CONTINUOUS DATA COLLECTION IN WIRELESS SENSOR NETWORKS

by

Dan Wang

B.S. Peking University, 2000

M.S. Case Western Reserve University, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
2007

APPROVAL

Name: Dan Wang

Degree: Doctor of Philosophy

Title of thesis: Continuous Data Collection in Wireless Sensor Networks

Examining Committee: Dr. Petra Berenbrink,
Chair

Dr. Funda Ergun, Senior Supervisor,
School of Computing Science,
Simon Fraser University

Dr. Jiangchuan Liu, Supervisor,
School of Computing Science,
Simon Fraser University

Dr. Jian Pei, SFU Examiner,
School of Computing Science,
Simon Fraser University

Dr. Ben Liang, External Examiner,
Department of Electrical and Computer Engineering,
University of Toronto

Date Approved: 01/18/2007

Abstract

Recently, it has come to be generally believed by academia and industry alike that the sensor network will have a key role to extend the reachability of the next generation Internet. A key characteristic of this network is that there is no single node in the network that is powerful enough to perform the assigned tasks. An application should be served via the cooperation of several nodes or even the entire network. The network serves as an information base, and is data driven, as opposed to a provider for the point-to-point connection.

The main challenge of this network is huge information organization, including information storage, searching and retrieval, especially in a continuous way. There are many specific and interrelated problems. We list a few examples. First, data accuracy: the correctness of the sensor network to represent the properties of the sensor field. Second, data search and retrieval delay; while low delay is always preferred, various applications have different delay constraints. Third, overhead; low transmission overhead is often the main consideration in system design, as it is directly related to the usage of energy, the most severely limited resource for sensors.

In this thesis, we first discuss load balanced sensor coverage, which provides a lower layer support for long run sensor data collection. We then concentrate on how to balance the parameters in data collection of the sensor networks, so that the user queries and applications can be satisfied with reasonable delay and low overhead. Based on different application specifics, we try to use a smaller number of sensors, less number of transmissions by exploring historical and topological information, coding techniques and data distribution information. Our analysis and experimental results show that our architecture and algorithms provide both theoretical and practical insights for sensor network design and deployment.

To my parents

Acknowledgments

I would first like to thank my senior supervisor Prof. Funda Ergun. What I learned from her is enormous. Her attitude towards research has greatly influenced me. Her support and encourage during all the past years are invaluable for the success of my Ph.D studies.

I want to thank Prof. Jiangchuan Liu. His support has made the road to completing this thesis smoother. Prof. Qian Zhang and Prof. Jianliang Xu have provided constant support and encourage during my research. The collaborations with them are precious. I also want to thank Prof. Jian Pei and Prof. Ben Liang for serving as my thesis examiners. Their suggestions have substantially enriched this thesis.

Many colleagues of mine not only provided help in my studies but also in my everyday life. The time with them is unforgettable.

Finally, nothing would happen without you, my parents. I love you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the advances in electrical engineering and embedded systems, micro sensors and reliable communication between them have become a reality, leading to the emergence of large sensor networks. A sensor network is a network consisting of a large number of small computing nodes called sensors and is connected to the outside world via more powerful nodes call base stations. A sensor typically consists of a data processing module, a sensing module, a transmission module and a power module and can be used for computation, data collection, storage and routing.

A Sensor Network Example: There are many different types of sensors, such as the tiny Berkeley mote and larger but more powerful UCLA WINS, etc., or even mobile sensors. In Fig. 1.1, we show an example sensor network consisting of a popular type of sensor nodes, the mica-2 series from Crossbow Inc. Fig. 1.2 (a) and (b) show the mica-2 node and mica-2 dot node respectively. In Fig. 1.2 (c) a mica-2 node is plugged in a base board, which is connected to the Internet. This node can be considered as a *base station*[1] for inter-connectivity between the sensor network and the outside world.

Sensor networks are usually deployed in an environment where traditional wired or wireless networks are not available/appropriate, so as to extend the reachability of the current infrastructured computer networks. The main duties for a sensor network are data collection and management. The intended uses of a sensor network include terrain monitoring,

[1]We sometimes call it a *server*. Both base station and server are inter-connection points between the sensor network and the outside world. A slight difference is that in our following chapters, base station is an anchor point with all-time connection to the sensor network; while a server may travel to the sensor network and the connection between the server and sensor network is intermittent.

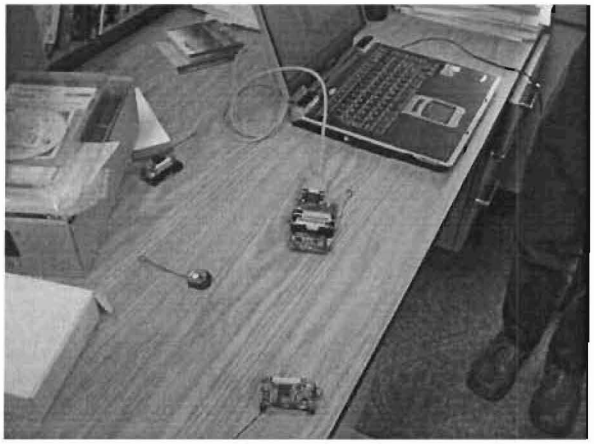
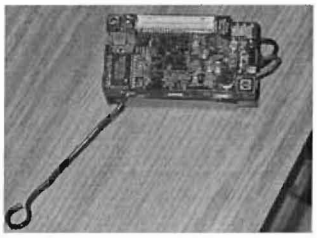
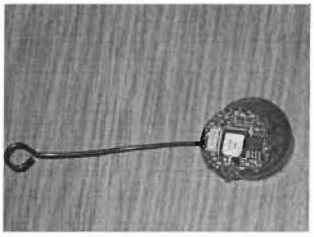


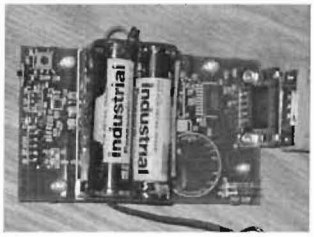
Figure 1.1: A small sensor network with mica sensors from Crossbow Inc.



(a) A mica-2 sensor



(b) A mica-2 dot sensor



(c) A base board with a mica-2 sensor; connecting to the Internet

Figure 1.2: Components of a mica-2 sensor network.

surveillance, and discovery [30] with a large number of applications such as geological tasks, military surveillance, search and rescue operations, building safety monitoring, and biological systems.

The major difference between a sensor network and the traditional network is that sensors are typically extremely small, low cost devices and sensors are tightly resource constrained. They not only lack long lifespan due to their limited battery resource but also possess little computational power and memory storage [2]. For example, a current mica-2 sensor has a programmable memory size of 128KB, a transmission bandwidth of 38.4Kbps and power support of two 5A batteries. As a result, one sensor can only collect a small amount of data from its adjacent environment and carry out a limited number of computations. In addition, sensors are less reliable devices both in packet transmission and survivability compared to the computers in the Internet. As a solution to these shortcomings, a single sensor is generally expected to work in cooperation with other sensors to provide service. The sensors are redundantly deployed in very large quantities and a sensor network usually consists of thousands or even tens of thousands of nodes.

Power conservation is the main focus of the current sensor network design as it is difficult and cost ineffective to recharge the sensor batteries. Main energy consumption in sensor networks is caused by packet transmission, both in sending and receiving. The energy cost is proportional to the payload of the transmission. It is also significantly affected by the length of the transmission. In [81], it is measured that $e = d^r$ where e is the energy, d is the distance between two sensors and r is a constant in [2, 6].

Data accuracy is also an important design parameter. Inaccuracy either comes from statistical error or systematic error. The former is primarily caused if the sensor network can not fully cover the sensor field and thus fail to represent the properties of the sensor field. The latter is mainly due to system design considerations, e.g., a result of trade-offs of different system parameters.

Delay is another important design concern. As the scale of a sensor network can reach thousands of nodes, and the requested service is expected to be answered cooperatively by a significant part or even the whole network, operations in a sensor network usually introduce a long delay.

Based on their unique features and capabilities, immense research activities have been undertaken in sensor networks. In this thesis, we are interested in a series of problems related to continuous data collection in sensor networks for a long period of time; the

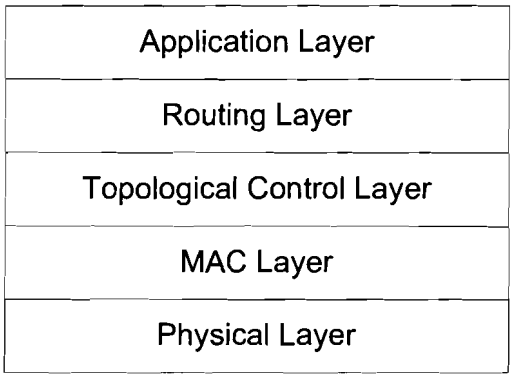


Figure 1.3: Protocol stack of a sensor network

demands, challenges and solutions.

1.1 An Overview of Sensor Network Architecture

Although there is no consensus of the protocol stack of the sensor network, a possible classification is given in Fig. 1.3.

The physical layer and MAC layer provide reliable transmission between sensors. The widely used MAC protocol currently is the short range 802.15.4; whereas longer transmission range is under active research and development [59].

Topological control layer is an important layer to sensor networks. This layer provides topological services such as coverage, connectivity, location, etc. A sensor has a transmission range and a sensing range. It functions not only as a node sending and receiving packets, but also as a sensing device, which collects readings from the surrounding environment. For the successful operation of a sensor network, at least two objectives should be achieved in topological control layer, i.e., high quality coverage and network-wide connectivity. For location aware applications, a sensor should also be able to estimate the its position in the sensor network.

The routing layer is closely related to application specifics. Some applications require point-to-point communication; and protocols like GPSR [43] are used. Some applications require in-network operation and process; and tree like topology may be used so that data from multiple children can be processed in the parent. Other routing algorithms such as clustering routing, multi-path routing, etc., are also related with different application

scenarios. It is unclear whether a general and unique routing layer suitable to all applications exists. Therefore, one may need to consider the routing layer and application layer together.

To build an efficient sensor network, one may take a bottom up approach. For example, one can start off by building an efficient and robust physical layer. Then he can select suitable protocols for MAC layer and high quality topology control schemes. Finally, he can work on efficient routing algorithms. One may also take a top down approach. For example, based on application specifics, he can find trade-offs of using fewer sensors and fewer transmissions. Then he can develop routing and topological control schemes to facilitate the architecture. Based on our experience, the sensor network is tightly coupled with application specifics. Sometimes even the sensor node is manufactured subject to application requirements. Thus, in this thesis, we first consider the requirements for different applications and then build the sensor network accordingly.

1.2 Sensor Coverage

A sensor application can hardly achieve its purpose without satisfiable coverage of the sensor field and an efficient sensor coverage is very important for a sensor network to function for a long period of time. A point in a sensor field is said to be covered at a time if this point is within the sensing range of an active sensor. The *k-coverage* is a common criterion, where any point in the sensor field should be covered by k sensors [72]. For many applications, it finds out that a deterministic k-coverage is too expensive and not necessary. Therefore, probabilistic coverage can be used and a point may not always be covered. Formally, for a point within the range of a few sensors, this point is covered with probability p if, at any certain time, the probability of at least one of the sensors is active is p. A user can specify a threshold of coverage ratio from [0, 1], which tunes the coverage quality of each point in the sensor field and sensors may switch between 'sleep' and 'active' states to save energy.

The coverage of sensor network is also related to the deployment methods of the sensor network. If the sensors are deployed manually, the coverage quality is easier to control. This is sometimes not cost-effective or, worse, impossible in many situations. A random deployment (e.g., from the air) is thus considered more feasible. Unfortunately, the unpredictable nature of the random deployment may lead to unfavorable distributions of the sensors and can hardly be compensated by static sensors only. Thus, mobile sensors are recently considered for sensor coverage problems. The advances in system designs have made this possible.

Mobile sensors, such as Robomote [71] and Khapera [60] can continuously function for 30 - 60 minutes with a moving speed of 20 - 100 cm/s. Unlike the static sensors, which are tightly constrained by the energy supplies, their batteries are easier to get recharged. Recent work also suggests that much longer working time and shorter recharging time can soon be expected in the near future [41].

The quality of sensor coverage is directly related to the quality of representation of the sensor network for the sensor field. Providing high quality of sensor coverage, as well as an energy efficient one, is thus a fundamental problem in this thesis.

1.3 Continuous Data Collection in Sensor Networks

1.3.1 Queries and Aggregation

One major use of the sensor networks is data collection and information retrieval. Each sensor collects data from its surroundings, stores the data if necessary, and forwards it to the base station. The interaction between sensors and the base station is usually done via *queries*. A simple query can have an SQL form as follows.

```
SELECT temp
  FROM sensors
  HAVING light > 50
```

where temp and light are the readings of temperature and light strength of a certain sensor.

Apart from collection raw readings from each individual sensor, queries can be used to collected aggregated information, such as MAXIMUM, MINIMUM, AVERAGE, QUANTILE, SUM [52][53], etc. An typical example is as follows [52]

```
SELECT TRUNC (temp / 10) AVERAGE temp
  FROM sensors
  GROUP BY TRUNC (temp / 10)
  HAVING AVERAGE (light) > 50
```

When replies are sent from the sensors to the base station, each intermediate sensor can perform some in-network processes to evaluate the data. For example, in a MAXIMUM query, an intermediate sensor can compare the data it received from the downstream sensors as well as its own; and only submit the maximum one upstream. This will reduce the

payload and consequently save on valuable transmission energy. We call this in-network process *aggregation.*

If the overall evaluation of the queries can not be answered accurately, these queries are called *approximate queries.* For applications that are more interested in the overall picture of the sensor field or the changing pattern of the data than specific data values of individual sensors, approximate queries are widely acceptable.

1.3.2 A Data Driven Network

The current Internet serves as a point-to-point provider and the architecture is client-server based. To search data or perform certain operation, a client will first locate a server by the server's address. Requests or commands are then sent to the server and the responses will be sent back to the client.

An implicit assumption for this client and server architecture is that the client knows a server which is capable to complete the client requests. The search for the server is address based. In contrast to this model, sensor network is data driven (also called data centric) due to the resource and power limitation of each individual sensor. Requests (for data) will be sent to the sensor network and the sensors will perform tasks depending on data values or data types. In [21] data centric routing is introduced to reduce energy consumption. The user request is first broadcast to each sensor in the sensor network. The routing of the replies (data) is performed where aggregation can be maximized according to the data values to reduce the payload. Data centric storage is introduced in [69] where the data will be sent to storage sensors according to the data types after they are collected. The data are thus stored in clusters. The queries from the base station can be forwarded to the sensors in an area concentrated with certain data type. The number of broadcasting messages can thus be reduced. Numerous studies have shown that the data centric paradigm is very suitable for the resource constrained sensor network in data collection.

