 96% energy cost with 10 storage nodes. The reason is that data accumulation to the storage nodes from the forwarding nodes consumes most of the

energy with respect to the query diffusion and reply. Moreover, when α is 0.9, the energy cost is even worse than the original cost, because the incurred query diffusion cost becomes larger than the benefits obtained.

The energy cost of random deployment in the dynamic tree model is shown in Fig. 2.10. In this model, the location of each storage node is broadcast to forwarding nodes so that they can choose the proper storage nodes to deliver data for the energy concern. In this way, we take full advantage of every storage node and maximize their contributions to the whole network. As shown in Fig. 2.10, random deployment performs much better in this dynamic tree model. The energy cost decreases very fast with increasing number of storage nodes, e.g., with 10 storage nodes (1% of total nodes), we can save energy by approximately 20%. Fig. 2.11 illustrates the impact of data reduction rate to the energy cost in the dynamic tree model. In this figure, α becomes an important parameter, because every storage node is in charge of many forwarding nodes in the dynamic tree model. A small decrease of α will reduce energy cost greatly. We also observe that our theoretical estimation matches well with the simulation results. Although our stochastic analysis uses some expected values and approximations, the maximum difference between the two curves is less than 5%.

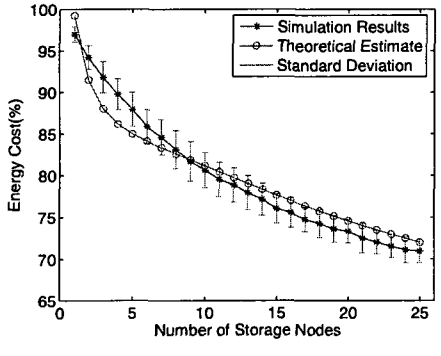


Figure 2.10: DT-RD: Energy cost with varying number of storage nodes (k), $n = 1000, r_d = r_q = s_d = s_q = 1$, $\alpha = 0.5$.

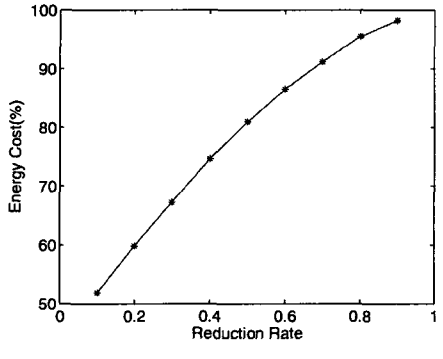


Figure 2.11: DT-RD: The impact of data reduction rate (α), $n = 1000, r_d = r_q = s_d = s_q = 1, k = 10$ and $\lambda_s = \frac{10}{\pi \cdot 5^2} = 0.127$.

As shown above, with random deployment, the dynamic tree model has a significant performance improvement over the fixed tree model. However, the locations of storage nodes need to be broadcast to all other nodes and the new tree is completely different from the originally constructed tree one. We consider a semi-dynamic tree model, in which local adjustments are applied to the originally constructed tree. For each storage node i, all the forwarding nodes within the transmission range of i that have a depth no less than i's depth select i as parent. In result, each storage node gains more descendants and accepts more raw data storage. Fig. 2.12 compares the energy costs of random deployment in three models (fixed tree, dynamic tree and semi-dynamic tree), as well as deterministic deployment in the fixed tree model. We use ST-RD to denote the newly introduced semi-dynamic tree model with random deployment. As shown in Fig. 2.12, DT-RD achieves the best performance, while FT-RD has the worst performance. Local adjustment in ST-RD improves the performance of the fixed tree model. In FT-RD, each storage node has no control about how many descendants it can have. Many storage nodes are deployed with few descendants, which explains why FT-RD delivers the worst performance. ST-RD allows each storage node to have some restrained flexibility in choosing its descendants, and has a better performance than FT-RD. DT-RD has more flexibility in choosing descendants, and we see a much improved performance.

2.4.1.3 Deterministic Deployment

Fig. 2.13 and Fig. 2.14 illustrate the performance of deterministic deployment in the fixed tree model. In the simulation, the locations of the storage nodes are obtained by Algorithm 2. Compared to the random deployment (FT-RD) and greedy algorithm, deterministic deployment significantly improve the performance by precisely computing the optimal positions to put the storage

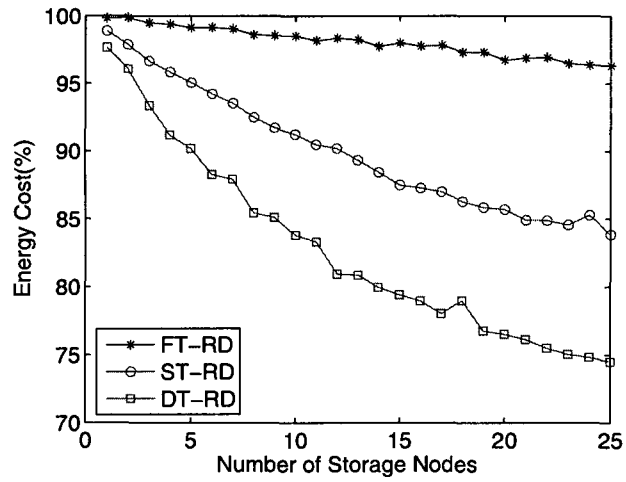


Figure 2.12: Comparison of energy costs with varying number of storage nodes (k), $n = 1000, r_d = r_q = s_d = s_q = 1$, $\alpha = 0.5$.

nodes.

In Fig. 2.13, we fix the reduction rate $\alpha = 0.5$, and vary the number of storage nodes from 2 to 25. With a few storage nodes, the energy cost is sharply reduced in Fig. 2.13. When k becomes larger, the slope gradually becomes flatter. As shown in the figure, we can save about 20% energy cost with 10 storage nodes, and 30% with 25 storage nodes. In addition, Fig. 2.14 shows the energy cost with varying reduction rate, when the number of storage nodes is fixed as 10. As we can see, the energy cost is nearly linear to the reduction rate α.

Intuitively, the performance in the fixed tree model depends on which level of the tree the storage nodes are deployed at. When storage nodes are close the the leaves, which often happens in FT-RD, the benefit of data reduction is limited. On the other hand, when storage nodes are deployed close to the sink as in the greedy algorithm, a large amount of raw data have to traverse a long path to reach the storage nodes still yielding high energy cost. The locations derived from our algorithm usually reside in the middle levels. Here is an example of the result. The following table shows the depth distribution in a particular tree structure. The depth of the sink is 0. When we

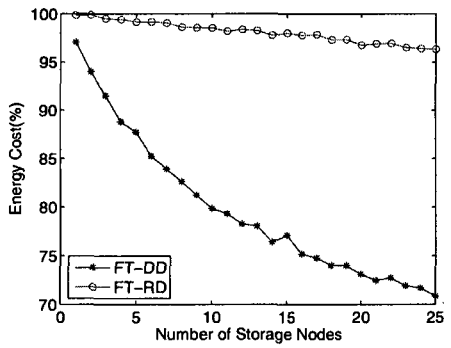


Figure 2.13: Comparison of FT-DD, FT-RD and Greedy: Energy cost with varying number of storage nodes (k), $n = 1000, r_d = r_q = s_d = s_q = 1$, $\alpha = 0.5$.

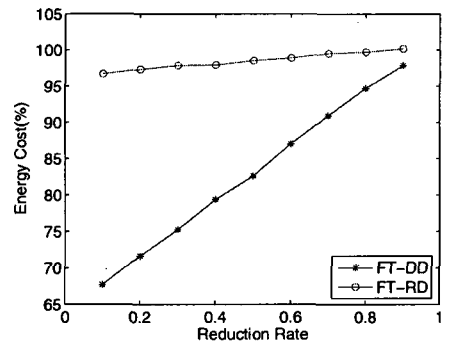


Figure 2.14: Comparison of FT-DD, FT-RD and Greedy: The impact of data reduction rate (α), $n = 1000, r_d = r_q = s_d = s_q = 1$, $k = 10$.

Depth	1	2	3	4	5	6	7	8	9	10	11
Number of sensors:	10	32	48	74	108	128	176	189	164	65	6

deploy 10 storage nodes in such a tree, the derived depths of storage nodes are 3,4,4,4,4,5,5,6,6,6.

2.4.1.4 Network Life

Finally, load distribution and network lifetime are shown in Fig. 2.15 and Fig. 2.16. We show the workloads of the most heavily-loaded 50 nodes in Fig. 2.15. In Fig. 2.16, we define lifetime as the time that 2% nodes are depleted of energy. In our setting, it means 20 sensors are out of operation. According to our simulation, this scenario will usually cause disconnection in the sensor network. As we can see in Fig. 2.15, FT-RD almost has no improvement on load-balancing and lifetime. In contrast, FT-DD lengthens the lifetime a lot with a small number of storage nodes, although the objective of our algorithm is to minimize the total energy cost. For example, with 15 storage nodes, the lifetime is increased by more than 60%. DT-RD does not perform well with only a few storage nodes because the sensors connecting storage nodes and the sink carry a lot of workloads for both raw data transmission and reply collection. The greedy algorithm is superior to DT-RD

by specifically reducing the energy cost of the most heavy loaded nodes.

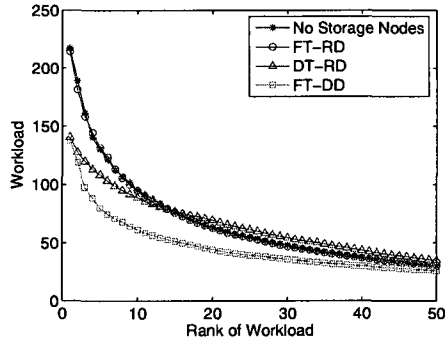


Figure 2.15: Comparison of load-balancing: Workload is measured as the size of the message sent by each sensor per unit time, $n = 1000, r_d = r_q = s_d = s_q = 1, k = 10$ and $\alpha = 0.5$.

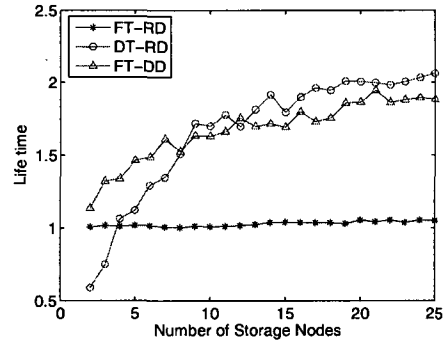


Figure 2.16: Comparison of lifetime with varying number of storage nodes (k): Values are normalized by the lifetime without storage nodes, $n = 1000, r_d = r_q = s_d = s_q = 1$, $\alpha = 0.5$.

2.4.2 Dynamic Tree Model

We have implemented the approximation algorithm and compared the performance of the algorithm with the optimal solution. We consider a network composed of 100 sensor nodes randomly deployed in a 100×100 square field, where the sink is in the center. We vary the number of storage nodes k (including the sink) from 2 to 15 with β taking $0.1, 0.15$, and 0.2 respectively. In our approximation algorithm implementation, we use GLPK package (GNU Linear Programming Kit [1]) to get the fractional solution in the first step of our algorithm. The optimal solution is done by using integer linear programming, which is provided by MIP (mixed integer program).

Fig. 2.17 shows the simulation results when the parameter β is set to 0.1, 0.15 and 0.2. We first calculate a maximum cost C_{max}, which is the energy cost when there is no storage node and every sensor sends data directly to the sink. The performance shown in the figures is the ratio over C_{max}. From the figures, we observe that our approximation algorithm achieves the optimal performance when the number of storage nodes is small, which is a valid assumption since a storage node

is expected to be in charge of tens of regular sensor nodes. When the number of storage nodes becomes larger, the disparity between the optimal solution and our approximation algorithm gets bigger. Even though the approximation algorithm has a high competitive ratio, our simulation shows that in practice, the algorithm performs well when the number of storage nodes is small.

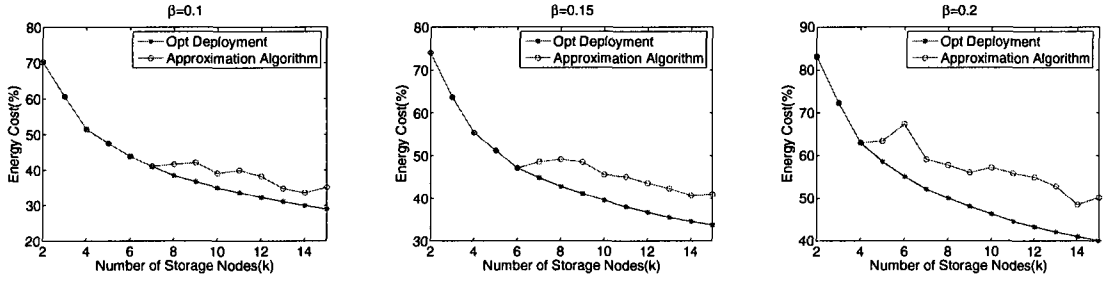


Figure 2.17: Select k storage nodes from 100 randomly deployed sensors and $\beta = 0.1, 0.15, 0.2$.

2.5 Summary

In this chapter, we have presented our work in network architecture design which focuses on storage placement in sensor networks. Our design improves the energy efficiency by deploying special storage nodes to build a hybrid sensor network with regular sensors. We have proposed the optimal algorithms for finding the best locations to place storage nodes in order to maximize performance gain. Our evaluation shows that guided by our algorithms, this hybrid structure significantly reduces the energy consumption in a sensor network.

Our solution in this chapter is a generic framework suitable for all queries. In fact, with more knowledge about specific queries, we can customize the protocol to become more efficient in terms of energy and time consumptions. Starting from the next chapter, we will focus on designing efficient protocols for particular queries, which may be combined with our work in this chapter depending on application models.

Chapter 3

Basic Query in Sensor Networks: Range Query

The previous chapter has discussed general design of network architecture that support all queries. From this chapter, we present upper layer protocol design for particular queries. We start with a basic query in sensor networks, range query, which is adopted in many applications. As mentioned in Chapter 1, we investigate this query with more practical settings. Generally, many sensor network applications may have other objectives or requirements besides efficiency. Multiple design objectives in the same sensor network may not coincide with each other and there are often tradoffs among them. It is challenging to optimize the query protocol with multiple conflicting objectives. Our work in this chapter, therefore, shows how to design an efficient protocol for the range query with other requirements that incur additional energy consumption. In particular, we focus on the applications with security and privacy concerns. Our solution achieves two goals in range query, protecting data privacy and verifying the reply, with low extra energy overhead.

3.1 Related Work

Data privacy and security have attracted lots of work in database system ([9–12, 63, 67, 131]). The database server might not be trusted in "Database as a Service" model [63] or outsourced database [67]. In [63], the authors considered privacy problems in a model, where the service provider might not be trusted and thus the data owner encrypts the data before sending it out. The authors proposed a data partition/bucketization scheme to allow the service provider to process queries without decryption. Finally, the results are decrypted and processed at the client site. In another work, the privacy issues of outsourced database are also discussed in [67]. B. Hore et al. investigated data bucketing scheme and analyzed the tradeoff between performance and privacy. They further gave two measurements of privacy and designed an algorithm to optimize the performance. Our work uses the same privacy metrics in [67] and [9], and apply them in sensor networks. In prior work, however, data providers are assumed to be just curious about sensitive data, but not to act in a malicious way. Our work, however, considers more powerful malicious attacks to a sensor network deployed in a hostile environment. Preserving privacy and detecting malicious behaviors are the two integral goals in our work. In addition, we consider communication efficiency in sensor networks, which is not presented in [67] and [9].

Another related research is privacy protection of documents stored on untrusted sites ([30, 56, 118]). D. Song et al. [118] described several schemes for keyword searching on encrypted data. Similarly, they considered the privacy issues with an untrusted storage server. The same problem is also discussed in [30]. Y. Chang et al. resolved the problem by using a dictionary and interactive protocol. In addition, P. Golle et al. proposed protocols particularly for conjunctive keyword search in [56]. The work in this line considers keyword search in a setting where the data provider is the one who issues the search operation. Our work considers a different application –

range query assuming the users query the data provided by sensors.

Prior research about privacy issue in sensor networks ([60,75,123,132,135]) focus on security and privacy for the location of the source sensor, not the data information. In addition, M. Shao et al. [112] and K. Ren et al. [109] apply cryptographic mechanism to provide security and privacy protection for data centric sensor networks and pervasive computing environment respectively. However, they do not consider data processing at storage sites.

In sensor networks, secure aggregation ([29, 68, 105, 125, 126]) is similar to our query reply verification. L. Hu and D. Evans [68] proposed a protocol to prevent intermediate aggregators transmitting false information by using MAC messages as a signature. In their design, one aggregator is able to verify the information from its children by the messages from its grandchildren. This scheme, however, does not work for the case where multiple nodes are compromised. In SIA [105], B. Przydatek et al. proposed an *aggregate-commit-prove* scheme to verify the aggregation result. Sampling theory is applied in the protocol, which enables the sink to estimate the probability that the result is within a tolerant error range. H. Chan et al. [29] extended this work to a hierarchical aggregation model with multihop communication. SDAP [125] is another solution to secure aggregation in a multihop sensor network. The authors divide the aggregation tree into groups and use a *commit-and-attest* scheme to enable the sink to verify the aggregates. However, all these approaches are not designed for privacy-sensitive data. In addition, the goal of [68] is to find malicious aggregators, not suspect data sources, and the schemes in [29,105,125] are designed only for aggregation queries. Their basic goal is to prevent malicious aggregators from forging the result. Our verification scheme, however, tries to detect the incorrect data from suspect data sources, i.e., the compromised storage nodes. In addition, some of the prior work is not suitable for range query, and some protocols release data information to other nodes, which breaches the

privacy in our problem.

3.2 Problem Formulation

3.2.1 System Model

We consider a sensor network consisting of storage nodes and regular sensors. The basic query-response model is illustrated in Fig. 3.1. We assume that every sensor generates environmental data values in a fixed rate and periodically submits the collected data to the closest storage node. For example, sensors monitor temperature every ten seconds and submit the data to storage nodes every one minute. Thus, each submission contains six temperature readings. We define an *epoch*, as the interval time between two consecutive submissions (one minute in the above example). Assume all sensors are synchronized so that they have agreement on the beginning and end of an epoch. After every epoch, the collected data is sent to the nearby storage nodes by sensors and archived there for future queries. The data messages from sensor s_i contain the following information:

$$s_i \rightarrow \text{Storage Node} : i, t, \{data1, data2, \ldots\},$$

where i is the sensor ID and t is the current value of the epoch counter. Data query from a user is directed to the storage nodes through the sink. In this work, we consider range queries in the following format $RangeQuery = \{t, [a,b]\}$, where t is the time slot (epoch) the user is interested in and $[a,b]$ is the specified data value range. For easy exposition, we only consider one-dimensional data in this work. In some applications, sensors may generate data with multiple attributes, which yield more complex range query. Our approach, however, can be easily extended to the query with multiple data types.

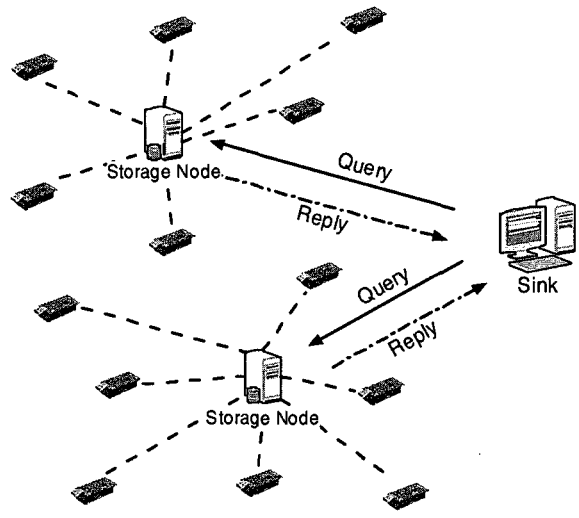


Figure 3.1: Two-tiered System Model (with two storage nodes)

3.2.2 Adversary Model and Security Goals

We assume that the adversary tries to launch the following two attacks. First, the adversary wants to obtain the sensitive data information from the sensor network, which violates *data privacy*. Leaking valuable data is a critical threat in many applications. The second attack is to breach *data fidelity*. For a user's query, the adversary tries to reply with wrong information and convince the user to accept it. We consider that both storage nodes and regular sensors might be compromised in a hostile environment. We suppose that a compromised node is fully controlled by the adversary. The adversary may utilize any compromised resource to launch attacks. In the rest of this subsection, we discuss the impacts of the compromised storage nodes and regular sensors, and propose our corresponding security goals.

3.2.2.1 Compromised Storage Nodes

Our major focus is on the compromised storage nodes. Since storage nodes host a lot of data collected from other regular sensors, compromising storage nodes will cause great damage to

the system. First, once compromising a storage node, the adversary easily obtains the privacy-sensitive data stored on the storage node. Second, the compromised storage nodes can help the adversary launch the *data fidelity* attack, because storage nodes are responsible for answering queries from the sink. After receiving a query, the compromised storage nodes may return arbitrary data as the reply. Therefore, the goal of this work is to protect data privacy and data fidelity. We aim to protect *data privacy* by designing a storage scheme, such that little information is exposed to storage nodes while fulfilling data queries. *Data fidelity* attack, however, is hard to prevent, because the compromised storage nodes under the control of the adversary may behave arbitrarily. Our countermeasure is an approach to enabling the sink to detect and reject the false reply so that applications will not be affected by misleading data.

3.2.2.2 Compromised Sensors

Regular sensors are data source in this system. If one regular sensor is compromised, the readings of the sensor will be exposed and the sensor may send forged data to storage nodes. Unfortunately, it is hard to prevent the data privacy attack and data fidelity attack in this scenario. However, the data from an individual sensor is minor in the whole network. Unless the adversary compromises a lot of regular sensors, this kind of attack has a very limited impact.

Compromised sensors, however, may be helpful for the adversary who has also compromised some storage nodes. The adversary may use the information from the compromised regular sensors to disclose other large amount of sensitive data, which are sent by other sensors or the compromised sensors in the past epochs. In addition, these information may help the adversary generate a false reply to fool the sink. Therefore, when we design a protection scheme, we ought to minimize the dependency among different sensors and epochs.

3.3 Methodology

3.3.1 Storage Scheme and Query Protocol

In this section, we propose our schemes to address the privacy and security issues discussed in the previous section. Our solution includes two components. The first is a privacy-preserving storage scheme for storage nodes to protect data privacy and the second is a query protocol to yield a verifiable reply for the sink to enable data fidelity. We will describe the details in the rest of this section. The following Table 3.1 lists some notations we will use in the rest of this chapter.

n/s_i	the number of sensors / sensor i
t	the epoch value
s	the amount of data generated in each epoch
$k_{i,t}$	the secret key of sensor s_i at epoch t
H	hash function
Q_i	the ith query in the query set
v_{max}/v_{min}	the maximum / minimum data value
α/δ	requirement of data lose ratio / confidence
VAR_p/EN_p	requirement of variance / entropy
PT_i	probability that a data is with tag T_i
d_{ss}	distance between a storage node and the sink
d_{avg}	average distance between a storage node and the associated sensors

Table 3.1: Notations

3.3.1.1 Privacy-Preserving Storage

We first discuss the protection of data privacy, i.e., preventing data from being disclosed to storage nodes. For this purpose, storing plaintext data on storage nodes is not desirable. Instead, each sensor must encrypt the data before sending them to the storage node. We assume that every sensor shares a secret key with the sink for a certain epoch, which makes up a one-way key chain. Let $k_{i,t}$ represent the secret key of sensor s_i at epoch t,

$$k_{i,t} = hash(k_{i,t-1}).$$

After an epoch, a new key is generated by the embedded hash function and the old key is erased from the sensor. The initial key $k_{i,0}$ can be preloaded before deployment. Secure protocols for key establishment such as MIB [86] can further protect the initial phase. Considering a long-term application, the overhead of this initial phase is negligible. In addition, the epoch counter t keeps increasing and will be reset to 0 periodically by applications. In the new cycle, the initial key will be the hash value of the last key in the previous cycle. In our design, compromising a sensor s_i as well as the nearby storage node does not lead to the disclosure of the data from s_i generated before the compromise. Each sensor possesses a distinct key chain so that compromising one sensor does not affect the security of another sensor's data. After the sink receives the query reply from storage nodes, the shared key between the sink and the corresponding sensor assists to decrypt the received data.

Leaking no information to the storage nodes provides good privacy, but does not help with replying a range query: the storage nodes have to send *all* the stored data back to the sink for a query request, which consumes too much energy. Our solution is to expose some information to the storage nodes while a good level of privacy is still maintained. We adopt the bucketing scheme

in [63], and associate a tag with each encrypted data. In this approach, the value domain is assumed to be discrete and divided into multiple buckets. There is no overlap or gap between consecutive buckets, i.e., every value is covered by exactly one bucket, and each bucket is assigned with a tag. Assume sensors and the sink have agreed on the same bucket partition in the initialization phase. When sending data to the storage nodes, sensors attach the corresponding tag to every encrypted data based on which bucket the data falls into. The data values with the same tag can be encrypted as a block. For example, a sensor s_i may send the following to the storage node:

$$\begin{aligned} s_i \rightarrow \text{Storage Node} \quad : \quad & i, t, \\ & \{\text{Tag1}, \{data1, data2\}_{k_{i,t}}\}, \\ & \{\text{Tag2}, \{data3\}_{k_{i,t}}\}, \ldots, \end{aligned}$$

where $data1$ and $data2$ are in the same bucket with Tag1.

For a user query $\{t, [a,b]\}$, the sink first translates the value range into a list of tags which are associated to the smallest set of buckets that cover the range $[a,b]$. Therefore, the query sent to storage nodes is composed of this list of eligible tags, instead of a and b, for example:

$$\text{Sink} \rightarrow \text{Storage Node} : t, \{\text{Tag1}, \text{Tag2}, \ldots\}.$$

Storage nodes will look up all the data generated in epoch t and return those whose tags are listed in the query. We will discuss how to define each bucket in the next section.

3.3.1.2 Verifiable Reply

As we mentioned earlier, if storage nodes behave maliciously, they may send back arbitrary data as the query reply. In this subsection, we discuss the counter schemes to detect the false reply

of a range query. More precisely, there are three possibilities for a storage node to cheat on a range query reply. First, a storage node can forge a non-existent data value for the query reply. The forged data can be easily detected because each valid data is encrypted by a key shared by the sink and the sensor who generates the data. Second, the storage node may reply with a valid encrypted data that is out of the required query range. The sink can also easily detect the cheating by decrypting the data and comparing with the query range. Third, a storage node may return partial portion of the requested data, which constructs an *incomplete reply*. In this work, we focus on detecting the *incomplete reply*.

Assume there are m tags, labeled as $T_1, T_2, \cdots, T_m$. Recall that when a sensor s_i sends data at the end of an epoch, all the data with the same tag are encrypted in bulk. If a storage node wants to drop the data with tag T_j, it has to drop the entire data block and pretends that no data with tag T_j has been received from sensor s_i in the specified epoch. In the next, we propose to use *encoding number* to detect the *incomplete reply*. We assume that the sink is aware of the association between sensors and storage nodes, i.e., which storage nodes store the data from which regular sensors. Our basic idea is to require a sensor to send the storage node an encoding number for a tag if the sensor has no data associated with the tag. This encoding number shares a similar format as a HMAC and is generated by a hash function on the secret key $k_{i,t}$. The encoding number will be requested by the sink, when the storage node claims that a sensor has no data with the tag. The sink is able to verify the authenticity of the received secrets. In this way, if a compromised storage node drops some data, it has to guess the encoding number to pass the verification at the sink. With careful design, our scheme can detect a false reply with high confidence.

The details of our design are as follows. For each tag T_j, every sensor s_i is able to generate a D_j-bit encoding number based on a predefined hash function H. Here D_j is a system parameter

and we will discuss how to set this value in the next section. Let $num(i,j,t)$ represent s_i's encoding number for tag T_j after epoch t. The encoding number is defined as

$$num(i,j,t) = H(j||k_{i,t}) \bmod 2^{D_j},$$

where $||$ means concatenating operation. After sending all data gathered during the past epoch to the storage node, each sensor also generates and sends the encoding numbers for those tags that have no data associated with to the storage node. For example, assume s_i generates some data with tag T_1, but no data with T_2 during epoch t. It will send to the storage node data in the following format:

$$\begin{aligned} s_i \rightarrow \text{Storage Node} \quad : \quad & i,t, \\ & \{T_1, \{data1, data2, \cdots\}_{k_{i,t}}\}, \\ & \{T_2, num(i,2,t)\}, \ldots \end{aligned}$$

To respond to a range query, in addition to finding all data matching the query range, a storage node generates a digest to show that it knows all the received encoding numbers for the tags within the query range. In fact, the storage node can send all received encoding numbers as a digest. However, to reduce the message size, our scheme uses a hashed value of the encoding numbers instead. First, for each encoding number in epoch t ($num(i,j,t)$), the storage node generates a hash value

$$c(i,j,t) = H(i||j||t||num(i,j,t)).$$

Then, the storage node concatenates these hash values $c(i,j,t)$ in the order of (i,j) pairs. This ordering is to enable the sink to reconstruct the digest later. Finally, the digest is obtained by

applying the hash function H on the concatenation of $c(i,j,t)$, Digest $= H(||c(i,j,t))$. This digest is included in the return message to the sink.

	T_1	T_2	T_3	T_4
s_1	X	X	001	101
s_2	X	X	X	010
s_3	101	X	011	X
s_4	X	110	010	100
s_5	X	100	X	X

Table 3.2: Each row represents the data sent by one sensor. We use 'X' to denote some data with the tag is generated, otherwise, a three-bit encoding number is received.

For example, assume there are 5 sensors $\{s_1, s_2, s_3, s_4, s_5\}$ and 4 tags $\{T_1, T_2, T_3, T_4\}$. Table 3.2 details the data received by storage nodes at epoch t. Consider a query for $\{T_1, T_2, T_3\}$, the digest is constructed as follows. We first generate

$$c(1,3,t) = H(1||3||t||001) \quad , \quad c(3,1,t) = H(3||1||t||101),$$

$$c(3,3,t) = H(3||3||t||011) \quad , \quad c(4,2,t) = H(4||2||t||110),$$

$$c(4,3,t) = H(4||3||t||010) \quad , \quad c(5,2,t) = H(5||2||t||100).$$

Then, we apply H to obtain the digest.

$$\begin{aligned} \text{Digest} \quad = \quad & H(c(1,3,t)||c(3,1,t)||c(3,3,t) \\ & ||c(4,2,t)||c(4,3,t)||c(5,2,t)). \end{aligned}$$

After calculating the digest, the storage node returns the following message to the sink:

$$\text{Storage Node} \rightarrow \text{Sink} : \text{Digest},$$

$$\{T_1, \{X\}_{k_{1,t}}, \{X\}_{k_{2,t}}, \{X\}_{k_{4,t}}, \{X\}_{k_{5,t}}\},$$

$$\{T_2, \{X\}_{k_{1,t}}, \{X\}_{k_{2,t}}, \{X\}_{k_{3,t}}\},$$

$$\{T_3, \{X\}_{k_{2,t}}, \{X\}_{k_{5,t}}\}.$$

When the sink receives the reply, it can reconstruct the encoding numbers and the digest based on the received data because it knows all secret keys. The sink compares it with the received digest and the validity of the reply is verified if they match.

3.3.1.3 Security Analysis

In this subsection, we discuss some potential security issues if storage nodes are compromised and how our protocols deal with them. It is possible that some regular sensors are also compromised by the same adversary.

Beach data privacy: Once a storage node is compromised, all the data stored there are disclosed to the adversary. In our scheme, however, these data are encrypted by symmetric keys. The adversary cannot obtain the data values unless they can break the symmetric key cryptosystem. In a feasible attack, the adversary can guess the data value according to the tag associated to the encrypted data. After compromising the storage node, the adversary is aware of the bucket partition, i.e., the value range each tag represents. Intuitively, for a tag representing a shorter value range, the adversary's guess is more likely to be closer to the actual value. Whether or not this attack can breach the privacy depends on the bucket partition and the application-specified requirements for privacy. In the next section, we will present how to quantify the privacy requirements and how to

define the buckets to satisfy these requirements.

Obtain each sensor's secret key: In our scheme, the adversary cannot obtain the secret key $k_{i,t}$ of sensor s_i at epoch t through eavesdropping or compromising storage nodes. The possible available information to the adversary is the encoding numbers for those tags the sensor has generated no data with. In our scheme, these encoding numbers are generated by a hash function on the secret $k_{i,t}$. Since it is computationally infeasible to invert a hash function, the adversary cannot derive the secret key from the encoding numbers.

Forge the digest: In order to launch an *incomplete reply* attack, the compromised storage node has to drop some data and generate a digest to pass the verification at the sink side. In our scheme, however, the adversary does not have enough information (all necessary encoding numbers) to surely generate a valid digest for the incomplete reply. The adversary may know partial encoding numbers for the digest. But it does not provide any clue about the valid digest generated by a hash function. Thus, the adversary can only forge the digest by guessing. In our scheme, we set the digest to be sufficiently long (e.g., 10-bit), so that a direct guess of the valid digest is very unlikely to be correct (e.g., with probability of $\frac{1}{2^{10}} < 0.1\%$). Another alternative for the adversary to forge the digest is to forge the missing encoding numbers and apply function H to generate a digest. We will discuss it in the next.

Forge the encoding numbers: The compromised storage node may forge some encoding numbers it has not received to generate a valid digest for an incomplete reply attack. In our scheme, each encoding number $num(i, j, t)$ is generated by a hash function and unique per sensor/tag/epoch. The adversary may receive other encoding numbers with one or two identical parameters, but cannot obtain any hints to derive the missing encoding number. Therefore, the encoding numbers can only be forge by blindly guessing. We will discuss the possibility of successfully guessing the

encoding numbers in the next section.

Malicious regular sensors: It is possible that some regular sensors may be faulty, dysfunctional, or even malicious after being compromised. The encoding numbers from those sensors may be incorrect or missing at storage nodes. In this case, storage nodes simply report to the sink about those abnormal sensors when replying a query. Since the main objective of this work is to detect malicious behavior, informing the sink of the faulty sensors is sufficient for further actions.

3.3.2 Finding the Optimal Parameters

In the previous section, we introduced a bucketing scheme to protect data privacy and encoding numbers to verify a reply. How to divide the value range into buckets and determine the length for encoding number is still a problem. In the rest of this section, we formulate the problem of setting parameters as an optimization problem with three system performance metrics, and discuss how to solve the problem in this setting.

Assume that a storage node is in charge of n sensors and each sensor generates s readings per epoch. Every data value is considered discrete at some precision level. Also we assume that the data generated by every sensor follows the same distribution $F(x)$ (the probability that a certain sensed value is x), which can be obtained from theoretical models or empirical data. In addition, the query characteristics, i.e., range specification and query frequency, need to be accounted as well to set the optimal parameters. We consider a complete range query set represented as $\{Q_i\}$,

$$Q_i = \{(t_i, [a_i, b_i])\}, a_i \in [v_{min}, v_{max}], b_i \in [a_i, v_{max}],$$

where v_{min} and v_{max} are the minimum and maximum values of the collected data, and t_i can be any past epoch, and there does not exist another Q_j, such that $a_i = a_j$ and $b_i = b_j$. Let L be the value

range, $L = v_{max} - v_{min} + 1$. Thus, there are $\frac{L(L+1)}{2}$ possible ranges in this set. For the purpose of a generalized analysis, we assume that the sink has the same possibility to receive a query for any range and it receives the queries to all possible ranges during c epochs.

3.3.2.1 System Performance Metrics

In this subsection, we introduce three performance metrics, which are crucial to the design of our scheme. Privacy and security metrics describe the robustness to data privacy and data fidelity attacks. Communication cost is the metric for energy efficiency. We define these metrics mathematically as follows.

Privacy Constraints: While bucketing scheme enables storage nodes to search data with tags, it may potentially lead to privacy breach. For example, let's consider an extreme case in which every distinct value has a unique tag. If a sensor is compromised, the value-tag mapping will be exposed to the adversary. He can derive all data values stored on the compromised storage node, even if the data is encrypted. Therefore, we should reduce this information leakage caused by the value-tag mapping. To do that, we need a way to measure the level of privacy for a bucket scheme.

In this work, we use *variance* and *entropy* to measure the privacy protection of a bucket as proposed in [67]. Essentially, we protect data against two types of privacy attacks. First, storage nodes may guess the actual value of stored data from the associated tag. *Variance* of value distribution of the data with a certain tag represents the protection level of this attack, i.e., the hardness to guess the data. Second, when query messages arrive, storage nodes may try to derive the exact value range (i.e., lower/upper bounds) from the list of tags in the query message. *Entropy* is chosen to measure this query privacy. Larger variance and entropy indicate better protection of privacy. In fact, our design does not restrict to these two measurements introduced in [67]. Some

applications may have different definitions of the privacy measurements and it is easy to modify our scheme accordingly.

For a given tag T_i defined by range $[l_i, h_i], l_i \leq h_i$, the variance and entropy can be calculated as follows. Let $\bar{E}_i$ be the expected value within this range and PT_i be the probability that a value belongs to this range,

$$\bar{E}_i = \sum_{x=l_i}^{h_i} F(x) \cdot x, \tag{3.1}$$

$$PT_i = \sum_{x=l_i}^{h_i} F(x). \tag{3.2}$$

According to the definitions of variance and entropy, we have

$$\text{variance} = \sum_{x=l_i}^{h_i} F(x)(x - \bar{E}_i)^2; \tag{3.3}$$

$$\text{entropy} = -\sum_{x=l_i}^{h_i} \frac{F(x)}{PT_i} \log \frac{F(x)}{PT_i}. \tag{3.4}$$

Applications may specify the requirements for these two metrics, indicated by VAR_p and EN_p respectively. In a valid bucketing plan, for any bucket, the variance and entropy must be greater than VAR_p and EN_p respectively. Thus, T_i is valid if its variance $> VAR_p$ and entropy $> EN_p$.

Security Constraints: Encoding number scheme proposed earlier is not perfectly secure. There is still a certain probability that the adversary can forge encoding numbers correctly to pass the verification, in particular when the length of the encoding number is short (say one bit, or less than one bit when multiple tags combined share one encoding number). We define the security level of a set of encoding numbers as follows:

Definition 1 **α-valid/false reply**: *We say a reply is α-valid if the dropped data is less than α portion of the total expected data. A reply, which is not α-valid, is called a* **false reply**.

Definition 2 (α, δ)**-secure encoding numbers**: *We say that a set of encoding numbers are* (α, δ)*-secure, if the confidence of accepting an* α*-valid reply, i.e., the probability of detecting false reply, is greater than* δ.

The first parameter α defines data fidelity, which is the fraction of data loss we can tolerate over the amount of data that should be returned for a range query. Data reply confidence δ, is the probability that we can detect a false reply. Given user specified α and δ, our resulting encoding numbers must be (α, δ)-secure.

Communication Cost: With security protection, extra communication cost is incurred in data collection and query reply. The objective in this problem is to minimize the communication cost during c epochs, which includes the cost of transferring data from sensors to storage nodes and from storage nodes to the sink. In this section, we analyze the costs and give an expression of the objective function.