1.3.3 Underlying Routing Support

For different applications and queries, the underlying routing architecture can be different. In certain scenarios, the base station can directly contact specific sensors. In other scenarios, multi-hop routing schemes are used. One popular routing scheme to assist data collection is the tree structure. During transmission, a parent node can aggregate the data it received

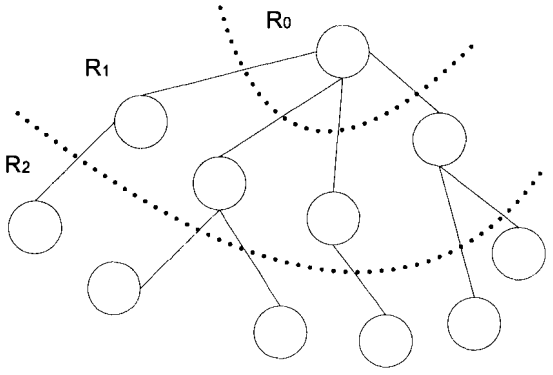


Figure 1.4: Tree construction by levels.

from its children to reduce the payload. A simple tree construction algorithm can be done by broadcasting from the base station [62]. The base station sets itself as the level zero node and broadcast a build tree message with its own level plus one. Each sensor will set its level to the smallest level it received. The build tree message will be recursively sent to all the nodes in the network. An example of this tree construction scheme is shown in Fig. 1.4.

1.3.4 Continuous Queries

Rather than obtaining a snapshot of the sensor field, most sensor applications are more interested in the data of the sensor field for a long period of time. Therefore, data need to be collected continuously. To handle it simply, continuous data collection can be achieved by a series of single data collection process. In this thesis, we obtain better results from different aspects in continuous data collection. A key technique (which makes a lot of intuitive sense) is to use previous/history information to assist future data evaluation.

1.4 Motivations and Research Challenges

In this thesis, we are interested in various aspects of data collection from a sensor field for a long period of time. From the architecture point of view, this requires network-wide collaboration. We consider efficient designs in the application layer, routing layer and topology control layer which concentrate on network level construction and optimization. From the technique point of view, we study several general methods that can be adopted

to control the system performance, e.g., load balancing between different sensors; using a subset of sensors if possible; and redundancy/load reduction whenever suitable. These techniques are applicable for different layers.

In this thesis, we first study the design of load balanced high quality sensor coverage. This provides the basis for efficient data collection for a long period of time. We then extend our focus on different aspects for efficient data collection based on application specifics. In this section, we outline some research challenges for both coverage and continuous data collection in sensor networks.

1.4.1 Sensor Coverage

In most studies on sensor coverage, only static sensors are used. The quality of coverage is noticeably affected by the initial deployment of the static sensors. For uneven sensor distributions, the sensors in a sparse area may have to stay active longer to ensure the coverage quality. The batteries of these sensors will be depleted earlier, making the area even sparser. In the extreme case, an area will become uncovered by any sensor, leaving a hole in the field. Unfortunately, such unfavorable sensor distributions are inevitable in many applications where a well-controlled or manual deployment is not practical.

Mobile sensors have the sensing capability as static sensors, but are able to move in a field, and their batteries are generally rechargeable. In other words, their lifetime is not bounded by the limited battery. While fully mobile sensor networks remain expensive and are complicated by information distribution between the mobile sensors, we envision that a hybrid network with both static and mobile nodes can be a cost-effective tool for coverage with unevenly distributed sensors. A related design was presented in [78], which suggested a one-time reposition of the mobile sensors after the initial deployment. This, however, does not fully utilize the movement capability of the mobile sensors.

For high quality and load balanced sensor coverage, we notice that several issues should be resolved. First, we need a better understanding of how the mobile sensors should be used; second, for a hybrid architecture, a clear division of the responsibilities between static and mobile sensors is needed; and third, the interaction between these two types of sensors should be defined.

1.4.2 Continuous Data Collection

Delay Sensitive Applications: Many sensor networks are redundant to compensate for the low reliability of the sensors and the environmental conditions. Since data from a sensor network is the aggregation of the data from individual sensors, the number of sensors in a network has a direct impact on the delay incurred in answering a query. In addition, significant delay is introduced by in-network aggregation [37][45][54], where intermediate nodes have to wait for the data values collected from their children before they can aggregate them with their own data.

A long delay is highly undesirable for time-sensitive applications such as critical condition monitoring and security surveillance [9]. As a result, there is increasing interest in research dealing with the delay problem [3][9][88][89].

In continuous data collection, the lifetime of the sensor network is long, i.e., the application is in favor of a large number of data collections. Each individual collection, however, may still be delay sensitive, e.g., for data collection in dangerous areas. We will illustrate more examples in the following sections.

Data Collection in Extreme Environments: Many recent studies have investigated data collection from harsh and extreme environments [20][80]. In these environments, the communications between sensors and the server (base station) can be expensive and scarce, and the data are collected occasionally. In each data collection, a fast data retrieval is usually desired [20]. Typical examples include the habitat monitoring system in Great Duck Island [55]; where some birds are notoriously sensitive to human intervention, and thus, data collection are done occasionally. In each collection, the presence of a human being should be minimized and, hopefully, far away from the habitat center to avoid direct impact on the birds. Applications of monitoring systems in chemical plants also share similar properties, where technicians occasionally approach the sensing area to collect data and each data collection should be performed quickly for safety purposes.

For these applications, the current popular tree based data collection and aggregation technique is not suitable. First, this technique can introduce a long delay in each data collection due to data searching and aggregation [45][76]. Second, this technique is beneficial if data can be aggregated so that the payload will be reduced in the intermediate nodes. If raw data are required, then the sensors close to the server will be burdened by uploading all data from the sensor network to the server. Third, in many situations, some part of

the sensor network may not be accessible due to failures. A better understanding of data collection, especially a continuous data collection, as well as the underlying routing support in these applications are greatly needed.

1.5 Related Work

Wireless sensor networks have received a lot of recent attention. A pioneer work discussing the challenges of sensor networks can be found in [21]. A general overview and a survey focusing on the routing protocols can be found in [2] and [3], respectively.

1.5.1 Coverage in Sensor Networks

In many sensor network applications, providing the desired field coverage or object protection is a key design objective. A typical coverage criterion is that every point of the field should be k-covered, which is studied in [72]. The k-coverage problem is further examined in [47], which proposes a sleeping/active schedule to minimize energy consumption. In [46], barrier coverage is considered, where the sensor networks can be used as barriers of, say, international borders. The problem is formulated as a k-multi-path problem and solved optimally if the sensors are centrally controlled. Distributed algorithms are also discussed in their work. Coverage of individual objects is studied in [13], which shows that the problem is NP-complete and heuristics are developed. Other related work include target tracing for mobile objects [90] and variable-quality of coverage [28]. Besides these theoretical studies, practical surveillance systems are also under active development; see for examples [29][86].

A closely related and yet orthogonal research direction is to find breach paths in a sensor protected area. A representative example is the *maximal breach path* [58]. Intuitively, the maximal breach path is a path traveling through the sensor network that has the least probability of being detected. The weight of maximal breach path shows the coverage quality of the sensor area. It is followed by *minimal and maximal exposure paths* [57][75] that focus on the paths with the least and most expected coverage.

In addition to coverage quality, network connectivity is also an important factor for successful operation of a multi-hop sensor network. The relation between coverage and connectivity is studied in [78], which suggests that if the transmission range of a sensor is twice of the sensing range, then the sensor network is connected if the area is covered in a convex region. Additional work in this direction can be found in [68][93].

Many studies propose grouping the sensors into grids [28][84][85], where all sensors in a grid are equivalent in their functionality, such as coverage capability. The surveillance systems in [28][86] further suggest that the sensors can be redundantly deployed and work in turn to extend the lifetime of the system.

Mobile sensors are recently used to assist coverage quality. In [50] the coverage is evaluated as the fraction of the covered area at a time instance. The authors conclude that, compared to using uniformly distributed static sensors, it is more beneficial if all sensors are mobile and are traveling in a random walk fashion.

A hybrid sensor network consisting of both static and mobile sensors is presented in [78], which compensates poor initial sensor distributions by strategically repositioning some mobile sensors. Similar work of the one-time reposition schemes can be found in [35][36][92].

These studies have given very solid understanding of high quality sensor coverage and provided ground for our study.

1.5.2 Data Routing and Aggregation

Data routing in sensor networks can be classified as flat routing and hierarchical routing. In flat routing, SPIN [31] is the first data centric protocol which uses flooding; directed diffusion [38] is proposed to select more efficient paths. Several variations and related protocols with similar concepts can be found in [10][17][66].

As an alternative, hierarchical routing has also been proposed for sensor networks. In LEACH [30], heads are selected for clusters of sensors; they periodically obtain data from their clusters. When a query is received, a head reports its most recent data value. An enhancement over LEACH can be found in [49]. In [88], energy is focused in a more refined way where a secondary parameter such as node proximity or node degree is included. Clustering techniques are studied in a different fashion in several papers, where [44] focuses on non-homogeneously dispersed nodes and [5] considers spanning tree structures.

In-network data aggregation is a widely used technique in sensor networks. Studies of MAX, MIN, AVERAGE, SUM can be found in [54][53][87]. Ordered properties such as QUANTILE are studied in [27]. A recent result in [12] considers power-aware routing and aggregation query processing together, building energy-efficient routing trees explicitly for aggregation queries.

Delay issues in sensor networks are mentioned in [45][54] where aggregation introduces high delay since each intermediate node and the source have to wait for the data values

from the leaves of the tree, as confirmed by [89]. In [37], where a modified direct diffusion is proposed, a timer is set up for intermediate nodes to flush data back to the source if the data from their children have not been received within a time threshold. In case of energy-delay tradeoffs, [89] formulates delay-constraint trees. A new protocol is proposed in [9] for delay critical applications where energy consumption is of secondary importance. In these algorithms, all of the sensors in the network are queried, resulting in $\Theta(N)$ processing time, where N denotes the number of sensors in the network, and incurs long delay. Embedding hierarchical architectures into the network where a small set of "head" sensors collect data periodically from their children/clusters and submit the results when queried [30][49][88] provides a very useful abstraction, where the length of the period is crucial for the tradeoff between the freshness of the data and the overhead.

The aggregation scheme works well if the data can be managed in the intermediate sensors to reduce the overall payload [30][38]. In some applications, one is interested in collecting the up-to-date raw data from the sensor network. These applications call for different solutions.

1.5.3 Network Coding

Many sensor networks are closely related to the *delay tolerant network* (DTN) or extreme network architecture [14][22][39]. A typical example is the ZebraNet in Africa [40], where researchers have to travel to the sensor network in person to collect data. Other recent examples can be found in [20][32][79][80]. One important feature of these networks is that each node needs to store data temporarily and submits data when the server approaches. These applications are failure prone and call for data redundancy and error control.

Coding is a powerful tool for redundancy management and error correction. It has been used in a large number of areas such as randomized data storage and packet transmission. Different kinds of coding techniques are also applied in sensor networks.

A typical coding scheme is *erasure codes* [8][48], in which a centralized server gathers all N data segments and builds C coded segments, $C \geq N$. If any N out of C coded segments are collected, the original data segments can be decoded [23][51]. A practical investigation of these codes can be found in [64]. These centralized operations are sometimes not suitable for application environment that involves a large quantity of tiny sensors. An alternative is network coding [1][91], which distributes the encoding operations to multiple nodes.

Network coding is first introduced in [1] to improve multicast throughput. As opposed

to erasure codes, where only the source can perform coding operation to the data packets, network coding allows each node in the network to combine data packets and construct codes. To maximize the benefit of network coding, linear network codes are constructed carefully such that the codes at each destination are decodable. Randomized network coding is introduced in [33], which adopts randomly generated coefficient vectors, and makes the calculation of linear network codes decentralized.

There are numerous recent studies applying conventional network coding and/or random linear coding in practical systems. Examples include network diagnosis [82], router buffer management [7], energy improvement in wireless networks [83], data gossiping [18], and in peer-to-peer networks [26].

Recently, network coding and its related extensions have been introduced in wireless sensor networks for ubiquitous data collection [20][80]. In these studies, the data segments to be collected are static and fixed. For continuous data collection, the sensor needs to remove obsolete data from a limited buffer to accommodate new ones. This is challenging if the data segments are coded together.

While many of the studies have encountered the problem of continuous data management, e.g., in [7][16][80][82], their common solution is to cut the data flow in *generations*, i.e., time periods, and combine all the original data segments in one generation. The length of a generation depends on the application and the choice is often experience based.

1.6 Contributions of this Thesis

The primary contributions of this thesis are listed as follows:

- We study a load balanced sensor coverage with a hybrid network consisting of both static and mobile sensors. Compared to previous studies, we fully utilize the movement capability of the mobile sensors. We design a protocol which optimally calculates the coverage contributions from the two types of sensors. We then propose the mobility model of the mobile sensors with random walk. Our experiment results show that our new hybrid network can significantly improve the lifetime of the sensor network with a small set of mobile sensors.

- We propose a layer architecture for delay sensitive applications in sensor networks. We trade-off delay with accuracy and obtain approximate queries with provable accuracy

guarantees. We show how to use history information to further reduce the delay for a series of queries. In addition, we optimize the structure of our system so that the energy consumption can be evenly distributed among each sensor.

- We develop partial network coding (PNC) for continuous data collection in an extreme network environment. PNC generalizes the existing network coding (NC) paradigm, an elegant solution for ubiquitous data distribution and collection. Yet, PNC allows efficient storage replacement for continuous data, where the conventional NC is not able achieve. We prove that the performance of PNC is quite close to NC, except for a sublinear overhead on storage and communications. We also address a set of practical concerns toward PNC-based continuous sensor data collection.

Chapter 2

Coverage in Sensor Networks

For a field with unevenly distributed static sensors, a quality coverage with acceptable network lifetime is often difficult to achieve. Recent advances in sensor technology have made it possible to deploy mobile sensors in the field. Exploring this possibility, a number of researchers have suggested a one time repositioning of the sensors after the initial deployment as a partial solution to the coverage problem. This solution, however, proves inadequate for balancing the sensor area and load in many applications. In this chapter we propose a hybrid sensor network with both static and mobile sensors, and fully exploit the movement capability of the mobile sensors. In our solution, the mobile sensors are always in motion to assist the static sensors; the occurrence probability of the mobile sensors in each grid, or their contribution for covering the grid, is adaptively determined according to the network configuration. From a statistical point of view, the overall coverage is enhanced, and energy consumption of the static sensors is more balanced.

We show the motivation of our idea in an example. Consider Fig. 2.1, where there are a number of static sensors and three mobile sensors to cover a field. Each sensor can cover its associated grid. If there are no mobile sensors, grid 6 will never be covered. If only one-time repositioning for the mobile sensors is employed, the coverage can be enhanced, but there will still remain grids with permanently fewer sensors than others.

The main challenges in designing such a hybrid network are; first, to clarify the necessary coverage contributions from the static and mobile sensors; and second, to find a mobility model for the mobile sensors to achieve the desired coverage contribution. In this chapter, we for the first time offer an analytical study on the above problems, with the results leading to a practical system design. Our model is general enough to match the moving capability

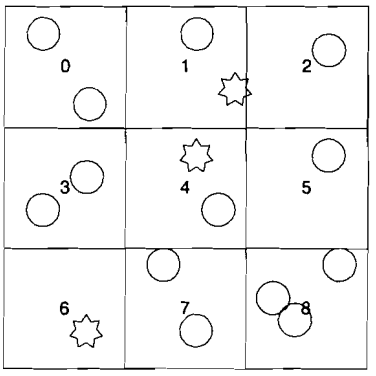


Figure 2.1: Field covered by a hybrid static and mobile sensor network, circles representing static sensors and stars representing mobile sensors.

of different mobile sensors and the demands from diverse applications.

2.1 Architecture Overview

2.1.1 Hybrid Network Model

The hybrid network in our study consists of both static and mobile sensor nodes, which collectively monitor a field of interest. As in previous studies [24][43][85], we assume that the field is divided into n^2 virtual grids, indexed from 0 to $n^2 - 1$[1]. This virtual grid structure is not special, and we will show in Section 2.6 that our analysis and algorithms can be easily extended to hexagon or other virtual structures. Through GPS or available positioning services [4][11], the sensors are aware of their location information and, hence, their associated grids. The size of each grid is $\frac{\sqrt{2}}{2}R \times \frac{\sqrt{2}}{2}R$, where R is the sensing range of a static sensor. As such, any active sensor in a grid can cover the whole grid. The sensing range of a mobile sensor can be smaller, e.g., $\frac{R}{2}$, as it can reposition itself to the center of its grid. An example of the grid structure is shown in Fig. 2.1.

Given that the static sensors in one grid are equivalent in coverage, they do not have to be active simultaneously, so as to save energy. Unfortunately, the deployment of the static sensors is often nonuniform; and even worse, holes (grids with no static sensors) can exist,

[1]In this paper, we use the grids to denote a grid of n^2 cells.

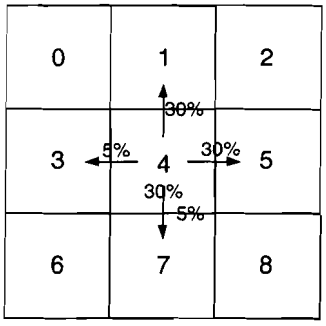


Figure 2.2: The movement of a mobile sensor. The probabilities for moving to or staying in a grid are determined according to the network configuration.

creating permanently uncovered regions[2]. The situation is very common when the sensors are distributed automatically through aircrafts or vehicles in complex terrains.

Our hybrid network addresses this problem by allowing assistance from the mobile sensors. The mobile sensors are always active, and can stay in a grid or move to neighboring grids, as shown in Fig. 2.2. This feature can therefore help with the covering of the holes in the field and reducing the load of the existing static sensors.

2.1.2 Performance Measurements

Since our main goal is covering related, we define a measure of how well a location is covered. Similar measurement is also used in [84].

Definition 2.1.1 *A sensor field is said to be δ-covered if, at any point in time, at least an expected $\delta \in (0, 1)$ fraction of the whole area is covered by one or more sensors.*

Assume that δ is the minimum coverage ratio required by the user, our objective is to ensure this quality, while maximizing the lifetime of the network.

It is worth noting that the battery of state-of-the-art mobile sensors is rechargeable [41]; hence, the lifetime of the whole network is bounded by that of the static sensors. We use the lifetime of the first dying out sensor as a measure for the system lifetime. This definition has been widely used in existing studies [15][88], and essentially suggests a load-balanced

[2]Even if the deployment is a globally uniform distribution, local fluctuations still would occur, resulting in uneven numbers of sensors in different grids.

operation for the static sensors. The effectiveness of this definition has been validated by our simulation results in Section 2.5. From a functional point of view, once the first static sensor dies, its grid needs additional assistance from the mobile/static sensors, which in turn increases the workload of other static sensors, resulting in a domino effect that quickly drains the power of the whole network. Thus, the death of the first sensor serves as a good signal to the end of the steady-state operation.

In summary, given a coverage requirement, the network lifetime depends on the activation models of the static sensors, which further depend on the sensor distribution and the potential contributions from the mobile sensors.

2.1.3 Working and Moving Models

Given the system model and the performance measures, a natural question is what kind of working and moving models of the sensors can achieve the coverage objective. In our basic framework, we adopt a random activation scheduling for the static sensors, and a random walk model for the mobile sensors. More specifically, our hybrid sensor network goes through the following stages:

1) *Parameter Initialization:* After deployment, one or more mobile sensors travel around the field and collect the distribution information of the static senors in all grids. The mobile sensors determine the movements of themselves as well as the activation probability of the static sensors. The mobile sensors then notify the static sensors of their activation probability.

2) *Field Monitoring:* Assume the time slots are discrete. In each time slot, a static sensor independently activates itself with the activation probability obtained in the initialization stage and then monitors its grid. Each mobile sensor independently decides to move into one neighboring grid or to stay in the current grid, and then monitors the grid where it resides.

The advantages of using a probabilistic operation over a deterministic one are many. First, our technique is easier to implement because it involves simple optimization in the initial stage for the sensors. Second, the behavior of each type of the sensors are statistically identical. This is useful especially for recharging or replacement of mobile sensors. The substitute mobile sensor can easily follow the mobility model and continue to monitor the sensor field, regardless of the current state of other sensors; whereas a deterministic scheme may involve re-optimization. Third, a probabilistic coverage is generally more resistent to

Notation	Definition
n	Grid dimension
N	Total number of grids, i.e., n^2
p	Activation probability for static sensors
R	Sensing range of a static sensor
δ	Required coverage ratio for the sensor field
$d(i)$	Density of the static sensors for grid i
l_i	The index of grid with density rank i
M	Number of mobile sensors
P_{ij}	Probability that a mobile sensor moves from grid i to j
π_j	Coverage ratio by a mobile sensor for grid j
π	Vector of π_i
m_i	Mobile sensor i
s_i	Static sensor i

Table 2.1: List of Notations for Chapter 2.

intruders that try to learn the sensor behavior.

Our hybrid architecture offers achievable and reasonably good solutions to the problem of the uneven distribution of static sensors. It is, however, worth emphasizing that the above framework provides only a flexible baseline for further design of hybrid systems. Many practical enhancements could be added to this basic framework, and we will discuss some of them as well.

For ease of exposition, we list the notations used throughout this chapter in Table 2.1.

2.2 Coverage Contributions from Static and Mobile Sensors

In our hybrid network, the coverage of a grid is achieved by the combined efforts of static and mobile sensors. A grid is said to be covered at time t if either a static sensor in this grid is active or a mobile sensor resides in the grid at time t. To balance the workload, it is desirable to assign the static sensors with an identical activation probability p. An illustrative example of coverage is shown in Fig. 2.3 (refer to Fig. 2.1 for the distribution of the sensors for this example).

We now identify the necessary long-term coverage contributions from the two types of sensors. Clearly, for grid i, $i = 0, 1, \ldots, n^2 - 1$, the contribution from a mobile sensor depends on the fraction of time that the mobile sensor will be present in this grid; in other words, the probability that it travels to the grid. We denote this probability by π_i. The

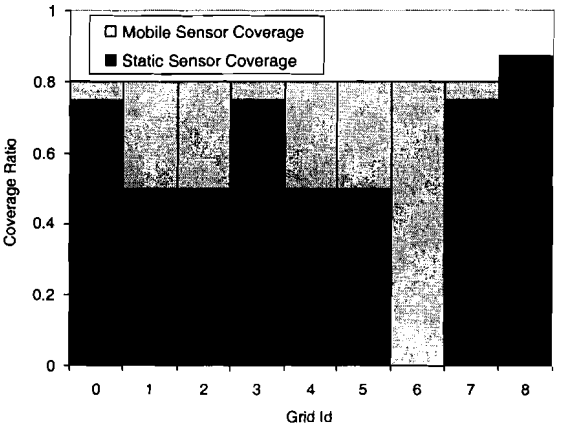


Figure 2.3: Coverage contributions from static and mobile sensors. Coverage requirement is $\delta = 0.8$, and activation probability of static sensors is $p = 0.5$.

contribution from a static sensor in the grid is equal to its activation probability: the higher this probability, the better the coverage will be.

We now focus on the optimal values of p and $\pi = [\pi_0, \pi_1, \ldots, \pi_{n^2-1}]$. In the next section, we will present a random walk model that achieves π.

To facilitate our discussion, we use $d(i)$ to represent the density of grid i, i.e., the number of static sensors in this grid. Let M be the number of mobile sensors in the network. Given coverage requirement δ, the following formulation maximizes the network lifetime:

$$\text{minimize} \quad p$$

$$s.t. \quad \pi_0 + \pi_1 + \ldots + \pi_{n^2-1} \leq 1 \tag{2.1}$$

$$(1-p)^{d(0)} \times (1-\pi_0)^M \leq 1 - \delta \tag{2.2}$$

$$(1-p)^{d(1)} \times (1-\pi_1)^M \leq 1 - \delta \tag{2.3}$$

$$\vdots$$

$$(1-p)^{d(n^2-1)} \times (1-\pi_{n^2-1})^M \leq 1-\delta \tag{2.4}$$

where Equation. (2.1) gives the contribution constraint of each mobile sensor, and Equations. (2.2) - (2.4) ensure the coverage ratio of all the grids.

```
Algorithm CalcContribution()
1  SortGrid();
2  for (K = 0; K < n^2; K++)
     /* (1-p)^{d(l_K)} <= 1-δ */
3    p = 1 - sqrt[d(l_K)](1-δ);
4    for (i = 0; i < K; i++)
       /* (1-p)^{d(l_i)} x (1-π_{l_i})^M <= 1-δ */
5      π_{l_i} = 1 - sqrt[M]((1-δ)/(1-p)^{d(l_i)});
6    if (Σ_{i=0}^{n^2-1} π_{l_i} > 1)
7      break;
8  AdaptP();
```

Figure 2.4: Algorithm CalcContribution()

We present algorithm CalcContribution() that solves this optimization problem (see Fig. 2.4). In CalcContribution(), we first invoke subroutine SortGrid() to sort the grids in ascending order of their densities. Let l_i represent the index of the grid with rank i after sorting, i.e., $d(l_0) \leq d(l_1) \leq \ldots \leq d(l_{n^2-1})$. We then search for $\mathcal{K}$, the rank after which the grids are dense enough to be covered by the static sensors only. We start searching for $\mathcal{K}$ from 0, and evaluate the p for the current setting of $\mathcal{K}$. If we can find a valid p and π_{l_i}, then we increase $\mathcal{K}$, until $\sum_{i=0}^{n^2-1} \pi_{l_i} > 1$ (intuitively, this says that the potential of the mobile sensors is fully exploited) or $\mathcal{K}$ reaches n^2. In this process, p is decreasing because additional assistance from the mobile sensors is introduced after each iteration.

Note that p is a real number but $\mathcal{K}$ is discrete. Hence, after the above process terminates, we in fact have an upper-bound on p corresponding to $\mathcal{K}-1$, and a lower-bound on p corresponding $\mathcal{K}$. To find the optimal and practical p, we invoke a subroutine AdaptP(), which performs a binary search for the p and adjusts π_{l_i} accordingly. The termination of this subroutine depends on the precision of p, which is usually a predefined value. In our experiments, the depth of the binary search is always smaller than a constant factor of four.

The complexity of this algorithm is N^2 where N represents the total number of grids;

and it does not depend on the number of sensors. In practice, if the field is very large and there are too many grids, it may take a long time for a single mobile sensor to collect all the field information. In this case, we can first do a simple uniform partition of the field according to the number of mobile sensors and let each mobile sensor be responsible for the information collection in a subfield. As such, the initialization phase can be remarkably shortened.

2.3 A Random Walk Model for Mobile Sensors

In the previous section, we obtained π, the long-term coverage contribution by the mobile sensors to the grids. It remains to show a concrete mobility model that can achieve this distribution. To this end, we demonstrate a viable and yet simple random walk model in this section.

2.3.1 Random Walk Model

In the random walk model, a mobile sensor will either stay in a grid, or move into an adjacent grid along four directions,[3] as shown in Fig. 2.2. We consider decisions depending only on the current grid where a mobile sensor resides. This results in a Markov chain where each grid is a state. We use P_{ij} to denote the transition probability from grid i to grid j. See Fig. 2.5 for an illustration. Given the long-run distribution π, this Markov chain obeys the following balance equations,

$$\pi_j = \sum_{k=0}^{n^2-1} \pi_k P_{kj}, \quad j = 0, 1, \ldots, n^2 - 1 \tag{2.5}$$

$$\sum_{k=0}^{n^2-1} \pi_k = 1 \tag{2.6}$$

$$\sum_{j=0}^{n^2-1} P_{kj} = 1, \quad \forall k \in [0, n^2 - 1] \tag{2.7}$$

$$0 \leq P_{ij} \leq 1, \quad \forall i, j \tag{2.8}$$

[3]For a mobile sensor in a boundary grid, it might have 3 or 2 directions to move only.

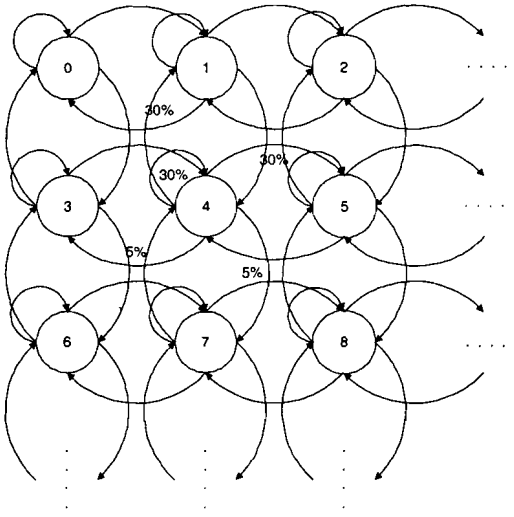


Figure 2.5: Markov chain for the random walk model.

$$P_{ij} = 0, \quad \forall i, j, \text{ grids } i, j \text{ not adjacent} \tag{2.9}$$

where the first four equations are standard steady-state constraints for Markov chains [42], and Equation (2.9) suggests that no transition is possible for two non-adjacent grids.