First, the bucketing scheme incurs a problem of *false positive* [67]. Some useless data are sent back together with the desired data. We define *false positive* as the total amount of the useless data received by the sink. Consider a tag T_i defined by the range of $[l_i, h_i]$. For a range query $[a, b]$, T_i yields no false positive if there is no overlap between $[a, b]$ and $[l_i, h_i]$, i.e., $b < l_i$ or $a > h_i$. However, if $l_i \leq b < h_i$, the data in the range of $[b+1, h_i]$, which size is $n \cdot s \cdot \sum_{x=b+1}^{h_i} F(x)$, are also returned. Considering the complete query set, for a certain b, a belongs to $[v_{min}, b]$, which yields $b - v_{min} + 1$ queries. Thus, the false positive in $[l_i, h_i]$ caused by the data out of a query's upper bound (between b and h_i, $(b, h_i]$) is

$$\sum_{b=l_i}^{h_i-1} (b - v_{min} + 1) \cdot n \cdot s \cdot \sum_{x=b+1}^{h_i} F(x).$$

Similarly, if $l_i < a \leq h_i$, the data in $[l_i, a-1]$ becomes false positive. In addition, we assume the

cost of transferring data is proportional to the data size and the distance between the sender and receiver. Therefore, considering the complete query set, the total cost for transferring the false positive incurred by T_i, denoted by CF_i, is

$$CF_i = d_{ss} \cdot n \cdot s(\sum_{j=l_i}^{h_i-1} (j - v_{min} + 1) \sum_{x=j+1}^{h_i} F(x) + \sum_{j=l_i+1}^{h_i} (v_{max} - j + 1) \sum_{x=l_i}^{j-1} F(x)),$$

where d_{ss} is the distance between the storage node and sink.

Similar to privacy protection, encoding number scheme incurs extra costs, too. First, when storage nodes reply to a query, a digest is attached to the message. The sensors relaying the message will consume more costs. This cost, however, is constant in this scheme. We do not have to consider it when determining buckets plan and encoding numbers. Second, when sensors send data to storage nodes, they need send the encoding numbers for the tags with no data associated as well. The cost of transferring encoding numbers depends on bucket partition, the length of each encoding number, the number of sensors in the proximity, and the distance between sensors and their closest storage nodes. For a tag T_i, the probability that one sensor has no data with T_i is $(1 - PT_i)^s$. Thus, the expected number of those sensors which have no data with T_i in an epoch is $n \cdot (1 - PT_i)^s$. This is the number of sensors that have to send the encoding number for T_i to storage nodes. Therefore, for each epoch, the expected communication cost for transferring the encoding numbers for T_i is

$$D_i \cdot n \cdot (1 - PT_i)^s \cdot d_{avg},$$

where d_{avg} is the average distance between sensors and the storage node and recall D_i is the length of the encoding number for T_i. Let CE_i be the cost of transferring the encoding numbers of T_i during c epochs,

$$CE_i = c \cdot D_i \cdot n \cdot (1 - PT_i)^s \cdot d_{avg}. \tag{3.5}$$

The secure protocols we proposed may also incur extra cost for computation such as hash operations. However, this extra cost for computation is negligible compared to the communication cost.

3.3.2.2 Problem Formulation

Considering all the metrics discussed above, our problem is formally defined as follows:

$$\begin{aligned} \text{Input:} \quad & F, VAR_p, EN_p, \alpha, \delta \\ \text{Output:} \quad & \text{Bucket partition } (T_i) \text{ \& encoding numbers } (D_i) \\ \text{Objective:} \quad & \min \sum_i (CF_i + CE_i) \\ s.t. \quad & \forall T_i, \text{variance} > VAR_p \text{ and entropy} > EN_p; \\ & \{D_i\} \text{ is } (\alpha, \delta)\text{-secure.} \end{aligned} \tag{3.6}$$

That is, given the sensed data distribution $F(x)$, privacy parameters VAR_p and EN_p, and security parameters α and δ, we aim to find the optimal bucket partition (T_i) and encoding numbers (D_i), such that the communication cost ($\sum_i (CF_i + CE_i)$) is minimized while the privacy requirements (in terms of variance and entropy) and the security requirement ((α, δ)-secure) are guaranteed.

3.3.2.3 Algorithm to Find the Optimal Parameters

As shown above, our problem boils down to determining the optimal bucket scheme and the optimal length for each encoding number. We call the bit length of an encoding number *encoding length* in the rest of this work. Our main algorithm uses dynamic programming to enumerate all bucket partition schemes. For each bucket partition, we first check the privacy constraints and call another algorithm to calculate the encoding lengths which can guarantee the security constraints.

Then, we can obtain the communication cost incurred by the bucket partition. After examining all bucket partition plans, our algorithm can find the optimal one with the minimum communication cost.

Main Algorithm: In this subsection, we describe the main algorithm to divide the value range into buckets such that the communication cost is minimized while the security and privacy constraints are satisfied. We use dynamic programming to resolve the problem in the following Algorithm 4. It basically is composed of two phases. In the first phase (lines 1-7), we enumerate all possible ranges $[i, j]$ by two loops. We first check if each range is eligible to be valid buckets according to the privacy constraints and store the results in a boolean array $valid[i, j]$. For each valid range $[i, j]$, i.e., $valid[i, j]$ is true, we calculate an encoding length $D[i, j]$ by another function *EncodingLength*. We will discuss the details of this function in the next subsection. Basically, for a given range, it returns the shortest encoding length that can guarantee the security constraint. Then in line 7, we compute the communication cost incurred by this range for transferring false positive data (Eq.(3.5)) and encoding numbers (Eq.(3.5)). The time complexity of this phase is $O(L^2 \cdot \max\{L^2, s\})$, where L is the value range as defined earlier. In the second phase, we define a two dimensional matrix M, where each element $M[i, j]$ stores the cost of the best solution to divide range $[i, j]$. We use dynamic programming to fill matrix M and finally $M[v_{min}, v_{max}]$ is the cost of the optimal bucket partition. We start from the smallest ranges with width 1 and calculate $M[i, j]$ in the ascending order of the range width $w = j - i$. Dividing $[i, j]$ can be regarded as a two-step process: defining the first bucket and recursively dividing the remaining range. Let $[i, k]$ be the first bucket. We enumerate all possible positions of k and $M[i, j]$ is obtained by the following equation,

$$M[i, j] = \min\{CE[i,k] + CF[i,k] + M[k+1, j]\},$$

where $k \in [i, j]$ and $valid[i,k] = true$. Additionally, another matrix P is used to record the pivot points of range partition. By tracing back from $P[v_{min}, v_{max}]$, we can obtain the optimal bucket partition. The time complexity of the second step is $O(L^3)$. Therefore, the algorithm terminates within $O(L^2 \cdot \max\{L^2, s\})$ steps.

Algorithm 4 Optimal Solution $(F, VAR_p, EN_p, \alpha, \delta)$

```
1: for i = v_min to v_max do
2:   for j = i to v_max do
3:     Calculate Ē[i, j] and PT[i, j] by Eq.(3.1) and Eq.(3.2)
4:     Calculate variance and entropy by Eq.(3.3) and Eq.(3.4)
5:     if variance > VAR_p and entropy > EN_p then
6:       valid[i, j] = true, D[i, j] = EncodingLength([i, j])
7:       COST[i, j] = Eq.(3.5)+Eq.(3.5)
8:     end if
9:   end for
10: end for
11: for w = 1 to v_max − v_min + 1 do
12:   for i = 1 to v_max − w do
13:     if valid[i, i + w] then
14:       M[i, i + w] = COST[i, j]
15:       for j = 1 to w − 1 do
16:         if valid[i, i + j] then
17:           cost = COST[i, i + j] + M[i + j + 1, i + w]
18:           if cost < M[i, i + w] then
19:             M[i, i + w] = cost
20:             P[i, i + w] = j
21:           end if
22:         end if
23:       end for
24:     end if
25:   end for
26: end for
```

Optimal Encoding Length: Here we present the details of *Encodinglength*. Apparently, a long bit length increases the communication cost, and this increase is non-negligible when a large number of sensors send encoding numbers during many epochs. The security level, i.e., the probability of detecting an incomplete reply, also increases with a long bit length, which hardens the process for a storage node to forge the encoding numbers. In this sub-problem, therefore, our goal is to find the optimal set of encoding lengths, which are (α, δ)-secure and yields the minimum communication cost.

To resolve this sub-problem, we first analyze the behavior of a malicious storage node, and then give an approximated estimation of the required encoding lengths. Essentially, malicious storage nodes intend to drop enough data to form a false reply and forge the missing encoding numbers to pass the verification at the sink. Let us consider a range query with a tag list $TQ = \{T_{q_1}, T_{q_2}, \cdots, T_{q_k}\}$ for the data collected in epoch t. Storage nodes are supposed to look up all data generated during epoch t and return the data whose tag is in TQ. We define two 2-dimension matrixes SD and N, where SD_{ij} represents the set of data from sensor s_i with tag T_j and N_{ij} is the

	T_1	T_2	$\cdots$	T_m
s_1	SD_{11}	SD_{12}	$\cdots$	SD_{1m}
s_2	SD_{21}	SD_{22}	$\cdots$	SD_{2m}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_n	SD_{n1}	SD_{n2}	$\cdots$	SD_{nm}

size of SD_{ij}, i.e., $N_{ij} = |SD_{ij}|$. Thus, the size of reply data for TQ is

$$R_N(TQ) = \sum_{i=1}^{n} \sum_{T_j \in TQ} N_{ij}.$$

A successful attack requires a malicious storage node to drop at least $\alpha \cdot R_N(TQ)$ data and forge the necessary encoding numbers to get approved. Consider the malicious storage node applies the optimal way to achieve this goal, i.e., drop those data with the minimum probability being detected. Let us regard all elements of SD as individual blocks and label those blocks which should be returned for TQ as $\{b_1, b_2, \cdots, b_r\}$. For example, assume there are 3 sensors and the tags listed in TQ are T_1 and T_2. The following Table 3.3 illustrates the data received by the storage node. In this case, we need consider 3 blocks as shown in Table 3.4. For a block b_j with tag T_q,

	T_1	T_2
s_1	X	X
s_2	O	O
s_3	O	X

Table 3.3: 'X' means there are data in the block received by the storage node. 'O' means no data are received.

	T_1	T_2
s_1	$b_1 = SD_{11}$	$b_2 = SD_{12}$
s_2		
s_3		$b_3 = SD_{32}$

Table 3.4: Renumber the three blocks that should be returned for TQ.

we associate an encoding length d_j with it, where $d_j = D_q$. One block is the minimum bulk of data the storage node can drop and if b_j is removed, the probability of successfully forging the encoding number is $\frac{1}{2^{d_j}}$. Thus, given $B = \{b_1, b_2, \cdots, b_r\}$ and $\{d_1, d_2, \cdots, d_r\}$, the storage node

need find a subset B' of B to

$$\text{maximize} \prod_{b_i \in B'} \frac{1}{2^{d_i}}$$

$$s.t. \sum_{b_i \in B'} |b_i| \geq \alpha \cdot R_N(TQ).$$

The objective is equivalent to maximize

$$\log \prod_{b_i \in B'} \frac{1}{2^{d_i}} = \sum_{b_i \in B'} \log \frac{1}{2^{d_i}} = -\sum_{b_i \in B'} d_i.$$

This problem is reducible to the 0/1 knapsack problem, which is known to be NP-hard. We define x_i as

$$x_i = \begin{cases} 1 & \text{if } b_i \notin B'; \\ 0 & \text{if } b_i \in B'. \end{cases}$$

Thus, the objective will be

$$\text{maximize}(-\sum_{b_i \in B'} d_i) \Rightarrow \text{maximize} \sum_{b_i \notin B'} d_i \Rightarrow \text{maximize } d_i \cdot x_i.$$

The constraint can also be expressed in the following form,

$$\begin{aligned} \sum_{b_i \in B'} |b_i| \geq \alpha \cdot R_N(TQ) \quad &\Rightarrow \quad \sum_{b_i \notin B'} |b_i| < (1-\alpha) \cdot R_N(TQ) \\ &\Rightarrow \quad |b_i| \cdot x_i < (1-\alpha) \cdot R_N(TQ). \end{aligned}$$

Then this problem is formulated as the 0/1 knapsack problem,

$$\text{maximize } d_i \cdot x_i$$

$$s.t. \quad |b_i| \cdot x_i < (1-\alpha) \cdot R_N(TQ), \;\; x_i \in \{0,1\},$$

where d_i is the value of item i, $|b_i|$ is the weight of item i, and $(1-\alpha)\cdot R_N(TQ)$ is the capacity of the bag.

To simplify the problem, we assume that the storage node applies a greedy algorithm as the attack strategy to select victim blocks. It first orders all blocks according to the values of $\frac{d_i}{|b_i|}$, where $|b_i|$ is the number of data in b_i. In the ascending order, the storage node drops the blocks with the smallest values until the total dropped data is larger than $\alpha \cdot R_N(TQ)$

Next, we present our algorithm to determine the optimal encoding lengths that are (α,δ)-secure for any possible query. We first give an algorithm to determine the optimal encoding lengths for a special category of queries, called *single tag query*, where the tag list in the query contains only one tag. Later we extend it to more general queries with multiple tags. Recall tag T_i is defined by a range $[l_i,h_i]$ and D_i denotes the encoding length of this tag. Algorithm 5 shows the detailed function of deriving a proper value of D_i. In the first step, we estimate the expected number of sensors which have t number of data with T_i, where $t \in [1,s]$, and store them in an array E_i. According to binomial distribution,

$$E_i[t] = n \cdot \binom{s}{t} \cdot PT_i^t \cdot (1-PT_i)^{s-t}.$$

Also, we calculate the expected total number of data with T_i as $sum_i = n \cdot s \cdot PT_i$. Secondly, we emulate the behavior of malicious storage nodes, dropping data by the greedy strategy. Since we are considering single tag queries, the encoding length d_j of every eligible block b_j is the same as D_i. Thus, the dropping order only depends on $\frac{1}{|b_j|}$, i.e., the block with the largest size $|b_j|$ will be dropped first. We start with the sensors which have s data with T_i, because $|b_j| \leq s$. Totally, they contribute $s \cdot E_i[s]$ data, but to drop all of them, we have to forge $E_i[s]$ encoding numbers. We continue to drop the data from the sensors which have $s-1$ data with T_i, and stop the procedure

Algorithm 5 EncodingLength ($T_i = [l_i, h_i]$)

for $t = 1$ to s **do**

$E_i[t] = n \cdot \binom{s}{t} \cdot PT_i^t \cdot (1 - PT_i)^{s-t}$

end for

$sum_i = n \cdot s \cdot PT_i, drop = 0, enum = 0$

for $t = s$ to 1 **do**

$drop = drop + E_i[t] \cdot t$

$enum = enum + E_i[t]$

if $drop > \alpha \cdot sum_i$ **then**

$enum = enum - (drop - \alpha \cdot sum_i)/t$

break

end if

end for

return $\lceil \frac{-\log(1-\delta)}{enum} \rceil$

when the dropped data is greater than $\alpha \cdot sum_i$. During this process, variable *drop* indicates the total amount of the dropped data, and variable *enum* records the number of encoding numbers the adversary has to forge. Thus, the estimated confidence of detecting a false reply is $1 - \frac{1}{2^{D_i \cdot enum}}$. To make it greater than δ, we have

$$1 - \frac{1}{2^{D_i \cdot enum}} > \delta \Rightarrow D_i > \frac{-\log(1-\delta)}{enum}.$$

To minimize the communication cost, we set D_i to $\lceil \frac{-\log(1-\delta)}{enum} \rceil$. The time complexity of Algorithm 5 is $O(s)$.

For multiple tag queries, we can apply the similar analysis as above. However, this step can

be skipped because of the following lemma.

Lemma 3.1 *If a set of encoding numbers are (α, δ)-secure for every single tag query, they are also (α, δ)-secure for multiple tag queries.*

Proof: Assume that a vector of encoding lengths $D = \{D_1, D_2, \ldots, D_m\}$ are (α, δ)-secure for any single tag query. Now let us consider a multiple tag query for a list of tags $\{T_{t_1}, T_{t_2}, \ldots\}$. As in Algorithm 5, we can estimate the expected total number of data for each tag, denoted by $\{sum_{t_1}, sum_{t_2}, \ldots\}$. The summary $\sum sum_{t_i}$ will be the expected return size of this query. Then, we will apply the greedy strategy to drop at least $\alpha \cdot \sum sum_{t_i}$ data. Meanwhile, we need count the encoding numbers that have to be forged. Let $enum_{t_i}$ be the number of dropped blocks of tag T_{t_i}. The confidence will be $1 - \prod \frac{1}{2^{D_{t_i} \cdot enum_{t_i}}}$. However, in this process, there must exist a tag T_{t_j} such that the dropped data of T_{t_j} is greater than $\alpha \cdot sum_{t_j}$. We already know that D_{t_j} guarantee the confidence of single tag query for T_{t_j}, which implies $1 - \frac{1}{2^{D_{t_j} \cdot enum_{t_j}}} > \delta$. Back to the confidence of this multiple tag query,

$$1 - \prod \frac{1}{2^{D_{t_i} \cdot enum_{t_i}}} > 1 - \frac{1}{2^{D_{t_j} \cdot enum_{t_j}}} > \delta.$$

Therefore, D can also guarantee the required confidence for multiple tag queries. ■

Thus, for any given bucket, Algorithm 5 can find the optimal encoding length satisfying the security constraint.

3.3.3 Rare Event Detection with Abnormal Values

In this part, we study event detection as a special application of range query. We consider a scenario that an event can be detected by the sensors in the proximity. For example, a vehicle traversing the field generates abnormal noise and vibration, which can be measured by nearby

sensors (illustrated in Fig. 3.2). Users can query the data in the range of abnormal values to detect the event and collect the relevant information.

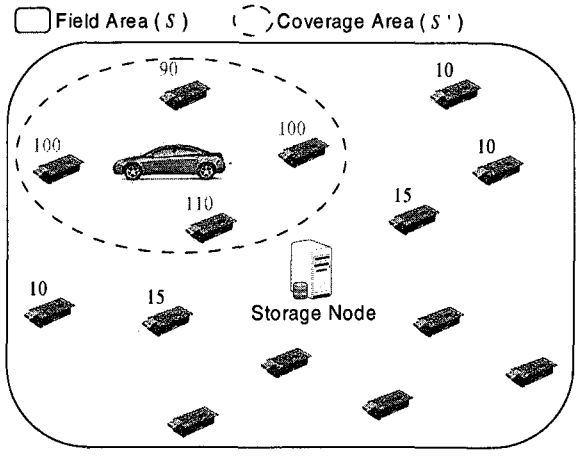


Figure 3.2: Example of event detection with range query: The sensors close to the passing vehicle measure abnormally high noise or vibration ([90,110] in this example), while the normal readings are much lower ([10-15]). Assume users know the prior information that the noise generated by a sedan is usually between 80 and 120. Thus, users can obtain the information about the event by querying the data in range [80-120].

The previously proposed schemes are suitable for general range query, but might be inefficient for detecting rare events. As we mentioned earlier, our schemes incur extra communication costs for transferring false positive data and encoding numbers. Although we may carefully design a bucket partition to minimize the false positive, the cost for transmitting encoding numbers inevitably escalates for rare events. Let us assume some tags are associated with abnormal value ranges that represent certain rare events. In most epochs, no such event occurs and every sensor has to send the corresponding encoding numbers for these tags to storage nodes. This extra cost caused by sending encoding numbers could be extremely high when accumulated over time in a large scale sensor network. Therefore, in this subsection, we propose an efficient encoding number scheme for rare event detection. For simplicity, we assume that each type of event can be detected by querying a special single tag. In reality, we may need query multiple tags for a certain event depending on the bucket partition parameters. In this subsection, however, we will not discuss the

bucket partition, but focus on the encoding number scheme. Our solution can be easily extended to the event covered by multiple tags.

The problem setting for event detection is slightly different from the previous problem in two aspects. First, we need to consider the coverage of an event, i.e., the proximity area of the event source where sensors can detect the event. This new parameter depends on the characteristics of events and the sensitivity of sensors. A larger coverage area tends to have more sensors detect the event. Second, in event detection applications, it is unnecessary for a storage node to send back all the received data about the same event. Event detection applications often take advantage of data redundancy in the sensed data among the sensors that detect the event to reduce the communication cost. The data from multiple sensors may collaboratively detect a rare event. However, after a certain threshold, obtaining more data does not yield much new information because of the redundancy. This threshold depends on the characteristics of the event and the sensed data. For example, the application may require the event data from five sensors no matter how many sensors actually detect the event. The sink will specify this threshold in the query. After receiving the query, a storage node will look up the hosted data and determine whether there exist data about the event, the storage node will bundle five of them (from five sensors) as the reply and send them back to the sink.

We modify the previous problem for this special case of rare event detection as follows. Assume a storage node is in charge of a field with area S as illustrated in Fig. 3.2. Sensors are randomly deployed on the field with a density λ and can be modeled as points of a Poisson process. Assume a rare event is associated with tag T, i.e., querying the data with T can detect the occurrence of this type of event. Let S' be the coverage area of an event. Here we use a simplified model for the rare event. When this rare event is not present, no sensor will generate data with tag

T. When an event occurs, on average $\lambda \cdot S'$ sensors will detect it. We assume that the application requires event data from $(1-\alpha) \cdot \lambda \cdot S'$ sensors if the event occurs. Note that parameter α here has a different meaning from the tolerance parameter in the previous sections. We still use α to keep consistent with our previous problem setting.

The adversary model in this problem is similar. A compromised storage node tries to drop partial or all the event data when some events have occurred and return less than $(1-\alpha) \cdot \lambda \cdot S'$ (could be none) event data as a reply. Therefore, our security goal is still to enable the sink to detect such kind of false reply with high probability.

In the rest of this subsection, we present a new encoding number scheme for rare event detection and derive the optimal parameters. Our scheme utilizes a sampling technique in order to efficiently report events. Instead of requiring all the sensors to send the encoding numbers when no event happens, we randomly choose a small set of v sensors as sample nodes to send the encoding numbers of T in each epoch. We assume that every sensor is aware of all sensor IDs in the field. In epoch t, each sensor calculates a pseudo-random function $R(t,i)$ for every sensor s_i. The top v sensors with the largest values of $R(t,i)$ are selected as sample nodes. If no event is detected in an epoch, each sample node will send out an encoding number to the storage node while a non-sample node will not. To reply a query for T, storage nodes are supposed to return the event data with tag T from $(1-\alpha) \cdot \lambda \cdot S'$ sensors. If there is no such data, i.e., no such event occurs, the storage node will send a digest generated by the encoding numbers received from sample nodes. After receiving the digest, the sink can apply the same pseudo-random function to derive the set of sample nodes and generate all the encoding numbers to verify the received digest. If the sink receives less than $(1-\alpha) \cdot \lambda \cdot S'$ event data or an invalid digest, it will discard the reply and consider the sending storage node as a malicious storage node for further investigation. Again,

remember we assume a simplified event model in which no sensor will generate data with tag T when there is no rare event.

To verify a reply is valid when an event happens, we consider two attacks the adversary may launch. First, the adversary may send partial event data ($< (1-\alpha) \cdot \lambda \cdot S'$) back. According to our policy for the sink, this reply will definitely be discarded. Second, the compromised storage node may pretend that it has not received any data about the event. In our scheme, the compromised storage node then has to send a digest back. If no sample sensor detects the event, this attack is certainly successful because the compromised storage node has obtained all necessary encoding numbers from sample nodes to generate a valid digest. However, if the storage node receives event data from some sample nodes (i.e., does not receive the encoding numbers from these sample nodes), it has to forge the encoding numbers to generate the digest for this attack to pass the verification at the sink side. In the next, we focus on the second attack and discuss how to determine the number of sample nodes v and the encoding length D for tag T such that the sink has a high probability ($> \delta$) to detect it.

Since we randomly pick v sensors as the sample nodes, the density of sample sensors is $\lambda_s = \frac{v}{S}$. Let $p(x)$ be the probability that exactly x sample sensors detect the event, which means there are x sample sensors in the area S'. Thus, according to Poisson process, we have

$$p(x) = \frac{(\lambda_s \cdot S')^x \cdot e^{-\lambda_s \cdot S'}}{x!}.$$

In this case, to drop all the data, the adversary has to guess x encoding numbers with a success probability of $\frac{1}{2^{x \cdot D}}$. Therefore, the probability we can detect the event by selecting v samples is $1 - \sum_x \frac{p(x)}{2^{x \cdot D}}$.

On the other hand, the communication cost for transmitting encoding numbers in each epoch

is $v \cdot D \cdot d_{avg}$, where d_{avg} is the average distance between a sensor and the storage node. Thus, we can find the optimal parameters by solving the following problem,

$$\text{minimize } v \cdot D$$
$$s.t. \quad 1 - \sum_x \frac{p(x)}{2^{x \cdot D}} > \delta.$$

Recall in our solution to this problem, the sink requires $(1-\alpha) \cdot \lambda \cdot S'$ event data or a digest from a storage node. It is possible that the actual number of sensors around the event source is less than the threshold $(1-\alpha) \cdot \lambda \cdot S'$, in which case a legitimate storage node can not provide sufficient event data and maybe can not generate a valid digest either. The sink then may regard this storage node as a malicious one and trigger a false alarm by mistake. The probability of the such a mistake depends on the threshold defined by α. Assume X sensors detect an event. The sink may trigger a false alarm if $X \leq (1-\alpha) \cdot \lambda \cdot S'$, whose probability is

$$Pr(X \leq (1-\alpha) \cdot \lambda \cdot S') = \sum_{X=0}^{(1-\alpha) \cdot \lambda \cdot S'} \frac{(\lambda \cdot S')^X \cdot e^{-\lambda \cdot S'}}{X!}.$$

As we will show in the evaluation, this probability can be neglected with a reasonable parameter setting.

3.4 Performance Evaluation

In this section, we evaluate our scheme based on simulation. We first evaluate the performance of the scheme for generic range queries and then show the simulation result for rare event detection. The generic privacy-preserving range query scheme is summarized in Algorithm 4, which calls subroutine Algorithm 5. To show the performance of these two algorithms in more detail, we first

examine Algorithm 5 in A and B to show that the resulting encoding length is sufficient to protect query reply, then in C we use real data sets to simulate Algorithm 4 and show the communication cost incurred by this approach. Furthermore, we present the data for rare event detection in the end.

3.4.1 Suggested Encoding Length

We first run Algorithm 5 to estimate the optimal encoding length for a single tag query. By default, we set $\{n, s, \alpha, \delta, PT_i\}$ to $\{100, 10, 0.1, 0.9, 0.1\}$. In the simulation, we fix four of these parameters and varies the remaining variable. The following figures (Fig. 3.3-Fig. 3.7) show the results of the encoding lengths suggested by Algorithm 5. On the one hand, higher confidence obviously requires longer encoding numbers, as shown in Fig. 3.4. On the other hand, the encoding length is also related to the tolerant size of the data loss. The more data loss we can tolerate, the shorter encoding length we require. To return a false reply, the adversary has to drop at least $\alpha \cdot N_i$ data, where N_i is the total number of data with tag T_i. We can use the expected value to express it, $N_i = PT_i \cdot n \cdot s$. Thus, the encoding length will be a non-increasing function over $\alpha \cdot PT_i \cdot n \cdot s$, which explains the trend of the curves in Fig. 3.3, Fig. 3.5, Fig. 3.6 and Fig. 3.7.

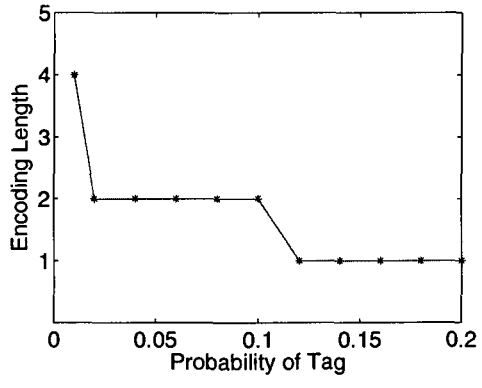


Figure 3.3: Encoding length vs. PT_i ($n = 100$, $s = 10$, $\alpha = 0.1$, $\delta = 0.9$)

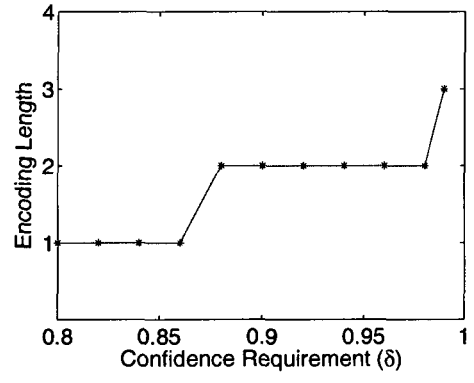


Figure 3.4: Encoding length vs. δ ($n = 100$, $s = 10$, $\alpha = 0.1$, $PT_i = 0.1$)

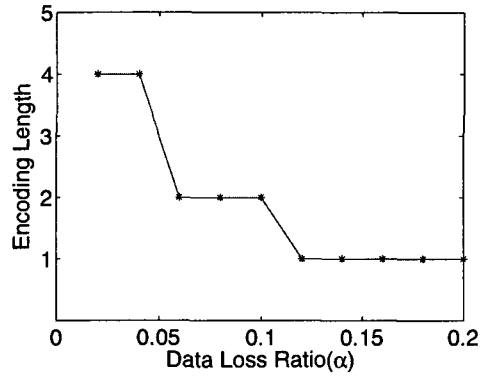


Figure 3.5: Encoding length vs. α ($n = 100, s = 10, \delta = 0.9, PT_i = 0.1$)

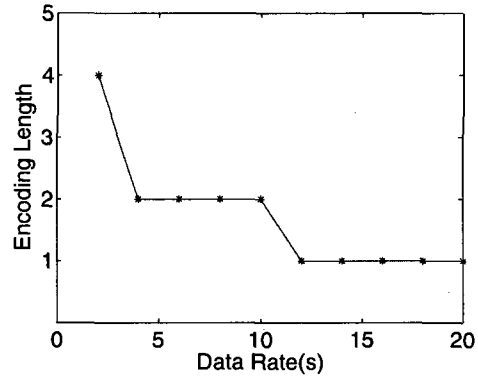


Figure 3.6: Encoding length vs. s ($n = 100, \alpha = 0.1, \delta = 0.9, PT_i = 0.1$)

Next, we examine the accuracy of this algorithm. Let k be the suggested encoding length and $conf(i)$ be the confidence achieved by using i-bit encoding numbers. We evaluate it from two aspects. First, we show the values of $conf(k)$ based on simulations to examine if k is sufficiently long to guarantee the security requirements, i.e., if $conf(k) > \delta$. Second, we show the values of $conf(k-1)$ if $k > 1$ to test whether k is optimal. If $conf(k) > \delta$ and $conf(k-1) \leq \delta$, then k is a perfect choice of the encoding length while $k-1$ fails the security requirements.

In this simulation, we randomly generate data based on the data distribution PT_i and simulate the behaviors of a malicious storage node. We run 10000 independent tests, and calculate the confidence, i.e., the probability of detecting a false reply at the sink. The simulation compares the values of $conf(k)$ and $conf(k-1)$ in Fig. 3.8-Fig. 3.12, where the dashed line without markers is the confidence requirement δ. As we can see, $conf(k)$ is always greater than δ while $conf(k-1)$ is not in most cases, which indicates that $k-1$ is not a proper encoding length for security protection.

Therefore, we conclude that Algorithm 5 gives a good guideline of selecting appropriate encoding lengths. Simulations have shown that the suggested length value is sufficient for security and also efficient in communication.

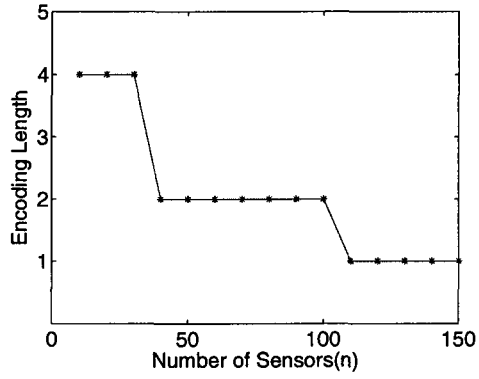


Figure 3.7: Encoding length vs. n ($s = 10$, $\alpha = 0.1$, $\delta = 0.9$, $PT_i = 0.1$)

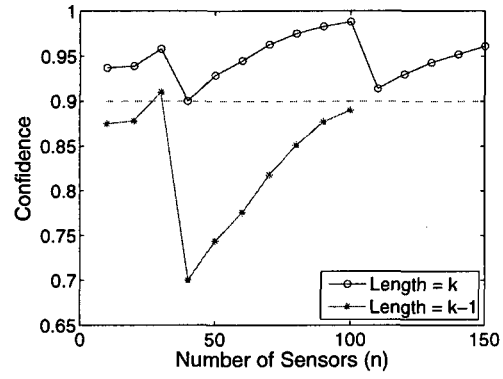


Figure 3.8: Confidence vs. n ($s = 10$, $\alpha = 0.1$, $\delta = 0.9$, $PT_i = 0.1$)

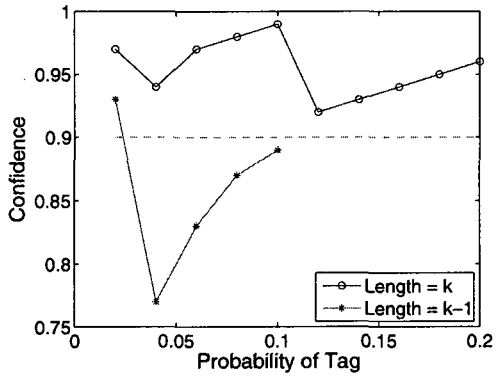


Figure 3.9: Confidence vs. PT_i ($n = 100$, $s = 10$, $\alpha = 0.1$, $\delta = 0.9$)

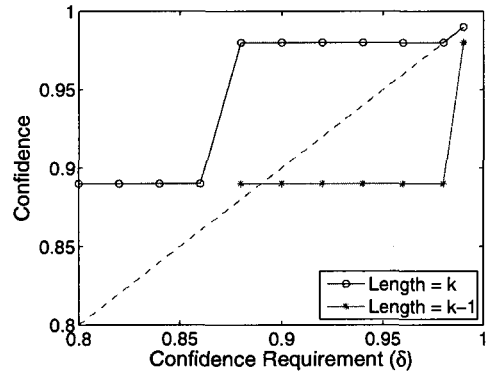


Figure 3.10: Confidence vs. δ ($n = 100$, $s = 10$, $\alpha = 0.1$, $PT_i = 0.1$)

3.4.2 One Bit Encoding Numbers

In this subsection, we are particularly interested in a special arrangement, where every encoding length is set to the smallest value 1, because it is the best case for communication cost. Since every encoding number has the same length, when the storage node drops data, it simply selects the largest block, no matter which tag it is associated with. We make small modifications to Algorithm 5 to estimate the confidence achieved by one bit encoding numbers.

We first examine the confidence for a single tag (T_i) query with varying PT_i. The following Fig. 3.13 shows the comparison of our estimation and simulation results. In this setting, our

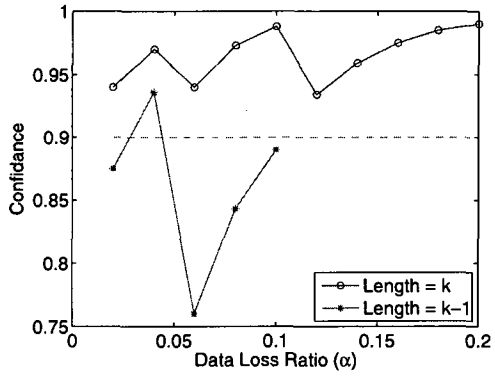


Figure 3.11: Confidence vs. α ($n = 100, s = 10, \delta = 0.9, PT_i = 0.1$)

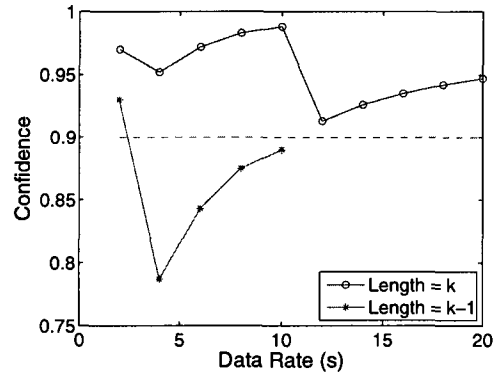


Figure 3.12: Confidence vs. s ($n = 100, \alpha = 0.1, \delta = 0.9, PT_i = 0.1$)

estimation is very close to simulation when $p \geq 0.08$. The result is also consistent with Fig. 3.3, in which 1-bit is suggested when $p \geq 0.12$.

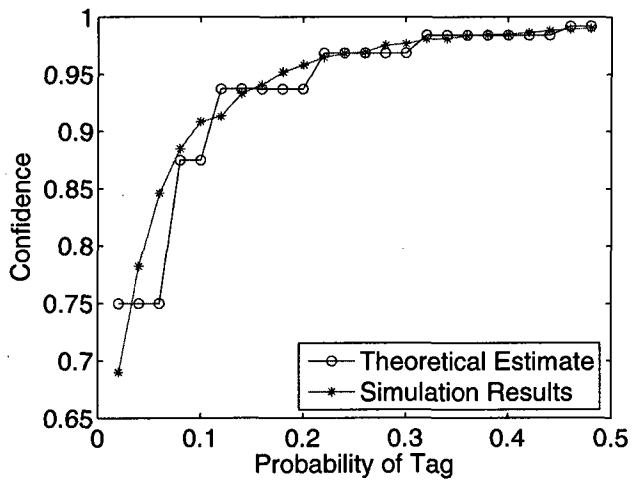


Figure 3.13: Confidence of 1-bit encoding number for single tag query, $n = 100, s = 10, \alpha = 0.1$.

Furthermore, we consider multiple tag range query in a more practical simulation. We adopt a data set with a normal distribution and find the optimal bucket partition which satisfies privacy constraints and yield minimum communication cost for false positive. The following Table 3.5 shows the bucket partition and the corresponding probability for each bucket.

Then we enumerate all 28 possible range queries in the tests. For each query, we generate

T_1	T_2	T_3	T_4	T_5	T_6	T_7
[20,30]	[31,38]	[39,46]	[47,53]	[54,60]	[61,68]	[69,80]
0.090	0.139	0.174	0.193	0.195	0.131	0.078

Table 3.5: The second row is the range partition of tags and the third row lists the probability of each tag.

random data for every sensor and apply the greedy algorithm to drop data. Since the encoding length is known as 1, we can easily derive a confidence of detecting the false reply. We repeat this process and use the average value as the result. Table 3.6 compares the simulation results with our algorithm. The cell in T_i row and T_j column represents the confidence for a query of $\{T_i, T_{i+1}, \ldots, T_j\}$. The values in parenthesis are our estimated confidence. As we can see, the estimation is very accurate for both single tag and multiple tag queries, where the largest difference is 0.016. We observe that one bit encoding numbers work well for popular tags or mild security re-

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
T_1	0.884(0.875)	0.987(0.984)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_2	–	0.930(0.938)	0.994(0.996)	0.999(0.999)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_3	–	–	0.947(0.938)	0.996(0.998)	0.999(0.999)	0.999(0.999)	0.999(0.999)
T_4	–	–	–	0.953(0.969)	0.997(0.998)	0.999(0.999)	0.999(0.999)
T_5	–	–	–	–	0.954(0.969)	0.994(0.996)	0.998(0.999)
T_6	–	–	–	–	–	0.925(0.938)	0.983(0.984)
T_7	–	–	–	–	–	–	0.868(0.875)

Table 3.6: Confidence Comparison of 1-bit Encoding Numbers: The row (column) index is the minimum (maximum) tag in the query. In each cell, the first value is simulation result and the second value in the parenthesis is our estimation.

quirements, and we can accurately estimate the confidence to determine whether one bit encoding numbers are suitable for a certain scenario.

3.4.3 Communication Cost

In this subsection, we present the performance of communication cost. We begin with the introduction to the data set and other environment settings used in our simulation. Then, we illustrate the two extra costs incurred by our protection scheme with varying parameters. First, during the periodical data report, sensors need to send encoding numbers to storage nodes for verifying the reply. Second, when storage nodes reply range queries, extra data (*false positive*) are transferred to the sink following the bucketing scheme. As we will show later, both encoding number scheme and bucket partition scheme are very efficient.

In this simulation, we use a real data set from Intel Lab [2], which is collected from 54 sensors during a one-month period. The details of the data set can be found at Intel Lab's web site [2]. After filtering out the incomplete and abnormal data, we adopt the data from 44 nodes in our simulation. We evenly divide the 40 sensors into 4 groups and place one storage node in each group, i.e., $n = 11$ for each storage node. We also retain their location coordinates and calculate d_{ss} and d_{avg} for Algorithm 4. We select the temperature data collected during 03/01/2004-03/10/2004 as the sensitive information and we round the data points to the precision of 0.5. In addition, we sample three different epoch lengths, 10 minutes, 20 minutes and 30 minutes and we assume that the whole query set is received in 24 hours. In our scheme, privacy requirements (VAR_p, EN_p) and security requirements (α, δ) also need to be specified. In this simulation, we fix security requirements ($\alpha = 0.1$, $\delta = 0.9$), set EN_p to $\{1, 1.5, 2\}$, and vary VAR_p from 0.4 to 1.2 with an interval of 0.2 to examine the performance. The following Table 3.7 presents the number of buckets our scheme derives for epoch=30min with different settings of the privacy requirements.

We first show the cost of transferring encoding numbers from sensors to storage nodes. For each sensor, let CE be the cost of sending encoding numbers, and let CD be the cost of transferring

	$VAR_p = 0.4$	0.6	0.8	1.0	1.2
$EN_p = 1.0$	13	8	6	5	4
1.5	8	8	6	5	4
2.0	5	5	5	5	4

Table 3.7: Number of buckets (epoch=30min)

the encrypted data to the storage node. We measure the ratio of $\frac{CE}{CD}$ in our simulation, which indicates the impact of sending encoding numbers. Since this ratio varies for different sensors, the average values are illustrated in Fig. 3.14-Fig. 3.16.

We observe that the less strict privacy requirement leads to the higher cost of transferring encoding numbers. Intuitively, the less strict privacy requirement allows smaller buckets, which provide more accurate information and can reduce the false positive. However, smaller buckets may increase the cost of sending encoding numbers because they yield a large number of buckets and each sensor probably has to send more encoding numbers in each epoch. In our simulation setting, every storage node is in charge of 11 sensors, which makes the false positive dominant in the extra cost compared with the cost of sending encoding numbers. Therefore, in order to minimize the total extra cost, our algorithm prefers to use fine-grained buckets in favor of reducing false positive. When the privacy requirement becomes less strict, our bucket partition probably will contain smaller buckets, which further increase the cost of sending encoding numbers. We also observe that the cost of encoding numbers decreases when the length of epoch increases. In a longer epoch, every sensor collects more data in each bucket following a certain distribution. It decreases the probability for each bucket to have no data in an epoch. Thus, increasing epoch length can reduce the number of non-data tags for each sensor, which require the sensor to send encoding numbers. Therefore, by suppressing more encoding numbers, a longer epoch incurs less

communication cost.

As a summary, we find that the encoding number scheme does not incur too much extra cost. Even for 10-minute epoch, the extra cost(CE) is less than 25% of CD in most cases. The performance mainly benefits from short encoding numbers derived in our protocol. In all the tested case, the length of an encoding number is no more than 4 bits. If we use the standard HMAC, e.g., 160 bits HMAC-SHA1, the cost of sending encoding numbers will be significantly increased (> 40 times).

The other extra cost in our scheme is the false positive. We measure the false positive as the number of useless data received by the sink, represented by CF. We also count the total number of data received by the sink, indicated as TN. The performance of the false positive is illustrated by the ratio of $\frac{CF}{TN}$ in Fig. 3.17.

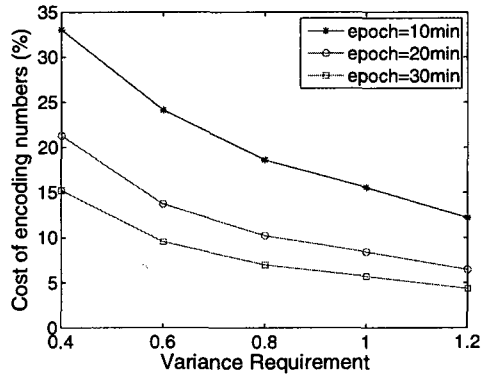


Figure 3.14: Cost of encoding numbers vs. VAR_p ($EN_p = 1$)

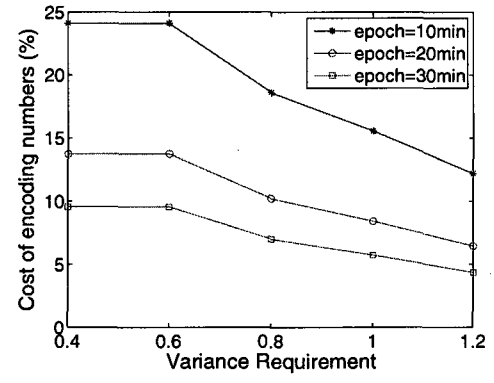


Figure 3.15: Cost of encoding numbers vs. VAR_p ($EN_p = 1.5$)

We observe that the false positive increases when the privacy requirements become more strict, because each of the resulting buckets probably include more data in order to yield the required variance and entropy. In our simulation, we also find that there is no big difference for varying epoch lengths. The reason is that in our simulation setting ($n = 11$), the false positive (CF in Eq.(3.6)) is much larger than the cost of encoding numbers (CE in Eq.(3.6)). Different epoch

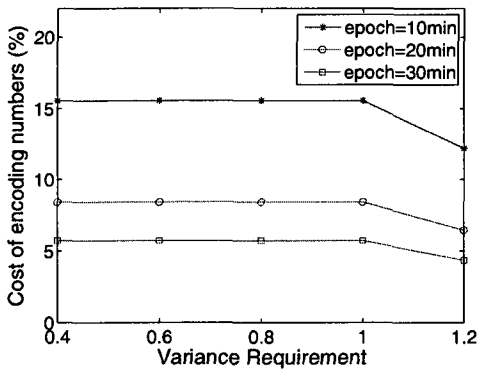


Figure 3.16: Cost of encoding numbers vs. VAR_p ($EN_p = 2$)

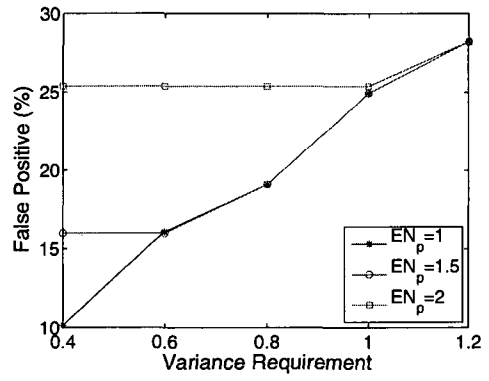


Figure 3.17: False Positive vs. VAR_p (epoch=10min)

lengths do not change CF, which is the dominant factor of the objective function Eq.(3.6). Thus, we obtain very similar bucket partitions for the varying epoch lengths. In Fig. 3.17, the epoch length is set to 10 minutes.

Based on our simulation results, bucketing scheme is also an efficient protection against privacy breach. The false positive (CF) takes less than 28% of the total data (TN) in all cases.

3.4.4 Event Detection

In this subsection, we test the encoding number scheme with sampling for rare event detection proposed in Section 3.3.3. In this setting, we randomly deploy 100 sensors in a 10×10 network field, i.e., $S = 100$. Assume the coverage area of an event be S', for convenience, we assume it is a $\sqrt{S'} \times \sqrt{S'}$ square area where the event source resides in the center. We set the coverage area from 15 to 50 with an interval 5. In addition, we set $\delta = 0.9$ and consider varying α at 0.4, 0.5 and 0.6 for defining the threshold of the desired event data. We first determine the number of sample nodes v and the encoding length D based on the previous analysis and the results are shown in Table 3.8.

Then, we randomly select v sensors and mark them as sample nodes. In the simulation, we

Coverage Area:	15	20	25	30	35	40	45	50
v:	30	23	18	15	13	11	10	9
D:	1	1	1	1	1	1	1	1

Table 3.8: Number of Sample Nodes (v) and Encoding Length (D)

randomly select a point as the event source. The sensors in the coverage area are supposed to detect the event. For each parameter setting, we conduct 10000 independent tests and present the average result in the following figures. Our evaluation considers three aspects. First, we consider if a legitimate storage node can be verified with high probability. Second, we examine if our encoding number scheme can detect a false reply with high probability. Finally, we evaluate the efficiency of the communication cost with our sampling scheme.

We first examine the probability that the number of sensors in the coverage area of an event is less than the specified threshold $(1-\alpha)\cdot\lambda\cdot S'$. This is also the probability that the sink triggers a false alarm and regards a legitimate storage node as a malicious storage node by mistake. The simulation result is presented in Fig. 3.18. As we can see, with a reasonable parameter setting, this probability of false alarm is very close to 0, especially when the coverage area is large.

In addition, we illustrate the confidence of detecting a false reply in Fig. 3.19. As we mentioned, this confidence is irrelevant to the threshold defined by α. According to Fig. 3.19, the confidence is obviously higher than the requirement $\delta = 0.9$ in all cases. It indicates that the derived v and D can guarantee the security requirement.

Moreover, we find this scheme significantly reduces the communication cost compared with encoding number scheme. For instance, when the event coverage are is 25, we decide to select $v = 18$ sample nodes to send encoding numbers ($D = 1$) every epoch. Thus, in this sampling scheme, $18 \times 1 = 18$ bits encoding numbers are sent every epoch. Compared with normal en-

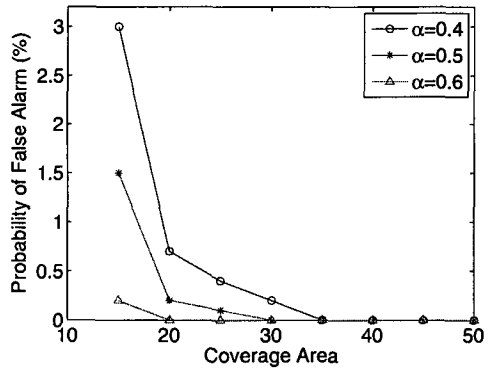


Figure 3.18: Probability of False Alarm in Event Detection: This figure illustrates the probability that the number of sensors in the event proximity is less than the threshold $(1-\alpha)\cdot\lambda\cdot S'$, in which case the storage node in charge of the area will be regarded by the sink as a malicious storage node by mistake.

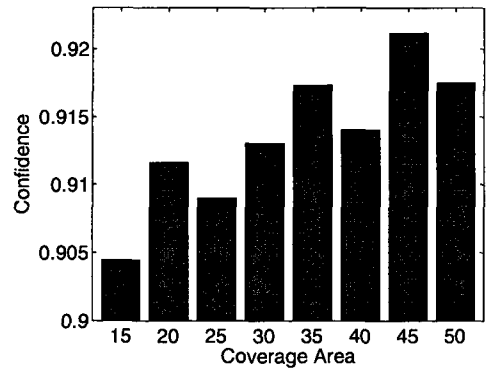


Figure 3.19: Confidence of Detecting a False Reply in Event Detection: This figure illustrates the probability for the sink to detect a false reply with varying coverage areas. The specified requirement for the confidence is $\delta = 0.9$.

coding number scheme, even if we use 1-bit length, there are 100 bits transmission per epoch. Therefore, the sampling scheme helps us reduce much communication cost and also achieve a desirable confidence.

3.5 Summary

Our work in this chapter considers a basic query in sensor networks, data range query, which is the building block in many applications. We focus on the security and privacy issues in this query which have been rarely discussed in the prior work. In particular, our solution preserves the privacy of sensor data and enables the sink to verify the query reply. Additionally, we have identified the trade-off between security protection and energy efficiency. Our solution satisfies the given security constraints with a minimal extra energy cost. Furthermore, we have studied an important application of event detection using our range query protocol with security mechanisms. Finally, we have evaluated our solution with both synthetic and real trace data.

As shown in this chapter, practical settings, e.g., the security concerns discussed here, often alter the protocol design for achieving efficiency. Our work on basic queries focuses on these practical scenarios in a pervasive computing environment. In the next chapter, we continue to discuss another basic query in RFID systems for practical applications.

Chapter 4

Basic Query in RFID Systems: Continuous Scans

This chapter focuses on another basic query in RFID systems. Compared to sensor networks, RFID is still in its infant stages of development. RFID applications are much less mature and less diverse than sensor network applications. While the previous chapter targets the applications with extra requirements, this chapter considers a general application model which is commonly seen in the literature. However, the prior work usually simplifies the problem settings and ignores practical constraints. In this chapter, we move a step forward by considering the basic RFID query in a more realistic environment.

In RFID systems, the most basic query is to collect all IDs from every RFID tag which has been the primary focus of the prior research. However, all the previous work only considers a single scanning process assuming all target RFID tags are within the reader's reading range at the time of scanning. This ideal assumption is not true in some practical scenarios. We often need a series of multiple scans to collect all the desired tags. In this chapter, as mentioned in Chapter 1,

we consider this continuous scan problem defined in both spatial and temporal domains.

In continuous scanning, an important property is that each individual scan usually is not independent. Some of the scanning processes may involve overlapping RFID tags. In the example illustrated in Fig. 1.5, the processes at adjacent locations inevitably have some tags in common in the reading ranges. Similarly, in temporal continuous scanning, some products may stay in the reading range for a long time, thus can be read by several consecutive scans. According to the above property, therefore, the simple solution that scans all the RFID tags in the reading range is inefficient In a series of continuous scanning processes, a lot of RFID tags may be present in multiple scans. This implies that each scan in the simple solution may collect a lot of redundant information which has already been gathered in the previous scans.

In this work, we propose algorithms to efficiently detect inventory changes for continuous scanning without collecting all the IDs. Our solution takes advantage of the information previously gathered about the inventory and only collects the IDs of the newly added tags, and removes the IDs that are no longer present.

4.1 Related Work

As mentioned, most of the previous research in RFID systems focused on the anti-collision protocols. For the problem of continuous scanning, however, collision is not the key crux. In the evaluation, we will show that typical anti-collision protocols are not suitable for this problem. Our solutions are still based on the ALOHA protocol, but specifically designed for continuous scanning. Adaptive splitting proposed in [101] is the only existing approach that can be applied to our problem. We will compare it with our solution in the evaluation and show that tree traversal is time consuming in a large scale system with long IDs (e.g., 96bit ID). Furthermore, some work

has extracted useful information based on the typical ALOHA scheme, e.g., counting the number of RFID tags [82, 83, 106]. However, this paper considers a more general task of collecting tag IDs. The intuition behind our scheme of checking the existing tags in Section 4.4.2 is similar to the Bloom filter, which has been well studied in the literature [10,23,35,64,99,130]. However, our algorithm has a different goal of minimizing the scanning time with a certain accuracy requirement. The protocol and the corresponding analysis are quite different. In addition, [64] proposed a method to build high accuracy bloom filter based on selectively choosing hash functions. In our improved scheme in Section 4.4.3, the pre-computation is motivated by a similar idea. Again, our problem has different goals and metrics.

4.2 RFID Background

An RFID system consists of RFID readers and RFID tags. An RFID tag contains an integrated circuit for storing and processing data, and an antenna for receiving and transmitting data. The most common passive RFID tags have no battery supply, and transmit data by backscattering the received signals from RFID readers.

4.2.1 Slotted ALOHA Protocol

Slotted ALOHA is a popular anti-collision protocol implemented by major RFID manufactures. We briefly review the protocol in this subsection because our design is built upon it. In this protocol, the RFID reader first broadcasts a number f to all tags indicating the following time is divided to f *slots*. These f slots form a *frame* and f is called the *frame size*. After receiving f, each tag will randomly pick a slot index from 0 to $f-1$ and load the index into a slot counter (sc). Usually, RFID tags use a pseudo-random number generator, which takes a random seed from the

reader and hashes the random seed with tag IDs to a slot index. Let r be the random seed sent by the reader, x be the tag ID, the slot index and the initial value of sc will be $hash(x||r)$ mod f. In the rest of this paper, we use $h_f(x,r)$ to represent this operation. In practice, this hash function is often implemented using the CRC checksum operation.

After sending the initial message with f and r, the reader then orderly scans every slot in the frame. The reader uses a 'slot end' command to close the current slot and start the next slot, which also triggers every tag to decrease its slot counter (sc) by one. If a tag's sc becomes zero, it will backscatter its ID in the coming slot. From the RFID reader's view, there are three possible scenarios for each slot. First, if only one tag T replies in a slot (the slot is called *single-reply slot*), the reader will send an acknowledgment of success (ACKS) to notify T that the data is successfully received. Tag T will then keep silent (inactive) in the rest of the session. Second, if multiple tags respond in the same slot, the reader will detect a signal collision (the slot is called *collision slot*). The reader will then send an acknowledgment of failure (ACKF) to notify the responding tags (>1) that it has failed to receive the data. These tags will keep active. Third, if no tag responds in a slot (called *empty slot*), the reader will close the slot immediately. At the end of a frame, if collisions have occurred in this frame, the reader will start a new frame, in which only the active tags will participate. Fig. 4.1 illustrates the state diagram of an RFID tag in the slotted ALOHA protocol.

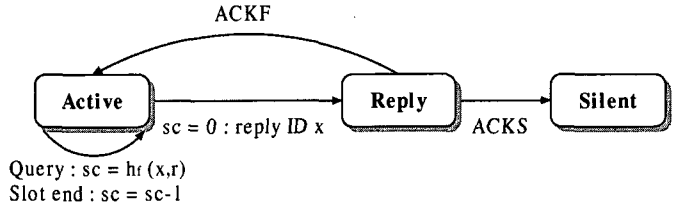


Figure 4.1: State Diagram of an RFID tag with ID x: There are three states and each directional edge is annotated by the transition condition or a pair of 'condition: action'. 'sc' denotes the slot counter and 'Query' includes the frame size f and random seed r.

4.2.2 Modified ALOHA Protocol

In this paper, we make a minor modification to the slotted ALOHA protocol, so that we can use it not only to collect IDs from RFID tags, but also to efficiently select a set of tags as details will be explained later. The only difference with the typical protocol is the transition from state 'Active' to state 'Reply'. In our design, when $sc = 0$, the tag has three options for the action, which is controlled by the reader. In the 'slot end' command, the reader adds two bits as an opcode, which instructs each tag with $sc = 0$ to act accordingly as follows. (1) **no reply:** If the opcode is 0, the tag will not reply anything. (2) **short reply:** If the opcode is 1, the tag will reply a random short binary string. The length can be as short as possible as long as the reader can detect the collision while more than one tag respond. Usually, the short reply is less than 10 bits. (3) **reply ID:** If the opcode is 2, the tag will reply its ID as the typical protocol. The modified state diagram of a tag is illustrated in Fig. 4.2. Note extra hardware design is needed to support this modified protocol. However, based on the functionality of current RFID tags, this modification is definitely feasible and can be easily implemented.

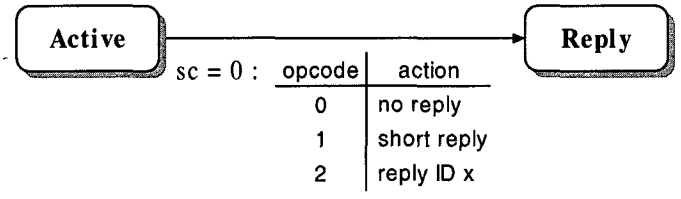


Figure 4.2: Modified Transition from 'Active' to 'Reply': The transition action when $sc = 0$ depends on the opcode in the previous 'slot end' command.

4.2.3 Slot Timing

In the slotted ALOHA protocol, timing of each slot, which is the time duration of the slot, is an important parameter for calculating total scanning time. In our algorithms, there are different slot

timings in accordance with three possible actions of RFID tags as mentioned earlier. First, for the action of 'no reply', we use t_{nr} to denote the slot timing. Second, for the action of 'short reply', we may have three different scenarios, empty slot, single-reply slot and collision slot. The timings of single-reply and collision slot are the same, indicated as t_{sr}, and the timing of an empty slot is shorter, indicated as t_{em}. Third, 'reply ID' action results in the same three scenarios as 'short reply'. The timing for an empty slot is t_{em} and the timing for single-reply and collision slot is t_{ID}. Table 4.1 summarizes the notations of slot timing used in this paper. These timings are hardware-dependent parameters, which will be quantified in the evaluation. A principal rule is $t_{nr} < t_{em} < t_{sr} \ll t_{ID}$.

	Empty	Single-Reply	Collision
No Reply	t_{nr}		
Short Reply	t_{em}	t_{sr}	t_{sr}
Reply ID	t_{em}	t_{ID}	t_{ID}

Table 4.1: Slot Timings

4.2.4 Optimal Frame Size for Collecting IDs

A part of our schemes also uses the typical slotted ALOHA protocol to collect IDs from all active RFID tags. Thus, we need set an appropriate frame size in order to achieve time efficiency. As we mentioned in the related work, a lot of previous work has derived the 'optimal' frame size for slotted ALOHA protocol. However, they all regard the number of slots as the measurement of scanning time, ignoring the varying timings for different types of slots. In this subsection, we analyze the optimal frame size used in our schemes.

Assume there are n active tags, and we use frame size f in the typical ALOHA. We will

successfully collect an ID only in a single-reply slot. For a certain slot, the probability of being an a single-reply slot is

$$p_1 = \frac{n}{f} \cdot (1 - \frac{1}{f})^{n-1} = \frac{n}{f} \cdot e^{-\frac{n}{f}}.$$

Thus, there is a single-reply slot every $\frac{1}{p_1}$ slots expectedly. Considering empty and non-empty slots have different time durations, these $\frac{1}{p_1}$ slots take a time of

$$\frac{1}{p_1}(p_0 \cdot t_{em} + (1 - p_0) \cdot t_{ID}),$$

where p_0 is the probability for a slot to be an empty slot, $p_0 = (1 - \frac{1}{f})^n = e^{-\frac{n}{f}}$. The optimal f minimizes the above formula, which indicates the minimum scanning time per ID collected. The above formula can be regarded as a function of $\frac{n}{f}$ denoted as $g(\frac{n}{f})$. Once t_{em} and t_{ID} are set, we can derive the optimal f by solving $g'(\frac{n}{f}) = 0$, where g' represents the derivative.

4.3 Problem Formulation

our objective is to efficiently scan RFID tags in a series of continuous scanning processes. In spatial continuous scanning, the objective is to efficiently collect the newly introduced RFID tags at each location. In temporal continuous scanning, besides collecting 'new' tags, the objective also includes efficiently detecting the RFID tags that have been moved away. Collecting all the tags in the reading range is definitely feasible, but very time-consuming especially in a setting of massive RFID tags. Our solutions utilize the previously gathered information to achieve the continuous scanning without collecting all IDs from every RFID tag.

In this problem, continuous scanning boils down to a general task with two overlapped sets of RFID tags $\overline{S}$ and S at each scanning location/time-point. One set (S) is the RFID tags in the

reading range we are about to scan. The other set ($\overline{S}$) is the previously gathered RFID tags that overlap with present set of tags S. Thus, assume we have scanned all the tags in $\overline{S}$, our problem is to scan the RFID set S. We first define two important types of tags for this problem as follows:

- **Unknown tags:** the present RFID tags whose IDs have not been collected in the previous scanning processes, i.e., $S - \overline{S}$. Let $U = |S - \overline{S}|$.
- **Missing tags:** the tags that have been previously scanned, but no longer exist, i.e., $\overline{S} - S$. Let $M = |\overline{S} - S|$.

Therefore, our goal is to efficiently collect IDs from *unknown tags* and for some applications (temporal continuous scanning), remove *missing tags* from the current inventory.

In this paper, we propose probabilistic solutions to the problem, which can not guarantee perfect accuracy. Some unknown tags may not be collected by our schemes and some missing tags may not be detected either. Additionally, for any accuracy requirement, a deterministic guarantee can not be provided. Therefore, we describe the accuracy constraint as follows. Let U' and M' respectively be the number of unknown tags and missing tags *after* applying our schemes. We quantify the accuracy by making them upper bounded by two requirements R_U and R_M respectively with more than α probability.

To summarize, given the previously obtained inventory $\overline{S}$, two upper bounds R_U and R_M, and a probability $\alpha \in (0, 1]$, we would like to use the minimal time to scan S, such that

$$\mathbf{Pr}(U' \leq R_U) > \alpha \text{ and } \mathbf{Pr}(M' \leq R_M) > \alpha.$$

In addition, our schemes are based on an assumption that the difference between $\overline{S}$ and S, characterized by U and M, is bounded. We use U_{max} and M_{max} to denote the bounds, i.e., $U =$

$|S-\overline{S}| \leq U_{max}$ and $M = |\overline{S}-S| \leq M_{max}$. Note U_{max} and M_{max} could be loose estimations. We will show the impacts of their accuracy in evaluation. We also try to relax this assumption and discuss a more general setting without prior knowledge about the difference between the two overlapped sets.

4.4 Methodology

Clearly, to achieve our objective, we need accomplish the following two tasks with a probability of at least α

1. to collect $\geq U - R_U$ unknown tags from $S-\overline{S}$;
2. to detect $\geq M - R_M$ missing tags from $\overline{S}-S$.

The rest of the section describes two algorithms, one for each task of the problem.

4.4.1 Collect Unknown Tags

The hurdle to efficiently collecting the unknown tags is that the tags in $\overline{S} \cap S$, which have already been read previously, will interfere and make the responses from the unknown tags buried in overwhelming "noises". Once we can suppress the known tags from responding while keeping the unknown tags active, applying the typical slotted ALOHA protocol would efficiently collect the unknown tags with no interference. Therefore, we propose a two-phase algorithm consisting of the **selecting phase** and the **collecting phase**. In the selecting phase, we select *only* unknown tags and keep them active. Note that the selected unknown tags could be a subset of $S-\overline{S}$. Meanwhile, we suppress and inactivate other tags including all the known tags in $\overline{S} \cap S$ and some unknown tags. In the collecting phase, we simply use the typical slotted ALOHA to scan all active tags.

The selecting phase is the key step in our algorithm and we still use the slotted ALOHA to select unknown tags. As we mentioned, in the slotted ALOHA, each tag randomly picks a slot in a frame based on the hashed value of its ID with a random seed sent by the reader. Since we know the set $\overline{S}$, the random seed r, and the hash function, it is straightforward to determine whether a particular slot would be occupied by a known tag. When checking the slots one by one, we signal the tags replying in those slots to be silent and keep the tags replying in other slots active. It is true that some slots may possibly be occupied by both known and unknown tags and our method will mistakenly miss the unknown tags. To counteract this, therefore, we run the two-phase algorithm using different random seeds for a number of rounds so that all unknown tags may be exposed.

Algorithm 6 (called the *CU* scheme) illustrates the protocol taking two parameters f and δ. How to determine the parameters is discussed later in this subsection. I is the resulting inventory which is expected to include sufficient unknown tags. The protocol first sets $I = \overline{S}$ and iteratively update I in the successive rounds. Inside the while loop, lines 3-15 present the selecting phase and line 16 is for the collecting phase with details omitted. In the selecting phase, the reader first generates a random seed r and broadcasts f and r. Each tag takes f as the frame size and picks the slot (from slot 0 to slot $f-1$) to respond denoted by the hashed value of its ID and r (line 7). A slot is called a ***pre-empty slot*** if no known tag responds in that slot. We use $PE(I,f,r)$ to denote the number of pre-empty slots after hashing all tag IDs in I with r onto f slots, that is,

$$PE(I,f,r) = |\{j | j \in [0,f), \forall x \in I, h_f(x,r) \neq j\}|.$$

Being aware of the known tags, the reader can emulate the hashing for each known tag ID in $\overline{S}$ and determine if a particular slot is pre-empty. The reader then sends commands to make the tags replying in the pre-empty slots stay active in the following stage while other tags are kept silent.

In the collecting phase, the reader simply collects the IDs of all active tags via typical slotted ALOHA protocol and adds them to I. In Algorithm 6, since an unknown tag randomly picks a slot to respond in each round, it has $\frac{PE(I,f,r)}{f}$ probability to be collected. We use a variable p to keep track of the probability that an unknown tag will not be identified after the current round (line 5). Obviously, p becomes smaller as the reader applies more rounds of queries. The process terminates when $p < \delta$. Fig. 4.3 illustrates an example.

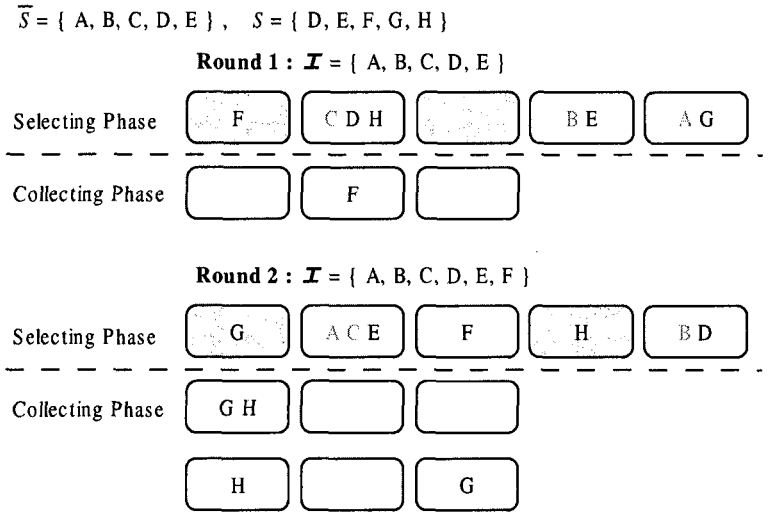


Figure 4.3: Example of Collecting Unknown Tags ($f = 5$): In this figure, each rectangle represents a slot and the dark rectangle indicates the pre-empty slot. In round 1, we first check 2 pre-empty slots and F will be the only active tag at the end of the selecting phase. We then apply the typical slotted ALOHA protocol (the frame size is set to 3) and successfully collect the unknown tag F. In round 2, we also check 2 pre-empty slots in the selecting phase. Note we do not select the third slot because F is already a known tag. Then, we apply the slotted ALOHA and collect IDs from G and H.

The protocol is heavily dependent on the parameters f and δ. The following theorems show how to choose these two parameters to achieve time efficiency and satisfy the accuracy requirement. Note the total scanning time includes the time spent in the selecting phase and the collecting phase. However, the settings of f and δ only affect the selecting phase, while the scanning time in the collecting phase depends on the number of newly collected RFID tags, which is related to the requirement parameter R_U. Therefore, in the following analysis, our objective is to minimize

Algorithm 6 $CU(f, \delta)$: Collect Unknown Tags

```
I ← S̄, p ← 1
while p ≥ δ do
    Reader powers on and generates a random seed r
    Reader calculates PE(I, f, r)
    p ← p · (1 − PE(I,f,r)/f)
    Reader broadcasts f and r to tags
    Each tag with ID x responds in slot h_f(x, r)
    for slot i = 0 to f − 1 do
        Reader sets opcode = 0 at the end of slot i − 1
        if slot i is a pre-empty slot then
            Reader sends ACKF
        else
            Reader sends ACKS
        end if
    end for
    Reader collects active tags by the typical slotted ALOHA (opcode = 2) and adds their IDs
    to I
    Reader powers off
end while
Return I
```

the scanning time in the selecting phase. Theorem 4.1 gives the condition that guarantees the accuracy requirement. Theorem 4.2 provides a bound for the expected number of the iterations in Algorithm 6 and Theorem 4.3 bounds the scanning time in the selecting phase by a function of the frame size. The results are utilized to find the optimal parameter f in Theorem 4.4.

Theorem 4.1 *After running Algorithm 6, we can guarantee* $\mathbf{Pr}(U' \leq R_U) > \alpha$ *if we set*

$$\delta \leq \frac{R_U + a}{2a + U_{max}} - \frac{\sqrt{a \cdot (a \cdot U_{max} + 2U_{max} \cdot R_U - 2R_U^2)}}{\sqrt{U_{max}} \cdot (2a + U_{max})},$$

where $a = (\Phi^{-1}(\alpha))^2$ *and* Φ *is the standard normal CDF.*

Proof: Recall that U and U' respectively represent the number of unknown tags before and after running Algorithm 6. For each unknown tag before running Algorithm 6, the event that it will not be collected by Algorithm 6 is independent and the probability of the event is $p < \delta$ according to the algorithm. Therefore, U' follows a binomial distribution $U' \sim B(U, p)$. Consider another binomially distributed random variable $z \sim B(U_{max}, \delta)$. Since $p < \delta$ and $U \leq U_{max}$, $\mathbf{Pr}(U' < R_U) > \mathbf{Pr}(z < R_U)$. When U_{max} is large, z is approximated by a normal distribution[1], $z \sim N(\mu, \sigma^2)$, where $\mu = \delta \cdot U_{max}$, $\sigma^2 = \delta \cdot (1 - \delta) \cdot U_{max}$. Therefore, based on the property of normal distribution,

$$\mathbf{Pr}(z < R_U) = \Phi(\frac{R_U - \mu}{\sqrt{2}\sigma}).$$

After plugging δ in the hypothesis, we have $\mathbf{Pr}(z < R_U) \geq \alpha$. Thus, $\mathbf{Pr}(U' < R_U) > \alpha$. ■

Theorem 4.2 *Algorithm 6 is expected to terminate after* $k = \ln\delta / \ln(1 - e^{-\frac{|\bar{S}| + U_{max}}{f}})$ *rounds.*

Proof: In each round (loop variable in line 2) in Algorithm 6, the set of currently known tags is I, and each slot (loop variable in line 8) in the frame is a pre-empty slot if no tags in I selects the

[1]The purpose of this approximation is to quickly calculate the CDF.

slot, which has a probability of $(1-\frac{1}{f})^{|I|}$. Thus, the expected value of $PE(I,f,r)$ is $f\cdot(1-\frac{1}{f})^{|I|}$. Since $|I| \leq |\overline{S}| + U_{max}$ in any round, the expected value of $\frac{PE(I,f,r)}{f}$ is

$$\frac{f\cdot(1-\frac{1}{f})^{|I|}}{f} \geq (1-\frac{1}{f})^{|\overline{S}|+U_{max}} = e^{-\frac{|\overline{S}|+U_{max}}{f}}.$$

After k rounds, the value of p in Algorithm 6 will be

$$\prod_{1}^{k}(1-\frac{PE(I,f,r)}{f}) < (1-e^{-\frac{|\overline{S}|+U_{max}}{f}})^k < \delta.$$

Thus, the algorithm will terminate after k rounds. ■

Theorem 4.3 *The expected scanning time of the selecting phase in Algorithm 6 is bounded by* $ST_c = k\cdot f\cdot t_{nr}$, *where k is the number of rounds in Algorithm 6.*

Proof: Apparently, Algorithm 6 spends totally $k\cdot f$ slots in the selecting phase. In each slot, the reader sets the opcode to 0 in the 'slot end' command. Thus, the timing of each slot in the selecting phase is t_{nr}. ■

Theorem 4.4 *The optimal value of the frame size is* $f = 1.443\cdot(|\overline{S}| + U_{max})$.

Proof: With Theorem 4.2 and Theorem 4.3, the expected scanning time of the selecting phase is bounded by

$$ST_c = \frac{\ln\delta}{\ln(1-e^{-\frac{|\overline{S}|+U_{max}}{f}})}\cdot f\cdot t_{nr}.$$

By solving $\frac{\partial ST_c}{\partial f} = 0$, we obtain the optimal value of the frame size, $f = 1.443\cdot(|\overline{S}| + U_{max})$. ■

4.4.2 Detect Missing Tags

As we mentioned earlier, temporal continuous scanning may need remove the missing tags from the inventory. In this subsection, we propose a scheme to detect missing tags and analyze the parameter setting.

To detect the missing tags, besides collecting all the present tags, another simple alternative is to check the existence of each tag in $\overline{S}$ by broadcasting its ID as a 'select' mask and listen to the response. If there is no response, the tag can be removed from the inventory. However, it is also inefficient to transmit all the IDs in $\overline{S}$ one by one, because the bit length of the data could be very long, e.g., a 96-bit ID in Gen2 standard [45] plus a 16-bit CRC checksum. Our solution is still based on tag replying in a certain slot according to the hashed value of the tag ID. Provided that a slot that should be occupied by a known tag becomes empty, we know the tag corresponding to that slot is missing. Similar to the previous problem of collecting unknown tags, some missing tags may be mapped to a slot with other present tags, and thus will not be detected. In this scheme, we also repeat the process for multiple rounds to achieve the accuracy requirement.

Our solution is illustrated in Algorithm 7 (called the *DM* scheme). I represents the returned inventory which is supposed to have removed the missing tags after Algorithm 7. I' represents the set of known tags we have not checked yet. Initially, both I and I' are set to $\overline{S}$. In each round inside the while loop, the reader first generates a random seed r and broadcasts f and r to tags. Each tag randomly picks a slot to respond according to the hash value of its ID. A slot is called an ***pre-single slot*** if only one tag in I' responds in the slot. With the knowledge of the known tags, the reader can calculate the indexes of the pre-single slots. The reader then checks each pre-single slot to determine if the known tag mapping to the slot is still present (x in line 9). If the pre-single slot is empty, the corresponding known tag ID x will be removed from I. If there is a reply, the reader

Algorithm 7 $DM(f)$: **Detect Missing Tags**

```
1: I ← S̄, I' ← S̄
2: Reader powers on
3: while |I'| > 0 do
4:    Reader generates a random number r
5:    Reader broadcasts f and r to tags
6:    Each tag with ID x responds at slot h_f(x, r)
7:    for slot i = 0 to f − 1 do
8:       if slot i is a pre-single slot then
9:          Reader sets opcode = 1 at the end of slot i − 1
10:         Find x ∈ I such that h_f(x, r) = i
11:         I' ← I' − {x}
12:         if no tags respond then
13:            I ← I − {x}
14:         else
15:            Reader sends ACKS
16:         end if
17:      else
18:         Reader sets opcode = 0 at the end of slot i − 1
19:         Reader sends ACKF
20:      end if
21:   end for
22: end while
23: Return I
```

will make no change on the inventory I and send ACKS to keep the responding tags silent in the following steps. Let $PS(I', f, r)$ be the number of pre-single slots yielded by I', f, and r. Thus, in each round, we check the existence of $PS(I', f, r)$ known tags. The whole algorithm terminates when $|I'| = 0$, i.e., we have checked all the known tags. Fig. 4.4 shows an example of checking pre-single slots.