Our problem now is to determine the transition probabilities P_{ij} in this system of equations to reach the stationary distribution π. This is the inverse of the traditional "given transition probability, find stationary distribution" problem in a Markov chain.

First of all, we need to ensure that the P_{ij} obtained can guarantee a limiting distribution π. By ergodic theorem [65], a Markov chain that is *aperiodic*, *irreducible* and *positive recurrent* has a limiting distribution[4]. Since there are only a finite number of states in our system, if our Markov chain is irreducible, it is positive recurrent. As such, if we ensure that the Markov chain is aperiodic and irreducible, it is sufficient to guarantee this π exists. For ease of discussion, we now assume that $\pi_k > 0$ for $k = 0, 1, \ldots, n^2 - 1$. We will generalize the solution later.

To ensure aperiodicity, we can set all the P_{ii} to be strictly positive. To ensure irreducibility, the mobile sensors cannot be trapped in a grid or a group of grids; hence, we

[4]Aperiodic means that $P_{ii} > 0$. Irreducible means that all states are reachable from all other states. Positive recurrent means that the sensor will return to a state within finite time.

have an additional set of constraints:

$$\forall i, \quad 0 < P_{ii} < 1, \tag{2.10}$$

which indicates that whenever a mobile sensor moves into a grid, the probability that it will stay in this grid should be strictly less than 1. A stronger condition is

$$P_{ij} > 0, \quad \forall i, j, \text{ grids } i, j \text{ are adjacent}, \tag{2.11}$$

which ensures that the mobile sensor always has chance to move into a neighboring grid. Equation. (2.8) can then be replaced by

$$0 < P_{ij} < 1, \quad \forall i, j \text{ that are adjacent} \tag{2.12}$$

It is not difficult to see that the above set of equations have multiple solutions. We now illustrate one solution set. Our strategy is to first find a set of solution to Equation. (2.5) and Equation. (2.6) and then try to satisfy all others. Notice that if $\pi_k P_{kj} = \pi_j P_{jk}$, Equation. (2.5) can be satisfied. We set $P_{kj} = \pi_j$ and $P_{jk} = \pi_i$ for all $P_{jk} \neq 0$ and $P_{kj} \neq 0$. This can always be achieved because either P_{kj} and P_{jk} are both strictly positive, or $P_{kj} = P_{jk} = 0$. We then set $P_{ii} = 1 - \sum_{j=0}^{n^2-1} P_{ij}$, and it is easy to verify that $P_{ii} > 0$. Therefore, Equations. (2.5), (2.6) and (2.7), (2.9) are satisfied. Since $\pi_k, \pi_j \neq 0, 1$ we have $P_{jk}, P_{kj} \neq 0, 1$, and Equations. (2.10), (2.12) are satisfied.

In summary, the solution set is

$$P_{jk} = \begin{cases} \pi_k & \forall j \neq k \text{ and } j, k \text{ are adjacent}; \\ 0 & \forall j \neq k \text{ and } j, k \text{ are not adjacent}; \end{cases} \tag{2.13}$$

$$P_{jj} = 1 - \sum_{k=0}^{n^2-1} P_{jk} \quad \forall j \tag{2.14}$$

Here we emphasize again that we assume $\pi_k > 0$ for $k = 0, 1, \ldots, n^2 - 1$. In Section 2.3.3, we will investigate an interesting impact of $\pi_k = 0$, where certain grids do not need assistance from the mobile sensors.

2.3.2 Boosting Movement

It is worth noting that the definition of coverage quality (Definition 2.1.1 in Section 2.1.2) does not account for the moving frequency of the mobile sensors, nor the convergence time

of the system. A *lazy movement* thus would achieve the same coverage requirement. An extreme example is one-time repositioning of the mobile sensors: a higher fraction of the sensor field can be covered, but the coverage could still be unbalanced or even with holes if the number of mobile sensors is not enough.

Our random walk model can effectively solve this problem by adaptively setting the transition probabilities, allowing a wide range of movement frequencies. The strategy is to adjust the existing solution within the constraints to obtain another viable solution set. Specifically, to satisfy Equation. (2.5), we only need to have $\pi_k P_{kj} = \pi_j P_{jk}$; thus setting $P_{kj} = \alpha\pi_j$ and $P_{jk} = \alpha\pi_k$ also works given $\alpha > 0$. Let $\alpha_l, \alpha_u, \alpha_r, \alpha_d$ denote the adjustment factors for the four directions. To achieve a higher moving frequency, we can increase $\alpha_l, \alpha_u, \alpha_r, \alpha_d$, and the constraints will still be satisfied as long as the sum of the outgoing probabilities in a grid is less than 1. In our experiments, we set a threshold for P_{ii}: if a $P_{ii}$is greater than the threshold, we increase the α's until all P_{ii}'s are less than the threshold, or there is no possible further reduction. We call the movement scheme after adjustment *aggressive movement.*

2.3.3 The Wall Effect and Solutions

We have assumed that π_i is non-zero in the previous Markov chain calculation. In practice, π_i can be zero for dense grids, i.e., those ranked higher than $\mathcal{K}$ in algorithm CalcContribution(). These grids will not get assistance from the mobile sensors and can simply be ignored in forming the Markov chain, if they are sparsely distributed. However, if a collection of such grids are connected, a *wall* can be formed, which partitions the field into two or more disjoint subfields. Given the presence of a wall (or multiple walls), a mobile sensor can not move freely in the whole field, and the expected distribution is no longer achievable. An example of this *wall effect* is shown in Fig. 2.6 where grids 3, 6, 9, 13 have dense static sensors and thus form a wall, splitting the fields into two subfields. Grid 0 and 4 also have dense static sensors. Compared to the wall grids, they still need some assist from mobile sensors. We call them *semi-walls* as these grids make traveling in subfield (0, 1, 2, 4, 5, 8, 12) difficult, i.e., it may take a long time for the mobile sensors in grids 1, 2, 5 to reach grid 8, 12. As such, the coverage of the non-wall grids strongly depends on the initial placement of the mobile sensors, and a strategic allocation of the mobile sensors to the subfields is thus necessary.

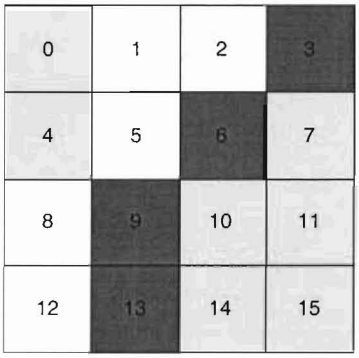


Figure 2.6: Wall effect. Darker grids have denser static sensors.

Mobile Sensor Allocation for Subfields: Assume that, after invoking algorithm Calc-Contribution() in the initial stage, the sensor field is divided into C subfields by walls. It is easy to see that the number of mobile sensors needed in each sub-field (excluding the wall grids) is independent of other subfields. We thus focus on a particular subfield, e.g., the kth one. Assume this subfield includes C^k grids, and similar to the notations used previously, let grid l_i^k be the ith rank in this subfield after sorting in ascending order of the densities, i.e., $d(l_0^k) \leq d(l_1^k) \leq \ldots \leq d(l_{C^k-1}^k)$. Let M^k be the number of mobile sensors to be assigned to this subfield. Our objective is to find the minimum M^k that provides the desired coverage for this subfield. This problem can be formulated as follows:

$$\text{minimize} \quad M^k$$

$$s.t. \quad \pi_{l_0^k} + \pi_{l_1^k} + \ldots + \pi_{l_{C^k-1}^k} \leq 1 \tag{2.15}$$

$$(1 - p_{min})^{d(l_0^k)} \times (1 - \pi_{l_0^k})^{M^k} \leq 1 - \delta \tag{2.16}$$

$$(1 - p_{min})^{d(l_1^k)} \times (1 - \pi_{l_1^k})^{M^k} \leq 1 - \delta \tag{2.17}$$

$$\vdots$$

$$(1 - p_{min})^{d(l_{C^k-1}^k)} \times (1 - \pi_{l_{C^k-1}^k})^{M^k} \leq 1 - \delta \tag{2.18}$$

where p_{min} is the optimal value of p obtained in CalcContribution. To maximize the expected network lifetime, this value should still be identical for all the static sensors, even in the presence of subfields.

We can iteratively reduce M^k starting from $M - \sum_{j=0}^{k-1} M^j$. We allocate mobile sensors to each subfield one by one and, for the kth subfield, we start with the remaining mobile sensors after assigning all $k-1$ subfields. We then calculate the the corresponding $\pi_{i^k_i}$ in each iteration. We stop until Equation. (2.15) is violated, (intuitively, this means that fewer sensors cannot provide necessary coverage). We thus obtain optimal M^k and $\pi_{i^k_i}$. Since the grids within the subfield all have $\pi_{i^k_i} > 0$, we can set the transition probabilities as before. The transition probabilities also guarantee that a mobile sensor will remain in its subfield during the random walk.

It is worth noting that after we calculate each M^k individually, it is possible that $\sum_{k=0}^{C} M^k > M$. This is because a sensor cannot be allocated fractionally. Given this negative impact of the walls, we need to increase p_{min} by decreasing $\mathcal{K}$; the contribution from the static sensors is thus increased. We continue until a $\mathcal{K}$ is found such that $\sum_{k=0}^{C} M^k \leq M$.

Subfield Partitioning: Besides the wall grids, other dense grids may have a very small π_i, implying that the mobile sensors should seldom visit them. Two examples are the grids 0 and 4 in Fig. 2.6. These two grids make a smooth walking in subfield (0, 1, 2, 4, 5, 8, 12) difficult and will significantly increase the convergence time of the system.

In the presence of semi-walls, we can further partition the subfields to balance the movement of the mobile sensors. Again, since the mobile sensors cannot be allocated fractionally, we have to strike a balance between the coverage and convergency. In our experiment, we set a threshold for the grids of semi-walls and show that the convergence time improves noticeably.

2.4 Sensor Collaborations

So far we have established the respective contributions from static and mobile sensors, and the activation and movement strategies for them. This framework is easy to implement as it involves node interactions in the initial period only, and all the remaining operations are randomly and independently performed in a distributed fashion. Within this basic framework, various node interactions/collaborations could be introduced to further enhance the baseline performance. To show this, we now outline a simple yet effective node collaboration

Time slots $[0, T_1]$, $[T_1, T_2]$, ..., $[T_i, T_{i+1}]$, ...
Slot length $T(\gg t)$; contention interval $[T_i - t, T_i + t]$;

Mobile sensor m_j

m_j **at time** $T_i - t$
m_j randomly picks a number $t_j \in [0, t]$.
determines next grid l to move in (could be unchanged)

m_j **at time** $T_i - t_j$
broadcasts Probe Message to sensors in grid l

m_j **upon receiving Probe Message from** m_k **in** $[T_i - t, T_i]$
if m_j will stay in l in $[T_{i+1}, T_{i+2}]$,
replies with Reject Message
else if m_j received probe messages from other $m_{k'}$,
replies with Reject Message
else records node m_k (as $m_{k'}$ in the next round)

m_j **at time** T_i
if no reject message received, enters grid l

m_j **upon receiving Probe Message from** s_k **in** $[T_i, T_i + t]$
replies with Reject Message

Static sensor s_j **in grid** l

s_j **upon receiving Probe Message from** m_k **in** $[T_i - t, T_i]$
if s_j received other probe messages,
replies with Reject Message
else records node m_k

s_j **at time** T_i
s_j randomly picks a number $t_j \in [0, t]$
determines activation with p

s_j **at time** $T_i + t_j$
if succeeded in activation,
broadcasts Probe Message to sensors in grid l
if no reject message, turns active

s_j **upon receiving Probe Message from** s_k **in** $[T_i, T_i + t]$
replies with Reject Message

Figure 2.7: Node Collaboration Protocol.

scheme.

The key idea here is to prevent overlapping coverage of a grid by multiple sensors. To this end, we introduce a sensor collaboration protocol with two contention phases (The pseudo-code for the collaboration scheme can be found in Fig. 2.7.). Without loss of generality, we consider a time slot starting at T_i of length T. The first phase $[T_i - t, T_i]$ is used for contention between mobile sensors to enter one certain grid; the second phase $[T_i, T_i + t]$ is used for suppressing multiple activation of the static sensors. Here, t is a fixed parameter such that $t \ll T$.

In $[T_i - t, T_i]$, mobile sensor m_j first decides which grid it will enter in the next time slot. Then, m_j randomly generates a number $t_j \in [0, t]$ and, at time $T_i - t_j$, sends a probe message to the sensors in the selected grid. If the grid has a mobile sensor or an active static sensor, it will allow m_j to enter in the next slot only if m_j is the first one sending the probe message. In $[T_i, T_i + t]$, each static sensor also generates $t_j \in [0, t]$, and, at time $T_i + t_j$, activates itself with probability p and broadcasts a probe message to its neighbors in the same grid. If a neighbor is a mobile or an already activated static sensor, it will reply by a reject message; The newly activated sensor thus has to deactivate itself to save energy.

It is easy to show that the power consumption and coverage quality with this sensor collaboration protocol are no worse than that of the basic framework.

2.5 Performance Evaluation

In this section, we evaluate the performance of the hybrid sensor network in field coverage through simulations. We developed an event driven simulator and focus on the following typical measures: coverage quality, network lifetime, and convergence time.

In our simulation, deployed 1000 static sensors in a field of 140m $\times$ 140m and the sensor field was partitioned into 100 virtual grids. The battery power for each sensor was 10000mAh, and can last for one day with persistent activation. We neglected the energy cost during dormant states.

We have examined the energy consumption status of the static sensors in our system. Fig. 2.8 shows the cumulative distribution curve of the residual energy after the death of the first sensor. We can see that at this time more than 70% of the sensors has residual energy less than 1000mAh (10% of the total energy reserve). It implies that the remaining operation time of the system is very limited, and the lifetime of the first dead sensor thus

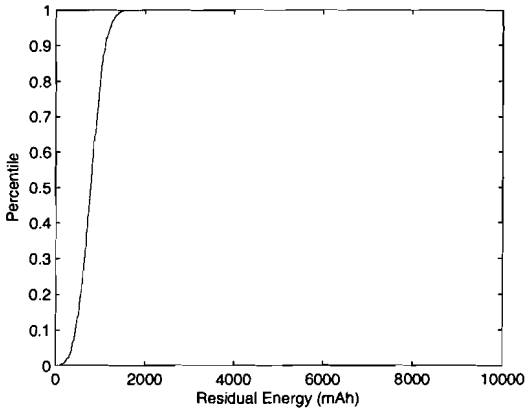


Figure 2.8: Residual energy after the death of the first sensor.