In Algorithm 7, only one parameter f, the frame size, needs to be determined to optimize the performance. To find the optimal value for f, we make use of the following results. Theorem 4.5 presents the condition that guarantees the accuracy requirement for the number of the undetected missing tags (M'). The scanning time in this algorithm, however, cannot be expressed by f in a closed form. Therefore, we present in Lemma 4.1 a program to estimate the number of iterations executed by Algorithm 7, and then the number of iterations can be used to express the scanning time in Theorem 4.6. By enumerating all possible values of f, we can find a suitable f to optimize the overall performance.

Theorem 4.5 *After running Algorithm 7, we can guarantee* $\mathbf{Pr}(M' \leq R_M) > \alpha$, *if* $f > -\frac{U_{max}}{\ln(1-\theta)}$, *where*

$$\theta = \frac{R_M + a}{2a + M_{max}} - \frac{\sqrt{a \cdot (a \cdot M_{max} + 2M_{max} \cdot R_M - 2R_M^2)}}{\sqrt{M_{max}} \cdot (2a + M_{max})},$$

$a = (\Phi^{-1}(\alpha))^2$ *and* Φ *is the standard normal CDF.*

Proof: In Algorithm 7, a missing tag is not detected only if some unknown tags respond in the pre-single slot it belongs to. Let q be the probability for this scenario,

$$q = 1 - (1 - \frac{1}{f})^{|S-I|} < 1 - e^{-\frac{U_{max}}{f}} < \theta.$$

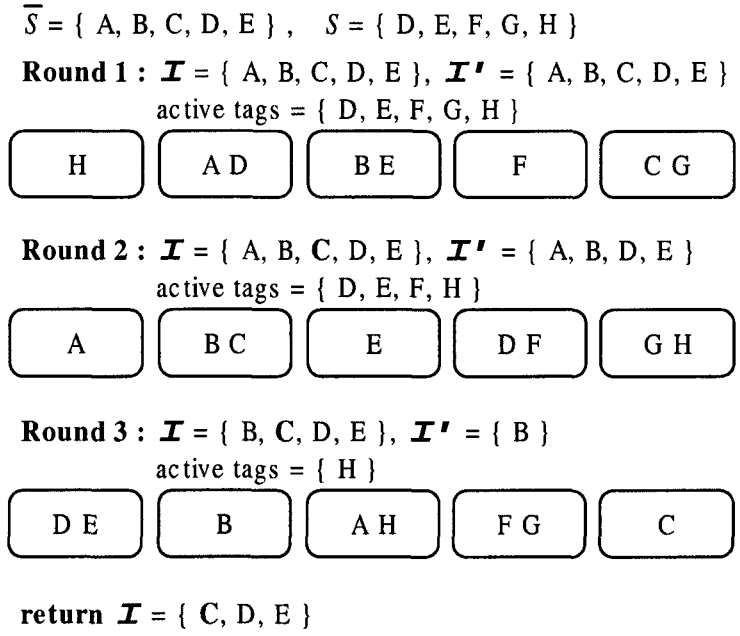


Figure 4.4: Example of Detecting Missing Tags ($f = 5$): In this figure, each rectangle represents a slot and the dark rectangle indicates the pre-single slot. I is the current inventory and I' represents the set of unchecked known tags. We also show the set of active tags in this figure. In round 1, there is only one pre-single slot. The reader supposes that the missing tag C is still present by mistake because the unknown tag G responds at the pre-single slot. In round 2, the reader checks 3 pre-single slots. We then remove A from I and confirm that D and E are still present. In the last round, the reader detects the missing tag B.

Recall M and M' are the number of missing tags before and after running Algorithm 7 respectively. Since detecting each missing tag is independent, M' follows a binomial distribution $M' \sim B(M, q)$. Consider another binomially distributed random variable $z \sim B(M_{max}, \theta)$. Since $q < \theta$ and $M \leq M_{max}$, $\mathbf{Pr}(M' < R_M) > \mathbf{Pr}(z < R_M)$. When M_{max} is large, z is approximated by a normal distribution, $z \sim N(\mu, \sigma^2)$, where $\mu = \theta \cdot M$, $\sigma^2 = \theta \cdot (1-\theta) \cdot M_{max}$. Therefore, based on the property of normal distribution,

$$\mathbf{Pr}(z < R_M) = \Phi\left(\frac{R_M - \theta \cdot M_{max}}{\sqrt{2\theta \cdot (1-\theta) \cdot M_{max}}}\right) = \alpha.$$

Thus, $\mathbf{Pr}(M' < R_M) > \alpha$. ■

Lemma 4.1 *Given f, we can estimate the number of iterations executed in Algorithm 7.*

Proof: Let n_i be the number of unchecked known tags after round i. In round i, for each slot, the probability that only one known tag in I selects the slot is

$$\frac{n_{i-1}}{f}(1-\frac{1}{f})^{n_{i-1}-1} = \frac{n_{i-1}}{f} \cdot e^{-\frac{n_{i-1}}{f}}.$$

Thus, the expected value of n_i is

$$\begin{aligned} n_{i-1} - PS(I', f, r) &= n_{i-1} - f \cdot \frac{n_{i-1}}{f} \cdot e^{-\frac{n_{i-1}}{f}} \\ &= n_{i-1} \cdot (1 - e^{-\frac{n_{i-1}}{f}}) \end{aligned}$$

We iteratively calculate n_i until $n_i < 1$, and k is estimated as $\min\{i | n_i < 1\}$. ■

Theorem 4.6 *Assume Algorithm 7 terminates after k round, the scanning time of Algorithm 7 is bounded by*

$$ST_d = |\overline{S}| \cdot t_{sr} + (k \cdot f - |\overline{S}|) \cdot t_{nr}.$$

Proof: Given k and f, the expected total number of slots is $k \cdot f$. Since every known tag is checked in a pre-single slot, there are $|\overline{S}|$ pre-single slots and the rest $k \cdot f - |\overline{S}|$ slots are no reply slots (lines 18-19 in Algorithm 7) each with duration t_{nr}. For each pre-single slot, there might be a response (with duration t_{sr}) or no response (with duration $t_{em} < t_{sr}$). Therefore, the expected total scanning time is bounded by ST_d. ■

Finally, based on the analysis above, we can enumerate all possible values of f (f is a bounded value in practice), and find the optimal value with the minimum scanning time ST_d.

4.4.3 Extension

Our previous analysis is based on two parameters U_{max} and M_{max}. In some applications, however, it may not be easy to estimate these two bounds or the estimations could be too loose to be meaningful. In this section, we present extensions to our schemes without the assumption of knowing U_{max} and M_{max}. Additionally, we propose an improved scheme for detecting missing tags based on pre-computation.

4.4.3.1 CU extension with estimating U_{max}

As we described earlier, our *CU* scheme is an iterative process and we collect some unknown tags in each round. In this extension, we utilize these information of the collected unknown tags to estimate U_{max}. Thus, our basic idea is to first set a rough estimation for U_{max}, and then iteratively revise it as our scheme proceeds. We expect the estimation of U_{max} converges towards the tighter bound as more iterations are executed.

The details of this extension is presented in Algorith 8. Initially, we set U_{max} to $|S|$, the total number of present tags, assuming all the tags are unknown tags. This number $|S|$ can be obtained by the previous work [83] with a small overhead. We define another variable t to count how many unknown tags have been collected since the algorithm starts.

In line 3, we set the frame size f as we discussed in Theorem 4 and keep updating p as in Algorithm 6. Recall that p is the probability for an unknown tag not to be collected after the current round. The selecting phase and collecting phase are the same as in Algorithm 6. We use variable c to denote the number of newly collected unknown tags and correspondingly update t in line 6. Lines 7-12 are the key part for this extension, which derives a new value for U_{max} based on the observations.

Algorithm 8 Collect Unknown Tags (Extension)

1: $U_{max} \leftarrow |S|, t \leftarrow 0, p \leftarrow 1$

2: **while** $U_{max} - t > R_U$ **do**

3: $f \leftarrow 1.443 \cdot (|\overline{S}| + U_{max})$

4: Use f in the selecting phase and execute lines 6-16 in Algorithm 1

5: Assume we collect c unknown tags in this round

6: $p \leftarrow p \cdot (1 - \frac{PE(I,f,r)}{f}), t \leftarrow t + c, tolp \leftarrow 0$

7: $sum \leftarrow \sum_{x=1}^{|S|} \begin{pmatrix} x \\ t \end{pmatrix} p^{x-t}(1-p)^t, x \leftarrow 1$

8: **while** $tolp < \alpha$ **do**

9: $tolp \leftarrow tolp + \begin{pmatrix} x \\ t \end{pmatrix} p^{x-t}(1-p)^t / sum$

10: $x \leftarrow x + 1$

11: **end while**

12: $U_{max} \leftarrow x$

13: **end while**

The following Theorem 4.7 and Theorem 4.8 guarantee the accuracy requirement after running Algorithm 8.

Theorem 4.7 *At any iteration,* $\mathbf{Pr}(U \leq U_{max}) > \alpha$.

Proof: At the beginning of the algorithm, there are U unknown tags. In each iteration, variable p records the probability that a certain unknown tag will have not been collected after the current iteration. Thus, the probability of collecting t unknown tags is

$$\binom{U}{t} p^{U-t}(1-p)^t.$$

In line 7, variable *sum* is the summary of the probability of collecting t unknown tags considering all possible values of U. We use *sum* as a normalizer in the following analysis. Basically, based on the observation of t,

$$\mathbf{Pr}(U = x) = \binom{x}{t} p^{x-t}(1-p)^t / sum.$$

In lines 8-12, we set the new value for U_{max} and guarantee that it satisfies the condition

$$\sum_{x<U_{max}} \mathbf{Pr}(U = x) > \alpha,$$

which is equivalent to $\mathbf{Pr}(U < U_{max}) > \alpha$. ■

Theorem 4.8 *After running Algorithm 8, we can guarantee* $\mathbf{Pr}(U' \leq R_U) > \alpha$.

Proof: Recall U' is defined as the number of remaining unknown tags, thus $U' = U - t$. According to Theorem 4.7, $\mathbf{Pr}(U \leq U_{max}) > \alpha$, then we have $\mathbf{Pr}(U' \leq U_{max} - t) > \alpha$. Since Algorithm 8 terminates when $U_{max} - t \leq R_U$, $\mathbf{Pr}(U' \leq R_U) \geq \mathbf{Pr}(U' \leq U_{max} - t) > \alpha$. ■

4.4.3.2 DM extension with estimating M_{max}

The similar idea can be also applied to *DM* scheme while M_{max} is not known. Initially, we set M_{max} to $|\overline{S}|$, assuming all the previously collected tags are missing. We define another variable t to count the number of missing tags that have been detected since the algorithm starts. According to the observation of t, we analyze if the current value of M_{max} is too loose. Basically, given a value of M_{max}, we derive the probability that such a value can yield the observation of t. If the probability is less than a small constant ε (e.g., 0.01), the value of M_{max} is considered too loose to be possible. The details are presented in Algorithm 9.

Algorithm 9 Detect Missing Tags (Extension)

1: $I \leftarrow \overline{S}, I' \leftarrow \overline{S}, M_{max} \leftarrow |S|, p \leftarrow 1$

2: **while** $|I'| > 0$ **do**

3: Derive the optimal frame size f using M_{max}

4: Execute lines 2-21 in Algorithm 7

5: $p \leftarrow p \cdot (1 - \frac{PS(I,f,r)}{|I|}), t \leftarrow |\overline{S}| - |I|$

6: $M_{max} \leftarrow \max\{x | \begin{pmatrix} x \\ t \end{pmatrix} p^{x-t}(1-p)^t > \varepsilon\}$

7: **end while**

8: Return I

4.4.4 Improved DM Scheme

Our basic *DM* scheme relies on the pre-single slots to detect the missing tags. The number of pre-single slots (*PS*) per round in *DM* is critical to the performance. Previously, we blindly choose the random numbers/seeds for the hash function and give the probabilistic performance analysis. It is easy to observe that some random numbers may result in more pre-single slots, which in turn

may improve the performance. This subsection discusses an improved *DM* scheme that inserts, between two consecutive scanning, a pre-computation step that searches for good random numbers for the next scanning.

Assume that RFID tags use 16-bit random numbers. Given a frame size f, the general optimization problem is to find the best set of random numbers from all possible random numbers, i.e., $[0, 2^{16})$, with the minimum scanning time while satisfying the accuracy constraints. This problem can be formulated as an integer programming and reducible to 0/1 knapsack problem. Due to the page limit, we omit the formal description here. Basically, the optimization is NP-hard and even approximation algorithms are not helpful because of the large size of the possible random numbers. Therefore, we present a heuristic solution based on greedy algorithm.

Our metric of the random number r is $PS(\overline{S}, f, r)$. We prefer to use the random number with the largest value of *PS* because *DM* can check more known tags in one round. In this pre-computation, however, $PS(\overline{S}, f, r)$ is dependent on the random numbers applied before r. Thus, we iteratively apply greedy algorithm to find a set of random numbers. The details are proposed in Algorithm 10. We consider the returned set *BSR* has a limited size *maxsize*. U represents the set of unchecked tags and N_r denotes the number of selected random numbers. We examine every possible random number and pick the one with the largest *PS*. $flag[i]$ records the number of tags responding at slot i, and $id[i]$ tracks the ID of the last tag replying at slot i. At the end of each iteration, we remove from U the tag IDs, which can be checked by the selected random number (lines 18-20), before continuing to select the next random number.

4.5 Performance Evaluation

Our evaluation is based on simulation. Here are some general parameter settings.

Algorithm 10 Finding the best set of random numbers for the improved *DM*

```
Calculate the frame size f and initialize BSR[maxsize]
U ← S̄, N_r ← 0
while U ≠ Φ and N_r < maxsize do
    for r = 0 to 2^16 − 1 do
        reset array flag[f] and id[f]
        for each x_i ∈ U do
            idx = hash(x_i||r) mod f
            flag[idx] ← flag[idx] + 1, id[idx] ← x_i
        end for
        ps ← 0
        for i = 0 to f − 1 do
            if flag[i] = 1 then ps ← ps + 1
        end for
        if ps > max then
            max ← ps, r' ← r, flag' ← flag, id' ← id
        end if
    end for
    for i = 0 to f − 1 do
        if flag'[i] = 1 then U ← U − id'[i]
    end for
    BSR[N_r] ← r', N_r ← N_r + 1
end while
Return BSR
```

Slot Timings: For slot timings, we keep the following ratios based on hardware and protocol specifications from major RFID manufactures,

$$t_{nr} : t_{em} : t_{sr} : t_{ID} = 1 : 1.5 : 3 : 30.$$

We set the unit time above to 0.25ms according to I-CODE description [103]. This unit time may vary for different hardware, but usually appears in the same scale.

Optimal Frame Size for Collecting IDs: As mentioned in Section 4.2.4, we need determine the optimal frame size when applying the typical slotted ALOHA protocol. According to the slot timing above and the analysis in Section 4.2.4, the optimal frame size for collecting n active tags is $f = 3.48n$. In the collecting phase of our CU scheme, n is estimated as the expected number of the selected unknown tags $n = \frac{PE}{f} \cdot U_{max}$.

Accuracy Confidence: The parameter α in our problem formulation is set to 0.99 in all our simulation. It indicates that our schemes must guarantee the accuracy specified by R_U and R_M with more than 0.99 probability.

Other Solutions for Comparison: We consider the following three solutions for comparison.

- *Collect All (CA):* In this solution, the reader collects all RFID tags via the typical slotted ALOHA protocol, ignoring the knowledge of $\bar{I}$. The initial frame size is set to $|S|$. The scanning time of this solution is proportional to the number of tags ($|S|$). We use CA to denote this solution.

- *Suppress Known (SK):* In this solution, the reader first broadcasts the known tag IDs one by one. The tag whose ID matches the broadcast ID will be suppressed to keep silent. The reader then uses the ALOHA protocol to collect the remaining active tags, which are all unknown tags. We use SK to denote this solution.

- *Adaptive Query Splitting (AQS):* This solution is proposed in [101] based on tree traversal scheme. The basic idea is to record the results when traversing the ID tree in the previous scanning. For the current scanning, instead of traversing the whole tree, AQS only checks those nodes where no collision has occurred in the previous scanning. We refer readers to [101] for details.

In the rest of this section, we present the scanning time of our schemes and compare with other solutions with different parameter settings. Spatial and temporal continuous scanning are separately evaluated. We conduct 100 independent trials for each parameter setting in the simulation, and illustrate the average values. The scanning time deviations of our schemes (< 0.2 second) and other solutions (< 0.5 second) are very small and omitted in the figures. In addition, in all our tested cases, the resulting inventories obtained by our algorithms always satisfy the accuracy requirement defined by R_U, R_M and α. In the rest of this section, the results about the accuracy are not explicitly presented. Instead, we focus on the performance of the scanning time.

4.5.1 Spatial Continuous Scanning

In this subsection, we present the performance results for spatial continuous scanning. First, we consider the scenario with two overlapping sets of tags. Then, we simulate a more complete case involving a series of continuous scanning processes. Finally, we examine the *CU* extension scheme proposed in Section 4.4.3.

4.5.1.1 Two overlapping sets

In this simulation, we consider two overlapped sets of tags S and S. Recall that we have collected the tags in $\overline{S}$ and try to collect the unknown tags in S. The default parameter setting is as follows.

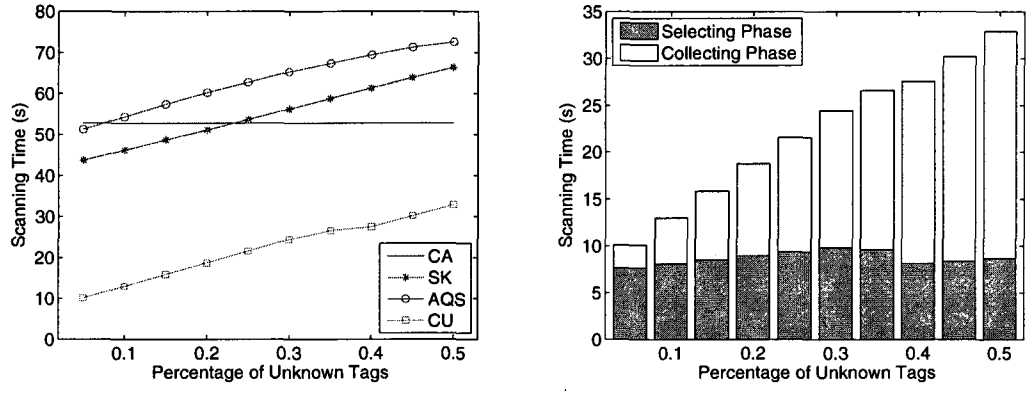
There are 5000 tags in both sets, i.e., $|\bar{S}| = |S| = 5000$. The number of unknown tags U is set to $U = 0.1 \cdot |S|$, which means 10% of current tags are unknown tags. For a certain value of U, the accuracy requirement R_U is set to $R_U = 0.1 \cdot U$, i.e., keeping 10% unknown tags uncollected is tolerated. In addition, U_{max} (the upper bound of U) is set to $1.2 \cdot U$, i.e., 20% more than the actual value of U. In the next, we vary one of these parameters while the other parameters are the same as in the default setting, and present the corresponding performance results.

Varying U: We first examine the performance with different number of unknown tags, which indicates the difference between the current set of RFID tags and that in the previous scanning. Fig. 4.5(a) compares our *CU* scheme with other three solutions. horizontal axis represents the ratio of $\frac{U}{|S|}$.

In Fig. 4.5(a), the CA scheme keeps the same performance for all the cases, since the total number of tags ($|S|$) does not change. The other three schemes all yield a linearly increasing performance when U increases. The performance of our *CU* scheme is superior to all other solutions. For example, in the default setting with $\frac{U}{|S|} = 0.1$, *CU* needs less than 13 seconds to finish while SK and CA need 46 and 53 seconds respectively. In the worst tested case with $\frac{U}{|S|} = 0.5$, *CU* still saves more than 40% scanning time compared with the CA scheme.

Fig. 4.5(b) further illustrates the scanning time of the *CU* scheme spent in the selecting phase and the collecting phase. The collecting phase consumes more time when U increases because it need collect more unknown tags. On the other hand, the scanning time of the selecting phase does not change too much ranging from 7 to 9 seconds. It is interesting to observe that the scanning time of the selecting phase is not monotonously increasing with U. Sometimes, when U increases, our scheme can select more unknown tags in a shorter time.

Varying $|\bar{S}|$ and $|S|$: In our simulation, we always set $|\bar{S}| = |S|$. Both $|\bar{S}|$ and $|S|$ indicate



(a) Overall Scanning Time

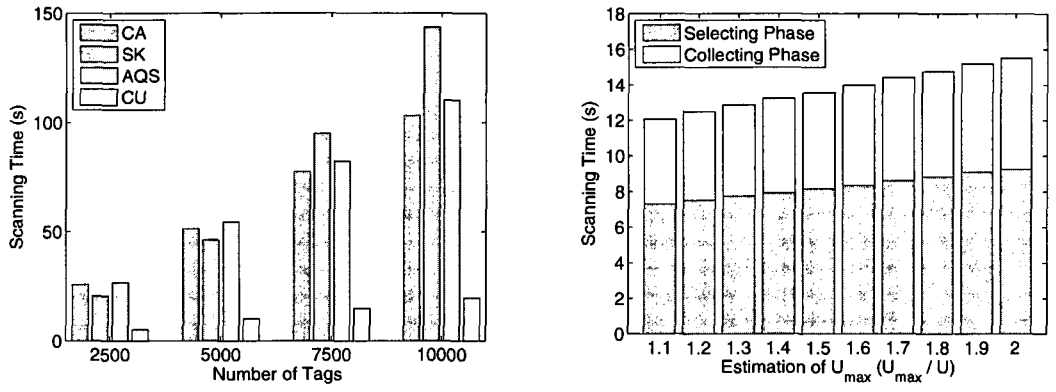
(b) Detailed Scanning Time

Figure 4.5: Varying U: (a) The overall scanning time (b) The scanning time spent on the selecting phase and the collecting phase. In our settings, $|S| = 5000, U/|S| = \{5\%, 10\%, \cdots, 50\%\}$

the scale of the RFID tags in the simulated scenarios. In this part, we present and compare the performance with varying number of RFID tags. We test four cases, where the number of RFID tags ranges from 2500 to 10000 with 2500 as the interval. Other parameters are derived by keeping the same ratio as in the default setting. We compare the scanning time in Fig. 4.6(a).

The performance of all schemes is proportional to the number of RFID tags. In all four cases, the scanning time of the *CU* scheme is less than 24% of that consumed by the CA scheme. The simulation results indicate that the *CU* scheme can significantly improve the performance in terms of scanning time, especially when dealing with a large amount of RFID tags. For example, when $|S| = 10000$, the CA scheme needs 104 seconds to finish. By contrast, the scanning time of *CU* is about 24 seconds.

Varying U_{max}: Our problem formulation assumes that U_{max} is the estimated bound for the actual number of the unknown tags (U). We also test the impact of U_{max} on the performance. In the default setting, the actual number of unknown tags is 500. We vary U_{max} in our tests from 550 (10% more than the actual number) to 1000 (100% more than the actual number). The results



(a) Varying $|S|(|\bar{S}|)$

(b) Varying estimations of U_{max}

Figure 4.6: The scanning time of spatial continuous scanning with different varying parameters. (a) The performance of the CA scheme is also presented as a comparison. $|\bar{S}| = |S| = \{2500, 5000, 7500, 10000\}$. (b) $U = 500$ and the ratio of U_{max}/M is set to $1.1, 1.2, \ldots, 2$.

are illustrated in Fig. 4.6(b), where the horizontal axis is the value of $\frac{U_{max}}{U}$ ranging from 1.1 to 2. We observe that a more accurate estimation slightly improves the performance, essentially in the selecting phase. The difference between the best case and worst case in Fig. 4.6(b) is less than 4 second. Therefore, our CU scheme can perform well even with a rough estimation of U_{max}.

4.5.1.2 A series of scanning processes

In the next, we simulate the scenario that a series of continuous scanning processes are needed to cover a large warehouse. We consider a simple floor plan with a straight aisle and stocking shelves beside it. A person carrying an RFID reader will move along the aisle and stop at certain locations to turn on the reader and scan the tags. Fig. 4.7 illustrates the setting in our simulation. Assume the reader will stop at 10 locations, $\{L1, L2, ..., L10\}$. The reading range of a reader is considered as a circle with radius R. The interval distance between any two consecutive scanning locations is set to $\frac{R}{2}$. For simplicity, we assume that the density of the products is a constant in the whole area, i.e., at any location, the number of the tags in the reading range is the same. With this setting, at

each location, there are about 30% unknown tags in the reading range. The total scanning time in the whole process is compared in Table 4.2. Compared with the CA scheme, the *CU* scheme dramatically reduces the scanning time by almost 50%.

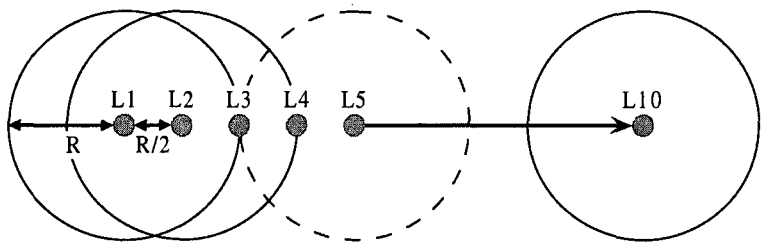


Figure 4.7: R is the reading range of the RFID reader, which moves towards a straight line and stop at 10 locations, $\{L1, L2, \cdots, L10\}$, to launch the scanning process. The interval distance between any two consecutive locations is $R/2$.

CA	SK	AQS	CU
528s	562s	628s	265s

Table 4.2: Total scanning time for a series of scanning processes

4.5.1.3 CU Extension

Finally, we evaluate the *CU* extension scheme, which only affects the selecting phase in the *CU* scheme. We present the performance in Fig. 4.8. Compared to the default *CU* scheme, the *CU* extension only incurs a small overhead in the selecting phase. Its performance is very close to the CU scheme when U is small. Even in the worst case in Fig. 4.8, the extra time spent by the *CU* extension is less than 3 seconds.

4.5.2 Temporal Continuous Scanning

We assume that temporal continuous scanning need not only collect the unknown tags, but also detect the missing tags. Thus, the implementation combines the *CU* and *DM* schemes. In our

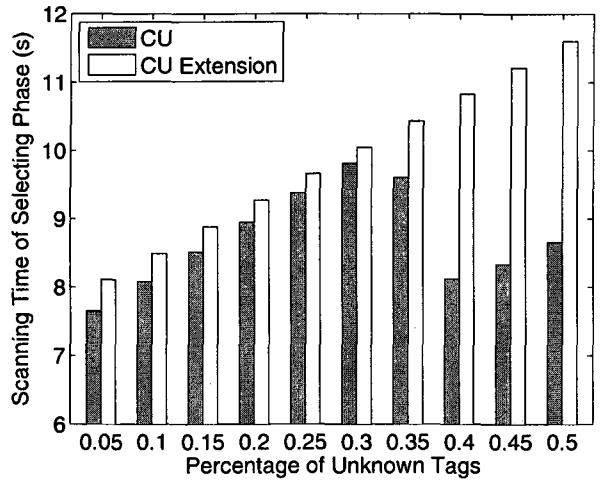


Figure 4.8: Selecting Phase in the *CU* Extension: This figure illustrate the scanning time spent on the selecting phase in the CU extension scheme with different number of unknown tags (U). The performance of the selecting phase in the *CU* scheme is also displayed as a comparison. ($|S| = 5000, U/|S| = \{5\%, 10\%, \cdots, 50\%\}$)

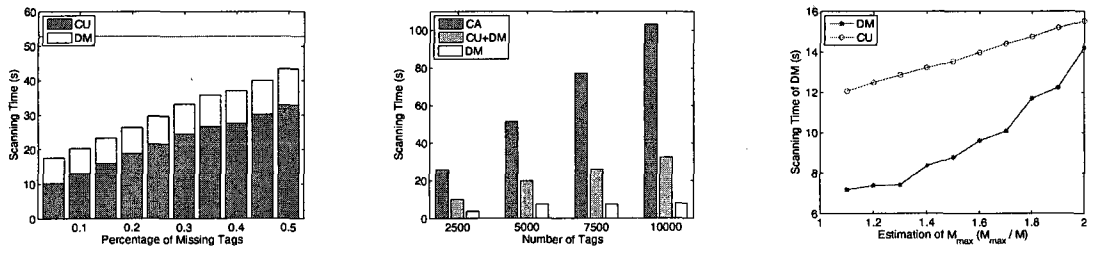
simulation, we first apply the *CU* scheme and then the *DM* scheme in each scanning process. In the next, we first present the simulation results when considering two overlapping sets. Then, we discuss the performance of the *DM* extension scheme and the improved *DM* scheme. We do not consider a series of scanning in an explicit section because its performance can be easily derived from the simple case with two overlapping sets.

4.5.2.1 Two overlapping sets

In the simulation, the default parameter setting is the same as that in spatial continuous scanning except that we need consider parameters for the missing tags. We set $|\overline{S}| = |S| = 5000$. Thus, the number of missing tags is the same as the unknown tags. We set $M = U = 0.1 \cdot |S|$ by default. The requirement for missing tags R_M is also set in the same way as R_U, $R_M = R_U = 0.1 \cdot M$. In addition, the estimation of M_{max} is to $M_{max} = 1.2 \cdot M$. Similar to spatial continuous scanning, in the next, we vary one of these parameters and examine the performance.

Varying M: We first test the impact of the number of missing tags M. Fig. 4.9(a) shows the simulation results. The horizontal axis represents the ratio of $\frac{M}{|S|}$ ranging from 0.05 to 0.5 with an interval of 0.05. We separately display the scanning time consumed by the CU and DM schemes. The summary of both parts is the total scanning time in temporal continuous scanning. The performance of other three schemes is the same as that in Fig. 4.5(a). In Fig. 4.9(a), we only compare our scheme with the CA scheme whose performance appears as a flat line.

According to Fig. 4.9(a), we observe that our scheme for temporal continuous scanning is also very efficient. For example, in the default setting with $\frac{M}{|S|} = 0.1$, our scheme consumes around 20 seconds, which is 37% of the time caused by the CA scheme. In addition, we find that the process of detecting missing tags is much shorter than collecting unknown tags and that increasing M does not incur too much time in the DM scheme. For example, the difference of DM's scanning time between $\frac{M}{|S|} = 0.05$ and $\frac{M}{|S|} = 0.5$ is less than 3 seconds.



(a) Varying $M(U)$: The flat line represents the performance of the CA scheme. ($|S| = 5000, M = U$, and $M/|S| = \{5\%, 10\%, \cdots, 50\%\}$)

(b) Varying $|S|(|\bar{S}|)$: The performance of the CA scheme is also presented as a comparison. ($|\bar{S}| = |S| = \{2500, 5000, 7500, 10000\}$)

(c) Varying estimations of M_{max}: In our settings, $M = 500$ and the ratio of M_{max}/M is set to $\{1.1, 1.2, \cdots, 2\}$.

Figure 4.9: The scanning time of temporal continuous scanning with different varying parameters.

Varying $|\bar{S}|$ and $|S|$: Similar to spatial continuous scanning, we consider 4 cases with 2500, 5000, 7500 and 10000 RFID tags respectively. We compare our scheme with the CA scheme in

Fig. 4.9(b). The performance of the other two schemes (SK and AQS) is the same as in Fig. 4.6(a). In Fig. 4.9(b), we also separately illustrate the scanning time spent on the *DM* scheme. As we can see from Fig. 4.9(b), the scanning time of the *DM* scheme is a small overhead compared with the *CU* scheme and only slightly increases with the number of tags. Thus our scheme that combines CU and *DM* together is still much more efficient than the CA scheme, especially in a large scale system. When $|S| = 10000$, our scheme consumes less than 30% time required by the CA scheme.

Varying M_{max}: Finally, we evaluate how the value of M_{max} affects the performance. The simulation results are presented in Fig. 4.9(c). The horizontal axis is the ratio of $\frac{M_{max}}{M}$ ranging from 1.1 to 2 with an interval of 0.1. As a comparison, the performance of the *CU* scheme is also displayed in the same figure. Similarly, we find that more accurate estimation of M_{max} leads to a better performance. Although the absolute difference of performance is not large, e.g., the difference is less than 7 seconds between $\frac{M_{max}}{M} = 1.1$ and $\frac{M_{max}}{M} = 2$, the *DM* scheme is obviously more sensitive to M_{max} compared with the *CU* scheme.

4.5.2.2 DM Extension and Improved DM

The performance of the *DM* extension and improved *DM* for varying M is illustrated in Fig. 4.10. We observe that the *DM* extension is slightly slower than the *DM* scheme with a difference less than 1 second. On the other hand, the improved *DM* scheme saves the scanning time by at least 1.4 seconds. In the best case when $\frac{M}{|S|} = 0.5$, the improved *DM* scheme reduces the scanning time by 41% compared with the *DM* scheme.

The improved *DM* scheme requires pre-computations which include numeric CRC16 operations. We examine the feasibility by implementing the pre-computation on a HP iPAQ pocket PC with a CPU running at 520MHz frequency. Our results show that each CRC16 takes 2.5ms

with 0.1ms derivation. With our default setting with $|\bar{S}| = 5000$, the pre-computation requires at most 130 seconds. With a reasonable scan frequency, it can definitely be completed between two consecutive scans.

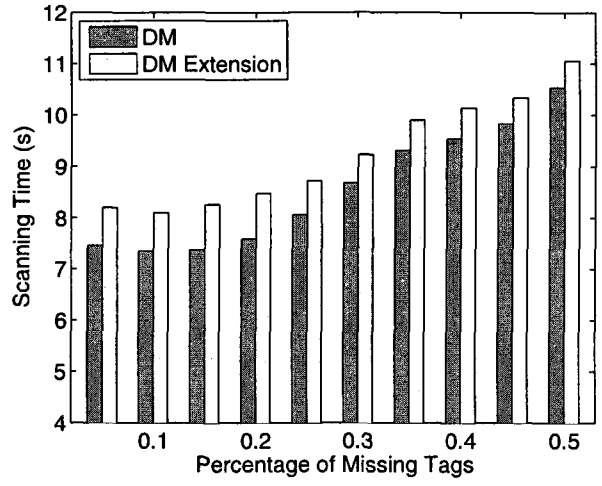


Figure 4.10: Scanning Time of the *DM* Extension and Improved *DM* Schemes: This figure compares *DM* extension schemes with the *DM* scheme for varying number of missing tags M. ($|S| = 5000, M/|S| = \{0.05, 0.1, \cdots, 0.5\}$)

4.6 Summary

This chapter focuses on a fundamental and practical query of continuously collecting all tag IDs via multiple scans. We have defined the problem in both spatial and temporal domains according to different application scenarios. In this problem, the key challenge is to deal with the overlapping RFID tags involved in multiple scans. Launching each scan from scratch is inefficient because redundant information is repeatedly collected. We have proposed efficient solutions based on randomized algorithm that utilizes the information obtained from previous scans to reduce the scanning time of the posterior scans. Our simulation results have shown a great improvement on the scanning time compared to the simple solution of collecting all IDs in each scan.

This work extends the scope of the previous research to a more practical environment. It shows the challenges and difficulties to design efficient query protocols with such weak pervasive devices as RFID tags. Meanwhile, it demonstrates that randomized algorithm is an effective tool for RFID systems because of the random selection in the slotted ALOHA scheme. In the following two chapters, we present our work on more complicated and challenging data mining queries in sensor networks and RFID systems. Randomized algorithm will also be applied to solve data mining query in RFID systems.

Chapter 5

Data Mining Query in Sensor Networks: Detecting Outliers

In the previous two chapters, we have discussed two basic queries that have been studied in the literature and our work has addressed the efficiency issues in these two queries with more practical settings. From this chapter, we present our work on data mining queries which are more complicated. However, the efficiency in data mining queries is valuable because the wireless infrastructure is treated as a database and data mining queries can extract useful information, thus can be widely used in applications. Therefore, efficient solutions for data mining queries are necessary to complete efficient design for the wireless infrastructure.

In this chapter, we focus on a classic data mining query of outlier detection in sensor networks. We consider two popular definitions of outliers in the database literature and develop efficient solutions to detect them. Different from research work in the database community, outlier detection in sensor networks is more challenging because the target data set is sparsely distributed on each sensor and gathering them at a central place is costly in terms of the energy consumption. In the

rest of this chapter, we present our efficient solutions based on histogram information.

5.1 Related Work

Outlier detection has been well studied in the database research community ([7, 8, 19, 26, 80, 81, 88, 107]) and there are several different definitions of outliers in the literature. Our work considers two popular distance-based definitions proposed in [80] and [107], where outliers are identified by examining data points' nearest neighbors. One major research goal of this problem in the database community is to efficiently detect outliers in a large-scale database ([80, 81, 107] and [19]). In sensor networks, however, data are generated by scattered nodes and transferred via wireless channels. The proposed approaches in the previous work can not be directly applied unless all data are gathered at one node, which is very costly in transmission. Another hot spot for database researchers is high dimensional outlier detection ([7, 8, 88] and [6]). This issue is not covered in our work, because sensor networks usually only generate low dimensional data, and different attributes, such as temperature and sound, may not be correlated to define outliers, thus can be considered separately as one-dimensional data.

Sensor networks are often treated as databases and SQL queries are common ways to collect data ([94] and [49]). A lot of research work ([34, 59, 70, 92, 115–117]) has been proposed to handle different types of queries efficiently. However, the distance-based outlier detection has been seldom discussed in this area. As close work, T. Palpanas et al. [104] study a model-based outlier detection in sensor networks. Normal behaviors are first characterized by predictive models and outliers are detected as the deviations. In [25], J. Branch et al. propose an in-network scheme for detecting outliers based on data exchanges among neighbors. However, their goal is to reveal outliers to every sensor and the cost is very expensive for common parameter settings in

the database literature. In [120], S. Subramaniam et al. present an online outlier detection scheme for sensor networks. Every sensor keeps a sliding window of the historic data and estimates the data distribution to detect the outliers. This method, however, consumes a lot of memory space and may not find all outliers.

Some queries of detecting outliers in our solution have a similar form as general range queries. Previous work in [92] and [70] has provided typical models, in which users can easily specify a range query and obtain the result in an efficient and reliable way. On the other hand, top-k query ([116] and [127]) and order statistical query ([115] and [59]) are similar to finding the rank-based outliers, which is a part of our work. However, none of these approaches is applicable to our problem, because the range parameter and the order of data in our problem do not depend on the data value, but the distance to neighboring data points. In addition, our problem requires outliers to be exactly returned without approximation.

5.2 Problem Formulation

An *outlier* represents a data point that is very different from the others. It has been defined in different ways in the database literature. In this work, we consider two popular definitions based on *distance*, which is defined as the the absolute difference between two data points. For each data point p, we can sort all the rest of the data points according to their distances to p in an ascending order. Suppose the sorted list is $p_1, p_2, \cdots, p_k, \cdots$. We have $|p_1 - p| \leq |p_2 - p| \leq \cdots \leq |p_k - p| \leq \cdots$. Let $D^k(p) = |p_k - p|$ represent the distance between data point p and its k-th nearest neighbor (KNN) (p_k). We can define two types of outliers as follows:

Definition 3 *A data point p is called an $O(d,k)$* outlier *if $D^k(p) \geq d$.*

Definition 4 *A data point p is called an* $O(n,k)$ outlier *if there are no more than* $n-1$ *other data points q, such that* $D^k(q) > D^k(p)$.

We are particularly interested in outlier detection in a sensor network composed of N sensors. The sensor network monitors the environment or any object and periodically generates data. Among all the data generated by the sensors, we would like to find all the outliers. We assume that a routing tree rooted at the sink has been constructed by using some simple routing algorithm, in which each sensor is linked to an up-stream and a down-stream node. An outlier detection algorithm will be built on this underlying communication tree. Depending on the choices of outliers, we aim to design efficient algorithms to respond to a query for outliers with parameters d and k for Definition 1 or n and k for Definition 2.

It is possible that the tree topology might be broken or updated due to the wireless link failures. This work considers a common practice that, when building the tree, only stable links are selected such that errors in transmission due to poor link quality can be reduced and the tree topology can be robust for a long time. We also assume that the communication cost of sending a packet between two nodes with a direct link is proportional to the packet size. For easy exposition, we make an assumption that each data point is represented as an integer with precision to the last digit. It is easy to transform any real data to this format, e.g., 12.34 will be converted to 1234 for precision 0.01. Our algorithms focus on the data points and return all the outliers, although many applications may require the algorithm to return the sensor ID or location of the outliers. An easy solution is, after running our algorithm and obtaining all the outliers, to let the sink diffuse the outlier data points to the sensor network so that the sensors holding the outlier data points will reply with their IDs or locations.

5.3 Methodology

5.3.1 Histogram Query

In this section, we introduce the motivation of using histogram information to identify outliers. We observe that both definitions are based on the value of $D^k(p)$, the distance between p and its KNN. In this section, we show that the histogram provides useful information to estimate $D^k(p)$ for each data point p, which helps identify outliers.

In this work, we use the equi-width histogram, because it is easy to aggregate equi-width histograms in sensor networks. We assume that the value range for all data points is uniformly divided into buckets with width w, and each bucket is assigned an ID, consecutively from 1 to the number of buckets. We define bucket i by a value range $[min_i, max_i)$, thus $w = max_i - min_i$ and $min_i = max_{i-1}$. After collecting the histogram, the sink will know the total number of data points in each bucket i, indicated by f_i. For any data point p in bucket i, we can estimate the bounds on $D^k(p)$ based on the histogram information. The following theorems aim to find a pair of values l_i and u_i for any bucket i, such that $\forall p$ in bucket $i, D^k(p) \in (l_i, u_i]$.

Theorem 5.1 *If $f_i > k$, then $l_i = 0$ and $u_i = w - 1$ are the lower and upper bounds for $D^k(p)$, where p is any data point in bucket i.*

Proof: We prove it by contradiction. Referring to Fig. 5.1, assume there exists a data point p

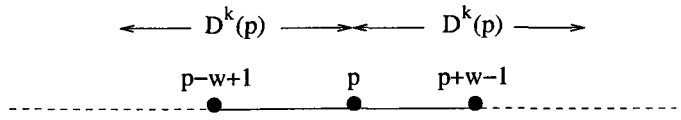


Figure 5.1: Bounds on $D^k(p)$ in Theorem 5.1

in bucket i, such that $D^k(p) > w - 1$. Let $Q = \{x | x \in (p - D^k(p), p + D^k(p))\}$. On one hand,

according to the definition of $D^k(p)$, $|Q| \leq k$. On the other hand,

$$D^k(p) > w-1 \Rightarrow (p-D^k(p), p+D^k(p)) \supseteq [p-w+1, p+w-1],$$

indicates that Q must include all data points in bucket i. Thus, $|Q| \geq f_i > k$ is a contradiction to $|Q| \leq k$. ■

Theorem 5.2 *We define a function as*

$$F(t,i) = \sum_{j=i-t}^{i+t} f_j.$$

If $f_i \leq k$, we can find an integer $s \geq 0$, such that $F(s,i) \leq k$ and $F(s+1,i) > k$. Then, $l_i = s \cdot w$ and $u_i = (s+2) \cdot w - 1$ are the lower and upper bounds for $D^k(p)$, where p is any data point in bucket i.

Proof: According to the condition $F(s,i) \leq k$, there are at most k data points from bucket $i-s$ to

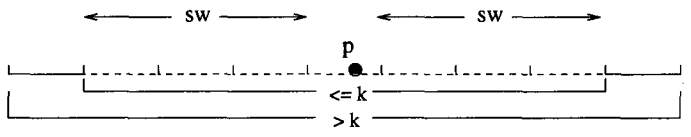


Figure 5.2: Bounds on $D^k(p)$ in Theorem 5.2

bucket $i+s$ inclusively. With the condition $F(s+1,i) > k$, we can derive that the KNN of data point p must be in either bucket $i-s-1$ or bucket $i+s+1$. Without loss og generality, assume data point q is the KNN of p and q is in bucket $i-s-1$, i.e., $D^k(p) = |p-q| = p-q$. Thus, we have

$$\begin{aligned} p-q &> min_i - max_{i-s-1} = s \cdot w; \\ p-q &\leq (max_i - 1) - min_{i-s-1} = (s+2) \cdot w - 1. \end{aligned}$$

Therefore, we can derive the bounds for $D^k(p)$ as

$$D^k(p) \in (s \cdot w, (s+2) \cdot w - 1].$$

■

As shown above, the histogram information helps us derive lower and upper bounds on $D^k(p)$ for any data point p. We will utilize these theorems in our outlier detection schemes.

5.3.2 Outlier Detection for $O(d,k)$

In this section, we propose a histogram-based protocol for detecting outliers defined by Definition 3. Our approach includes two stages. In the first stage, we divide the data value range into uniform buckets and collect equi-width histogram information. The histogram provides us with useful information to identify data points as outliers or non-outliers. However, the histogram information may not be sufficient to identify every data point. We call those data points *potential outliers* if they cannot be identified by the histogram. In the second stage, the sink gathers the potential outliers from the sensor network and checks their distance to the KNN. Eventually, the sink will identify all outliers.

In the rest of this section, we introduce a basic scheme which uses a single-round histogram information collection and an enhanced scheme which refines the histogram through multiple-round histogram collections.

5.3.2.1 Basic Scheme

In this subsection, we present a basic scheme for $O(d,k)$ outlier detection with a single-round of histogram collection.

Obtain v_{min} and v_{max}: In the first step, the sink queries for the minimum and maximum data values in the sensor network in order to calculate the value range. Let v_{min} and v_{max} be the minimum and maximum values received by the sink. In this step, every sensor sends at most $\log(v_{min} \cdot v_{max})$ bits of information.

Collect Histogram: In the second step, the sink collects the histogram from the sensor network. To obtain the histogram, every sensor and the sink have to agree on the same bucket partition, which can be specified by the bucket width w and the value range $[v_{min}, v_{max}+1)$[1]. For an easy exposition, we fix the bucket width w to d. We will explain why we set this width rather than other values in the next step. In this step, the sink diffuses a query to the sensor network including d, v_{min} and v_{max}, as well as the other parameter, k. Every non-leaf node sends $\log(k \cdot d \cdot v_{max} \cdot v_{min})$ bits of information to forward the query to its children. Let l be the width of $[v_{min}, v_{max}+1)$ defined as $l = v_{max} - v_{min} + 1$. Sensors divide the value range $[v_{min}, v_{max}+1)$ into $\lceil \frac{l}{d} \rceil$ uniform buckets with width d after they receive the histogram query, i.e., the ith bucket is defined by $[v_{min} + (i-1) \cdot d, v_{min} + i \cdot d)$. Starting from the leaf nodes, each sensor aggregates the number of data points in each bucket from all its descendants. Let g_i^j be the number of data points generated by sensor j in bucket i, and f_i^j be the histogram summary of bucket i sent by sensor j. For a leaf node j, $f_i^j = g_i^j$. For a non-leaf node j, assume it receives summaries from its children $c_1, c_2, \cdots$. All these summaries are aggregated as well as its own summary,

$$f_i^j \Leftarrow g_i^j + f_i^{c_1} + f_i^{c_2} + \cdots.$$

Finally, the aggregated number of data points in each bucket is forwarded to j's parent. In this step, we do not have to maintain the exact histogram as long as we can apply Theorems 5.1 and 5.2

[1] We use an inclusive lower bound and exclusive upper bound to define a value range.

later. Therefore, if $f_i^j > k+1$, we will reset it as $k+1$ in order to reduce communication costs. In this way, every node transfers at most $\lceil \frac{l}{d} \rceil \cdot \log(k+1)$ bits of data back and the sink finally obtains the value of f_i.

Collect Outliers and Potential Outliers: In the third step, the sink applies Theorem 5.1 and Theorem 5.2 on every bucket i based on f_i to assign l_i and u_i. If u_i is set by Theorem 5.1, we will get $u_i = w-1 < d$. On the other hand, if l_i is set by Theorem 5.2 and greater than 0, we must have $l_i = sw \geq d$. Based on the definition of $O(d,k)$, the sink analyzes the received histogram information as follows:

- Case 1: If $u_i < d$, all data points in bucket i are non-outliers and bucket i is called a *non-outlier bucket*;
- Case 2: If $l_i \geq d$, all data points in bucket i are outliers and bucket i is called an *outlier bucket*;
- Case 3: Otherwise, bucket i is called a *potential outlier bucket* and the data points in it are called *potential outliers*.

As we mentioned earlier, w can be set to other values. If $w > d$, however, Theorem 5.1 will not help us identify non-outlier buckets because there is no derivation from $u_i = w-1$ to $u_i < d$. On the other hand, if $w < d$, more detailed information is obtained and more outlier buckets are identified. However, smaller buckets incur more communication costs and some non-outlier buckets might be overlooked. Therefore, without any prior knowledge of the data, d is a conservative value for w to achieve good performance.

The non-outliers identified in Case 1 can be ignored, because they should not be returned in the result. For Case 2, the sink can send another query to indicate the outlier buckets and collect

all the identified outliers from every sensor. To process the potential outliers in Case 3, we use a simple method to first obtain all potential outliers and then, in the next step, find the distance to their individual KNNs to determine whether each potential outlier is actually an outlier or not.

Considering all three cases above, the sink diffuses a query, which includes a vector of length $\lceil \frac{l}{d} \rceil$,

$$\{q_1 q_2 \cdots q_{\lceil \frac{l}{d} \rceil}\},$$

where q_i is a one-bit flag satisfying

$$q_i = \begin{cases} 0 & \text{if bucket } i \text{ is a non-outlier bucket;} \\ 1 & \text{otherwise.} \end{cases}$$

In other words, $q_i = 1$ indicates that all data points in bucket i need to be returned, because they are either outliers or potential outliers. After receiving this query, every sensor will scan its own data set and return all data points in the marked buckets along the routing tree. The query diffusion cost for each non-leaf sensor is $\lceil \frac{l}{d} \rceil$ bits. However, the cost of collecting potential data depends on the histogram obtained in the previous steps. Suppose N_o is the number of the identified outliers and N_{po} is the number of the potential outliers. Thus, the number of data points needed in this step is $N_o + N_{po}$. We assume the communication cost is proportional to the data size and the distance between the sender and receiver. Therefore, the cost of collecting data in this step is estimated as

$$(N_o + N_{po}) \cdot \log v_{max} \cdot avgDist,$$

where $avgDist$ is the average hop distance between the sink and sensors.

Diffuse Potential Outliers and Count the Number of Neighbors within d: In the last step, the sink combs through the collected data points in the potential outlier buckets. Some data points

may be immediately identified as outliers or non-outliers.For example, data points in a potential outlier bucket will be identified if the data points in the two neighboring buckets are also collected. Unfortunately, some data points remained can not be identified as either outliers or non-outliers. The sink sends those remaining potential outliers to the sensor network in order to find the actual outliers among them. The query is formed as $\{p_1, p_2, \ldots,\}$, which includes all the remaining potential outliers. Every sensor will forward the query to its children until the query reaches the leaf nodes. To answer such a range query, starting from the leaf nodes, every sensor sends a vector of summaries, one for each data point, to its parent, $\{f'_1, f'_2, \ldots,\}$, where f'_i is the number of the data points within $[p_i - d, p_i + d]$, i.e., the number of p_i's neighbors within distance d. A non-leaf sensor will sum up the summaries from its children as well as the summary of its own data set and forward the aggregated summaries to its parent. Similar to the second step, if $f'_i > k+1$, we will reset it to $k+1$.

This step can be optimized by filtering out some unnecessary diffusion at each node based on the histogram obtained previously. For example, consider a potential outlier p, which belongs to bucket $i = \lceil \frac{p - v_{min} + 1}{d} \rceil$. If a sensor j finds $f^j_{i-1} + f^j_{i+1} = 0$, according to the previous histogram, there is no need to diffuse p_i to its children, because all possible neighbors of p_i in the subtree rooted at this sensor are in bucket j, so we can immediately set the summary for p to f^j_i.

Eventually, the sink receives a value for each potential outlier, which represents the number of data points within distance d of p_i. The sink may simply scan the summary list and determine whether the ith potential outlier is actually an outlier by determining if $f'_i \leq k$.

In this final step, the sink diffuses all the potential outliers collected to every sensor. The diffusion cost is at most $N_{nl} \cdot N_{po} \cdot \log v_{max}$, where N_{nl} is the number of non-leaf nodes. The cost of transferring the return summaries is bounded by $N \cdot N_{po} \cdot \log(k+1)$. To summarize, the total cost

of the basic scheme, denoted by C_{basic}, is estimated as

$$\begin{aligned} C_{basic} &= N \cdot \log(v_{min} \cdot v_{max}) + N_{nl} \cdot \log(k \cdot d \cdot v_{min} \cdot v_{max}) + N \cdot \lceil \frac{l}{d} \rceil \cdot \log(k+1) \\ &+ N_{nl} \cdot \lceil \frac{l}{d} \rceil + (N_o + N_{po}) \cdot \log v_{max} \cdot avgDist \\ &+ N_{nl} \cdot N_{po} \cdot \log v_{max} + N \cdot N_{po} \cdot \log(k+1). \end{aligned} \tag{5.1}$$

5.3.2.2 Enhanced Scheme

A drawback of the basic scheme is that if there are many potential outliers, i.e., N_{po} in Eq.(5.1) is very large, collecting and diffusing them will incur a large communication cost. In this section, we propose an enhanced scheme that refines some of the histogram before we query for the potential outliers. We expect that more rounds of histogram queries can help us prune out more data points, i.e., the number of potential outliers can be further reduced. In the following, our enhanced scheme only considers at most one extra round of histogram collection. The algorithm and analysis, however, can be used for more rounds of histogram collection in the same manner.

The first two steps of the enhanced scheme are quite similar to the basic scheme. After receiving the histogram in step 2, however, the sink has two options. First, the sink can follow step 3 in the basic scheme, collecting the potential outliers. In the other option, the sink can send a query for another histogram with a new bucket width $w = d' < d$ and then continue step 3 in the basic scheme. One more histogram with a smaller bucket width leads to more accurate information that helps to reduce ambiguous potential outliers. However, collecting more detailed histogram incurs extra communication cost. Thus, we need to determine if refining histogram is worthwhile and if so what is the appropriate bucket width for the new query. According to Eq.(5.1), if we do not

query for more histogram, the estimated cost of the remaining steps, denoted by $Cost'$, is

$$\begin{aligned} Cost' &= N_{nl} \cdot \lceil \frac{l}{d} \rceil + (N_o + N_{po}) \cdot \log v_{max} \cdot avgDist \\ &\quad + N_{nl} \cdot N_{po} \cdot \log v_{max} + N \cdot N_{po} \cdot \log(k+1). \end{aligned} \tag{5.2}$$

In the following, we analyze the cost of collecting more histogram and propose an algorithm to determine the optimal value of the new width d'. The minimum cost achieved by using d' will be compared with $Cost'$ to determine which option is better.

In this enhanced scheme, we keep the first two steps of the basic scheme with the following changes:

- In the first step, besides v_{min} and v_{max}, the sink also queries for the total hop distance to the sink and the number of non-leaf nodes. These two values can be aggregated at each node and we use $tolDist$ and N_{nl} to represent the results received by the sink respectively.
- In the second step of collecting the histogram, the upper limit of data points count f_i^j is set to $k \cdot d$, instead of $k+1$. We will explain it in the analysis later.

After step 2, we set $avgDist = \frac{tolDist}{N}$ and estimate $Cost'$ as defined in Eq.(5.2).

Next, we estimate the cost after step 2 if we send one more histogram query. Essentially, the extra round of histogram query only targets at potential outlier buckets as well as their related neighboring buckets. Assume the new bucket width is set to $d' = \frac{d}{B}$ for the next round of histogram query. To reuse the previously collected histogram information, we choose B to be an integer. The query sent by the sink includes B and a vector of bits which mark the potential outlier buckets identified by Theorems 5.1 and 5.2,

$$\{B, q_1 q_2 \cdots q_{\lceil \frac{l}{d} \rceil}\},$$

where $q_i = 1$ if bucket i is a potential outlier bucket. Thus, the cost of query diffusion is $N_{nl} \cdot (\log B + \lceil \frac{l}{d} \rceil)$. After receiving the query message, each sensor will know the new bucket width d' and the potential outlier buckets. The sensors divide each of potential outlier buckets and their neighboring buckets into B uniform sub-buckets. Similarly, every sensor generates a histogram for the new sub-buckets. This information is aggregated bottom-up along the routing tree and finally reaches the sink. One optimization that each sensor can apply is to transfer the first $B-1$ summaries for each target bucket i instead of B summaries, because its parent already knows the total number of data points in bucket i, the last summary can be derived from the available information. The number of buckets involved in the reply can be easily counted as follows:

$Count \leftarrow 0$

if $q_1 + q_2 > 0$ **then** $Count \leftarrow Count + 1$

if $q_{\lceil \frac{l}{d} \rceil - 1} + q_{\lceil \frac{l}{d} \rceil} > 0$ **then** $Count \leftarrow Count + 1$

for $i = 2$ to $\lceil \frac{l}{d} \rceil - 1$ **do**

  if $q_{i-1} + q_i + q_{i+1} > 0$ **then** $Count \leftarrow Count + 1$

In the return stage, each sensor transfers at most $B \cdot Count \cdot \log(k+1)$ bits of information. Therefore, the total cost of querying and collecting the refined histogram is

$$N_{nl} \cdot (\log B + \lceil \frac{l}{d} \rceil) + N \cdot B \cdot Count \cdot \log(k+1). \tag{5.3}$$

Assume after obtaining more histogram information, we identify EN_o outliers and EN_{po} potential outliers. Following step 3 in the basic scheme, we need collect outlier data and further

check potential outliers. Similar to Eq.(5.2), the estimated cost of the remaining steps is,

$$N_{nl} \cdot \lceil \frac{l}{d} \rceil + (EN_o + EN_{po}) \cdot \log v_{max} \cdot avgDist$$
$$+N_{nl} \cdot EN_{po} \cdot \log v_{max} + N \cdot EN_{po} \cdot \log(k+1). \tag{5.4}$$

Therefore, the total cost incurred by refining histogram and its consequence after step 2 is estimated as

$$Cost(d') = \text{Eq.(5.3)+Eq.(5.4)}.$$

In this problem, we aim to find the optimal d' such that $Cost(d')$ is minimized and compare it with $Cost'$ to decide if refining histogram is worthwhile.

In order to calculate $Cost(d')$, we first estimate EN_o and EN_{no} in Eq.(5.4) based on the histogram information collected in step 2. We assume that data are randomly distributed in each bucket. Let us take a close look at a potential outlier bucket i. After the refined histogram query, we will get B summaries for bucket i as well as bucket $i-1$ and $i+1$. Each new summary is responsible for a sub-bucket of the original buckets. Let us label the jth sub-bucket of the original bucket i as bucket $b_{(i-1)B+j}$. Let $f'_{j'}$ be the number of data points in sub-bucket $b_{j'}$. In the following, we estimate the probabilities that a sub-bucket j' is a non-outlier bucket or outlier bucket, indicated by $P_{no}(j')$ and $P_o(j')$ respectively. The probability of being a potential outlier bucket is $1 - P_{no}(j') - P_o(j')$. Thus, EN_o and EN_{po} can be derived as

$$\begin{aligned} EN_o &= \sum_{b_{j'}} f'_{j'} P_o(j'), \\ EN_{po} &= \sum_{b_{j'}} f'_{j'} (1 - P_{no}(j') - P_o(j')). \end{aligned}$$

To ensure $b_{(i-1)B+j}(1 \leq j \leq B)$ is a non-outlier bucket, we must have

$$\sum_{q=(i-1)B+j-B+1}^{(i-1)B+j+B-1} f'_q > k.$$

As illustrated in Fig. 5.3, the left side can be derived as

$$\begin{aligned}\sum_{q=(i-2)B+j+1}^{iB+j-1} f'_q &= \sum_{q=(i-2)\cdot B+j+1}^{(i-2)\cdot B+B} f'_q + \sum_{q=(i-1)\cdot B+1}^{(i-1)\cdot B+B} f'_q + \sum_{q=i\cdot B+1}^{i\cdot B+j-1} f'_q \\ &= f_i + \sum_{q=(i-2)B+j+1}^{(i-1)B} f'_q + \sum_{q=iB+1}^{iB+j-1} f'_q.\end{aligned}$$

Thus, a sub-bucket $b_{(i-1)B+j}$ is a non-outlier bucket if

$$\sum_{q=(i-2)B+j+1}^{(i-1)B} f'_q + \sum_{q=iB+1}^{iB+j-1} f'_q > k - f_i.$$

Let a be the first term (the number of the data points in the rightmost $B-j$ sub-buckets of bucket $i-1$) and b be the second term (the number of the data points in the leftmost $j-1$ sub-buckets of bucket $i+1$). Thus, $P_{no}((i-1)B+j)$ is the probability that $a+b > k-f_i$. We define a function $P(x,y,z)$ to be the probability that the number of data points in the leftmost or rightmost y sub-buckets of bucket x is z. Based on the assumption of random data distribution in every bucket,

$$P(x,y,z) = \binom{f_x}{z} (\frac{y}{B})^z (\frac{B-y}{B})^{f_x - z}.$$

Thus, $P_{no}((i-1)B+j)$ can be calculated as

$$\sum_{a=0}^{f_{i-1}} (P(i-1, B-j, a) \cdot \sum_{b=k-f_i-a+1}^{f_{i+1}} P(i+1, j-1, b)).$$

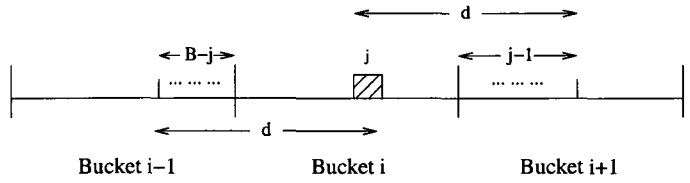


Figure 5.3: Data involved in identifying a non-outlier sub-bucket

On the other hand, we can claim that $b_{(i-1)B+j}$ is an outlier bucket if the following condition is satisfied:

$$\sum_{q=(i-1)B+j-B}^{(i-1)B+j+B} f'_q \leq k.$$

By similar analysis as above, it becomes

$$\sum_{q=(i-2)B+j}^{(i-1)B} f'_q + \sum_{q=iB+1}^{iB+j} f'_q \leq k - f_i,$$

as shown in Fig. 5.4. Let a be the first term and b be the second term. Then, $P_o((i-1)B+j)$ can

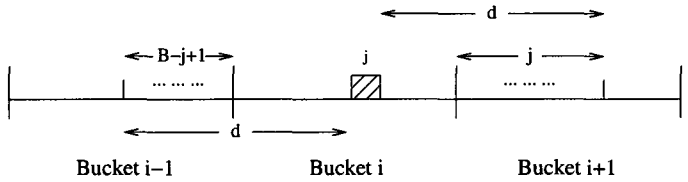


Figure 5.4: Data involved in identifying an outlier sub-bucket

be calculated as

$$\sum_{a=0}^{f_{i-1}} (P(i-1, B-j+1, a) \cdot \sum_{b=0}^{k-f_i-a} P(i+1, j, b)).$$

Besides the above analysis, we use a short-cut estimation if a potential bucket i's neighboring buckets reach the histogram limit $k \cdot d$. Assume $f_{i+1} = k \cdot d$, expectedly, the frequency of each value in bucket $i+1$ is k. Thus, every data point in bucket i has a high probability to be a non-outlier. For such a bucket i, we skip the probabilistic analysis and directly increase EN_o by f_i.

Algorithm 11 determines the optimal bucket width d' for the second round of histogram query. Initially, we scan every bucket and use an array M to mark the potential outlier buckets,

$$M[i] = \begin{cases} 1 & \text{if bucket } i \text{ is a potential outlier bucket;} \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm is constructed by two embedded loops. In the outer loop (lines 7–22), we try different bucket width $d' = \frac{d}{B}$ by testing all possible B. For each B, we estimate the cost incurred by this round of refined query and the subsequent steps for raw data collection. The optimal width yields the minimum value of $Cost$, which is tracked by the variables $optB$ and min in Algorithm 11. If the final value of min is no less than $Cost'$, there is no need to conduct an extra round of histogram query. Otherwise, we set $d' = \frac{d}{optB}$ and do the refined histogram query.

The inner loop of this algorithm (lines 9–17) checks every bucket to estimate the cost. There will be B sub-buckets for each requested bucket and reporting a histogram of them needs $B \cdot \log(k+1)$ bits of data. A bucket i will be involved only if it is a potential outlier bucket or one of its neighboring buckets is a potential outlier bucket. Additionally, EN_o and EN_{po} are accumulated in the inner loop when checking potential outlier buckets.

The implementation of EstNO used in Algorithm 11 is shown in Algorithm 12. Basically, for every sub-bucket, we calculate its probability of being a non-outlier bucket. The loop variable j set from 2 to $B-1$ is the index of sub-buckets. The first and last sub-buckets are special cases, which are handled in lines 16–22. For each sub-bucket j, q_1 is the probability that a data point in bucket $i-1$ resides in the rightmost $B-j$ sub-buckets of bucket $i-1$ and q_2 is the probability that a data point in bucket $i+1$ is within the leftmost $j-1$ sub-buckets of bucket $i+1$. In this algorithm, a represents the number of data points in the rightmost $B-j$ sub-buckets of bucket $i-1$ and b is the number of data points in the rightmost $j-1$ sub-buckets of bucket $i+1$. We

enumerate all possible combinations of a and b, which satisfy $a+b > k - f_i$, and calculate the probabilities for values a and b, indicated by p_1 and p_2 respectively. The sum of the product $p_1 p_2$ for all possible cases becomes the probability that sub-bucket j is a non-outlier bucket. This value is recorded in variable p. On average, there are $\frac{f_i}{B}$ data points in sub-bucket j, so $p\frac{f_i}{B}$ is the expected number of non-outliers in sub-bucket j. After checking every bucket, we store the total number of non-outliers in r and return it as the result.

Algorithm 13 shows the details of EstO used in Algorithm 11. It has a quite similar structure to Algorithm 12. In this algorithm, we have the same definition of variables p_1, p_2, a, and b. However, a sub-bucket is estimated as an outlier bucket if $a+b \leq k - f_i$.

Algorithm 11 Find the Optimal Bucket Width

1: **for** $i = 1$ to $\lceil \frac{l}{d} \rceil$ **do**

2: **if** $f_i \leq k$ and $f_{i-1} + f_i + f_{i+1} > k$ **then**

3: $M[i] = 1, N_{po} = N_{po} + f_i$

4: **end if**

5: **end for**

6: $min = Cost' =$ Eq.(5.2)

7: **for** $B = 2$ to d **do**

8: $Cost = N \cdot (\log B + \lceil \frac{l}{d}) \rceil$

9: **for** $i = 1$ to $\lceil \frac{l}{d} \rceil$ **do**

10: **if** $M[i-1] + M[i] + M[i+1] > 0$ **then**

11: $Cost = Cost + N \cdot (B-1) \log(k+1)$

12: **end if**

13: **if** $M[i] = 1$ **then**

14: $e = \text{EstO}(B, i), EN_o = EN_o + e$

15: $EN_{po} = EN_{po} + f_i - \text{EstNO}(B, i) - e$

16: **end if**

17: **end for**

18: $Cost = Cost + EN_o \cdot avgDist \cdot \log v_{max} + EN_{po} \cdot (avgDist \cdot \log v_{max} + N_{nl} \cdot \log v_{max} + N \cdot \log(k+1))$

19: **if** $Cost < min$ **then**

20: $optB = B, min = Cost$

21: **end if**

22: **end for**

23: **if** $min = Cost'$ **then**

24: there is no need for more histogram query

25: **else**

Algorithm 12 EstNO(B, i)

1: $t = k - f_i$

2: **for** $j = 2$ to $B - 1$ **do**

3: $q_1 = \frac{B-j}{B}$, $q_2 = \frac{j-1}{B}$, $p = 0$

4: **for** $a = 0$ to f_{i-1} **do**

5: $p_1 = \begin{pmatrix} f_{i-1} \\ a \end{pmatrix} q_1^a (1 - q_1)^{f_{i-1} - a}$

6: **if** $a > t$ **then** $p_2 = 1$

7: **else if** $t + 1 - a > f_{i+1}$ **then** $p_2 = 0$

8: **else**

9: **for** $b = t + 1 - a$ to f_{i+1} **do**

10: $p_2 = p_2 + \begin{pmatrix} f_{i+1} \\ b \end{pmatrix} q_2^b (1 - q_2)^{f_{i+1} - b}$

11: **end for**

12: $p = p + p_1 p_2$

13: **end for**

14: $r = r + p \frac{f_i}{B}$

15: **end for**

16: **if** f_{i-1} or $f_{i+1} > t$ **then**

17: $p = 0$

18: **for** $a = t + 1$ to f_{i-1} or f_{i+1}

19: $p = p + \begin{pmatrix} f_{i-1} \text{ or } f_{i+1} \\ a \end{pmatrix} (\frac{B-1}{B})^a$

20: **end for**

21: $r = r + p \frac{f_i}{B}$

22: **end if**

Algorithm 13 EstO(B, i)

1: $t = k - f_i$

2: **for** $j = 2$ to $B - 1$ **do**

3: $q_1 = \frac{B-j}{B}, q_2 = \frac{j-1}{B}, p = 0$

4: **for** $a = 0$ to t **do**

5: $p_1 = \begin{pmatrix} f_{i-1} \\ a \end{pmatrix} q_1^a (1 - q_1)^{f_{i-1}-a}$

6: **for** $b = 0$ to $t - a$ **do**

7: $p_2 = p_2 + \begin{pmatrix} f_{i+1} \\ b \end{pmatrix} q_2^b (1 - q_2)^{f_{i+1}-b}$

8: **end for**

9: $p = p + p_1 p_2$

10: **end for**

11: $r = r + p\frac{f_i}{B}$

12: **end for**

13: $p = 0$

14: **for** $a = 0$ to t

15: $p = p + (\begin{pmatrix} f_{i-1} \\ a \end{pmatrix} + \begin{pmatrix} f_{i+1} \\ a \end{pmatrix}) \cdot (\frac{B-1}{B})^a$

16: **end for**

17: $r = r + p\frac{f_i}{B}$

18: return r

5.3.3 Outlier Detection for $O(n,k)$

In this section, we present a solution to detect the outliers defined by Definition 4. Given k and n, we sort all data points according to the distances to their KNNs. Let the sorted points be $p_1, p_2, \ldots,$ where $D^k(p_i) \geq D^k(p_j)$ for $i < j$. The first n data points, $p_1, \ldots, p_n$, are all the $O(n,k)$ outliers we are looking for.

Our approach is still based on equi-width histogram. The sink sends histogram queries for multiple iterations and tries to find a suitable cut-off value c that separates $D^k(p_n)$ and $D^k(p_{n+1})$. The histogram collected in each iteration gives us an estimation for the range of c and helps filter out the buckets that are out of our interests. Then we use the next query to obtain more detailed histogram of the buckets that possibly hold outliers. This query process is repeated till we find all outliers. Note this approach does not fetch and check potential outliers as in the last step of finding the $O(d,k)$ outliers. Checking a potential outlier in this problem is very costly when k is large, because every sensor has to send k data values (k nearest neighbors of the potential outlier).

In the following, we first show that we can estimate bounds for the cut-off value c based on the histogram information. We try to find a pair of values L_c and U_c, such that $c \in (L_c, U_c]$. Suppose the sink sends a histogram query with bucket width w. After receiving the histogram, we first apply Theorem 5.1 and Theorem 5.2 on every bucket i to calculate l_i and u_i. Then we calculate L_c and U_c according to the following theorems.

Theorem 5.3 *Consider the histogram collected with bucket width w. We have*

$$L_c = \max \{x | \sum_{l_i \geq x} f_i > n, x \text{ is multiple of } w\} < c.$$

Proof: In the above equation, the condition, $\sum_{l_i \geq x} f_i > n$, means that there are more than n data points (p) satisfying $D^k(p) \geq x$. Based on the definition of the cut-off value c, $x < c$. Thus, any x

satisfying the condition can be an exclusive lower bound of c. ■

Theorem 5.4 *Consider the histogram collected with bucket width w. We have*

$$U_c = \min \{x | \sum_{u_i \geq x} f_i \leq n,\ x+1 \text{ is multiple of } w\} \geq c.$$

Proof: The condition, $\sum_{u_i \geq x} f_i \leq n$, means that the number of all possible data points (p) satisfying $D^k(p) \geq x$ is less than or equal to n. According to the definition of c, $x \geq c$. Thus, any x satisfying the condition can be an inclusive upper bound of c. ■

Our solution is shown in Algorithm 14. We start a histogram query with an initial bucket width w_{init}. Based on the received histogram, we obtain l_i and u_i for each bucket i and calculate L_c and U_c (lines 4–6). Then, we categorize buckets as follows:

- Case 1: If $u_i \leq L_d$, bucket i is a *non-outlier bucket*;
- Case 2: If $l_i \geq U_d$, bucket i is an *outlier bucket*;
- Case 3: Otherwise, bucket i is a *potential outlier bucket.*

Similar to the $O(d,k)$ outlier detection, we ignore the non-outliers in case 1 and send another query to collect the data values in the outlier buckets (lines 7–10). For case 3, we query for more histogram information of potential outlier buckets, which are marked by the variable q_i (lines 12–16). The bucket width of the new query is set to the half of the current bucket width. Upon receiving the query, sensor nodes calculate the histogram of the marked buckets (by q_i) with new bucket width, and send it back to the sink in a bottom-up direction. We repeat this process until all outliers are found. In the worst case, we need $\log w_{init}$ iterations.

Algorithm 14 Find $O(n,k)$ Outliers

```
1:  w = w_init, hasPO = true, q_1 = q_2 = ··· = 1
2:  while w > 1 and hasPO do
3:      Send a query with < w, q_1 q_2 ··· > and collect the
        histogram with bucket width w
4:      Calculate l_i and u_i for each bucket i
5:      L_d = max {x | Σ_{l_i ≥ x} f_i > n}
6:      U_d = min {x | Σ_{u_i ≥ x} f_i ≤ n}
7:      for i = 1 to ⌈l/w⌉ do
8:          q_i = (l_i ≥ U_d)
9:      end for
10:     Send a query with {q_i} and collect the outliers
11:     hasPO = false
12:     for i = 1 to ⌈l/w⌉ do
13:         if l_i < U_d and u_i > L_d then
14:             q_i = 1, hasPO = true
15:         else q_i = 0
16:     end for
17:     w = w/2
18: end while
```

5.4 Performance Evaluation

Our evaluation is based on simulations. We conduct examinations on both real data trace and synthetic datasets. In the following, we will show the performance results separately.

5.4.1 Real Data Trace

In the first simulation, we use real datasets from Intel Lab [2]. The data were collected from 54 sensors during a one-month period. The details of the dataset can be found at Intel Lab's web site [2]. We consider a 100×100 network field, where the sink is placed in the center. We deploy 54 sensor nodes randomly in the field and assume sensors communicate in a multi-hop fashion. The communication range is set to 18 for good connectivity in a random topology. Two important parameters used in our algorithms, the number of non-leaf nodes and the average hop distance, are shown in Table 5.1. The values are average measurements of 1000 connected random topologies.

In this simulation, we select the entire temperature records on two dates (03/01 and 03/20) as two datasets. The dataset for 03/01 represents a regular temperature distribution with mean value around 24 degrees. The dataset for 03/20, however, displays a large deviation from the average value. In this dataset, for some reason, 50 degrees is reported for many times, and a lot of data are sparsely scattered between 35 degrees and 50 degrees. We use precision 0.01 to round temperature values and scale them by 100 times in order to obtain integer values. The relevant parameters are also listed in Table 5.1 and Table 5.2.

In the following, we show the performance of our algorithms in terms of the total communication cost for finding all the outliers, which is the sum of all sensors' communication costs. We assume that the cost of transferring a message is proportional to the payload size, which includes the actual data size and necessary control information, e.g., the message type. Thus, in the fol-

Table 5.1: Network Setup

Number of Sensors (N)	54
Number of Non-leaf Nodes (N_{nl})	25.7
Radio Range	18
Average Hop Distance ($avgDist$)	4.26

Table 5.2: Data Characteristics

	03/01 Dataset	03/20 Dataset
Number of Data Points	91468	76871
Maximum Value	3424	5008
Minimum Value	1499	363
Value Range ($v_{max} - v_{min}$)	1926	4646

lowing, the total communication cost is measured by the total size of the messages transferred in the whole network. We first measure the communication costs for the *centralized scheme* through 1000 independent simulation runs and use the average value as the baseline. In the centralized scheme, the whole network transfers 575K bytes data for the 03/01 dataset on average and 514K bytes for the 03/20 dataset. The deviations for two datasets are 119K bytes and 113K bytes respectively. In addition, to evaluate our algorithms, we conduct 100 independent simulations for each parameter setting. We normalize the average communication costs in our algorithms against the baselines of the centralized scheme and show them as percentage values in the rest of this section.

5.4.1.1 $O(d,k)$ Outlier Detection

To compare the basic scheme and enhanced scheme with different parameters, we vary d and k separately. First, we fix $k = 100$ and vary d from 20 to 70 for the 03/01 dataset and from 50 to

450 for the 03/20 dataset. Fig. 5.5 shows the numbers of outliers with various d. We find the two datasets differ dramatically. For the 03/01 dataset, when we set $d = 70$ (i.e., 0.7 degree in original data), no outlier exists in the entire set. For the 03/20 dataset, however, when we use a large distance with $d = 100$ (1 degree in the original data), 126 outliers appear. We keep increasing d to 400 (4 degree), we still find one outlier. This figure indicates that the 03/20 dataset contains more scattered data points and yields more outliers for a certain (d, k) setting.

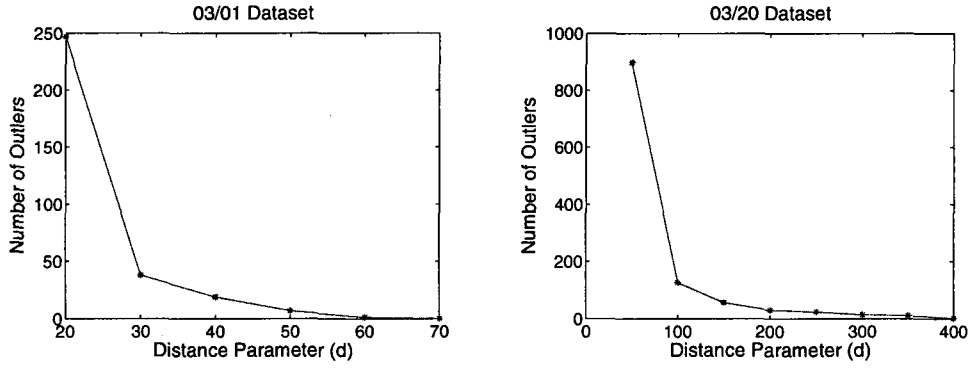


Figure 5.5: Number of outliers for varying d ($k = 100$)

The performance of basic and enhanced schemes is illustrated in Fig. 5.6. First, as shown, the enhanced scheme is always superior to the basic scheme. Secondly, both schemes greatly reduce communication costs. In the worst case in Fig. 5.6, the basic scheme consumes less than 5.5% of the cost of the centralized scheme.