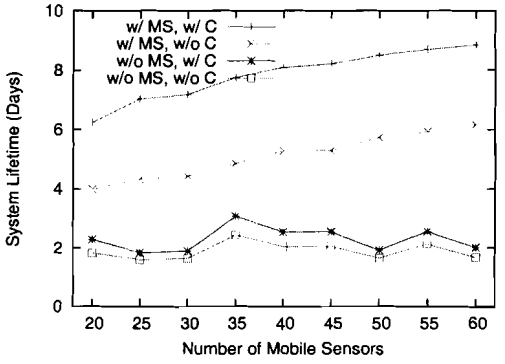


Figure 2.9: System lifetime as a function of additional sensors.

serves as a legible measure for the system lifetime.

Unless otherwise specified, the following default parameters were used in our simulation: The expected coverage quality was $\delta = 0.85$, and the length of each time slot was 1 minutes. Each point in our figures was the average of 100 independent experiments.

2.5.1 Contribution of Mobile Sensors

In first set of experiments, we deployed different number of mobile sensors in the field to observe their effectiveness. In Fig. 2.9, we show the network lifetime as a function of the number of mobile sensors. The number of mobile sensors varies from 20 to 60, which

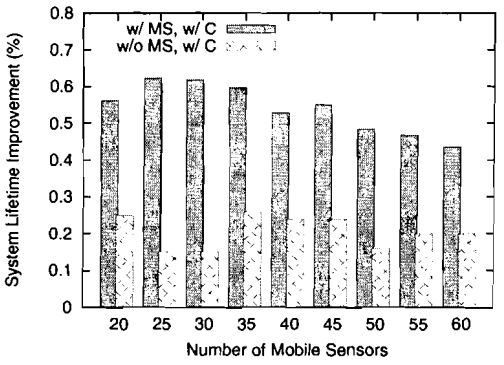


Figure 2.10: System lifetime improvement with or without collaborations.

accounts for only a small portion of all the sensors. For comparison, we also plot the result with static sensors only; to ensure fairness, in this case, we deployed additional static sensors (the same amount as mobile sensors), which are equipped with extra-batteries to remain active throughout the experiments. In our figures, we use w/ MS, w/o MS to denote the experiments with or without mobile sensors; w/ C, w/o C to denote the experiments with or without using the sensor collaboration protocol.

We observe that the use of mobile sensors substantially increases the network lifetime. For example, consider the case where there are 50 mobile sensors, the lifetime (w/ MS, w/o C) is three times longer than without mobile sensors (w/o MS, w/o C). In addition, we see that the lifetime improves steadily when more mobile sensors are deployed. On the contrary, by adding a few static sensors only, there is no clear improvement of the system lifetime. Node collaboration also improves the life time for both cases, but more substantially if mobile sensors are used. The improvement percentage is plotted in Fig. 2.10. We can see that without mobile sensor (w/o MS, w/ C), there is a 10% to 20% lifetime improvement with sensor collaboration compared to without collaboration. If mobile sensors are used, this effect is much pronounced. This is because without mobile sensors, the lifetime is constrained by the grids with fewer sensors, resulting in smaller chance of suppressing redundant activations. Since node collaboration substantially improves the system performance, for the rest of our experiments, we will focus on the performance of the system with collaboration only.

We next consider the effect of two different distributions of the static sensors. First, we deployed the static sensors randomly and uniformly. Second, we added some bias on the

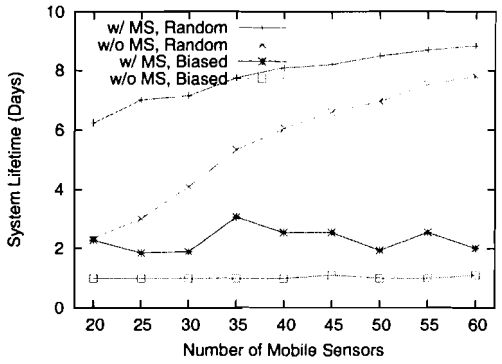


Figure 2.11: System lifetime for uniform and biased distributions of static sensors.

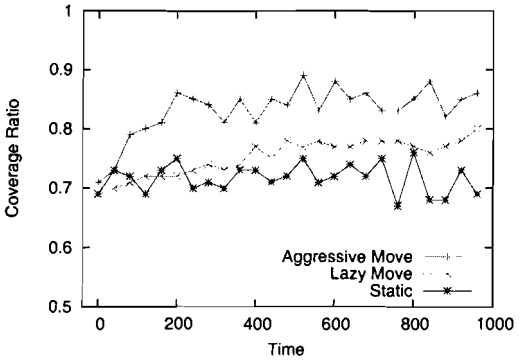


Figure 2.12: Coverage ratio as a function of running time for varying movement patterns.

distribution, where the right side of the sensor field was two times denser than the left side of the sensor field. Fig. 2.11 shows the comparison results. Not surprisingly, the lifetime has reduced in biased distribution since the system is more stressed. With assistance from mobile sensors, however, the situation improves fast; for example, with 20 mobile sensors, the lifetime is only marginally better than with no mobile sensors at all, whereas with 60 mobile sensors, the lifetime is less significantly affected by the bias of the distribution. This clearly shows the inherent adjustment capability of using mobile sensors.

2.5.2 Convergence Time

We now consider the convergence time of the network, in particular, the effect of moving speed of the mobile sensors. We simulated 50 mobile sensors and 1000 static sensors in the

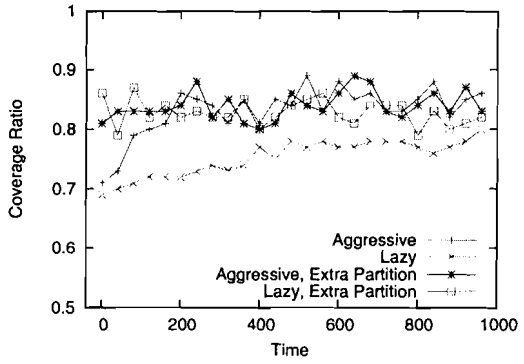


Figure 2.13: Coverage ratio as functions of running time with partitioning.

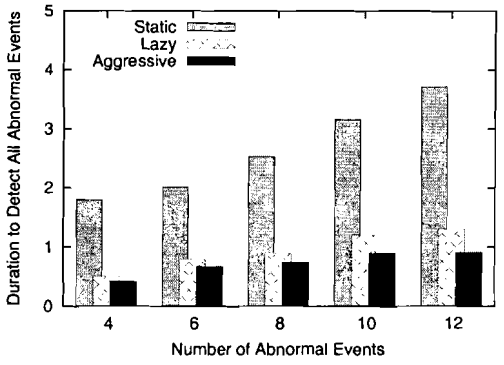
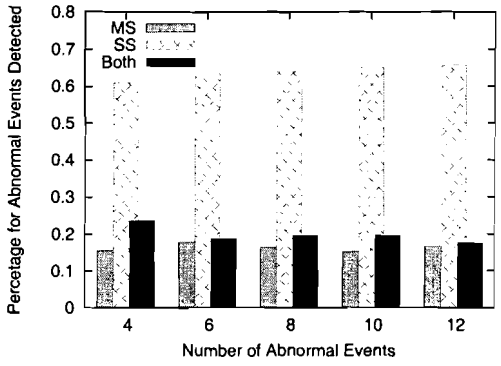


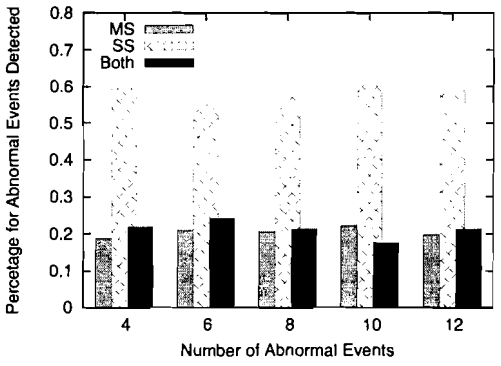
Figure 2.14: Duration to detect all abnormal events.

sensor field. In initialization, the whole sensor field was partitioned into subfields by walls. All mobile sensors belonging to the same subfield were dispatched to the grid with the highest index in this subfield. Fig. 2.12 shows the coverage quality over time for both aggressive and lazy movements. We see that if there are high transition probabilities between adjacent grids, the convergence time is much smaller. For example, with aggressive movement, the system reaches 85% coverage after 200 minutes, while lazy movement has yet to reach this ratio after 1000 minutes. We can also see from Fig. 2.12, that the coverage ratio with static sensors only is only around 70%.

We consider the effect of finer partitioning of the subfields. From Fig. 2.13, we see that finer partition improves the convergence time with both aggressive and lazy movements.



(a) Mobile sensors with lazy movement.



(b) Mobile sensors with aggressive movement.

Figure 2.15: Abnormal event detection. SS: Detected by static sensors only; MS: Detected by mobile sensors only; Both: Detected by both.

These experiments clearly show that the walls and semi-walls in the field would remarkably affect the convergence of the system, and our allocation algorithms for the mobile sensors can effectively solve this problem.

2.5.3 Aggressive Movement in Event Detection

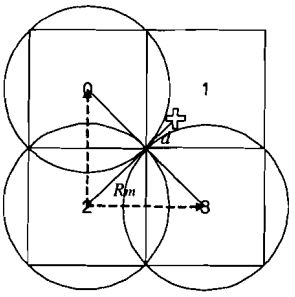
While finer partitioning makes the the convergence time of lazy movement close to that of aggressive movement, we argue that aggressive movement can be much more effective than lazy movement in abnormal event detection.

We randomly generated abnormal events in the sensor field. In Fig. 2.14, we show the time needed to detect all these events for three strategies, namely, aggressive movement, lazy movement and without mobile sensors. Not surprisingly, the more abnormal events there are, the longer it takes to find all of them. We see that with aggressive movement, the detection time is not only shorter than the other two, but also increases more slowly when the number of abnormal events increases. The gain obtained from aggressive movement compared to lazy movement is around 5% to 15%. Notice that this is achieved neither by increasing the number of the mobile sensors nor by increasing their physical speeds, but simply by improving the transition probabilities between the grids. Finally, note that the detection time of using static sensors only is remarkably longer than the other two. In fact, in some tests, the events can never be fully detected if the grids has no any static sensor; we set an expiration time of 20 in such cases, which explains the high average detection time.

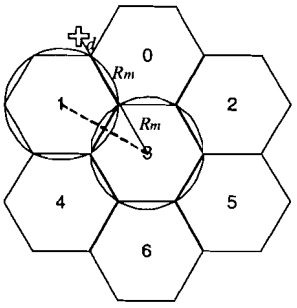
To further understand the contributions from static and mobile sensors, we show in Fig. 2.15 the ratio of the abnormal events detected by different types of sensors, namely, static, mobile, or both. We see that the static sensors are still the main source in coverage, detecting 55% to 60% of the abnormal events alone. The mobile sensors detect around 20% and for the other 20% cases, static and mobile sensors observe the abnormal events simultaneously. Fig. 2.15 (a) and (b) demonstrate the scenario where the mobile sensors adopt lazy movement and aggressive movement strategies. We can also see that, if aggressive movement is adopted, the mobile sensors become more effective in detecting abnormal events.

2.6 Generalizing Grid Structure

We have assumed a square grid structure for the field in our study, which has also been widely adopted in this research area. A limitation of the square grid structure is its inflexible



(a) Square grid structure.



(b) Hexagon structure.

Figure 2.16: Different underlying structure.

moving directions. We show an example in Fig. 2.16. In Fig. 2.16, abnormal event is shown as a cross. Consider an abnormal event is $R_m + d$ away from a mobile sensor, where R_m is the sensing range of a mobile sensor and d is a small distance. If the event happens in the upper-right direction (see Fig. 2.16 (a)), then because the two grids are not adjacent, the mobile sensor will detect it after moving at least twice. A hexagon structure, like that in the cellular network, will perform better in this case. As shown in Fig. 2.16 (b), only if the abnormal event is more than $2R_m$ away from the mobile sensor can it avoid being possibly detected in the next sensor movement.

Our hybrid network and mobility models are not restricted to the square grid structure. They can easily accommodate the hexagon or even more general polygon structures. In fact, algorithm CalcContribution() does not depend on the grid structure. Only some transitions are to be added in the Markov chain, e.g., for the hexagon structure, six transitions are needed against the four for the square grid case, and other calculations remain unchanged.

2.7 Conclusion

In this chapter, we proposed a hybrid sensor network model with static and mobile sensors. We offered an optimal algorithm for calculating the coverage contributions from each type of sensors to maximize the network lifetime. We further showed how the contributions can be achieved, as well as enhancements based on collaborations among sensors. The results from our simulations demonstrate that a small set of mobile sensors can noticeably improve the coverage quality and extend the network lifetime.

Chapter 3

Delay Sensitive Applications

Limiting the query delay in a sensor network is crucial, yet difficult for many sensor applications. While most of the known techniques that deal with delay-related issues focus on delay-energy efficiencies, they lack provable guarantees for the accuracy of their results. In this chapter, we provide trade-off between delay and accuracy. We propose an architecture which provides provable guarantees. We make a key observation that the fewer nodes queried, the smaller the delay is; but also accuracy gets worse.

Our architecture consists of layers, where each layer contains a subset of the sensor nodes chosen uniformly at random from the whole set, and lower layers contain a superset of the higher layers. In this model, data collection is done by broadcasting a query to a particular layer. Only sensors in that layer will reply the query, resulting in an aggregation tree with fewer hops, and thus smaller delay.

The key difference between our layered architecture and hierarchical architectures is that each sensor in our network only represents itself and submits its own data for each query, without the need to act as a "head" of a cluster of sensors. Therefore, when queried, a sensor will always submit fresh data. Unfortunately, the reduction in delay comes with a price tag; since only a subset of the sensors submit their data, the accuracy of the answer to the query is compromised.

We perform our study in the context of five key "properties" of the network, MAX, MIN, QUANTILE, AVERAGE and SUM. We analyze, given a user-defined accuracy level, what layer of the network should be queried for these properties. We show that different queries do show distinct characteristics which affect the delay/accuracy tradeoff. We also show that for certain types of queries such as AVERAGE and SUM, additional statistical information

obtained from the history of the environment can help further reduce the number of sensors involved in answering a query, and explore the new trade-offs implied.