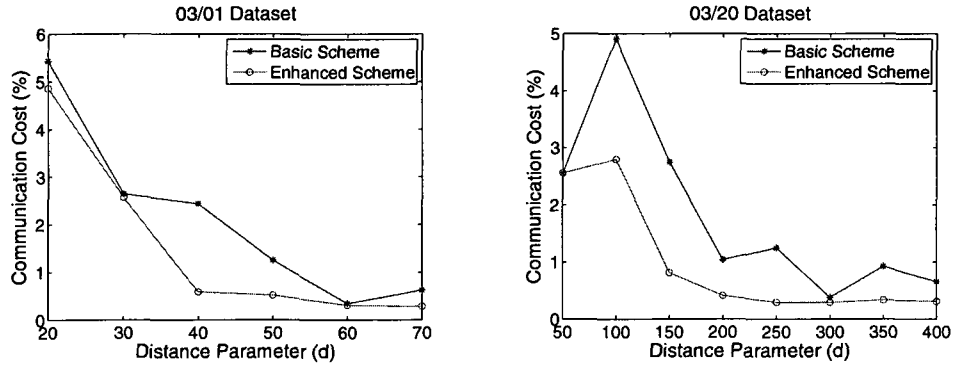


Figure 5.6: Communication costs for varying d ($k = 100$)

In the following, we will analyze and compare the performance of the basic and enhanced schemes. The communication cost in both schemes is comprised of histogram collection and raw data transfer (collection and diffusion). Considering the same value range defined by $[v_{min}, v_{max} + 1)$, larger bucket width yields smaller number of buckets and less cost in histogram collection. On the other hand, larger bucket width provides less detailed histogram information, which may increase the number of potential outliers and the cost in transferring raw data.

In the 03/01 dataset, when we query for outliers, most non-outliers are identified after the first round of histogram collection, and the number of potential outliers is limited. Thus, the cost of histogram collection is the dominant factor compared with the raw data transfer. As shown in Fig. 5.6, the performance keeps decreasing along the increasing d. In the 03/20 dataset, a lot of data are sparsely distributed over an abnormal range and the number of outliers is dramatically larger than that in the 03/01 dataset. Since we set larger bucket width for the 03/20 dataset, the cost of histogram collection is less than that in the 03/01 dataset. On the other hand, as we mentioned above, larger bucket width may increase the cost of raw data transfer due to insufficient histogram information. Therefore, the cost of histogram collection is no longer dominant as the difference with the cost of raw data transfer is alleviated. Sometimes, raw data transfer is even more significant than histogram collection. These two types of cost interact with each other and show unstable curves for the 03/20 dataset in Fig. 5.6. The enhanced scheme outperforms the basic scheme in both datasets by filtering out more potential outliers.

In our simulation, we also fix d and study the performance on variable k. Fig. 5.7 shows the change of the number of outliers and Fig. 5.8 is the performance comparisons. Similarly, we find that the enhanced scheme is better than the basic scheme and both schemes are very efficient. In this case, the cost of histogram collection is fixed for basic scheme and the communication cost

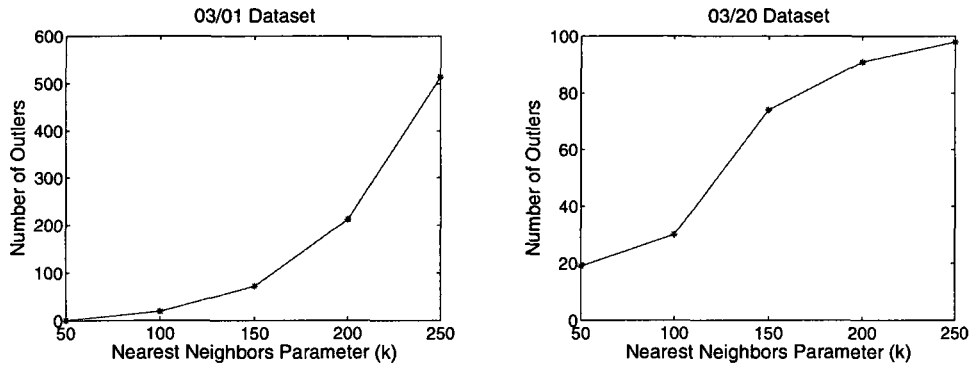


Figure 5.7: Number of outliers for varying k ($d = 40$ for the 03/01 dataset and $d = 200$ for the 03/20 dataset)

only depends on the number of potential outliers. For a given k, the data points with roughly k neighbors within distance d, have a high probability to be potential outliers, because it is hard to distinguish these data points by coarse histogram information. Thus, the trend of the curves in Fig. 5.7, which indicates the number of nearby potential outliers, has an impact on the communication cost. As we can see, for the 03/01 dataset, there is a sharp increase of outliers when $k \in [150, 250]$, which means that many potential outliers will be transferred as raw data when we search for outliers. Correspondingly, we see an increase of communication cost around that range in Fig. 5.8. Additionally, in the 03/20 dataset, the number of outliers has a jump from $k = 100$ to $k = 150$. It also yields an increased cost of basic scheme in Fig. 5.8. For both datasets, the enhanced scheme smooths the impact of the increased potential outliers and significantly improves the performance.

As a summary, in this trace-driven simulations, our proposed approaches are very efficient for the $O(d,k)$ outlier detection. The enhanced scheme consumes less than 4% of the cost of the centralized scheme in most cases.

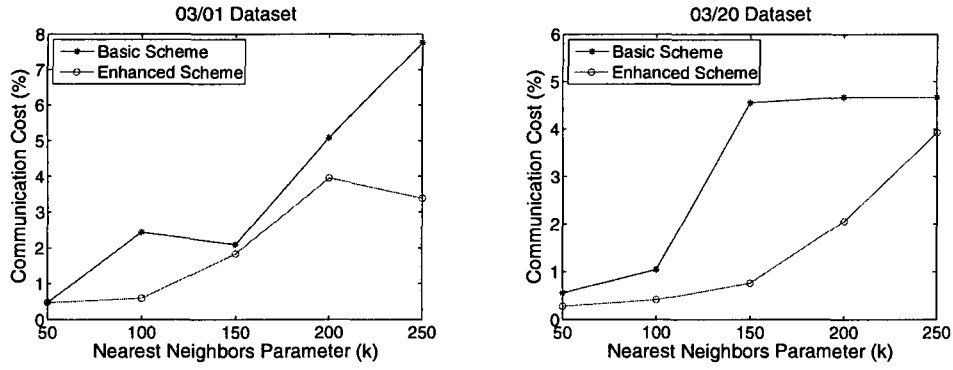


Figure 5.8: Communication costs for varying k ($d = 40$ for the 03/01 dataset and $d = 200$ for the 03/20 dataset)

5.4.1.2 $O(n,k)$ Outlier Detection

In this simulation, we set $k = 100$, and vary n from 10 to 80 for both datasets. The initial bucket width is set to a large value of 1500, i.e., $w_{init} = 1500$ in Algorithm 14. Fig. 5.9 shows the values for $D^k(p_n)$ and the communication cost is presented in Fig. 5.10.

The simulation results show that our approach is cost-efficient for the $O(n,k)$ outlier detection. Compared with the centralized scheme, our approach significantly reduces the communication cost. For the abnormal 03/20 dataset, it takes less than 1% of the cost to find all top-80 outliers. For the normal 03/01 dataset, our scheme consumes less than 1.5% of the cost in all the cases.

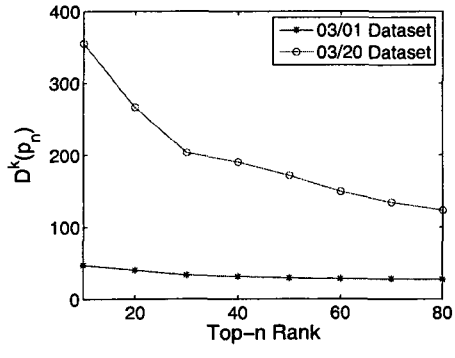


Figure 5.9: Values of $D^k(p_n)$ for varying n ($k = 100$)

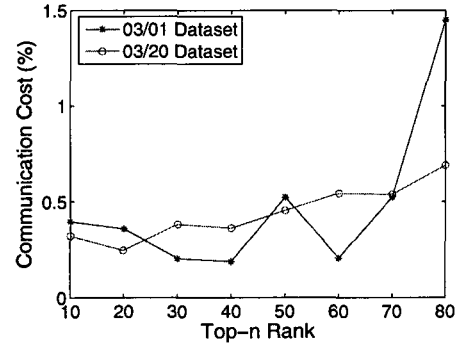


Figure 5.10: Communication costs for varying n ($k = 100$)

In details, we observe that the communication cost is related to the value of $D^k(p_n)$. When the

value of $D^k(p_n)$ drops greatly, the communication cost is small. On the contrary, when the curve of $D^k(p_n)$ becomes flat, the corresponding communication cost is increased. This relationship can be explained from two perspectives. First, our scheme tries to estimate a range for the cutoff value in each round, thus when the algorithm terminates, the estimated range must reside between $D^k(p_n)$ and $D^k(p_{n+1})$, i.e., $L_c > D^k(p_{n+1})$ and $U_c < D^k(p_n)$. Therefore, if $D^k(p_n)$ and $D^k(p_{n+1})$ are very close, the final estimated range will be very small. To obtain such accurate estimation, we have to use a small bucket width, which requires a large number of iterations. Second, if the value of $D^k(p_n)$ changes dramatically, it is easy to distinguish the non-outlier data points around the cut-off value. Thus, in each round, the number of the potential outlier buckets, which actually contain no outlier, will become small. This further decreases the cost of collecting histogram.

5.4.2 Synthetic Data Sets

Our second set of simulations uses synthetic data. Again we consider a 100×100 square network field, where the sink is placed in the center. We deploy 100 sensor nodes randomly in the field and the communication range is set to 14 for good connectivity in a random topology. The number of non-leaf nodes and the average hop distance are shown in Table 5.3. The numbers are averaged over 1000 connected random topologies.

Table 5.3: Network Setup

Number of Sensors (N)	100
Number of Non-leaf Nodes (N_{nl})	46.65
Radio Range	14
Average Hop Distance ($avgDist$)	5.32

This simulation is performed on two synthetic data sets, denoted by Dataset1 and Dataset2.

Each data set contains 10 ranges with dense data distribution {[0, 1000), [1500, 2500), [3000, 4000), $\cdots$, [13500, 14500)} and 10 ranges with sparse data distribution {[1000, 1500), [2500, 3000), [4000, 4500), $\cdots$, [14500, 15000)}. As we can see, the dense and sparse ranges alternate with each other, and the widths of each dense range and sparse range are 1000 and 500 respectively. We randomly distribute 10000 data points into each dense range. In addition, we generate a small number of data for sparse range. We intend to inject outliers among these data, especially the data in the central area of the sparse range. The number of sparse data is 1000 in Dataset1 and 100 in Dataset2. The parameters of the two data sets are listed in Table 5.4. For each data set, we randomly distribute the data to sensors and let every sensor hold the same amount of data.

Table 5.4: Data Characteristics

	Dataset1	Dataset2
Number of Sparse Data	1000	100
Number of Dense Data	100000	
Number of Clusters	10	
Width of Each Cluster	1000	
Width of Sparse Area	500	
Maximum Value	14999	
Minimum Value	0	

In the following, we show the performance of this set of simulations and discuss the results. The communication costs are also normalized against the cost of centralized scheme and appear as percentage values.

5.4.2.1 $O(d,k)$ Outlier Detection

Similar to the previous evaluation, we first fix $k = 100$ and vary d from 50 to 450 at an interval of 50. Fig. 5.11 shows the numbers of outliers with various d in Dataset1 and Dataset2. The curves are nearly linear with a turning point at $d = 250$.

Fig. 5.12 illustrates the performance comparison between the basic scheme and the enhanced scheme. In this outlier detection, most data in the dense ranges appear as non-outliers after the first histogram query, so the performance basically depends on identifying those data in the sparse ranges. As we mentioned, both schemes incur two types of costs, for histogram collection and raw data transfer. For a fixed value range $[v_{min}, v_{max} + 1)$, large bucket costs less in histogram collection, but it may increase the number of potential outliers. When analyzing the histogram of bucket i, we check $f_{i-1} + f_i + f_{i+1}$ to determine if bucket i is an outlier bucket. If the neighboring buckets overlap with the dense range, bucket i probably will become a potential outlier bucket. Thus, as d increases, more data in sparse ranges are marked as potential outliers after the first query. For example, in Dataset1, when $d = 50$, we can estimate that every sparse data point, whose distance to a dense range is less than 100, will become a potential outlier. Since we randomly distribute the sparse data, expectedly, there are $\frac{200}{500} \cdot 100 \cdot 9 + \frac{100}{500} \cdot 100 = 380$ potential outliers. When $d = 100$, this number is doubled to 760. At the same time, the cost of histogram query decreases very slowly. When d changes from 50 to 100, the number of buckets is only reduced from 300 to 150. So the raw data transfer dominates. Fig. 5.12 shows the increasing communication cost. The basic scheme reaches the maximum cost when $d = 250$, where every data point in sparse ranges becomes potential outliers. After the point of $d = 250$, as we continue increasing d, more sparse data will be identified as non-outliers after the first query, which reduces the number of potential outliers. Compared with the basic scheme, the enhanced scheme uses another histogram query to

reduce the number of outliers. As our simulation shows, the performance gain is much more than the cost incurred.

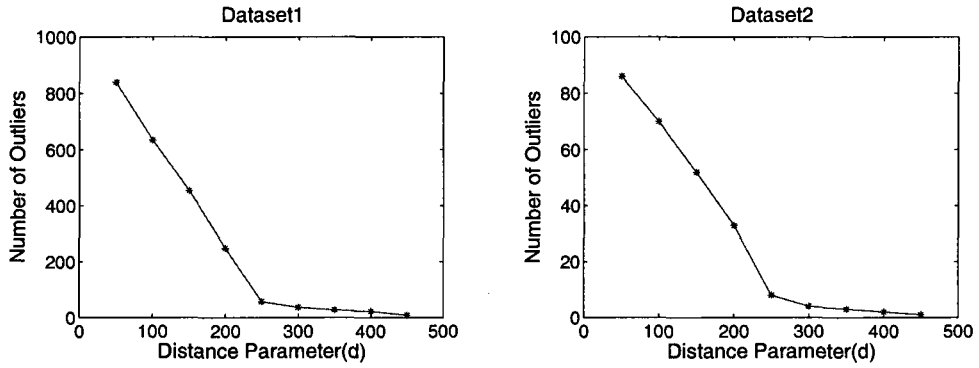


Figure 5.11: Number of outliers for varying d ($k = 100$)

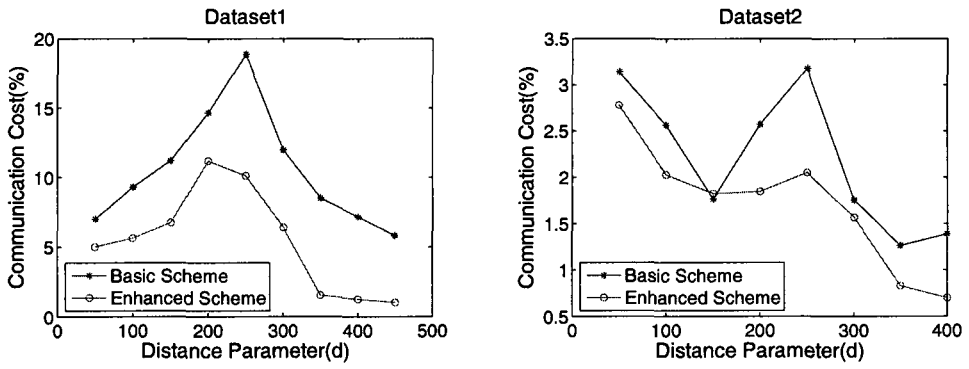


Figure 5.12: Communication costs for varying d ($k = 100$)

In Dataset2, due to the smaller number of sparse data, the costs are much lower than Dataset1. Using similar analysis mentioned earlier, when $d = 50$, the expected number of potential outliers is 38. Compared with the histogram of 300 buckets, raw data transfer is no longer dominant. As a result, we see a decreasing trend along increasing bucket size because histogram collection costs less for larger bucket. The costs of histogram collection and raw data transfer interact with each other and show an unstable curve for Dataset2. For example, when $d = 250$, we will have 100 potential outliers while the number of buckets is 60. Therefore raw data transfer becomes a major factor again. Also the enhanced scheme outperforms the basic scheme except when $d = 150$. This

abnormality is due to the fact that our cost estimations of fetching raw data and conducting more histogram queries are not accurate. When the actual costs of these two options are close, we may have chosen sending another histogram query by mistake, which causes the enhanced scheme to yield a little more cost than the basic scheme.

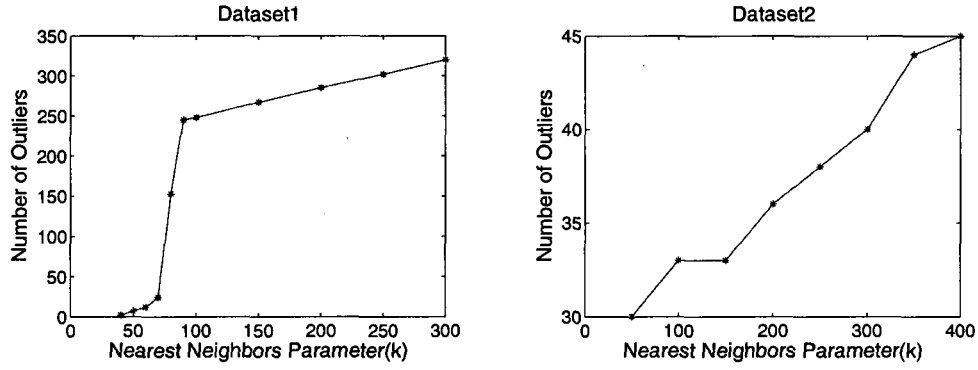


Figure 5.13: Number of outliers for varying k ($d = 200$)

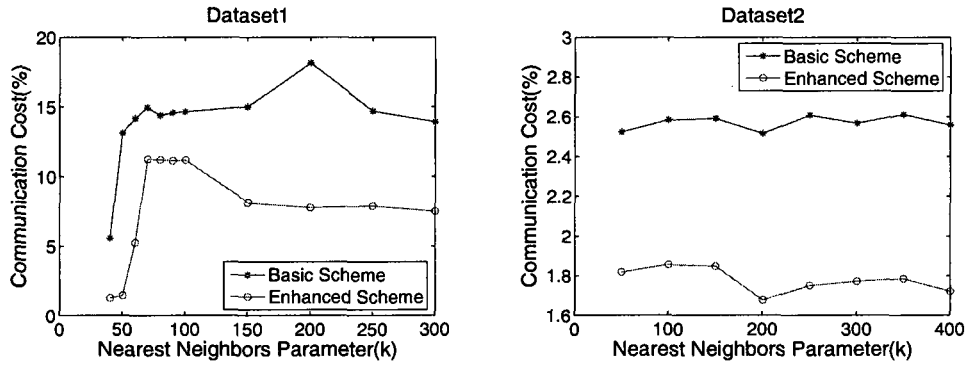


Figure 5.14: Communication costs for varying k ($d = 200$)

In our simulation, we also set $d = 200$ and study the performance on variable k. Fig. 5.13 shows the change of outliers and Fig. 5.14 is the performance comparison. Similarly, the communication costs depend on the number of potential outliers. For a given k, the data points with roughly k neighbors within distance d, have a high probability to be potential outliers, so the tangents of the curves in Fig. 5.13 indicate the trend of communication costs. As we can see, for Dataset1, there is a sharp increase of the number of outliers when $k \in [70, 90]$, which means many

potential outliers appear when we search for outliers. Correspondingly, we see a burst of communication cost around that range in Fig. 5.14. When $k > 100$, the number of outliers increases steadily, which incurs an almost constant cost shown in Fig. 5.14. On the other hand, in Dataset2, the number of outliers grows in a steady and linear fashion. It yields a very low communication cost and a flat curve in Fig. 5.14. In both data sets, we observe, the enhanced scheme significantly improves the performance.

5.4.2.2 $O(n,k)$ Outlier Detection

In this simulation, we set $k = 100$, and vary n from 5 to 1000 for Dataset1, from 5 to 100 for Dataset2. Fig. 5.15 shows the values for $D^k(p_n)$ and the performance is presented in Fig. 5.16.

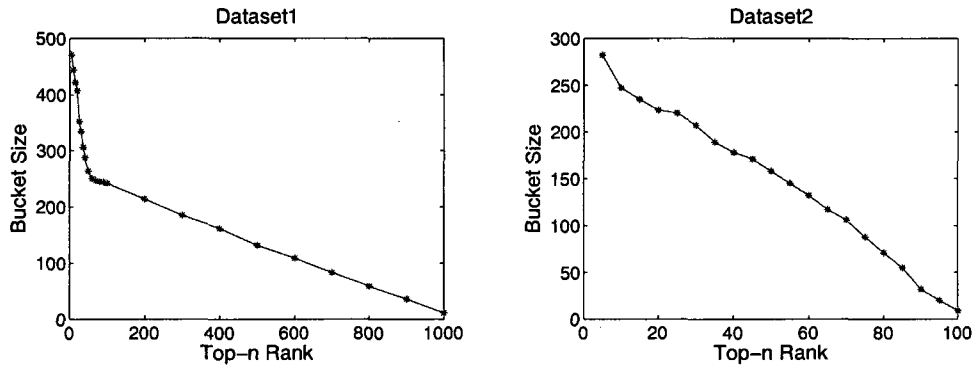


Figure 5.15: Values of $D^k(p_n)$ for various n ($k = 100$)

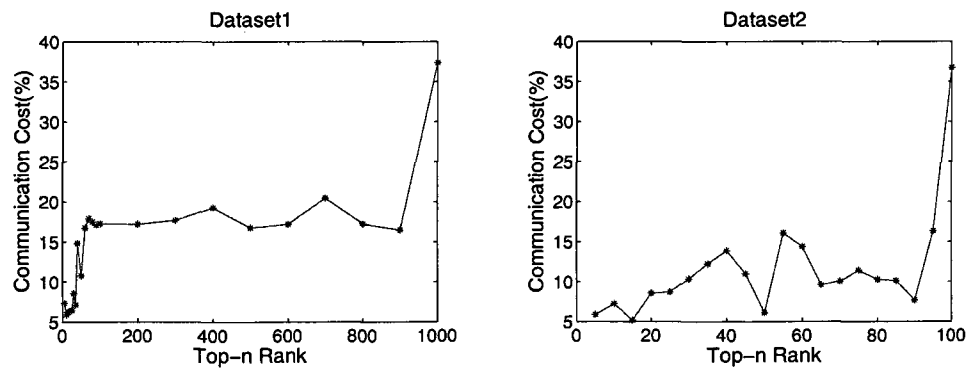


Figure 5.16: Communication costs of $O(n,k)$ outlier detection (normalized against the centralized scheme). We set k to 100 and vary n from 5 to 1000 for Dataset1, from 5 to 100 for Dataset2.

Using an analysis similar to the real data trace, we find that when the curve's tangent is sharp in Fig. 5.15, the communication cost remains low. For Dataset1, the initial phase in Fig. 5.15 is steep and the communication cost is very low. But after the point of $n = 60$, the curve becomes less steep, which leads to an increase of communication cost in Fig. 5.16. In Dataset2, the value curve of $D^k(p_n)$ drops constantly and faster than that in Dataset1. It results in a much lower communication cost in Fig. 5.16. Both data sets have a huge cost increase at the end of the curve. The reason is that, when n is large, we have to count the densely packed data points, whose $D^k(p)$ values are very close to each other.

5.5 Summary

Outlier data represents a complex form of abnormal data which is extremely important in many applications. Outlier detection has been well studied in the database community, but the existing solutions cannot be applied in sensor networks unless all the data from every sensor is gathered at a central place which is a costly process in practice. Our solutions utilize small-sized histogram information to analyze the data set and filter out undesired data without collecting them. According to our simulation, the proposed solutions accurately find all outliers with low energy consumption. From this work, we have demonstrated an appropriate way to handle complicated queries in a sensor network. Compared to collecting all the raw data, it is more efficient to collect a small amount of rough information first and analyze it to reduce further energy consumption. This process can be repeated for multiple rounds if applicable. This method, however, requires some information processing and data relaying on each device, thus only works with sensors, but not RFID tags. In the next chapter, we present our work on a data mining query in RFID systems which is more challenging because of the hardware limitation.

Chapter 6

Data Mining Query in RFID Systems: Finding Popular Categories

Similar to sensors, RFID tags also contain useful data information and data mining queries are desirable in RFID systems too. In the previous chapter, our solution to the data mining query in sensor networks relies on each sensor's ability of data processing and transmission. RFID tags, however, represent a category of even weaker pervasive computing devices that do not hold the same ability as sensors. They can hardly process data and there is no communication between RFID tags. Therefore, data mining queries with RFID tags are more challenging and need a completely different design from the previous chapter to achieve efficiency.

In the literature, there is very little work on complicated queries in RFID systems. Most of the prior work collects all tag IDs before replying to any query. This universal solution, however, may not be efficient for a particular query. In this chapter, we investigate a typical data mining query of finding popular items and develop efficient algorithms without collecting all IDs.

6.1 Related Work

For a reader to successfully receive data from multiple tags, anti-collision protocols must be designed so that replied data from multiple tags will not be garbled because of collision. In general, two approaches are used to regulate collision. The first is based on the ALOHA protocol [24, 28, 45, 48, 66, 89, 97, 110, 121, 124, 129, 133]. A representative protocol used in RFID systems is the framed ALOHA [97], a variation of ALOHA [4]. In this protocol, a frame is divided into multiple time slots. The communication is initialized when the reader broadcasts a frame size, i.e., the number of slots in the frame. Every RFID tag responds only in a particular slot in the current frame. The reader can successfully receive data in a certain slot if only one tag picks the slot for transmission. This process is repeated until all tag data are collected. The second approach uses the tree traversal technique [33, 36, 69, 87, 98, 100, 101, 134]. The reader broadcasts an ID prefix, and those tags whose IDs match the prefix will respond. If a collision is detected, the reader will append '0' or '1' to the prefix and send new prefixes again. It is equivalent to traversing a binary tree, where each tag's ID is a leaf node. The expansion of prefix stops if only one tag responds. The goal of the above anti-collision protocols is to collect all the IDs, which can definitely solve our problem of finding popular categories. However, as we will show in evaluation, they are not efficient. Interestingly, we do use the framed ALOHA and a tree-traversal-like method in this work, but with a totally different purpose.

In the database community, mining RFID data has drawn considerable attention [57,58,74,91]. Their problems are formulated at a high level, where all RFID data are already stored in a central database. Our work considers the problem where none of the RFID data has been collected.

The research work in [83] is the closest to this work. The authors consider the problem of estimating the number of tags without collecting the tag IDs. Based on the framed ALOHA, their

algorithms analyze the numbers of empty slots, single-reply slots and collision slots to obtain approximated information. By carefully tuning the parameters for multiple iterations, their solutions can quickly estimate the number of RFID tags with high accuracy. [82] uses a similar analysis for anonymous tracking in RFID systems. In this work, our *TCS* scheme is based on a similar analysis. However, we consider a more complex problem of finding popular categories. Directly applying the algorithms in [83] cannot efficiently resolve it.

Another relevant research is finding popular items in streaming data [37,39,54,55,77]. Similar ideas of group testing [42] are adopted in [37] to maintain a small set of counters to find frequent items in data streams, thus achieving memory efficiency. In this work, our goal is to reduce the scanning time and the assumption of scanning all the data in one pass in the data streaming algorithms is impossible.

6.2 Problem Formulation

We consider that, within the reading range of a reader, there are n products each of which is attached with an RFID tag, that is, n tags $(t_1, \ldots, t_n)$ in total. Every RFID tag contains a unique ID represented by a bit string, which consists of several fields [45]. We assume that one of the fields specifies the category the product belongs to. The bit string in the field is called *category ID*. Depending on the applications, a category ID can be as generic as the origin of country, or as specific as a brand and model number. We assume that we know the set of distinct category IDs of the tags considered in this scenario, denoted as $C = \{C_1, \ldots, C_m\}$. For each tag t_j, we use c_j to represent its category ID. We will also discuss the scenario without knowing C in an extension of our scheme.

In this work, popular categories are defined by an application specific threshold. Let F_i be the

number of products in category C_i.

Definition 5 *Given a threshold $\alpha \in (0,1)$, C_i is a popular category if $F_i \geq \alpha \cdot n$.*

Our goal is to find a category set R, which contains popular categories of products in the warehouse. To this end, we are going to design randomized algorithms. This requires us to slightly modify the problem in the randomized setting as follows. Given α, $\beta \leq \alpha$, and $\delta \in (0,1)$, we would like to **minimize the scanning time** and find a category set R such that with probability larger than $1-\delta$, the following two accuracy constraints are satisfied:

1. **Completeness Constraint**: $\{C_i | F_i \geq \alpha \cdot n\} \subseteq R$;

2. **Population Constraint**: $\forall C_i \in R, F_i \geq \beta \cdot n$.

We name the first constraint *completeness constraint*, since it requires returning all popular categories. The second constraint is called *population constraint*, as it defines the lower bound of the population of any returned category.

Here we briefly explain the rationale of this problem formulation. Ideally, we would like to return all popular categories, i.e., $\{C_i | F_i \geq \alpha \cdot n\} \subseteq R$, and only the popular categories. However, our randomized setting may return some unpopular categories. To control what extraneous categories may be returned, we introduce another parameter $\beta \leq \alpha$, which defines a lower bound for the population of any returned category. It requires that any $C_i \in R$ must have no fewer than $\beta \cdot n$ products, i.e., $\forall C_i \in R, F_i \geq \beta \cdot n$. A strict requirement may set $\beta = \alpha$. In practice, however, applications usually tolerate a certain level of inaccuracy. For example, it is meaningful to return a category with fewer than $\alpha \cdot n$ products as a popular category. With the requirement of β, the population of each resulting category, although maybe less than $\alpha \cdot n$, is confined to be close to $\alpha \cdot n$. Furthermore, to save scanning time, the number of products in each category is estimated by a

probabilistic algorithm. Thus, we can not provide deterministic guarantee for the two constraints. Instead, another parameter $\delta \in (0,1)$ is defined as a probabilistic guarantee which specifies the maximum allowed probability that our returned results fail to satisfy the two constraints.

In the rest of this chapter, our schemes often use a 'select' operation: the tags satisfying a certain condition will stay active while the others will keep silent. In a 'select' command, two types of conditions can be specified. First, the reader can broadcast a prefix bit string *mask* and each tag t_j will check if its category ID matches the received prefix, i.e., if the first $|mask|$ bits of c_j is the same as *mask*, where $|mask|$ is the bit length of *mask*. Second, the reader can broadcast three numbers, r, u, and v, and each tag t_j will check the following condition, $h(r, c_j) \bmod u = v$, where h is a hash function. We use $h_u(r,x)$ to indicate $h(r,x) \bmod u$ in the rest of this chapter. In both cases, an RFID tag will keep active only when the specified condition holds.

Our communication model is based on the framed ALOHA. We assume that an RFID reader is able to distinguish the slots with no reply, single reply, or multiple replies. We define these slots as empty slot, single-reply slot, or collision slot respectively. In the typical ALOHA scheme, the duration of a non-empty slot (single-reply or collision) is much longer than that of an empty slot, because tags transfer the whole ID with CRC (Cyclic Redundancy Check) in a non-empty slot. In our approaches, every tag does not transfer the long ID, but a short random bit string (usually < 10 bits [83]), as long as the RFID reader can detect the presence of the signal. Thus, all slots in our approaches have similar durations. In the rest of this chapter, we call an empty slot or a slot transferring short bit strings as *short slot*, and a slot transferring IDs as *long slot*. We use S and L to denote the lengths of a short slot and long slot respectively. In addition, our schemes use the algorithm presented in [83] to estimate the total number of active tags. For total n' active tags, the algorithm, denoted as $\Omega(a,b)$ for $a,b \in (0,1)$, gives an estimation of $\tilde{n}'$ for n', such that

with probability larger than a, $1-\frac{b}{2} \leq \frac{\tilde{n}'}{n'} \leq 1+\frac{b}{2}$. Let $|\Omega|$ be the scanning time of Ω. As claimed in [83], $|\Omega|$ is independent of n. Table 6.1 lists some notations used in the following sections.

$n/\tilde{n}$	number of tags / estimation of n
$n'/\tilde{n}'$	number of active tags/ estimation of n'
C_i/F_i	category ID / number of products in C_i
t_j/c_j	RFID tag / t_j's category ID

Table 6.1: Summary of Notations

6.3 Methodology

We propose and compare different solutions in this section. First, we describe three straightforward, but impractical solutions. Then, we introduce the Threshold Checking Scheme (*TCS*), which is an important component in our solutions. Finally, we propose our schemes, group testing with *TCS* and tree traversal with *TCS*.

6.3.1 Simple Solutions

The first simple solution is to collect all tag IDs by using the framed ALOHA. Then, we can scan the data and find all popular categories. We call this solution *identification scheme*. In this solution, we have to use long slots to correctly receive the IDs. As analyzed in the prior work [28, 48, 110], the number of slots needed is proportional to the number of tags n. It is inefficient when n is very large.

Alternatively, we can use Ω to resolve the problem. The algorithm is described in Algorithm 15. For each category, the reader broadcasts the category ID so that the tags in the category

stay active while the other tags keep silent. Then, we apply Ω to estimate the number of active tags and compare the result with the threshold. Since Ω can obtain a good estimation with a

Algorithm 15 Check Each Category

1: Run Ω to obtain $\tilde{n}$

2: **for** $i = 1$ to m **do**

3: Reader broadcasts C_i

4: Tag t_j stays active if $c_j = C_i$

5: Run Ω to obtain $\tilde{n}'$

6: **if** $\tilde{n}' \geq \alpha \cdot \tilde{n}$ **then** $R = R \cup \{C_i\}$

7: return R

certain setting, Algorithm 15 is able to find all popular categories with a very high probability and the scanning time is $m(L + |\Omega|)$. In practice, this solution is not efficient either, because we may have hundreds of categories (large m) and $|\Omega|$ could be thousands of short slots for a certain accuracy [83].

Another alternative solution is the sampling scheme. We can randomly select a set of sample RFID tags, and collect all IDs from them. Then we can easily determine the popular categories in this small set. Assuming these samples effectively reflect the whole set of RFID tags, the popular categories found in samples can be returned as the results. The details are presented in Algorithm 16. The performance of this scheme heavily depends on the sample size and has a tradeoff with the accuracy. We will evaluate this scheme in Section 6.4.

The probability that this sampling scheme can identify a popular category is

$$1 - \sum_{k=0}^{\alpha n'-1} \frac{C_{n'}^{k} \cdot F_i^k \cdot (n - F_i)^{n'-k}}{n^{n'}}$$

Algorithm 16 Sampling Scheme

1: Randomly select a set S' of RFID tags from n tags

2: $n' = |S'|$

3: **for** $i = 1$ to m **do**

4: $\quad F'_i = |\{ t_j \mid t_j \in S' \text{ and } c_j = C_i \}|$

5: $\quad$ **if** $F'_i \geq \alpha \cdot n'$ **then** $R = R \cup \{C_i\}$

6: **end for**

7: return R

where $\frac{C^k_{n'} \cdot F_i^k \cdot (n-F_i)^{n'-k}}{n^{n'}}$ is the probability that k tags from a category C_i are sampled. We will show in the evaluation that this probability of identifying a popular category becomes low when F_i is close to $\alpha \cdot n$ and the sample rate (n'/n) is low.

6.3.2 Threshold Checking Scheme (TCS)

Our algorithms are based on a scheme that estimates whether the number of currently active tags (n') exceeds a given threshold. We call this scheme Threshold Checking Scheme (TCS). The details are presented in Algorithm 17. The input includes a frame size f and other two parameters $\tau_1, \tau_2 \leq f$. The reader first broadcasts the frame size f. RFID tags follow the basic framed ALOHA protocol and respond at a random time slot. During this frame, the reader keeps counting the numbers of empty slots and collision slots, recorded in N_0 and N_c respectively. In the end, the reader will compare N_0 and N_c with τ_1 and τ_2 to determine the returned value of TCS. We intentionally avoid using the number of slots for single tag reply (N_1) because N_1 is not a monotonous function of the number of tags. N_0 and N_c, however, are monotonous decreasing and increasing

Algorithm 17 $TCS(f, \tau_1, \tau_2)$

1: Reader broadcasts f

2: Each tag randomly picks a time slot to reply

3: Reader obtains N_0 and N_c

4: **if** $(N_0 \leq \tau_1)$ and $(N_c > \tau_2)$ **then** return true

5: **else** return false

functions of the number of tags respectively. This gives us a simple way to check if n' is greater than the given threshold. We omit the detailed analysis here and refer the interested reader to [83].

By carefully choosing f, τ_1, and τ_2, we can have a high confidence that if the number of active tags exceeds a given threshold the protocol returns true. In the following lemmas and theorems, we give the analysis for the protocol assuming there are n' active RFID tags. More specifically, we show the results on $Suc(n')$, which is defined as the probability that $TCS(f, \tau_1, \tau_2)$ returns true when applied to n' active tags. These lemmas and theorems are crucial for the analysis of our algorithms which will be presented later.

Lemma 6.1 *When n' and f are large*[1]*, N_0 and N_c approximately follow a normal distribution , $N_0 \sim N(\mu_0, \sigma_0)$, and $N_c \sim N(\mu_c, \sigma_c)$, where μ_0, σ_0, μ_c and σ_c are defined as follows.*

$$
\begin{aligned}
\mu_0(n', f) &= f \cdot e^{-\frac{n'}{f}}; \\
\sigma_0^2(n', f) &= f \cdot e^{-\frac{n'}{f}}(1 - (1 + \frac{n'}{f})e^{-\frac{n'}{f}}); \\
\mu_c(n', f) &= f(1 - (1 + \frac{n'}{f})e^{-\frac{n'}{f}}); \\
\sigma_c^2(n', f) &= f \cdot e^{-\frac{n'}{f}}((1 + \frac{n'}{f}) - (1 + \frac{2n'}{f} + (\frac{n'}{f})^2 + (\frac{n'}{f})^3)e^{-\frac{n'}{f}}).
\end{aligned}
$$

[1]We consider general rules of thumb for approximating a binomial distribution to a normal distribution.