The algorithm that we propose for our architecture is fully distributed; there is no need for the sensors to keep information about other sensors. Using the fact that each sensor is independent of others, we show how to balance the power consumption at each node by reconstructing the layered structure periodically. This results in an increase in the life expectancy of the whole network.

3.1 System Architecture

3.1.1 Preliminaries

We assume our network consists of N sensors, denoted $s_1, s_2, \ldots, s_N$, deployed uniformly in a square area with side length D. A base station acts as an interface between the sensor network and the users, receiving queries which follow a poisson distribution with mean interval length λ.

We embed a layered structure on our network, with L layers, numbered 0, 1, 2, ..., $L-1$. We use $r(l)$ to denote the transmission range used on layer l: during a transmission taking place on layer l, all sensors on layer l communicate using $r(l)$ and can reach one another, in one or multiple hops. We use R to denote the maximum transmission range of the sensors. Let $e(l)$ be the energy needed to transmit for layer l. The energy spent per sensor for a transmission is $e(l) = r(l)^\alpha$ where $2 \leq \alpha \leq 6$ [81]. Initially, each sensor has B units of energy, which decreases with each transmission.

3.1.2 Network Construction

We now expound on how this structure is constructed. In our scheme, each sensor decides, without communicating with the outside world, to which layer(s) it will belong. We assume that all the sensors have access to a value $0 < p < 1$ and L the value of which will be present shortly (these values may be hardwired into the sensors). Let us consider the decision process that a generic sensor s_i undergoes. All sensors, including s_i, exist in the base layer 0. Inductively, if s_i exists on some layer l, it will, with probability p, *promote* itself to layer $l+1$, which means that s_i will exist on layer $l+1$ *in addition to* all the lower layers $l, l-1, , \ldots, 0$. If on some layer l', s_i makes the decision not to promote itself to layer

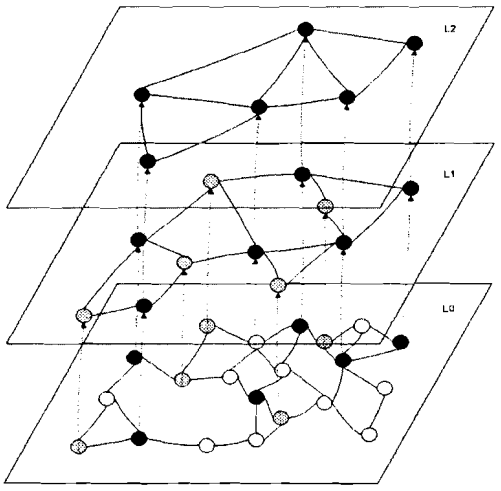


Figure 3.1: A layered sensor network; a link is presented whenever the sensor nodes in a certain layer are within transmission range

$l'+1$, s_i stops the randomized procedure and does not exist on any higher layers. If s_i promotes itself to the highest layer $L-1$, it stops the promotion procedure since no sensor is allowed to exist beyond layer $L-1$. Thus, any sensor will exist on layers $0, 1, \ldots, k$ for some $0 \leq k \leq L-1$. Fig. 3.1 shows the architecture of a sensor network with three layers.

Our layered architecture have the following properties as follows:

a) The layers are numbered 0 through $L-1$, with the base layer labeled 0.

b) The base layer contains all sensors $s_1, \ldots, s_N$.

c) The sensors on layer l form a subset of those on layer $l-1$, for $1 \leq l \leq L-1$.

d) The expected number of sensors on each layer drops exponentially with the layer number.

Since our construction does not assume the existence of any mechanism of synchronization, it is possible that some sensors may be late in completing its procedure for promoting itself up the layers. Since the construction scheme works in a distributed fashion, this is not a problem – the late sensor can simply promote itself using probability p and join its related layers in its own time. In addition, if new sensors join the sensor network, they can perform the same operations. In general, this scheme can be updated without touching the existing structure.

Whenever the base station has a query, the query is sent to a specific Layer. Those and only those sensors existing on this layer are expected to take place in the communication. This can be achieved by reserving a small field (of $\log \log N$ bits) in the transmission packet for the layer number. Once l is specified by the base station (the method for which will be explained later), all of the sensors on layer l communicate using transmission range $r(l)$. The transmission range can be determined by the expected distance of two neighboring sensors on layer l, i.e. $r(l) = \frac{D}{\sqrt{N/2^l}}$, and can be enlarged a little further to ensure higher chances of connectivity.

3.1.3 Specifying the Structure of the Layers

Note that in the construction of the layers, the sensors do not promote themselves indefinitely; this is because if there are too few sensors on a layer, the inter-sensor distance will exceed the maximum transmission range R. Rather, we "cut off" the top of the layered structure, not allowing more than L layers where $L = \Theta\left(\log \frac{N}{(\frac{D}{R}+1)^2}\right)$.

In what follows, we assume that the promotion probability $p = \frac{1}{2}$. We analyze the effect of varying p when appropriate and in our simulations.

3.1.4 Data Collection and Aggregation

Given a layered sensor network constructed as above, we now focus on how a query is injected into the network and an answer is returned.

When the base station has a query to make, it first determines which layer is to be used for this query. Let this layer be l. The base station then broadcasts the query using communication range $r(l)$ for this layer. In this message, the base station specifies the layer number l and the query type (in this chapter, we study MAX, MIN, QUANTILE, AVERAGE and SUM). Any sensor on layer l that hears this message will relay information using communication range $r(l)$; those sensors not on layer l will simply ignore this message.

After the query is received by all the sensors on layer l, a routing tree rooted at the base station is formed. Each leaf node then collects its data and sends it to its parent, which then aggregates its own data with the data from its children, relaying it up to its parent. Once the root has the aggregated information, it can obtain the answer to the query.

Note that our schemes are independent of the routing and aggregation algorithms used in the network. Our goal is to specify the layer number l which will reduce the number

of sensors, as well as the number of messages, used in responding to a query. Once l is determined, the distribution of the query and the collection of the data can be performed in a number of ways, such as that proposed in [12]. In fact, once the layer to be used for a particular query has been identified, the particular routing/aggregation algorithm to be used is transparent to our algorithm.

3.2 Evaluation of the Accuracy and the Latency

In this section we explore how the accuracy of the answers to queries and the latency relate to the layer which is being queried.

In general, we would like to be able to obtain the answers to the queries with as little delay as possible. The delay is a function of the number of sensors whose data are being utilized for a particular query. Thus, the delay is reflected by the layer to which the query is sent. Obviously we would like to get as accurate answers to our queries as possible. When a query utilizes data from all the sensors, the answer is accurate; however, when readings from only a subset of the sensors are used, errors are introduced. We now analyze how these concerns of delay and accuracy relate to the number of sensors queried, and thus to the layer used.

We measure accuracy in terms of the absolute deviation of the computed answer $\hat{a}$ to a query from the exact answer a^*. The accuracy requirement stipulates that this deviation not exceed ϵ in most cases. More precisely, we would like to have $Pr[|\hat{a} - a^*| \geq \epsilon] \leq \delta$, for some given $0 \leq \epsilon, \delta \leq 1$. Here we refer to ϵ as the *accuracy* parameter (or error bound) and δ as the *confidence* parameter.

To explore the relation between the layer l to which a query has been sent and the accuracy of a answer, we represents the number of sensors on each layer as a random variable. In the next lemma, we investigate which layer must be queried if one would like to have input from at least k sensors.

Lemma 3.2.1 *Let* $l < \log N - \log\left(k + ln\frac{1}{\delta} + \sqrt{ln\frac{1}{\delta}(2k + ln\frac{1}{\delta})}\right)$, *where* $k \leq$ *the expected number of sensors on layer* l. *Then, the probability that there are fewer than* k *sensors on layer* l *is less than* δ.

Proof: Define random variable Y_i for $i = 1, \ldots, N$ as follows. $Y_i = 1$ if s_i is promoted to layer l; and $Y_i = 0$ otherwise. Clearly, $Y_1, \ldots, Y_N$ are independent. $Pr[Y_i = 1] = 1/2^l$,

and $Pr[Y_i = 0] = 1 - 1/2^l$. On layer l there are $Y = \sum_{i=1}^{N} Y_i$ sensors. Therefore, $Pr[Y < k] = Pr[Y < \frac{k}{E[Y]}E[Y]] < e^{-(1-\frac{k}{E[Y]})^2 E[Y]/2}$ by Chernoff's inequality. Since $E[Y] = N/2^l$, to have $e^{-(1-\frac{k}{E[Y]})^2 E[Y]/2} < \delta$, we must have $l < \log N - \log\left(k + ln\frac{1}{\delta} + \sqrt{ln\frac{1}{\delta}(2k + ln\frac{1}{\delta})}\right)$ □

In what follows, we analyze the accuracy and the latency in the context of certain types of queries.

3.2.1 MAX and MIN Queries

In general, exact answers to maximum or minimum queries cannot be obtained unless all sensors in the network contribute to the answer, since any missed sensor might contain an arbitrarily high or low data value. The following theorem is immediate.

Theorem 3.2.2 *The queries for MAX and MIN must be sent to the base layer to avoid arbitrarily high error.*

3.2.2 QUANTILE Queries

Due to similar reasons as the MAX and MIN queries, we cannot obtain an exact quantile by querying a proper subset of the sensors in the network. Thus, we introduce an approximate notion of quantile.

Definition 3.2.1 *The ϕ-quantile ($\phi \in (0, 1]$) of an ordered sequence S is the element whose rank in S is $\phi|S|$.*

Definition 3.2.2 *An element of an ordered sequence S is the ϵ-approximation ϕ-quantile of S if its rank in S is between $(\phi - \epsilon)|S|$ and $(\phi + \epsilon)|S|$.*

The following lemma shows that a large enough subset of S has similar quantiles to S.

Lemma 3.2.3 *Let $Q \subseteq S$ be picked at random from the set of subsets of size k of S. Given error bound ϵ and confidence parameter δ, if $k \geq \frac{ln\frac{2}{\delta}}{2\epsilon^2}$, the ϕ-quantile of Q is an ϵ-approximation ϕ-quantile of S with probability at least $1 - \delta$.*

Proof: The element with rank $\phi|Q|$ in Q[1] does not have rank within $(\phi \pm \epsilon)|S|$ in S if and only if either of the following holds: a) More than $\phi|Q|$ elements in Q have rank

[1]Wherever rank in a set is mentioned, it should be understood that this rank is over a sequence obtained by sorting the elements of the set.

less than $(\phi - \epsilon)|S|$ in S, or b) more than $(1 - \phi)|Q|$ elements in Q have rank greater than $(\phi + \epsilon)|S|$ in S.

Since $|Q| = k$, the distribution of elements in Q is identical to the distribution where k elements are picked uniformly at random without replacement from S. This is due to the fact that any element of S is as likely to be included in Q as any other element in either scheme, and both schemes include k elements in Q. Consequently, we can think of the construction of Q as k random draws without replacement from a *0-1 box* that contains $|S|$ items, of which those with rank less than $(\phi - \epsilon)|S|$ are labelled "1" and the rest are labelled "0". For $i = 1, \ldots, k$, let X_i be the random variable for the label of the ith element in Q. Then $X = \sum_{i=1}^{k} X_i$ is the number of elements in Q that have rank less than $(\phi - \epsilon)|S|$ in S. Clearly, $E[X] = (\phi - \epsilon)k$. Hence $Pr[X \geq \phi k] = Pr[X - E[X] \geq \phi k - (\phi - \epsilon)k] = Pr[X - E[X] \geq \epsilon k] = Pr[\frac{X}{k} - E[\frac{X}{k}] \geq \epsilon]$. This is at most $e^{-2\epsilon^2 k}$, by Hoeffding's Inequality[2].

Similarly, it can be shown that the probability that more than $(1 - \phi)|Q|$ elements in Q have rank greater than $(\phi + \epsilon)|S|$ in S is also at most $e^{-2\epsilon^2 k}$. Setting $2e^{-2\epsilon^2 k} \leq \delta$, we have $k \geq \frac{ln\frac{2}{\delta}}{2\epsilon^2}$. □

We now show which layer we must use for given error and confidence bounds.

Theorem 3.2.4 *Let l be a layer such that $l < \log N - \log\left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, then the probability of the ϕ-quantile of this layer is a ϵ-approximation ϕ-quantile of the whole network is at least $1 - \delta$.*

Proof: The probability that layer $l < \log N - \log\left(k + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2k + ln\frac{2}{\delta})}\right)$ has fewer than k sensors is less than $\frac{\delta}{2}$, according to lemma 3.2.1. By lemma 3.2.3, if the number of sensor nodes on layer l is at least $\frac{ln2(\frac{2}{\delta})}{2\epsilon^2} = \frac{ln\frac{4}{\delta}}{2\epsilon^2}$, the probability that the ϕ-quantile on layer l is ϵ-approximation ϕ-quantile of the sensor network is at least $1 - \frac{\delta}{2}$. Hence the answer returned by layer $l < \log N - \log\left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$ is ϵ-approximation ϕ-quantile of the sensor network with probability at least $1 - \delta$. □

[2]Note that Hoeffding's Inequality applies to random samples chosen without replacement from a finite population, as shown in Section 6 of Hoeffding's original paper [34], without the need for independence of the samples.

3.2.3 AVERAGE and SUM Queries

AVERAGE and SUM queries are tightly correlated queries; the AVERAGE is just SUM/N. Since we know the number of the sensors in advance, we just analyze the AVERAGE queries in this section and do not explicitly explain the SUM queries.

We now consider approximating the average data value over the whole sensor network by querying a particular layer. The below lemma indicates that the expectation of the average data value of an arbitrary layer is the same as the average of the base layer, which is the exact average of the sensor network.

Lemma 3.2.5 *Let $a_1, a_2, \ldots, a_N$ be the data values collected by the nodes $s_1, s_2, \ldots, s_N$ of the sensor network. Let k be the number of sensors on layer l. Let $X_1, X_2, \ldots, X_k$ be the random variables representing the values collected by these k sensors. Let $\overline{X} = \frac{1}{k}\sum_{i=1}^{k} X_i$. Then $E[\overline{X}] = \frac{1}{N}\sum_{i=1}^{N} a_i$.*

Proof: Since during the construction of the layers, each sensor independently promotes itself to layer l with the same probability, $Pr[X_i = a_1] = Pr[X_i = a_2] = \ldots = Pr[X_i = a_N] = \frac{1}{N}$, for $i = 1, 2, \cdots, k$. Then $E[X_i] = \frac{1}{N}(a_1 + a_2 + \cdots + a_N)$. Hence $E[\overline{X}] = E[\frac{1}{k}\sum_{i=1}^{k} X_i] = \frac{1}{k}\sum_{i=1}^{k} E[X_i] = \frac{1}{k}\frac{k}{N}(a_1 + a_2 + \cdots + a_N) = \frac{1}{N}\sum_{i=1}^{N} a_i$. □

We thus propose that the average returned by the queried layer be output as the average of the whole network. The next theorem shows that, given the appropriate layer, this constitutes an ϵ-approximation to the actual average with probability at least $1 - \delta$.

Theorem 3.2.6 *Assume the data value at each sensor come from the interval $[a, b]$, and let l be such that $l < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, then the probability that the average of the data values on layer l deviates from the exact average by more than ϵ is less than δ.*

Proof: Let k be the number of sensors on layer l. As we have explained in lemma 3.2.3, these k sensors can be considered as random samples drawn without replacement from all N sensors. Let $X_1, X_2, \ldots, X_k$ be the random variables describing the data values of the k sensors on layer l. Then $a \leq X_i \leq b$ for $i = 1, 2, \cdots, k$. Let $\overline{X} = \frac{1}{k}\sum_{i=1}^{k} X_i$. By lemma 3.2.5, $E[\overline{X}]$ is the exact average over the entire sensor network. For any $\epsilon > 0$, $Pr[|\overline{X} - E[\overline{X}]| \geq \epsilon] \leq 2e^{\frac{-2k\epsilon^2}{(b-a)^2}}$, by Hoeffding's Inequality. Setting $2e^{\frac{-2k\epsilon^2}{(b-a)^2}} \leq \delta/2$, we have $k \geq \frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2}$. By Lemma 3.2.1, the probability that layer $l < \log N - \log\left(k + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2k + ln\frac{2}{\delta})}\right)$ has

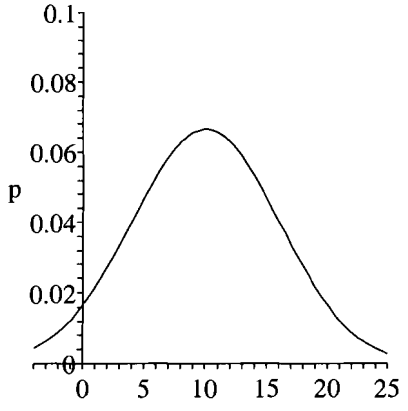


Figure 3.2: Temperature changes in Δt time where Δt = [2am, 12pm]

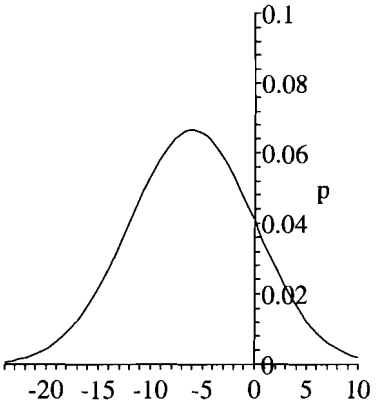


Figure 3.3: Temperature changes in Δt time where Δt = [12pm, 8pm]

fewer than k sensors is less than $\frac{\delta}{2}$. Thus, if we send an AVERAGE Query to layer $l < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, the probability that the estimated average deviates from the exact average by more than ϵ is less than $\frac{\delta}{2} + \frac{\delta}{2} = \delta$. □

Utilizing Statistical Information about the Behavior of Data: It is possible to reduce the delay further given the access to additional information regarding the characteristics of the objects that the sensor network is monitoring. In what follows, we show that if we have the knowledge of the change in data values over time, it can be used to improve the quality of our estimates in continuous data collection.

To make use of the statistical information regarding the change in the value, we adopt a history-based approach, where we assume that we know the distribution of the change of the environment to be a normal distribution with mean μ. Formally, let d_t be the data value at time t, then $d_{\Delta t} = d_{t+\Delta t} - d_t$ is distributed according to normal distribution with mean μ (See Fig. 3.2, 3.3 for examples of the temperature rises from 2am to 12pm and falls from 12pm to 8pm in a day, and follow normal distribution). The change in the average temperature of all the sensors also follows a normal distribution since the sum of normal distributions is still a normal distribution with mean and variance equal to the sum of the individual means and variances.

The intuition behind our strategy is as follows. First we obtain an initial estimate avg

of the average data value in the network, which, by our computations above, is likely to be close to the true average. After one unit of time, the true average is likely to have changed by some value close to μ. Thus, $avg + \mu$ is likely to be a good estimate for the average for that point in time. However, errors have been introduced into our estimate. One cause for possible error is the fact that only a subset of the sensors have been queried. The other contribution to the error comes from our inability to know the exact change in the data value; we only know from the normal distribution that the change is "likely" to be "around" μ. Since the quantity of the error, as well as its likelihood increases with each step of this procedure, we need to make sure that our error bound and confidence parameter remain at acceptable levels.

To ensure low error, we adopt a multi-stage approach to our estimation of the average. In the first stage, which we call *Query Average*, we query a relatively large subset of the sensors – more precisely, we query a low enough layer to obtain an error of $\epsilon_1 < \epsilon$ with confidence level $\delta_1 < \delta$. This high guarantee will leave some room for extra error to be incurred in the following stages.

In the following stage (after one time unit has elapsed), which we call *Test Average*, and the subsequent stages, we will query higher layers, involving a smaller number of sensors, to see whether the expected change pattern is followed. The result of doing this is that either (a) we will see something that falls within our expectation, and boost the confidence to an acceptable level or (b) we will observe an "anomaly", that is, a deviation from expected behavior, which we will attempt to resolve by querying a lower layer with a larger number of sensors. In case of (a), we will have obtained a fast and acceptable answer by querying only a very small number of sensors. Case (b) on the other hand is, by definition of the normal distribution, an anomaly that will not happen often. In the unlikely event of an "accident", in the form of an atypical value, our system will experience a longer query time for the sake of accuracy. In the long run, we will see more "expected" cases and will observe a lower average query time.

Before we go into the specifics of the algorithm, we present an example (Fig. 3.4). Suppose the possible change for temperature after a time unit follows a normal distribution with $\mu = 10$ and $\sigma = 4$. Suppose $\epsilon = 8$ and $\delta = 0.2$. In the first stage, we see that we get the average data value $avg_1 = 60°\text{F}$ by using error bound, say $\epsilon_1 = 2$ and a confidence level, say 0.9, (i.e., confidence parameter $\delta_1 = 0.1$). After 10 hours, we expect that the temperature changes to $avg_1 + \mu = 70°\text{F}$. Let $\hat{\epsilon}$ and $\hat{\delta}$ denote the confidence interval and confidence level

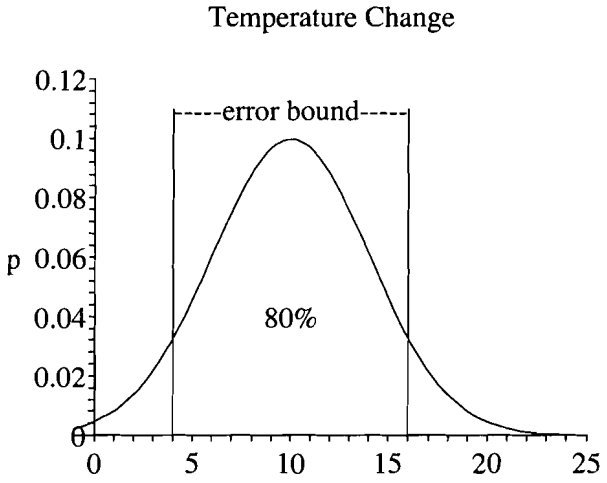


Figure 3.4: Calculating the second stage error bound according to a normal distribution.

of the normal distribution. To ensure an error within the user specified bound of 8, then $\hat{\epsilon} \leq 6$, as shown in Fig. 3.4, resulting a confidence level of $\hat{\delta} = 0.8$. Therefore, after this period of time, the overall confidence level is $0.8 \times 0.9 = 0.72$, i.e., the error probability is larger than the 0.2 specified as acceptable. To boost the confidence level to 0.8, we need to query a few more sensors with an error bound of $\epsilon_2 = 8$ and a much looser error probability of $\delta_2 = \frac{0.2}{1-0.72} = 0.714$. If the returned value of Test Average is 70, we will return this value. Otherwise, if the returned value falls outside of 70 ± 8, this indicates that an anomaly might be present,[3] in which case we perform Query Average to determine the new temperature value. If the returned value falls within 70 ± 8, to ensure the error bound, we perform Test Average with a more stringent error bound until an anomaly is found or the new average value is confirmed.

Below we explain our algorithm in higher detail and analyze its properties mathematically. Fig. 3.5 shows Algorithm Query Average. It takes as input the error and confidence parameters ϵ, δ. We assume that Query(l) returns the average data value for sensors on layer l.

Our next algorithm (Fig. 3.6) shows how to perform Test Average given Query Average. It takes as input error and confidence parameters ϵ, δ, as well as the mean μ and standard

[3]Here we use the word anomaly to indicate a situation whose likelihood is small according to the given normal distribution.

Algorithm QueryAvg (ϵ, δ)
1 Select $\epsilon_1 < \epsilon$ and $\delta_1 < \delta$.
2 $l_1 = \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1} + \sqrt{ln\frac{2}{\delta_1}(2\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1})}\right)$
3 avg_1 =Query(l_1).
4 **return** avg_1

Figure 3.5: Algorithm Query Average

Algorithm TestAvg $(i, \epsilon, \delta, \mu, \sigma, avg_1, \epsilon_1, \delta_1)$
1 $\hat{\epsilon} = \epsilon - \epsilon_1$.
2 Calculate $\hat{\delta} = Pr(\mu - \epsilon_n < X < \mu + \epsilon_n)$ by Normal Distribution.
3 **if** $1 - \hat{\delta} \times (1 - \delta_1) < \delta$,
4 $avg_i = avg_1 + \mu$, **return** avg_i
5 **else**
6 $k = 0$, $\delta_i = \frac{\delta}{1-\hat{\delta}\times(1-\delta_1)}$
repeat
7 $\epsilon_k = \epsilon - k$, $\epsilon_i = \epsilon_k$
8 $l_i = \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i} + \sqrt{ln\frac{2}{\delta_i}(2\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i})}\right)$
9 **if** (Query$(l_i) \leq avg_1 + \mu + k$)
and (Query$(l_i) \geq avg_1 + \mu - k$)
10 **then** $avg_i = avg_1 + \mu$, **return** avg_i
11 **else** $k = k + 1$
12 **if** $k \geq \epsilon$, Goto QueryAvg

Figure 3.6: Algorithm Test Average

deviation σ of the distribution of the change in the data value. It also takes avg_1 which is the average obtained from Query Average and the round number i.

In Line 2 of Algorithm Test Average, we calculate the probability that the change will fall within interval $\hat{\epsilon}$. In Lines 1-5, if the Query Average has already guaranteed the error probability, we do not perform any further queries. This might occur when the number of sensors queried in Query Average is large enough. Line 12 displays the threshold where we should initiate a new QueryAvg query.

Theorem 3.2.7 *Assume the data value collected by each sensor is bounded by $[a, b]$ and the change in the average of the values at all sensors follows a normal distribution with mean μ and standard deviation σ. The probability that algorithm QueryAvg and TestAvg will deviate from the exact average by more than ϵ is less than δ.*

Proof: Let us first consider QueryAvg. Choose any ϵ_1, δ_1 such that $\epsilon_1 < \epsilon$ and $\delta_1 < \delta$. By Theorem 3.2.6, we can obtain the desired accuracy by sending queries to any layer l_1 where $l_1 < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1} + \sqrt{ln\frac{2}{\delta_1}(2\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1})}\right)$.

For TestAvg, let $\overline{Y}_i$ denote the average value over all the sensors at round i. Define $\Delta_i = \overline{Y}_i - \overline{Y}_1$. We know *a priori* that the probability distribution for each Δ_i is a normal distribution with mean μ_i and variance σ_i^2.

In round i, $\hat{\epsilon} = \epsilon - \epsilon_1$ is the confidence interval for normal distribution, hence $\hat{\delta}$ is probability that the change in the value of the average will not exceed $\hat{\epsilon}$.

Therefore with probability $1-\hat{\delta}\times(1-\delta_1)$, we can guarantee an error bound of $\epsilon_1+\hat{\epsilon} < \epsilon$. If $1-\hat{\delta}\times(1-\delta_1) < \delta$ then our query satisfies both bounds ϵ and δ, and we can compute the value to be returned from the value in the previous round and the expected change. Otherwise, we choose $\delta_i = \frac{\delta}{1-\hat{\delta}\times(1-\delta_1)}$ which ensures that the confidence level δ will be bounded in the ith round. For error bound ϵ_i in the ith round, since we don't know the returned value, we can use all $\epsilon_i = \epsilon_k$ where $\epsilon \geq \epsilon_k \geq 0$ as long as the returned value $avg_i \pm (\epsilon_k)$ is bounded by $(avg_1+\mu)\pm\epsilon$. If that happens, the change is confirmed. Otherwise, to bound ϵ and δ, a new QueryAvg must be performed. In our algorithm, we reduce ϵ_i iteratively from ϵ to 0, and use ϵ_i and δ_i to query layer $l_i < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i} + \sqrt{ln\frac{2}{\delta_i}(2\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i})}\right)$ so that the number of sensors in TestAvg increases little by little and the process stops as early as possible. □

3.2.4 The Effect of Promotion Probability p

The promotion probability p will only affect the logarithmic base of the system. Thus, we present comparable theorems without detailing the proofs.

Theorem 3.2.8 *Let l be a layer such that $l < \log_{\frac{1}{p}} N - \log_{\frac{1}{p}}\left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, then the probability of the ϕ-quantile of this layer is a ϵ-approximation ϕ-quantile of the whole network is at least $(1-\delta)$.*

Theorem 3.2.9 *Assume the data value at each sensor come from the interval $[a,b]$, and let l be such that $l < \log_{\frac{1}{p}} N - \log_{\frac{1}{p}}\left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, then the probability that the average of the data values on layer l deviates from the exact average by more than ϵ is less than δ.*

The algorithms of QueryAvg and TestAvg can be adjusted where the queries are sent to layer respected to theorem 3.2.8 and 3.2.9 and theorem 3.2.7 also follows.