Proof: Refer to [83]. ■

Theorem 6.1 *When n' and f are large,*

$$Suc(n') = \frac{1}{4}(1 + erf(\frac{\tau_1 - \mu_0}{\sqrt{2}\sigma_0})) \cdot (1 - erf(\frac{\tau_2 - \mu_c}{\sqrt{2}\sigma_c})),$$

where erf is the error function of the standard normal distribution[2]*, and variables μ_0, σ_0, μ_c and σ_c are defined in lemma 6.1.*

Proof: Based on the properties of normal distributions,

$$\begin{aligned} Pr(N_0 \leq \tau_1) &= \Phi(\frac{\tau_1 - \mu_0}{\sigma_0}) = \frac{1}{2}(1 + erf(\frac{\tau_1 - \mu_0}{\sqrt{2}\sigma_0})); \\ Pr(N_c > \tau_2) &= 1 - \Phi(\frac{\tau_2 - \mu_c}{\sigma_c}) = \frac{1}{2}(1 - erf(\frac{\tau_2 - \mu_c}{\sqrt{2}\sigma_c})). \end{aligned}$$

Therefore,

$$Suc(n') = Pr(N_0 \leq \tau_1) \cdot Pr(N_c \geq \tau_2) = \frac{1}{4}(1 + erf(\frac{\tau_1 - \mu_0}{\sqrt{2}\sigma_0})) \cdot (1 - erf(\frac{\tau_2 - \mu_c}{\sqrt{2}\sigma_c})).$$

■

Theorem 6.2 *$Suc(n')$ is an increasing function of n', i.e., if $n'_1 \geq n'_2$, $Suc(n'_1) \geq Suc(n'_2)$.*

Proof: Obviously, compared with a group with n'_2 tags, a group with n'_1 tags tends to have less empty slots and more collision slots. ■

Theorem 6.3 *Given a list $\{u_1, \ldots, u_q\}$ and a number $v > 0$, if $\sum u_i = z$, then*

$$\sum Suc(u_i) \leq \frac{z}{v} + (q - \frac{z}{v}) Suc(v).$$

[2]In our implementations, continuity correction is applied.

Proof: We divide the list into two sets, $S_1 = \{i|u_i \geq v\}$ and $S_2 = \{i|u_i < v\}$. Obviously, at most $\frac{z}{v}$ elements belong to S_1. Therefore,

$$\begin{aligned}
\sum Suc(u_i) &= \sum_{i \in S_1} Suc(u_i) + \sum_{i \in S_2} Suc(u_i) \\
&\leq |S_1| \cdot 1 + (q - |S_1|) \cdot Suc(v) \\
&= |S_1| \cdot (1 - Suc(v)) + q \cdot Suc(v) \\
&\leq \frac{z}{v} + (q - \frac{z}{v}) Suc(v).
\end{aligned}$$

■

6.3.3 Group Testing with *TCS*

In this section, we propose a solution based on group testing with *TCS*. We first divide the tags into groups according to their category IDs. The tags with the same category ID belong to the same group and each group may contain the tags in multiple categories. We then apply *TCS* to check the number of tags in each group. The intuition is that many categories with few tags may be grouped together and thus can be easily identified as unpopular categories in a simple group test. The groups with sufficient tags are labeled as potential popular groups, which may include popular categories or have no popular categories (when a certain number of unpopular categories contribute adequate number of tags). Our algorithm continues to shuffle all categories into different groups and apply the *TCS* tests to the new groups again. This process is repeated for a prescribed number of rounds and in the end, the testing history is able to reveal all popular categories.

The details of our protocol are illustrated in Algorithm 18. The whole process consists of T rounds (line 3) and in each round all tags are distributed into W groups by a hash function $h(r, C)$,

Algorithm 18 Group Testing

```
Run Ω to obtain ñ
Calculate parameters T, W, f, τ1, and τ2
for k = 1 to T do
    for g = 0 to W − 1 do
        Reader broadcasts a random seed r_k, W, and g
        Tag t_j stays active if h_W(r_k, c_j) = g
        M[k, g] = TCS(f, τ1, τ2)
    end for
end for
for C_i ∈ C do
    check=true
    for k = 1 to T do
        if (not M[k, h_W(r_k, C_i)]) then
            check=false
        end if
    end for
    if check then
        R = R ∪ {C_i}
    end if
end for
return R
```

where r is a random seed and C is a category ID. A tag t_j is in group g if $h_W(r, c_j) = g$ (recall $h_W(r, c_j)$ denotes $h(r, c_j) \bmod W$). We use a different random seed to shuffle the categories in each round. Thus, Algorithm 18 totally generates T random seeds, denoted by $\{r_1, r_2, \ldots, r_T\}$. Throughout the algorithm, all tags form $T \times W$ groups, labeled as $G(k, g)$ for $k \in [1, T]$ and $g \in [0, W-1]$, such that

$$G(k, g) = \{C_i | \; h_W(r_k, C_i) = g\}.$$

In the rest of this chapter, we use $|G(k, g)|$ to denote the number of the tags whose category IDs belong to $G(k, g)$. In round k, the reader broadcasts r_k, W, and g (line 5) to select the RFID tags mapping to group $G(k, g)$. We then run $TCS(f, \tau_1, \tau_2)$ to examine the number of RFID tags in $G(k, g)$. We record the results in a matrix M: $M[k, g] = true$ means that there might be popular categories in group $G(k, g)$. Otherwise, if $M[k, g] = false$, all the categories in $G(k, g)$ are considered as unpopular categories. Thus, as shown in lines 8-15, a category will be returned, only if the group it belongs to in every round passes the test. Fig. 6.1 illustrates an example of group testing with 10 categories.

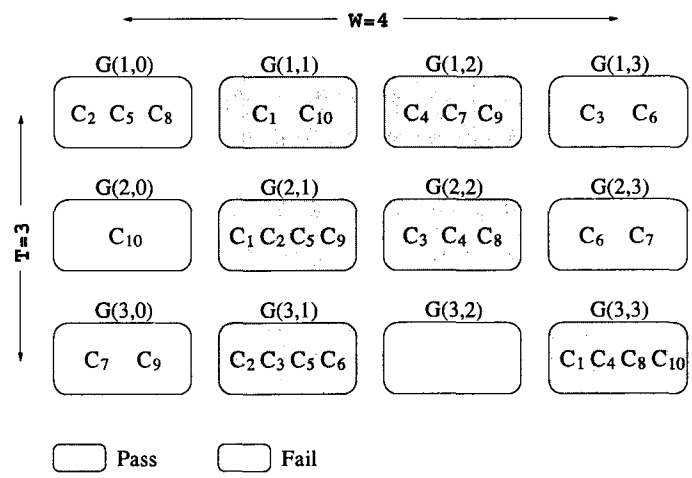


Figure 6.1: There are 10 category IDs, with parameters $W = 4$ and $T = 3$. Based on the test results, C_1 and C_4 will be returned as popular categories.

In the following, we show how to choose these parameters to minimize the scanning time

while the constraints are satisfied. Theorem 6.4 and Theorem 6.5 give the conditions that provide the probabilistic guarantee for the completeness constraint and population constraint (stated in Section 3) respectively. Theorem 6.6 expresses the scanning time by the parameters. Combining them, we can find the optimal parameters with the minimum scanning time while satisfying the two constraints.

Specify the constraints: Since *TCS* is probabilistic and group testing is essentially a randomized algorithm, a popular category may be filtered out of the resulting set and an unpopular category may survive all tests and be present in R. The following two theorems specify the conditions for the parameters to satisfy the accuracy constraints.

Theorem 6.4 *The completeness constraint is satisfied with more than* $1-\delta$ *probability if* $(1-\delta\cdot\alpha) \leq Suc(\alpha\cdot n)^T$.

Proof: Consider a popular category C_i, assume C_i belongs to $G(k,g)$. Let $t = |G(k,g)| \geq F_i \geq \alpha\cdot n$. $G(k,g)$ will pass the *TCS* test with probability of $Pr(M[k,g] = true) = Suc(t)$. According to Theorem 5.2,

$$Pr(M[k,g] = true) \geq Suc(\alpha\cdot n).$$

The probability that any of the T groups that C_i belongs to will fail in the *TCS* test is at most $1-Suc(\alpha\cdot n)^T \leq \delta\cdot\alpha$. Based on the definition of a popular category, there are at most $\frac{1}{\alpha}$ popular categories. Thus, by union bound, the probability that no popular category is missing (all popular categories pass all the T tests) is greater than $1-\delta\cdot\alpha\cdot\frac{1}{\alpha} = 1-\delta$. ■

Theorem 6.5 *The population constraint is satisfied with more than* $1-\delta$ *probability if there exists* u, *such that*

$$(\frac{n-\beta\cdot n}{W(u-\beta\cdot n)}(1-Suc(u))+Suc(u))^T \leq \delta.$$

Proof: We prove the theorem by showing that for any unpopular category C_i, i.e., $F_i < \beta \cdot n$, the probability to be returned in R is less than δ. Assume in a certain round, C_i belongs to a group G and let $t = |G|$. The probability that group G passes a TCS test is $Suc(t)$. For any given u,

$$\begin{aligned} Suc(t) &= Pr(t \geq u)Suc(t) + Pr(t < u)Suc(t) \\ &\leq Pr(t \geq u) + (1 - Pr(t \geq u))Suc(u). \end{aligned}$$

Let X denote the number of tags in group G which do not belong to category C_i, i.e., $X = t - F_i$. The expectation of X is $E(X) = \frac{n-F_i}{W}$. According to Markov's inequality,

$$\begin{aligned} Pr(t \geq u) &= Pr(X \geq u - F_i) \leq \frac{n - F_i}{W(u - F_i)} \\ &= \frac{1}{W}(1 + \frac{n-u}{u-F_i}) < \frac{1}{W}(1 + \frac{n-u}{u - \beta \cdot n}). \end{aligned}$$

Therefore,

$$\begin{aligned} Suc(t) &\leq Pr(t \geq u)(1 - Suc(u)) + Suc(u) \\ &< \frac{n - \beta \cdot n}{W(u - \beta \cdot n)}(1 - Suc(u)) + Suc(u). \end{aligned}$$

Considering T rounds of tests, C_i will be returned in R with probability of $Suc(t)^T < \delta$. ■

Express the scanning time: Here we express the scanning time used in Algorithm 18. In a simple estimation, we need test $T \cdot W$ groups and each test consumes one long slot and f short slots. Thus, in total, Algorithm 18 takes $T \cdot W \cdot (L + f \cdot S)$. We find, however, that it is not necessary to check all groups. In every round, we recognize some unpopular categories, thus the remaining

possible popular categories become fewer and fewer. If one group contains only known unpopular categories, we can skip the *TCS* test for it. We analyze the scanning time in the following series of theorems and lemmas. Theorem 6.6 bounds the expected scanning time utilizing the result from Lemma 6.3. Lemma 6.2 is an auxiliary lemma that helps prove Lemma 6.3.

Lemma 6.2 *Given $a \in (0,1)$, $x < b < n$, and $c \geq 1$, $(a+(1-a)\frac{n-x}{W\cdot(b-x)})^c$ is a convex function of x.*

Proof: Let $g = (a+(1-a)\frac{n-x}{W\cdot(b-x)})^c$. The lemma is proved if the second derivative of g is positive. Let $h = \frac{n-x}{W\cdot(b-x)} > 0$. We have

$$h' = \frac{n-b}{W\cdot(b-x)^2} > 0, h'' = \frac{2(n-b)}{W\cdot(b-x)^3} > 0.$$

The first derivative of g is $g' = c\cdot(1-a)\cdot(a+(1-a)h)^{c-1}h'$, and the second derivative is

$$g'' = c\cdot(1-a)\cdot(((c-1)(1-a)(a+(1-a)h)^{c-2}h')\cdot h' + (a+(1-a)h)^{c-1}h'') > 0.$$

■

Lemma 6.3 *Let m_k be the expected number of possible popular categories after the k-th iteration in line 3 of Algorithm 18 and $m_0 = m$. Given $u > 0$ and $v \leq \frac{W\cdot u-n}{W-1}$, then $\forall k \in [1,T]$, m_k is bounded by*

$$\frac{n}{v} + (m - \frac{n}{v})(Suc(u) + (1-Suc(u))\frac{n-1}{W(u-1)})^k.$$

Proof: For a category C_i, let $p_{i,k}$ be the probability that C_i will still be considered as a possible popular category after the k-th iterations, $m_k = \sum_i p_{i,k}$. Similar to Theorem 6.5, for any given u,

$$p_{i,k} \leq (Suc(u) + (1-Suc(u))\frac{n-F_i}{W(u-F_i)})^k.$$

We divide all categories into two sets, $S_1 = \{C_i | F_i > v\}$ and $S_2 = \{C_i | F_i \leq v\}$. We have,

$$\begin{aligned}\sum p_{i,k} &= \sum_{C_i \in S_1} p_{i,k} + \sum_{C_i \in S_2} p_{i,k} \\ &\leq |S_1| + \sum_{C_i \in S_2} (Suc(u) + (1 - Suc(u)) \frac{n - F_i}{W(u - F_i)})^k.\end{aligned}$$

According to Lemma 6.2, the right side of the above inequality is a convex function of F_i. To maximize the right hand side, for each category $C_i \in S_2$, F_i takes value of either 1 or v, by the property of a convex function. Suppose $t_1 = |\{C_i | F_i = v\}|$ and $t_2 = |\{C_i | F_i = 1\}|$ when the maximization is achieved. Therefore, $\sum p_{i,k}$ is bounded by

$$\begin{aligned}&|S_1| + t_1 \cdot (Suc(u) + (1 - Suc(u)) \frac{n - v}{W(u - v)})^k + t_2 \cdot (Suc(u) + (1 - Suc(u)) \frac{n - 1}{W(u - 1)})^k \\ &= |S_1| + t_1 + t_2 \cdot (Suc(u) + (1 - Suc(u)) \frac{n - 1}{W(u - 1)})^k.\end{aligned}$$

Let $\lambda = (Suc(u) + (1 - Suc(u)) \frac{n-1}{W(u-1)})^k \leq 1$, we have

$$\begin{aligned}\sum p_{i,k} &\leq |S_1| + t_1 + t_2 \cdot \lambda \\ &= |S_1| + t_1 + (m - |S_1| + t_1) \cdot \lambda \\ &= m \cdot \lambda + (|S_1| + t_1) \cdot (1 - \lambda).\end{aligned}$$

Since the right side of the above inequality is an increasing function of $|S_1| + t_1$ (the number of categories with no less than v tags) and $|S_1| + t_1$ is at most $\frac{n}{v}$, we have

$$\sum p_{i,k} \leq \frac{n}{v} + (m - \frac{n}{v})(Suc(u) + (1 - Suc(u)) \frac{n - 1}{W(u - 1)})^k.$$

■

Theorem 6.6 *The expected scanning time is bounded by*

$$ST = (L + f \cdot S) \cdot W \cdot \sum_{k=1}^{T} (1 - (1 - \frac{1}{W})^{m_{k-1}}), \tag{6.1}$$

where m_{k-1} is expressed by the bound derived in Lemma 6.3, replacing k with $k-1$.

Proof: Let X_k be the number of groups we need check in the k-th iteration. For a certain group, the probability that all the tags in it belong to known unpopular categories is $(1 - \frac{1}{W})^{m_{k-1}}$. Thus, the expected value of X_k is $E(X_k) = W(1 - (1 - \frac{1}{W})^{m_{k-1}})$. Obviously, it is an increasing function of m_{k-1}. Thus, ST bounds the expected scanning time when we express it with the upper bound of m_{k-1}. ■

Solve the optimization problem: In summary, given α, β, δ, n and m, our problem is to determine the values of T, W, f, τ_1 and τ_2 in the following optimization problem.

$$\begin{aligned} & \text{minimize } ST \text{ (Eq.(6.1))} \\ s.t. \quad & (1 - \delta \cdot \alpha) \leq Suc(\alpha \cdot n)^T; \\ & \exists u, (\frac{n - \beta \cdot n}{W(u - \beta \cdot n)}(1 - Suc(u)) + Suc(u))^T \leq \delta. \end{aligned}$$

Since all these parameters are bounded integers, we can find the optimal set of parameters by discretizing them and enumerating all possible values. The process basically includes five loops to enumerate all possible discrete values for the five parameters. We also apply some optimization strategy to speed up the process.

6.3.4 Tree Traversal

Group testing can be applied differently. In this sub-section, we combine group testing with divide-and-conquer. We first divide all tags into W groups based on their category IDs and run

TCS for each group, which is the same as the first round in the previous solution. However, in this scheme, we do not shuffle all categories into groups in each of the remaining rounds. We ignore those groups that fail to pass the *TCS* tests and suppose there are no popular categories in them. Each of the groups which pass the test is further divided into W sub-groups and we apply *TCS* to each sub-group. This dividing process is repeated recursively until *TCS* test fails or there is only one category in the group, in which case that category will be returned as a popular category. Fig. 6.2 illustrates an example.

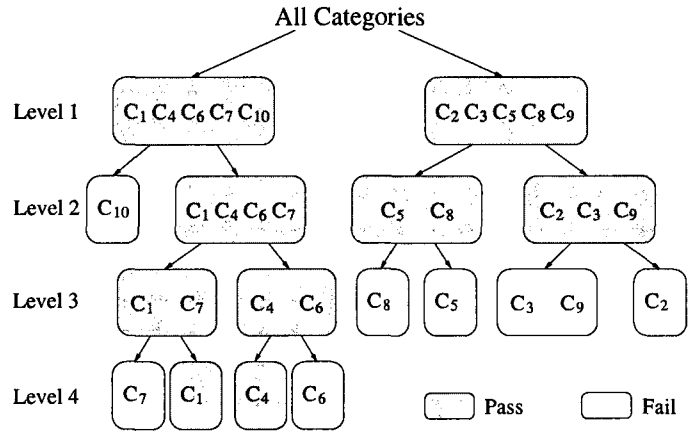


Figure 6.2: In this example, there are 10 categories with parameter $W = 2$. Based on the test results, C_1 and C_4 will be returned as popular categories.

Conceptually, this scheme is equivalent to a depth-first tree traversal on a W-ary tree, where each leaf is a category and each non-leaf node represents a group of categories that appear as leaves of the subtree rooted at it. Different from the previous scheme, this scheme uses multiple random seeds and group indices to define a group. For example, a node at level 1 (a direct child of the root) is defined by a pair composed of a random seed and a group index as in the previous scheme. However, to select a group represented by a level 2 node, we need first select the tags belonging to its parent node, and then divide them into W sub-groups by another random seed. Thus, we need two pairs of random seeds and group indices to define a level 2 node. Inductively,

for a node at level l, the group it represents is defined by l pairs of random seeds and group indices. Thus, we denote a node by a vector of random seeds $\{r_k\}$ and a vector of group indices $\{v_k\}$,

$$\text{Node } (\{r_k\}, \{v_k\}) = \{C_i | \ \forall k, h_W(r_k, C_i) = v_k\}.$$

Since passive RFID tags are memoryless devices, when visiting a node on the tree, the reader has to provide all random seeds and group indices to select the corresponding group. Algorithm 19 presents the details of traversing a node. The first call is to traverse the root (level 0), where both $\{r_k\}$ and $\{v_k\}$ are empty.

Algorithm 19 Traverse Node $(\{r_k\}, \{v_k\})$ at Level l

```
for k = 1 to l do
    Reader broadcasts W, v_k, and r_k
    Each tag t_j stays active if h_W(r_k, c_j) = v_k
end for
if TCS(f, τ_1, τ_2) = true then
    Reader generates a new random seed r
    for v = 0 to W − 1 do
        Traverse Node ({r_k} ∪ {r}, {v_k} ∪ {v}).
    end for
end if
```

Specify the constraints: Similar to the previous sub-section, the following Theorem 6.7 and Theorem 6.8 give the conditions that guarantee the completeness constraint and population constraint. Lemma 6.4 is needed by the proof of Theorem 6.7.

Lemma 6.4 *Consider a leaf node at level l. Given $u \geq 1$,*

$$Pr(l \leq u) = (1 - \frac{1}{W^u})^{m-1}.$$

Proof: For a certain category C_i, the probability that a different category falls in the same group at level l is $\frac{1}{W^l}$. The probability that none of the other $m-1$ categories share the same hashed values is $(1 - \frac{1}{W^l})^{m-1}$. ■

Theorem 6.7 *The completeness constraint is satisfied with more than $1 - \delta$ probability if there exists u, such that*

$$1 - (1 - \frac{1}{W^u})^{m-1} Suc(\alpha \cdot n)^u \leq \delta \cdot \alpha.$$

Proof: Assume a popular category is represented by a leaf node at level l. It must pass l *TCS* tests to be returned, which has a probability of at least $Suc(\alpha \cdot n)^l$. Given a parameter $u \geq 1$, the probability that a popular category will be returned is more than $Pr(l \leq u) \cdot Suc(\alpha \cdot n)^u$. Applying Lemma 6.4 and union bound, this theorem can guarantee the accuracy requirement. ■

Theorem 6.8 *The population constraint is satisfied with more than $1 - \delta$ probability if $Suc(\beta \cdot n) < \delta$.*

Proof: Any returned category in this scheme must pass the test as a leaf node, i.e., without tags in any other category in the same group. Therefore, $Suc(\beta \cdot n) < \delta$ guarantees that with more than $1 - \delta$ probability, an unpopular category will not pass the test by its own. ■

Express the scanning time: In this tree traversing process, when visiting a node at level l, we need l long slots to transmit the random seeds and group indices which define the node. Then we need f short slots for each *TCS* test. Theorem 6.9 bounds the expected scanning time.

Theorem 6.9 *Given u, the expected scanning time of the tree traversal scheme is bounded by*

$$ST = W \cdot \sum_{l=0}^{\log_W m-1} ((l+1) \cdot L + f \cdot S) \cdot (\frac{n}{u} + (W^l - \frac{n}{u})Suc(u)). \tag{6.2}$$

Proof: Assume a node i is at level $l+1$. Let N_i be the number of tags whose category IDs belong to the group represented by i. The probability that i is visited is less than $Suc(N_j)$, where j is i's parent at level l.

Let us consider a balanced W-ary tree, with W^l nodes at level l. The expected number of nodes visited at level $l+1$ is at most $W \cdot \sum_j Suc(N_j)$. According to Theorem 6.3

$$\sum Suc(N_j) \leq \frac{n}{u} + (W^l - \frac{n}{u})Suc(u).$$

Visiting node i requires $l+1$ long slots for the reader to broadcast random numbers and group indices and f short slots for the TCS test. Thus, considering all levels, the expected scanning time is bounded by ST. ■

Therefore, our goal is to find the optimal parameters to

$$\begin{aligned} & \text{minimize } ST \text{ (Eq.(6.2))} \\ s.t. \quad & \exists u, 1 - (1 - \frac{1}{W^u})^{m-1} Suc(\alpha \cdot n)^u \leq \delta \cdot \alpha; \\ & Suc(\beta \cdot n) < \delta. \end{aligned}$$

Similar to the previous scheme, all the involved parameters are integers and bounded. Thus, we are able to enumerate all possible values and find the optimal parameters.

6.3.5 Extension

6.3.5.1 Without Knowledge of C

All previous solutions are based on the assumption that the set of present category IDs is known. In fact, with minor modifications, our schemes are also suitable for the scenario where category IDs are unknown.

Obtain m: In our schemes, m is an important factor in setting other parameters. In this extension, our first step is to use Ω to estimate m. We can let the reader send a random seed r and a frame size f as usual and have each tag t_j respond at slot $h_f(r, c_j)$. In this way, all the tags in a group will reply at the same slot, acting as a single tag. Thus, we can count the number of empty slots and use Ω to estimate the number of distinct categories.

Group Testing: If we use group testing, the analysis of the scanning time will be different. Without the category ID information, we have to exam all $T \cdot W$ group. We can easily find the optimal parameter setting with this modified objective. For each group that passes a TCS test, we need use a simple query tree scheme to find the category IDs in the group. For each category ID, we check the other groups it belongs to. If all of them pass the tests, we return this category as a popular category.

Tree Traversal: We can also use the tree traversal scheme in this extension. Without the category ID information, however, we have to determine if the traversing process reaches leaf nodes. An effective way is to observe the number of empty sub-groups of a node. If all sub-groups but one are empty, then with more than $1 - \frac{1}{W}$ probability, the node is a leaf node. If this scenario has occurred for several times (k times) while we keep dividing the non-empty sub-group, then with probability more than $1 - \frac{1}{W^k}$, the node is a leaf node. With a heuristic value of k, we can confirm a leaf node with high probability in this means. After locating a leaf node, we can

easily obtain the category ID by using a prefix mask to query each bit. Assume the category ID is represented by B bits. We can locate it in B slots.

6.3.5.2 Continuous Monitoring

A unique advantage of group testing method is that it can be used for continuous online popular categories discovery. For example, in a shipping port monitoring system, goods may come through the monitoring gate in bulk and bursty fashion, or in a large warehouse, a reader cannot reach all the tags in stock. In both scenarios, finding the popular categories is different from the case that all tags are within the range of a reader, in which case the tag information can be retrieved any time. Group testing approach can conform to this dynamic environment so that the popular categories can be found by only estimating the number of tags that fall in each of the predetermined number of groups. Our algorithm can be slightly modified to suit this case.

6.4 Performance Evaluation

We evaluate the performance of our schemes via simulations. By default, we set $n = 10000$, $m = 100$, $\alpha = 0.1$, $\beta = 0.05$, and $\delta = 0.01$. In addition, $|\Omega(a,b)|$ is estimated as 2000 short slots for $a = 0.99$ and $b = 0.05\%$ according to [83], and we assume that the duration of a long slot is 5 times that of a short slot, i.e., $L = 5S$. For the rest of the evaluation, we denote group testing with TCS as GT, and tree traversal with TCS as TT. All results are the averaged results of 1000 independent trials.

6.4.1 Distribution Models for Data Sets

The performance of our schemes is heavily dependent on the product distribution in all categories. The following distribution models are considered in our evaluation.

- Uniform Distribution: In this distribution, we intentionally introduce some popular categories, and uniformly distribute the remaining tags to the other unpopular categories. We use $UD(k)$ to denote the uniform distribution with exactly k popular categories. For this distribution, each popular category is assigned $\alpha \cdot n$ tags, and other $m-k$ categories have $\frac{n-k \cdot \alpha \cdot n}{m-k}$ tags.

- Max/1 Distribution: We denote this distribution as $M1(X)$, where X is the maximum number of tags in one category. In this distribution, each category has either X tags or only 1 tag. Since the total number of tags is n, there are $\lfloor \frac{n-m}{X-1} \rfloor$ categories with X tags and $m - \lfloor \frac{n-m}{X-1} \rfloor$ categories with 1 tag.

- Zipf Distribution: We also consider the Zipf distribution, which is commonly found in the real world. This distribution, denoted as $ZD(n,Z)$, is specified by two parameters. The first parameter is the total number of tags and the second parameter Z defines the upper bound of the population for each category, i.e., the number of tags in each category ranges from 1 to Z. For each category, the probability of having $i \in [1,Z]$ tags is $\frac{c}{i^{\theta}}$, where c is the normalization constant and θ characterizes the distribution. In our data set $ZD(n,Z)$, we tune the value of θ such that the total number of tags is n.

6.4.2 Alternative Solutions

6.4.2.1 Simple Solutions

We begin with presenting the performance of the simple solutions mentioned in Section 6.3.1. For the first identification scheme, we conduct 1000 simulations with an initial frame size $f = 10000$. At the end of each frame, the new frame size is set to the number of the tags which have not been collected. With the default setting, the time consumed in our simulations is about 122k short slots on average and the deviation is less than 2k short slots. For the other simple scheme (Algorithm 15), the scanning time is estimated based on $|\Omega| = 2000$. Checking each category needs 2000 short slots to finish Ω. Thus, with the default setting, Algorithm 15 requires $100 \times 2000 = 200$k short slots. These two simple solutions are both very costly, as we will show later when comparing with our schemes.

6.4.2.2 Sampling Scheme

In the sampling scheme, we collect all IDs from every sample RFID tags. Thus, similar to the identification scheme, the scanning time of the sampling scheme is proportional to the sample size. For example, with our default setting of $n = 10000$, if the sample size is 10%, i.e., 1000 RFID tags, the scanning time will be roughly 10% of 122k short slots. Therefore, if the sample size is small, the sampling scheme can be very efficient.

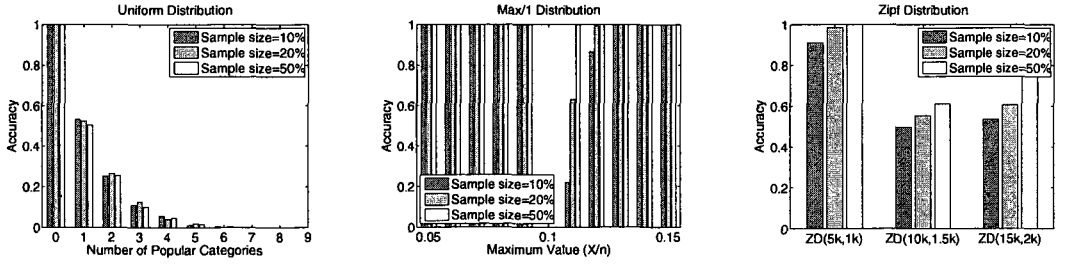


Figure 6.3: The accuracy of the sampling scheme with different product distribution. Sample size is set to $10\% \cdot n$, $20\% \cdot n$, and $50\% \cdot n$.

However, the major problem of the sampling scheme is the accuracy. When using a small sample size, the sampling scheme can hardly guarantee the accuracy. It may miss some popular categories with more than $\alpha \cdot n$ tags and report some categories with less than $\beta \cdot n$ tags. Fig. 6.3 illustrates the accuracy of the sampling scheme under different workloads. We conducted 1000 tests for each parameter setting, and the accuracy represents the percentage of correct results. As we can see, the sampling scheme is extremely sensitive to the product distribution and its accuracy dramatically vary with different parameters. Even with a sample size of 50% of all tags, the sampling scheme still cannot guarantee a high accuracy as our schemes. In practice, therefore, the sampling scheme may not be a feasible alternative to accurately finding popular items.

6.4.3 Scanning Time

6.4.3.1 Varying Number of Tags

We first evaluate our schemes by varying the number of RFID tags n. Fig. 6.4, Fig. 6.5 and Fig. 6.6 present the performance of GT and TT under the uniform, Max/1 and Zipf distributions respectively.

We observe that, when n increases, *TCS* tests in both GT and TT require larger frame sizes. This is because the number of tags involved in *TCS* test cases for GT and TT increases, i.e., each group in GT and each node in TT contain more tags. It is intuitive that, for *TCS* to achieve the same accuracy, a test case with more tags requires a larger frame size. If the frame size remains the same, the increased number of tags will overwhelm most slots in the frame with collisions engendering an inaccurate estimation.

Under the uniform distribution (Fig. 6.4), the average number of tags in one category (< 15) is far less than the threshold ($\alpha \cdot n = 500, 1000$ and 1500). Both schemes can efficiently identify

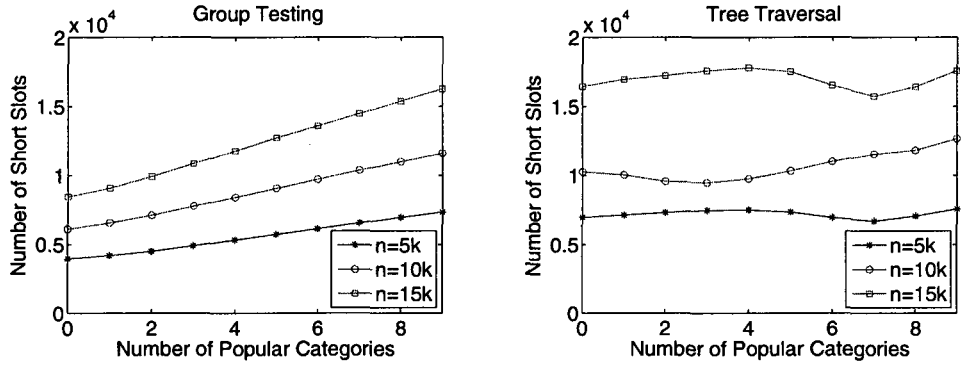


Figure 6.4: Scanning time for the uniform distribution with varying n

the groups with popular categories. In the GT scheme, the scanning time is approximately proportional to the number of popular categories. However, the scanning time of the TT scheme does not change much along axis x. In both schemes, a larger n yields more scanning time primarily because of the increase of the frame size in *TCS*.

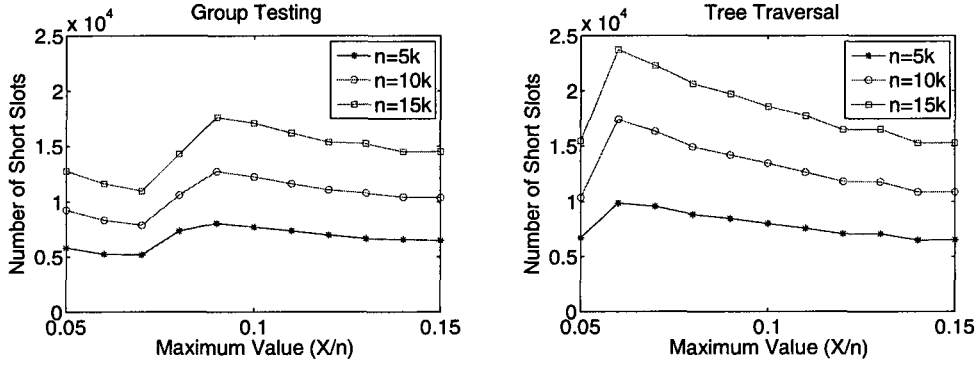


Figure 6.5: Scanning time for the $M1$ distribution with varying n

For the $M1(X)$ distribution (Fig. 6.5), we vary the maximum value X from $0.05n$ to $0.15n$. Let us call a category with X tags a *large category*, and a group containing at least 1 large category a *large group*. Basically, a large group has a higher probability to pass the *TCS* tests. The value of X has two impacts on the performance. On the one hand, the growth of X increases the probability that a large group can pass the *TCS* tests. The consequence is that we have to apply more *TCS* tests to eliminate the unpopular categories. On the other hand, when X increases, there are fewer large

categories and groups in the protocol, which helps filter out the unpopular categories quickly. In Fig. 6.5, both schemes are fast at the starting phase, because when X is small, all categories (even large categories) are unpopular and every group has a small probability to pass the TCS tests. Thus, both schemes quickly eliminate all categories and return no popular category. When X grows, the first impact becomes visible, and a sharp increase appears for both schemes, though the peak values are reached at different values of X. We also observe there is a slight decline for GT before the peak value because of the second impact. When X keeps increasing, the second impact becomes dominant and both schemes show a decreasing scanning time after the peak values. For a fixed value of X/n, the scanning time is nearly proportional to n.

Fig. 6.6 presents the performance under the Zipf distribution. In our data sets, there are usually one or two popular categories. Most categories are unpopular with the number of tags scattered between 1 and $\alpha \cdot n$. Since a considerable number of unpopular categories have tags close to the threshold, our schemes take more time to identify them as unpopular compared to the uniform distribution ($UD(1)$ or $UD(2)$), in which the sizes of the unpopular and popular categories diverge dramatically.

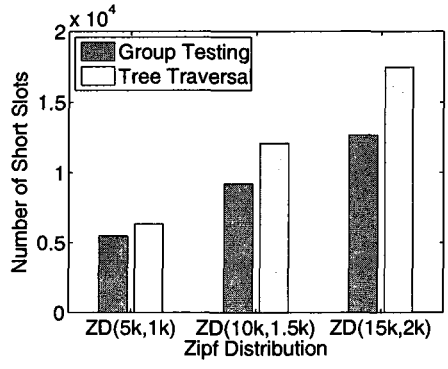


Figure 6.6: Scanning time for the Zipf distribution with varying n

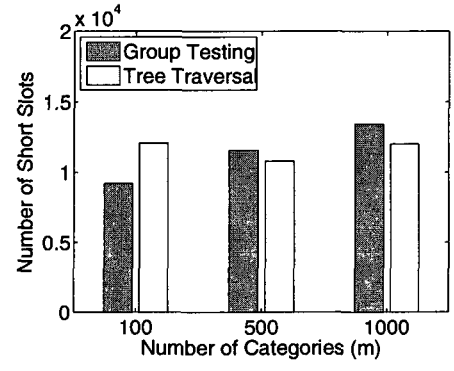


Figure 6.7: Scanning time for the Zipf distribution with varying m

6.4.3.2 Varying Number of Categories

We also evaluate the performance of the GT and TT schemes with a varying number of categories m. The results are illustrated in Fig. 6.8, Fig. 6.9 and Fig. 6.7.

In Fig. 6.8 and Fig. 6.9, we find that with other parameters fixed, the scanning time is increasing when m increases. However, the curves for $m = 500$ and $m = 1000$ are quite close. In Fig. 6.7, the performance of TT for varying m is almost the same, and the scanning time of GT is slightly increased when m increases.

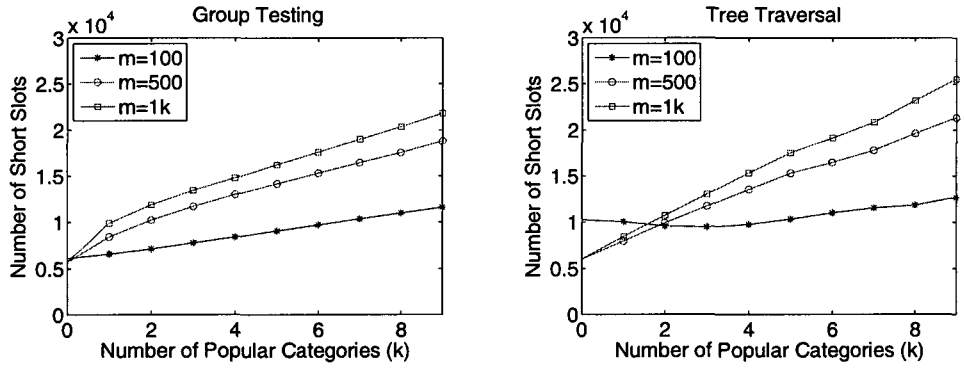


Figure 6.8: Scanning time for the uniform distribution with varying m

In all three distributions, the number of popular categories in each tested case is primarily determined by other parameters rather than m. Thus, with all other parameters fixed, the case with a larger m has almost the same number of popular categories and more unpopular categories which have to be filtered out. Thus, our schemes need run more *TCS* tests to identify these unpopular categories. However, unlike n, the impact of m is not proportional to the value of m.

6.4.3.3 Comparing to Simple Solutions

Both GT and TT are very efficient in finding the popular categories. Recall that simple solutions in Section 6.3.1 need at least 122k short slots with our default setting. We use 122k as a baseline to

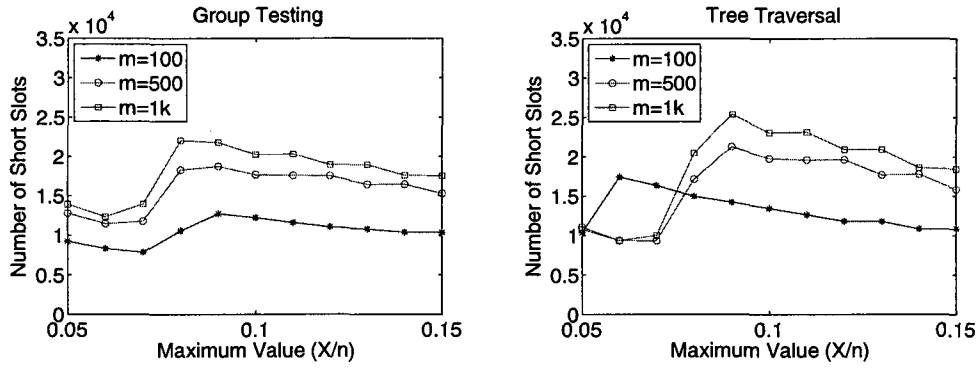


Figure 6.9: Scanning time for the $M1$ distribution with varying m

compare with our schemes. In most of the tested cases, the scanning time of our schemes with the default setting is less than 15k short slots, which is about 12% of the baseline. In the scenario that only a few popular categories exist, e.g., $UD(1)$, $UD(2)$, our schemes only require $< 4\%$ of the baseline to finish. We also observe that the group testing scheme is superior to the tree traversal scheme in most cases, especially when the number of tags in some unpopular categories is close to the threshold.

6.4.4 Tightness of Bounds

Our analysis in Theorem 6.5 uses Markov inequality, a loose bound that holds for arbitrary random variables. Theorem 6.5 is further referred in Lemma 6.3 and Theorem 6.6 to derive a upper bound of the expected scanning time. Thus, inherently the bound in Theorem 6.6 is relatively loose for any specific case. To understand how well the theoretical bound matches the reality, we compare our estimated scanning time with the simulation results in this subsection.

In the default setting, our algorithm estimates that the expected scanning time of the GT scheme is fewer than 14516 short slots. We compare this estimation with the results (mean scanning time) found in our simulations in the following table. For each distribution, we select the worst observed performance. According to the results, our estimation is very close to the actual

performance (the worst case is $UD(9)$ with 12734 short slots).

	Our Bound	UD	$M1$	ZD
Number of short slots	14516	12734	11615	9196

6.4.5 Other Issues

This subsection covers the discussions on some other issues:

1. **Accuracy requirements:** In all our simulations, both the completeness constraint and population constraint always hold with more than $1 - \delta$ probability.

2. **Other varying parameters:** When examining the scanning time, we also vary the parameters α and β, and find two basic trends. First, if α and β become closer, our schemes need more time to find popular categories. Second, if we keep their difference constantly, increasing one of them reduces the scanning time.

3. **Compare TCS with Ω:** Group testing can also be combined with algorithm Ω, because Ω obtains more accurate estimation than our TCS test. However, in the tested cases, the frame size for TCS is between 115 to 247 slots, much less than $|\Omega| = 2000$. Based on the results in [37], group testing with Ω will use smaller parameters T and W. The scanning time, however, is still much larger than that in our schemes with TCS.

6.5 Summary

In this chapter, we have discussed a data mining query of finding popular categories in RFID systems. It is difficult for weak devices like RFID tags to efficiently respond to such a complicated query because of the lack of processing and coordinating ability. Similar to our work in Chapter 4, we have adopted randomized algorithm in our solution. Combined with group testing technique,

we have proposed efficient protocols to reply to this complicated query. This work includes several novel means of analyzing the accuracy of the results and the scanning time, which can be applied to other similar problems. More importantly, the combination of randomized algorithm and group testing has been first introduced to accomplish complicated tasks in RFID systems.

Chapter 7

Conclusions and Future Work

This chapter concludes the dissertation by summarizing the contributions and proposing several future research directions.

7.1 Contributions

When Mark Weiser first introduced the concept of pervasive computing in the early 1990s, he was describing computers on pens or pads, and imaging a room with hundreds of wireless computing devices. Around twenty years later, technology has already caught up with his description and gone further towards his imagination. Computer hardware has evolved over the years, becoming more powerful, less expensive and smaller. The most importantly, computers have been *pervasive* devices in our daily life. Consumer electronics such as PDA, cell phone, media player, and calculator, are carried by people all the time. A lot more computing devices are weaved in our working and living space. For example, the modern kitchen appliances are configured by computer chips. Sensors are deployed to monitor machine status in a factory, the structural conditions on a bridge, and to detect wide fire in a forest. Computer-based medical devices have been equipped to help

health care. Many more pervasive computing devices and applications are being developed every day. Following this trend of development, we are substantially moving closer to the new era of pervasive computing referring to 'many computers per person'.

Besides the hardware devices, pervasive computing applications heavily rely on a wireless infrastructure that connects all the devices and delivers various useful data information. This infrastructure transparently links pervasive computing devices with each other and serves the upper application layer with the requested data across the whole environment. For designing such a wireless infrastructure, the most fundamental goal is to achieve efficiency, in particular, to save energy and time consumption in data provision. Most of pervasive computing devices are powered by batteries and saving energy is critical for prolonging their life time for service. Quick response to a query is also a highly desirable feature for pervasive computing applications, especially those with real-time requirements.

Designing an efficient wireless infrastructure, however, is challenging because of the hardware limitations. In practice, most pervasive computing devices are inexpensive, but weak in ability. There is a big gap between pervasive computing devices and regular computers in terms of all computation and communication resources. With this characteristic, many existing approaches designed for traditional computer systems can hardly be applied in pervasive computing environments. In addition, researchers encounter new problems and challenges that have never occurred in regular computer systems.

The main contribution of this dissertation is to solve critical efficiency problems in designing the wireless infrastructure and to explore appropriate methodologies for achieving efficiency with weak pervasive computing devices. This dissertation has investigated two representative infrastructures, sensor networks and RFID systems, both of which generate a large amount of data in

applications. To achieve efficiency, we have presented novel techniques for organizing network architecture and optimizing query protocols. Specifically, we have designed optimal algorithms for deploying storage sites in a sensor network and we have proposed efficient protocols for basic and complicated queries in sensor networks and RFID systems. We have developed solutions to address these representative and important problems in designing the wireless infrastructure.

First, we have proposed a novel two-tiered hybrid sensor network with special storage nodes to support in-network storage model and reduce the energy cost. In this storage model, we have worked on the problem of determining the locations of storage nodes, which is critical to the energy efficiency. Two practical models are considered, fixed tree model and dynamic tree model. We have developed optimal algorithms for the first model and an approximation algorithm for the second one. In addition, our work has presented performance analysis for a common practice of random deployment in both models.

Second, we have developed efficient protocols for two basic queries, range query in sensor networks and continuous scans in RFID systems. We have presented the first solution to provide security and privacy protection in range query. Particularly, we have developed a privacy preserving storage scheme and a range query protocol that enables the sink to verify the reply to the query. Our solutions satisfy the given security requirements with the minimum energy overhead. Additionally, we have proposed the first solution to launch continuous multiple scans in RFID systems without collecting all IDs in each scan. The key idea is to apply randomized algorithm to identify and avoid collecting the redundant RFID tags that have been previously gathered. We have discussed continuous scans in spatial and temporal domains and developed efficient solutions for both scenarios.

Finally, we have investigated complicated data mining queries in pervasive computing envi-

ronments. We have first designed efficient outlier detection algorithms in sensor networks based on histogram information. The basic idea is to collect small-sized histogram and apply analysis on it to filter unnecessary data transmission. In addition, we have developed the first protocol to efficiently find popular categories in an RFID system. Our solution includes a quick protocol for estimating the number of RFID tags based on randomized algorithm and a group testing scheme combined with the estimation protocol. The intuition is still to avoid collecting all RFID tags when responding to this complicated query.

7.2 Future Work

Following the work in this dissertation, there are several clear directions for future work. We start with two general directions and propose some particular problems later.

- **Complex data formats:** While demonstrating the methodology and design principles for addressing efficiency issue, this dissertation mainly considers simple data format, e.g., single dimensional sensor data and one RFID tag per item. In practice, some applications involve more complicated data formats. For example, sensors may measure data with multiple attributes and in an RFID system, each item may contain several tags. This reflects more complex problem settings for the topics discussed in the dissertation. For example, the range query may contain several value ranges each for a data attribute, outliers may be defined in the basis of multiple dimensional data, and categories in RFID systems may be classified by the combined information from several tags in an item. All these new settings raise new challenges and issues. Our work in this dissertation can be extended to solve some of the new problems. For the others, we need to explore new solutions.

- **System implementations:** Most of this dissertation focuses on theoretical and algorithmic design. It is a natural enhancement to consider system implementations in the future work which imply two aspects of efforts. First, we need to consider complete protocols that are compatible with the current hardware and software systems. For example, our work on sensor networks has focused on the data flow in the protocols, but ignored low level communication stacks such as duty-cycle MAC protocol, and packet coding and retransmission. To develop a complete protocol considering all these factors, our work needs to be further refined for achieving efficiency. Second, we need to consider practical issues for system implementation. For example, there might be signal loss in some RFID systems, i.e., the command from an RFID reader and the response from RFID tags cannot be successfully received. It leads to inaccurate or even misleading results in our solutions. In the future work, we need to first model and characterize the signal loss, and then integrate it into algorithm design as an input parameter.

Besides the above two general directions, there are some concrete new problems we can consider in the future work.

- **Multi-tier storage in sensor networks:** This is a follow-up problem from our work on storage placement in sensor networks, where we introduced storage nodes and built a two-tier network structure. In the future development, it is likely to build a multi-tier storage system in a sensor network, where each tier represents different storage hardware in terms of storage capacity, hardware cost and other functionalities. The applications benefit from this multi-tier structure by carefully assigning diverse data on these storage sites. Sensor data may have different types/sizes, e.g., small single-value data vs. large media data. Some data may be frequently queried while others may not. Some data may be more important

and some may be out-dated. Considering the diversity of sensor data, it is more appropriate to utilize a multi-tier storage system than a two-tier system. In this problem, the location of each storage site in each tier is still important to the efficiency performance. We are interested in investigating the storage placement problem in this direction.

- **Continuous / local outlier detection in sensor networks:** The outlier detection protocols in this dissertation only consider one-time detection. In practice, some applications need to continuously monitor the outlier data. To solve this problem, we can periodically launch our one-time detection protocols. However, it is inefficient to start from scratch every time because there might be minor change on the outlier data and histogram information compared to the previous runs. The better solution should utilize all available information from the last run and focus on the new data since then to identify new outliers or the existing outliers which are no longer abnormal. In addition, this dissertation has solved outlier detection with a global setting, i.e., all data form the target data set. However, some applications are interested in local outliers, i.e., the outliers are defined by the sensor data within a physical region. For example, if a sensor network is deployed to monitor the temperature in a building. Most of the rooms are kept in 75F. But one room's is configured to control the temperature at 68F. If a sensor in that special room reports a temperature reading of 75F, it is normal if we consider the global data set. However, when comparing to the readings from local sensors nearby, it becomes a outlier data. In practice, there is often no clear boundary for defining the desired region, e.g., in an open field. It is challenging and interesting to gradually expand the target set of sensors to identify a meaningful region and detect the outlier data at the same time.

- **Efficiently identify a group of items in RFID systems with security requirements:** This problem is about how an RFID reader can determine whether the RFID tags in the reading range contain a certain group of IDs. In the traditional solution, the reader broadcasts each ID in the group one by one and the tags matching the IDs will respond. By gathering the responses, the reader is able to determine the result. This solution, however, is not secure in a hostile environment because the adversary can easily eavesdrop the tag IDs. In the future work, we are looking for a solution based on randomized algorithm. The basic intuition is that since we are aware of the group of IDs, we can deterministically predict their responding behaviors (i.e., how to choose slot to reply) if they are present. Therefore, we may seek a certain pattern from all the slots to identify the group. However, the challenge is that we need to consider other tags in the reading range that may introduce noises to obstruct the solution.

7.3 Final Remarks

This dissertation focuses on the critical efficiency problems, particularly time efficiency and energy efficiency, in building wireless infrastructures for pervasive computing environments. In a general framework, we have found a system can be more efficient by a better strategy of organizing the underlying network architecture. We have proposed a new hybrid structure for sensor networks with special storage nodes and developed optimal algorithms for placing storage nodes to improve efficiency. In addition, we have realized that each particular query ought to be treated differently with a specially designed protocol in order to achieve more efficiency. We have investigated representative basic queries and data mining queries in sensor networks and RFID systems. For basic queries, this dissertation enhances the prior work by considering more practical problem settings.

In particular, we have shown for a sensor network with multiple objectives besides efficiency, how to design a query protocol to strike the tradeoff between them. We have further identified that the prior work on the basic query in RFID systems is inefficient in realistic scenarios. Using randomized algorithm, we have developed compatible RFID protocols for the basic query with much reduced scanning time. Finally, this dissertation has explored data mining queries which are useful in applications, but rarely discussed in pervasive computing environments. According to the different characteristics of sensors and RFID tags, we have demonstrated different techniques to efficiently process data mining queries. In sensor networks, we have proposed multiple round protocols that gradually collect coarse but small-sized information with corresponding analysis. With a careful design, it is an efficient way to handle complicated queries by filtering out redundancy and unnecessary data transmissions. In RFID systems, we have found that randomized algorithm is also an appropriate tool to deal with data mining queries. Combining with other techniques such as probability analysis, group testing, and combinational optimization, we have developed protocols that work with weak RFID tags to efficiently respond to the complicated data mining query.

Bibliography

[1] GLPK (GNU Linear Programming Kit), available [online] http://www.gnu.org/software/glpk/glpk.html.

[2] Intel Lab Data, avilable [online] http://berkeley.intel-research.net/labdata/, April 2004.

[3] KAREN AARDAL, F. A. CHUDAK, AND DAVID B. SHMOYS. A 3-approximation algorithm for the k-level uncapacitated facility location problem. *Inf. Process. Lett.*, 72(5-6):161–167, 1999.

[4] N. ABRAMSON. The ALOHA system - another alternative for computer communications. In *Proceedings of the AFIPS Conference*, volume 37, pages 295–298, 1970.

[5] ALEXANDER A. AGEEV. Improved approximation algorithms for multilevel facility location problems. In *APPROX '02: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 5–13, London, UK, 2002. Springer-Verlag.

[6] C. AGGARWAL AND S. YU. An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB Journal*, 14(2):211–221, 2005.

[7] CHARU C. AGGARWAL. Re-designing distance functions and distance-based applications for high dimensional data. *SIGMOD Rec.*, 30(1):13–18, 2001.

[8] CHARU C. AGGARWAL AND PHILIP S. YU. Outlier detection for high dimensional data. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 37–46, New York, NY, USA, 2001. ACM Press.

[9] DAKSHI AGRAWAL AND CHARU C. AGGARWAL. On the design and quantification of privacy preserving data mining algorithms. In *Symposium on Principles of Database Systems*, 2001.

[10] RAKESH AGRAWAL, ALEXANDRE EVFIMIEVSKI, AND RAMAKRISHNAN SRIKANT. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 86–97, New York, NY, USA, 2003. ACM Press.

[11] RAKESH AGRAWAL, JERRY KIERNAN, RAMAKRISHNAN SRIKANT, AND YIRONG XU. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 563–574, New York, NY, USA, 2004. ACM Press.

[12] RAKESH AGRAWAL, RAMAKRISHNAN SRIKANT, AND DILYS THOMAS. Privacy preserving OLAP. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 251–262, New York, NY, USA, 2005. ACM Press.

[13] JOON AHN AND BHASKAR KRISHNAMACHARI. Fundamental scaling laws for energy-efficient storage and querying in wireless sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 334–343, New York, NY, USA, 2006. ACM Press.

[14] SANJEEV ARORA, PRABHAKAR RAGHAVAN, AND SATISH RAO. Approximation schemes for euclidean k-medians and related problems. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, New York, NY, USA, 1998. ACM Press.

[15] VIJAY ARYA, NAVEEN GARG, ROHIT KHANDEKAR, KAMESH MUNAGALA, AND VINAYAKA PANDIT. Local search heuristic for k-median and facility location problems. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, New York, NY, USA, 2001. ACM Press.

[16] F. BACCELLI, M. KLEIN, M. LEBOURGES, AND S. ZUYEV. Stochastic geometry and architecture of communication networks. *J. Telecommunication Systems*, 7:209–227, 1997.

[17] FRANCOIS BACCELLI AND SERGEI ZUYEV. Poisson-Voronoi spanning trees with applications to the optimization of communication networks. *Operations Research*, 47(4):619–631, 1999.

[18] SEUNG JUN BAEK, GUSTAVO DE VECIANA, AND XUN SU. Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation. *IEEE JSAC Special Issue on Fundamental Performance Limits of Wireless Sensor Networks*, 22(6):1130–1140, August 2004.

[19] STEPHEN D. BAY AND MARK SCHWABACHER. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD '03: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 29–38, New York, NY, USA, 2003. ACM Press.

[20] SAGNIK BHATTACHARYA, HYUNG KIM, SHASHI PRABH, AND TAREK ABDELZAHER. Energy-conserving data placement and asynchronous multicast in wireless sensor networks. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pages 173–185, New York, NY, USA, 2003. ACM Press.

[21] BORIS JAN BONFILS AND PHILIPPE BONNET. Adaptive and decentralized operator placement for in-network query processing. In *IPSN '03: Proceedings of Information Processing in Sensor Networks*, 2003.

[22] PHILIPPE BONNET, JOHANNES GEHRKE, AND PRAVEEN SESHADRI. Towards sensor database systems. In *Proceedings of the Second International Conference on Mobile Data Management (MDM '01)*, pages 3–14, London, UK, 2001. Springer-Verlag.

[23] FLAVIO BONOMI, MICHAEL MITZENMACHER, RINA PANIGRAH, SUSHIL SINGH, AND GEORGE VARGHESE. Beyond bloom filters: from approximate membership checks to approximate state machines. *SIGCOMM Comput. Commun. Rev.*, 36(4):315–326, 2006.

[24] MAURIZIO A. BONUCCELLI, FRANCESCA LONETTI, AND FRANCESCA MARTELLI. Tree slotted ALOHA: a new protocol for tag identification in RFID networks. In *WOWMOM '06*, 2006.

[25] JOEL BRANCH, BOLESLAW SZYMANSKI, CHRIS GIANNELLA, RAN WOLFF, AND HILLOL KARGUPTA. In-network outlier detection in wireless sensor networks. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, Lisboa, Portugal, July 2006. IEEE Computer Society.

[26] MARKUS M. BREUNIG, HANS-PETER KRIEGEL, RAYMOND T. NG, AND JÖRG SANDER. LOF: identifying density-based local outliers. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, New York, NY, USA, 2000. ACM Press.

[27] ADRIANA BUMB AND WALTER KERN. A simple dual ascent algorithm for the multilevel facility location problem. In *APPROX '01/RANDOM '01: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 55–62, London, UK, 2001. Springer-Verlag.

[28] JAE-RYONG CHA AND JAE-HYUN KIM. Novel anti-collision algorithms for fast object identification in RFID system. In *ICPADS '05*, 2005.

[29] HAOWEN CHAN, ADRIAN PERRIG, AND DAWN SONG. Secure hierarchical in-network aggregation in sensor networks. In *CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 278–287, New York, NY, USA, 2006. ACM Press.

[30] YAN-CHENG CHANG AND MICHAEL MITZENMACHER. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of the 3rd Applied Cryptography and Network Security*, New Yrok, USA, June 2005.

[31] MOSES CHARIKAR AND SUDIPTO GUHA. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 378, Washington, DC, USA, 1999. IEEE Computer Society.

[32] MOSES CHARIKAR, SUDIPTO GUHA, ÉVA TARDOS, AND DAVID B. SHMOYS. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1999. ACM Press.

[33] HO-SEUNG CHOI, JAE-RYON CHA, AND JAE-HYUN KIM. Fast wireless anti-collision algorithm in ubiquitous ID system. In *Vehicular Technology Conference*, pages 4589–4592, 2004.

[34] DAVID CHU, AMOL DESHPANDE, JOSEPH M. HELLERSTEIN, AND WEI HONG. Approximate data collection in sensor networks using probabilistic models. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 48. IEEE Computer Society, 2006.

[35] JULIA CHUZHOY AND YUVAL RABANI. Approximating k-median with non-uniform capacities. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–958, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[36] I. CIDON AND M. SIDI. Conflict multiplicity estimation and batch resolution algorithms. *IEEE Trans. Inf. Theor.*, 34(1):101–110, 1988.

[37] GRAHAM CORMODE AND S. MUTHUKRISHNAN. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.

[38] SWADES DE. On hop count and euclidean distance in greedy forwarding in wireless ad hoc networks. *IEEE Communication Letters*, 9, 2005.

[39] ERIK D. DEMAINE, ALEJANDRO LEORTIZ, AND J. IAN MUNRO. Frequency estimation of internet packet streams with limited space. In *ESA '02*, 2002.

[40] ALAN DEMERS, JOHANNES GEHRKE, RAJMOHAN RAJARAMAN, NIKI TRIGONI, AND YONG YAO. The cougar project: a work-in-progress report. *SIGMOD Rec.*, 32(4):53–59, 2003.

[41] PETER DESNOYERS, DEEPAK GANESAN, HUAN LI, MING LI, AND PRASHANT SHENOY. PRESTO: A predictive storage architecture for sensor networks. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS X)*, Santa Fe, New Mexico, June 2005.

[42] DINGZHU DU AND F. HWANG. *Combinatorial Group Testing and Its Applications*. World Scientific Publishing Company, 1993.

[43] E. J. DUARTE-MELO AND M. LIU. Data-gathering wireless sensor networks: organization and capacity. *Computer Networks (COMNET)*, 43(4):519–537, Nov. 2003.

[44] CHENG TIEN EE, SYLVIA RATNASAMY, AND SCOTT SHENKER. Practical data-centric storage. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, USA, May 2006.

[45] EPCGLOBAL. Class 1 generation 2 UHF air interface protocol standard version 1.0.9.

[46] QING FANG, JIE GAO, LEONIDAS GUIBAS, VIN DE SILVA, AND LI ZHANG. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *INFOCOM '05: Proceedings of the 24th Conference of the IEEE Communication Society*, volume 1, pages 339–350, March 2005.

[47] ABRAHAM D. FLAXMAN, ALAN M. FRIEZE, AND JUAN C. VERA. On the average case performance of some greedy approximation algorithms for the uncapacitated facility location problem. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 441–449, New York, NY, USA, 2005. ACM Press.

[48] CHRISTIAN FLOERKEMEIER AND MATTHIAS WILLE. Comparison of transmission schemes for framed ALOHA based RFID protocols. In *SAINT-W '06*, 2006.

[49] WAI FU FUNG, DAVID SUN, AND JOHANNES GEHRKE. COUGAR: the network is the database. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 621–621, New York, NY, USA, 2002. ACM Press.

[50] DEEPAK GANESAN, DEBORAH ESTRIN, AND JOHN HEIDEMANN. Dimensions: why do we need a new data handling architecture for sensor networks? *SIGCOMM Comput. Commun. Rev.*, 33(1):143–148, 2003.

[51] DEEPAK GANESAN, BEN GREENSTEIN, DEBORAH ESTRIN, JOHN HEIDEMANN, AND RAMESH GOVINDAN. Multiresolution storage and search in sensor networks. *Trans. Storage*, 1(3):277–315, 2005.

[52] DEEPAK GANESAN, BEN GREENSTEIN, DENIS PERELYUBSKIY, DEBORAH ESTRIN, AND JOHN HEIDEMANN. An evaluation of multi-resolution storage for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 89–102, New York, NY, USA, 2003. ACM Press.

[53] JOHANNES GEHRKE AND SAMUEL MADDEN. Query processing in sensor networks. *IEEE Pervasive Computing*, 03(1):46–55, 2004.

[54] ANNA C. GILBERT, SUDIPTO GUHA, PIOTR INDYK, YANNIS KOTIDIS, S. MUTHUKRISHNAN, AND MARTIN STRAUSS. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, pages 389–398, 2002.

[55] ANNA C. GILBERT, YANNIS KOTIDIS, S. MUTHUKRISHNAN, AND MARTIN STRAUSS. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB '02*, 2002.

[56] P. GOLLE, J. STADDON, AND B. WATERS. Secure conjunctive keyword search over encrypted data. In *Proceedings of the 2004 Applied Cryptography and Network Security Conference*, pages 31–45. LNCS 3089, 2004.

[57] HECTOR GONZALEZ, JIAWEI HAN, AND XIAOLEI LI. Flowcube: constructing RFID flowcubes for multi-dimensional analysis of commodity flows. In *VLDB '06*, 2006.

[58] HECTOR GONZALEZ, JIAWEI HAN, AND XIAOLEI LI. Mining compressed commodity workflows from massive RFID data sets. In *CIKM '06*, 2006.

[59] MICHAEL B. GREENWALD AND SANJEEV KHANNA. Power-conserving computation of order-statistics over sensor networks. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 275–285, New York, NY, USA, 2004. ACM Press.

[60] M. GRUTESER, G. SCHELL, A. JAIN, R. HAN, AND D. GRUNWALD. Privacy-aware location sensor networks. In *Proceedings of Workshop on Hot Topics in Operating Systems (HotOS)*, 2003.

[61] SUDIPTO GUHA AND SAMIR KHULLER. Greedy strikes back: improved facility location algorithms. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 649–657, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[62] P. GUPTA AND P. R. KUMAR. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, 2000.

[63] HAKAN HACIGUMUS, BALAKRISHNA R. IYER, CHEN LI, AND SHARAD MEHROTRA. Executing SQL over encrypted data in the database service provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 2002.

[64] FANG HAO, MURALI S. KODIALAM, AND T. V. LAKSHMAN. Building high accuracy bloom filters using partitioned hashing. In *SIGMETRICS '07*, 2007.

[65] WENDI HEINZELMAN, ANANTHA CHANDRAKASAN, AND HARI BALAKRISHNAN. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of International Conference on System Sciences*, Maui, HI, USA, Jan 2000.

[66] P. HERNANDEZ, J.D. SANDOVAL, F. PUENTE, AND F. PEREZ. Mathematical model for a multiread anticollision protocol. In *IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, Aug. 2001.

[67] BIJIT HORE, SHARAD MEHROTRA, AND GENE TSUDIK. A privacy-preserving index for range queries. In *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004.

[68] L. HU AND D. EVANS. Secure aggregation for wireless networks. In *Proceedings of Workshop on Security and Assurance in Ad hoc Networks*, Jan. 2003.

[69] D. HUSH AND C. WOOD. Analysis of tree algorithms for RFID arbitration. In *ISIT*, 1998.

[70] CHALERMEK INTANAGONWIWAT, RAMESH GOVINDAN, AND DEBORAH ESTRIN. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 56–67, New York, NY, USA, 2000. ACM Press.

[71] CHALERMEK INTANAGONWIWAT, RAMESH GOVINDAN, DEBORAH ESTRIN, JOHN HEIDEMANN, AND FABIO SILVA. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[72] KAMAL JAIN, MOHAMMAD MAHDIAN, AND AMIN SABERI. A new greedy approach for facility location problems. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 731–740, New York, NY, USA, 2002. ACM Press.

[73] KAMAL JAIN AND VIJAY V. VAZIRANI. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[74] SHAWN R. JEFFERY, MINOS GAROFALAKIS, AND MICHAEL J. FRANKLIN. Adaptive cleaning for RFID data streams. In *VLDB'2006*, 2006.

[75] PANDURANG KAMAT, YANYONG ZHANG, WADE TRAPPE, AND CELAL OZTURK. Enhancing source-location privacy in sensor network routing. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 599–608, Washington, DC, USA, 2005. IEEE Computer Society.

[76] BRAD KARP AND H. T. KUNG. GPSR: greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, New York, NY, USA, 2000. ACM Press.

[77] RICHARD M. KARP, SCOTT SHENKER, AND CHRISTOS H. PAPADIMITRIOU. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, 2003.

[78] HYUNG SEOK KIM, TAREK F. ABDELZAHER, AND WOOK HYUN KWON. Minimum-energy asynchronous dissemination to mobile sinks in wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 193–204, New York, NY, USA, 2003. ACM Press.

[79] YOUNG-JIN KIM, RAMESH GOVINDAN, BRAD KARP, AND SCOTT SHENKER. Geographic routing made practical. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.

[80] EDWIN M. KNORR AND RAYMOND T. NG. Algorithms for mining distance-based outliers in large datasets. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 392–403, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[81] EDWIN M. KNORR AND RAYMOND T. NG. Finding intensional knowledge of distance-based outliers. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 211–222, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[82] M. KODIALAM, THYAGA NANDAGOPAL, AND WING CHEONG LAU. Anonymous tracking using RFID tags. In *INFOCOM '07*, 2007.

[83] MURALI KODIALAM AND THYAGA NANDAGOPAL. Fast and reliable estimation schemes in RFID systems. In *MobiCom '06*, 2006.

[84] MADHUKAR R. KORUPOLU, C. GREG PLAXTON, AND RAJMOHAN RAJARAMAN. Analysis of a local search heuristic for facility location problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 1–10, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[85] DENIS KRIVITSKI, ASSAF SCHUSTER, AND RAN WOLFF. A local facility location algorithm for sensor networks. In *DCOSS '05: First IEEE International Conference Distributed Computing in Sensor Systems*, pages 368–375, 2005.

[86] CYNTHIA KUO, MARK LUK, ROHIT NEGI, AND ADRIAN PERRIG. Message-in-a-bottle: user-friendly and secure key deployment for sensor nodes. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 233–246, New York, NY, USA, 2007. ACM.

[87] CHING LAW, KAYI LEE, AND KAI-YEUNG SIU. Efficient memoryless protocol for tag identification. In *Proceedings of the 1th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 75–84. ACM, August 2000.

[88] ALEKSANDAR LAZAREVIC AND VIPIN KUMAR. Feature bagging for outlier detection. In *KDD '05: Proceeding of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 157–166, New York, NY, USA, 2005. ACM Press.

[89] SU-RYUN LEE, SUNG-DON JOO, AND CHAE-WOO LEE. An enhanced dynamic framed slotted ALOHA algorithm for RFID tag identification. In *MOBIQUITOUS '05*, 2005.

[90] MING LI, DEEPAK GANESAN, AND PRASHANT SHENOY. PRESTO: Feedback-driven data management in sensor networks. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, San Jose, CA, USA, May 2006.

[91] YUNHAO LIU, LEI CHEN, JIAN PEI, QIUXIA CHEN, AND YIYANG ZHAO. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. In *PerCom '07: Proceedings of the 5th Annual IEEE International Conference on Pervasive Computing and Communications*, 2007.

[92] SAMUEL MADDEN, MICHAEL J. FRANKLIN, JOSEPH M. HELLERSTEIN, AND WEI HONG. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Opererating System Review*, 36(SI):131–146, 2002.

[93] SAMUEL MADDEN, MICHAEL J. FRANKLIN, JOSEPH M. HELLERSTEIN, AND WEI HONG. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 491–502, New York, NY, USA, 2003. ACM.

[94] SAMUEL R. MADDEN, MICHAEL J. FRANKLIN, JOSEPH M. HELLERSTEIN, AND WEI HONG. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[95] GAURAV MATHUR, PETER DESNOYERS, DEEPAK GANESAN, AND PRASHANT SHENOY. CAPSULE: An energy-optimized object storage system for memory-constrained sensor devices. In *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, Boulder, Colorado, USA, 2006. ACM Press.

[96] GAURAV MATHUR, PETER DESNOYERS, DEEPAK GANESAN, AND PRASHANT SHENOY. Ultra-low power data storage for sensor networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, pages 374–381, New York, NY, USA, 2006. ACM Press.

[97] BOB METCALFE. Steady-state analysis of a slotted and controlled ALOHA system with blocking. *SIGCOMM Comput. Commun. Rev.*, 5(1):24–31, 1975.

[98] A. MICIC, A. NAYAK, D. SIMPLOT-RYL, AND I. STOJMENOVIC. A hybrid randomized protocol for RFID tag identification. In *IEEE International Workshop on Next Generation Wireless Networks*, 2005.

[99] MICHAEL MITZENMACHER. Compressed bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, 2002.

[100] J. MYUNG AND W. LEE. An adaptive memoryless tag anti-collision protocol for RFID networks. In *IEEE ICC*, 2005.

[101] JIHOON MYUNG AND WONJUN LEE. Adaptive splitting protocols for RFID tag collision arbitration. In *MobiHoc '06: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.

[102] JAMES NEWSOME AND DAWN SONG. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 76–88, New York, NY, USA, 2003. ACM Press.

[103] NXP SEMICONDUCTORS. I-code smart label RFID tags.

[104] THEMISTOKLIS PALPANAS, DIMITRIS PAPADOPOULOS, VANA KALOGERAKI, AND DIMITRIOS GUNOPULOS. Distributed deviation detection in sensor networks. *SIGMOD Rec.*, 32(4):77–82, 2003.

[105] BARTOSZ PRZYDATEK, DAWN SONG, AND ADRIAN PERRIG. SIA: secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 255–265, New York, NY, USA, 2003. ACM Press.

[106] CHEN QIAN, HOILUN NGAN, AND YUNHAO LIU. Cardinality estimation for large-scale RFID systems. In *PerCom '08: Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 30–39, Washington, DC, USA, 2008. IEEE Computer Society.

[107] SRIDHAR RAMASWAMY, RAJEEV RASTOGI, AND KYUSEOK SHIM. Efficient algorithms for mining outliers from large data sets. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 427–438, New York, NY, USA, 2000. ACM Press.

[108] SYLVIA RATNASAMY, BRAD KARP, SCOTT SHENKER, DEBORAH ESTRIN, RAMESH GOVINDAN, LI YIN, AND FANG YU. Data-centric storage in sensornets with GHT, a geographic hash table. *Mobile Networks and Applications*, 8(4):427–442, 2003.

[109] KUI REN, WENJING LOU, KWANGJO KIM, AND ROBERT DENG. A novel privacy preserving authentication and access control scheme for pervasive computing environment. *IEEE Transactions on Vehicular Technology*, 2006.

[110] F. C. SCHOUTE. Dynamic frame length ALOHA. *IEEE Transactions on Communications*, 31:565–568, April 1983.

[111] RAHUL C SHAH, SUMIT ROY, SUSHANT JAIN, AND WAYLON BRUNETTE. Data MULEs: Modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SPNA)*, Anchorage, AK, USA, May 2003.

[112] M. SHAO, S. ZHU, W. ZHANG, AND G. CAO. pDCS: Security and privacy support for data-centric sensor networks. In *INFOCOM '07: Proceedings of the 26th IEEE International Conference on Computer Communications*, Anchorage, Alaska, USA, 2007. ACM Press.

[113] SCOTT SHENKER, SYLVIA RATNASAMY, BRAD KARP, RAMESH GOVINDAN, AND DEBORAH ESTRIN. Data-centric storage in sensornets. *SIGCOMM Computer Communication Review*, 33(1):137–142, 2003.

[114] DAVID B. SHMOYS, E. TARDOS, AND KAREN AARDAL. Approximation algorithms for facility location problems (extended abstract). In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, New York, NY, USA, 1997. ACM Press.

[115] NISHEETH SHRIVASTAVA, CHIRANJEEB BURAGOHAIN, DIVYAKANT AGRAWAL, AND SUBHASH SURI. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 239–249, New York, NY, USA, 2004. ACM Press.

[116] ADAM SILBERSTEIN, REBECCA BRAYNARD, CARLA ELLIS, KAMESH MUNAGALA, AND JUN YANG. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE '06: Proceedings of 22nd International Conference on Data Engineering*. IEEE Computer Society, 2006.

[117] ADAM SILBERSTEIN, KAMESH MUNAGALA, AND JUN YANG. Energy-efficient monitoring of extreme values in sensor networks. In *SIGMOD '06: 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, Illinois, USA, June 2006.

[118] DAWN XIAODONG SONG, DAVID WAGNER, AND ADRIAN PERRIG. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 2000. IEEE Computer Society.

[119] UTKARSH SRIVASTAVA, KAMESH MUNAGALA, AND JENNIFER WIDOM. Operator placement for in-network stream query processing. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 250–258, New York, NY, USA, 2005. ACM.

[120] S. SUBRAMANIAM, T. PALPANAS, D. PAPADOPOULOS, V. KALOGERAKI, AND D. GUNOPULOS. Online outlier detection in sensor data using non-parametric models. In *VLDB'2006: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 187–198. VLDB Endowment, 2006.

[121] HARALD VOGT. Efficient object identification with passive RFID tags. In *International Conference on Pervasive Computing*, LNCS. Springer-Verlag, 2002.

[122] SERDAR VURAL AND EYLEM EKICI. Analysis of hop-distance relationship in spatially random sensor networks. In *MobiHoc '05: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 320–331, New York, NY, USA, 2005. ACM Press.

[123] YAWEN WEI, ZHEN YU, AND YONG GUAN. Location verification algorithms for wireless sensor networks. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, Toronto, Canada, 2007. ACM Press.

[124] J.E. WIESELTHIER, A. EPHREMIDES, AND L.A. MICHAELS. An exact analysis and performance evaluation of framed ALOHA withcapture. *IEEE Transactions on Communications*, pages 125–137, Feb. 1989.

[125] YI YANG, XINRAN WANG, SENCUN ZHU, AND GUOHONG CAO. SDAP: a secure hop-by-hop data aggregation protocol for sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.

[126] FAN YE, HAIYUN LUO, SONGWU LU, AND LIXIA ZHANG. Statistical en-route detection and filtering of injected false data in sensor networks. In *INFOCOM '04: Proceedings of the 23th IEEE International Conference on Computer Communications*, Hong Kong, 2004. ACM Press.

[127] D. ZEINALIPOUR-YAZTI, Z. VAGENA, D. GUNOPULOS, V. KALOGERAKI, V. TSOTRAS, M. VLACHOS, N. KOUDAS, AND D. SRIVASTAVA. The threshold join algorithm for top-k queries in distributed sensor networks. In *DMSN '05: Proceedings of the 2nd International Workshop on Data Management for Sensor Networks*, pages 61–66, New York, NY, USA, 2005. ACM Press.

[128] DEMETRIOS ZEINALIPOUR-YAZTI, SONG LIN, VANA KALOGERAKI, DIMITRIOS GUNOPULOS, AND WALID A. NAJJAR. MicroHash: An efficient index structure for flash-based sensor devices. In *Proceedings of the FAST '05 Conference on File and Storage Technologies*, San Francisco, California, USA, December 2005. USENIX.

[129] JIA ZHAI AND GI-NAM WANG. An anti-collision algorithm using two-functioned estimation for RFID tags. In *ICCSA (4)*, pages 702–711. Springer, 2005.

[130] JIAWEI ZHANG. Approximating the two-level facility location problem via a quasi-greedy approach. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 808–817, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[131] NAN ZHANG AND WEI ZHAO. Distributed privacy preserving information sharing. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 889–900. VLDB Endowment, 2005.

[132] YANCHAO ZHANG, WEI LIU, YUGUANG FANG, AND DAPENG WU. Secure localization and authentication in ultra-wideband sensor networks. *IEEE Journal on Selected Areas in Communications*, 2006.

[133] BIN ZHEN, MAMORU KOBAYASHI, AND MASASHI SHIMIZU. Framed ALOHA for multiple RFID objects identification. In *IEICE TRANSACTIONS on Communications*, 2005.

[134] FENG ZHOU, CHUNHONG CHEN, DAWEI JIN, CHENLING HUANG, AND HAO MIN. Evaluating and optimizing power consumption of anti-collision protocols for applications in RFID systems. In *ISLPED '04*, 2004.

[135] JIANMING ZHOU, WENSHENG ZHANG, AND DAJI QIAO. Protecting storage location privacy in sensor networks. In *QShine '07: Proceedings of the 4th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Vancouver, Canada, 2007. ACM Press.

VITA

Bo Sheng

Bo Sheng received his Bachelor of Science degree in Computer Science from Nanjing University (Nanjing, China) in 2000. He was admitted to the Ph.D. program in Computer Science Department at the College of William and Mary in 2004, and became a Ph.D. candidate in 2005. His major research interests include wireless networks and embedded systems with an emphasis on the efficiency and security problems.