3.3 Energy Consumption

It can be readily observed that in our system higher layer sensors will be transmitting at longer ranges than their lower layer counterparts. Given that any high layer sensor is also present in all the lower layers, if nothing is done to balance out the energy consumption, the higher layer sensors will get depleted much faster than the lower layer ones. To balance out the energy consumption, our system reconstructs the layered network periodically by deciding each layer from scratch, so that the top layer sensors change over time. An appropriate timing scheme for the reconstruction will lead to relatively uniform energy consumption across the sensors in the network. Note that the frequency of reconstructions has no expected effect on accuracy, since we are as likely to be stuck with a "good" sample of sensors (in which case reconstruction is likely to give us a worse sample) as with a "bad" one. Given the above and the overhead of building a new aggregation tree for each new construction, we would like to infrequently repeat this procedure to make the energy consumption more even across the network.

Let the lifetime of the network be defined as the time between its initial construction and the first time that a sensor runs out of power [88]. We investigate the relationship between the timing of the reconstructions and the expected lifetime of our system in our simulations. In this section, we analyze our system assuming that each sensor has sufficient power to let it undergo several reconstructions, and that we run sufficiently many reconstructions. Ideally, we have a totally symmetric scenario where the service that each sensor has performed on each layer is identical across sensors. Since the layers are chosen in a randomized fashion, given a large enough number of reconstructions one expects to see that most sensors have served on most layers.

The energy spent by each sensor for a query directly depends on the distance between the sensor and its neighbors. Recall that since there are an expected $(N/2^l)$ nodes on layer l, the transmission range is set to be $r(l) = \frac{D}{\sqrt{N/2^l}}$. Therefore, the energy spent by each sensor for each query on layer l is $e(l) = \left(\frac{D}{\sqrt{N/2^l}}\right)^\alpha$, which is what we will use below to estimate the overall system lifetime.

Overall Lifetime of the System: We assume that the queries are uniformly distributed across different layers due to the error bounds and confidence levels coming independently from the users.

We now present a theorem which estimates the expected lifetime of our system depending on the network parameters.

Theorem 3.3.1 *In a setting where each level is equally likely to be queried, the expected lifetime of our system is* $E(t) = \frac{BL(\sqrt{N})^{\alpha}(1-(\sqrt{2})^{\alpha-2})}{\lambda D^{\alpha}(1-(\sqrt{2})^{L(\alpha-2)})}$

Proof: We assume that each layer has the same probability $\frac{1}{L}$ of being queried. The probability that a sensor s exists on layer l is $\frac{1}{2^l}$, therefore, the energy consumption for s is $\sum_{l=0}^{L-1} \frac{1}{2^l} e(l) \frac{1}{L}$. Let the life expectancy of s be t. Recall that B is the battery power, λ is the incoming query interval following (assumed) Poisson distribution, and $e(l) = \left(\frac{D}{\sqrt{N/2^l}}\right)^{\alpha}$. We have $\sum_{l=0}^{L-1} \frac{1}{2^l} e(l) \frac{1}{L} \lambda t = B$. Therefore, the expected lifetime of the system is $E(t) = \frac{BL(\sqrt{N})^{\alpha}(1-(\sqrt{2})^{\alpha-2})}{\lambda D^{\alpha}(1-(\sqrt{2})^{L(\alpha-2)})}$ □

3.4 Numerical Results

We use numerical simulations to test the performance of our system, as well as to observe the effects of the parameters of the algorithm and the re-election time on the performance.

3.4.1 System Settings

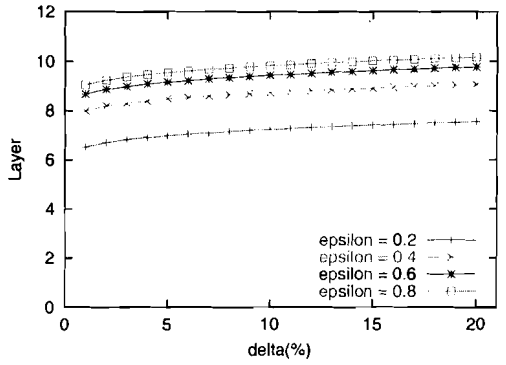
We set the default number of sensors to be $N = 10000$. The default promotion probability is $p = \frac{1}{2}$. We focus on QUANTILE and AVERAGE queries in our simulations.

3.4.2 The Relationship Between Layer and Accuracy

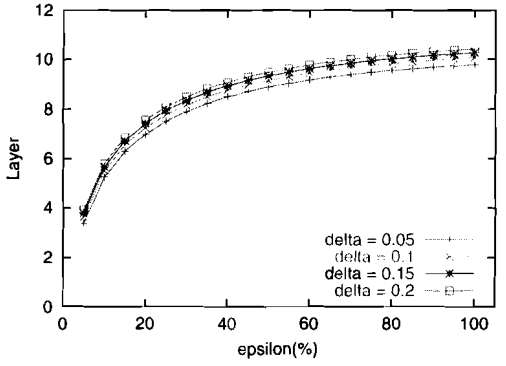
We first evaluate the relationship between the layer answering a query and parameters relating to the quality of the answer to the query[4].

QUANTILE Queries: We first study QUANTILE Queries in Fig. 3.7. It can be seen clearly that, as the error bound ϵ and confidence parameter δ increase, the layer that the

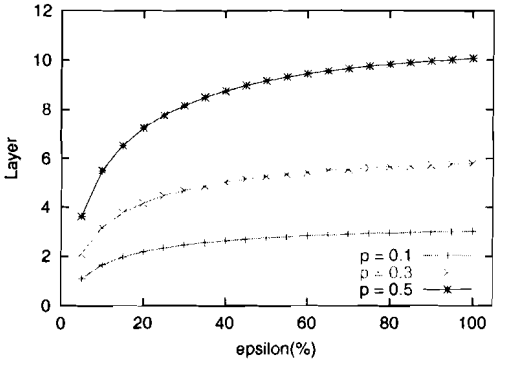
[4]It should be understood that the actually layer queried should be rounded by floor of the value shown in the figure.



(a) Layer as a function of δ

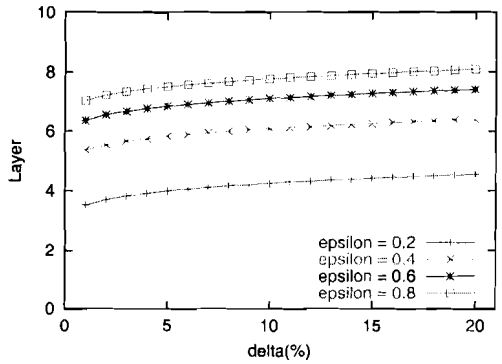


(b) Layer as a function of ϵ

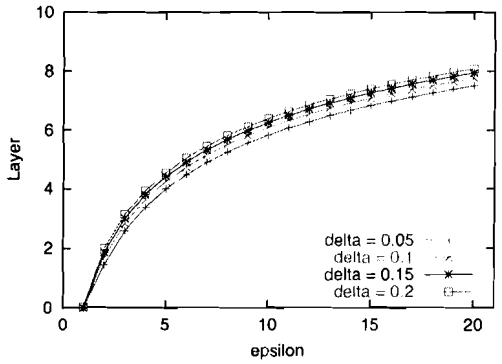


(c) Layer as a function of p

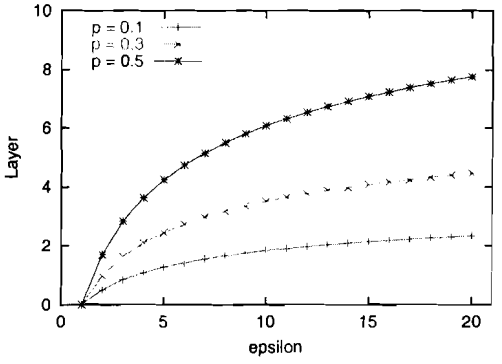
Figure 3.7: Numerical results for QUANTILE Queries.



(a) Layer as a function of δ



(b) Layer as a function of ϵ



(c) Layer as a function of p

Figure 3.8: Numerical results for AVERAGE Queries.

query should be sent to also increases, as confirmed by our computations. Here it can be observed that even though the layer number monotonically increases with both parameters, ϵ has more impact on it than δ as can be seen from Fig. 3.7 (a) and (b). This is because the confidence parameter δ can easily be improved using standard boosting techniques from probability theory and randomized algorithms. In fact, repeating the algorithm $O(\log k)$ times and returning the median answer will improve δ to δ/k, since the probability of getting an incorrect answer $(\log k)/2$ times is at most $\delta^{\log k}$, which is $O(\delta/k)$. On the other hand, to reduce ϵ by a constant factor k, $O(k)$ repetitions of the experiment are needed. Fig. 3.7 (c) shows the relationship between the promotion probability p and the layer number. As p increases, the variation in the layer number for the same query is more obvious. This is because there are fewer layers for smaller p and the choice of layer is more coarse-grained than for larger p.

AVERAGE Queries: We show the relationship of the layer number with the confidence parameter δ, error bound ϵ and promotion probability p for average queries in Fig. 3.8. We observe the same effect as with the QUANTILE queries, i.e., ϵ has a larger impact than δ, for the same reason. This gives us hints for building test queries as we have additional statistical information.

To investigate the question of how to choose the parameters introduced in our algorithms for QueryAvg and TestAvg, we fix the following parameters and vary the others. The upper and lower bounds of the data values are $a = 20$ and $b = 100$. We set the error bound and confidence parameter to be $\epsilon = 15.1$ and $\delta = 0.25$ respectively. The mean and standard deviation for the normal distribution are set to $\mu = 20$ and $\sigma = 8$. We choose ϵ_1 in the range [6, 15], and use two different δ_1, 0.05 and 0.2. We compare the combined algorithm of QueryAvg and TestAvg with QueryAvg only, i.e., using ϵ and δ directly.

Note that, when $p = 0.5$, the expected number of sensors in each layer increases by 2 as we go down each layer. To reduce the overall delay, every query performed on some layer $i - 1$ rather than layer i must be compensated by 2 or more runs of TestAvg performed on layer $i + 1$ rather than layer i, or 1 or more on $i + 2$ rather than $i + 1$, etc. Thus, the combination of QueryAvg and TestAvg is more profitable when the change in the data is highly predictable. Thus, QueryAvg and TestAvg can be used for emergency monitoring applications in stable environments whereas QueryAvg alone can be used in applications of data acquisition in changing environments.

Fig. 3.9 (a) shows the effect of ϵ_1 and δ_1 on QueryAvg. Fig. 3.9 (b) shows the effect

of ϵ_1 and δ_1 on TestAvg. In our simulations we see that regardless of the choice of ϵ_1 and δ_1, the QueryAvg procedure will query a larger number of sensors and the TestAvg procedure will query a smaller number of sensors, comparing to using QueryAvg only. In QueryAvg, however, varying δ_1 has relatively larger effect. We also note that ϵ_1 changes more for QueryAvg than for TestAvg. As a result we suggest choosing a larger ϵ_1 and δ_1 for QueryAvg so that we can save more for QueryAvg and pay less in TestAvg.

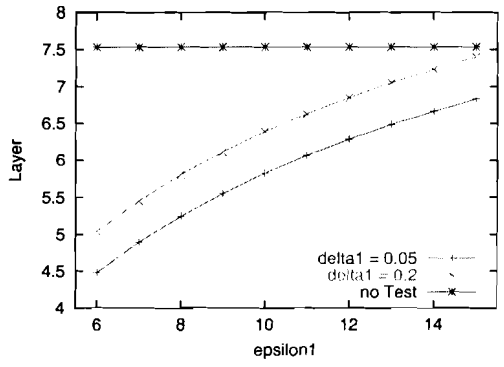
The effect of TestAvg also depends on the readings of the second phase, (or ith query of TestAvg from the QueryAvg). If the ith query is far from the expected value, then we have to narrow down ϵ_i to investigate the sensor network with higher accuracy. For example, let r denote the output of TestAvg, in our algorithm, if the answer of $r \pm \epsilon_i$ is out of the range of $avg_1 + \mu \pm \epsilon$ then we set ϵ_i more stringently. This, however, leads to possibly involving a larger number of sensors for this query. In theory, we stop at $\epsilon_i = 0$, however, in practice, we can stop earlier and move to QueryAvg for efficiency reasons. We observe this effect in Fig. 3.9 (c) where we set $\epsilon_1 = 13$ and $\delta_1 = 0.2$. We see that in this setting, there are reasonable chances that we will stop with gains since when the results from TestAvg fall within $avg_1 + \mu \pm 5$, fewer sensors are queried. This also confirms that ϵ has a greater impact than δ for our architecture.

Next, we study the effect of σ, the standard deviation of the normal distribution. σ represents the rate of change in the data. One can see in Fig. 3.10 that σ has a big influence on efficiency. The discontinuity of the lines in Fig. 3.10 (a) and (b) indicates that QueryAvg has tested more sensors than required, making some of the following rounds of TestAvg unnecessary.

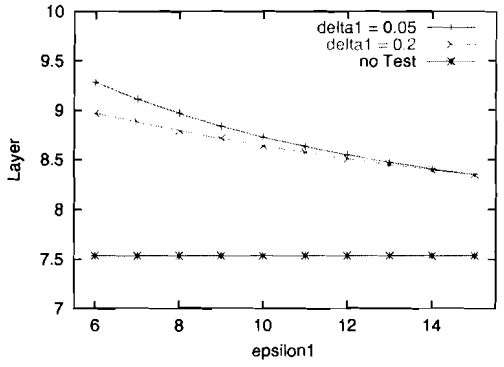
3.4.3 Energy Consumption Evaluation

In this section we consider the effect of our algorithms on the energy consumption. We assume that our sensor network occupies an area of 100×100, with uniformly distributed positions, and each sensor has 5000 units of energy. We also assume that the queries are generated according to Poisson distribution with mean value $\lambda = 20$. The data queries are generated uniformly from the set {MAX, MIN, QUANTILE, AVERAGE}. The query parameters are assumed to be uniformly generated within the bounds $0 < \delta < 0.5$, and $0 < \epsilon < 0.5$ for QUANTILE. For AVERAGE, ϵ is proportional to the bounds a, b which we set to be 20 and 100.

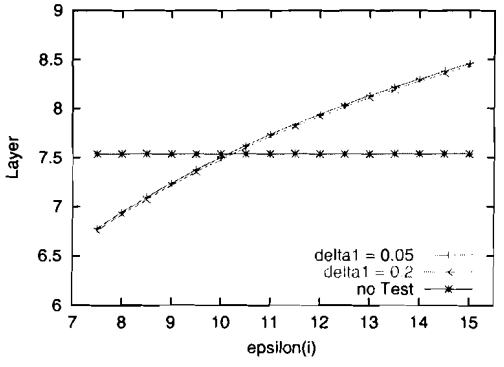
In our experiments, we use the simplest data gathering technique, flooding, where the



(a) Layer as a function of ϵ_1

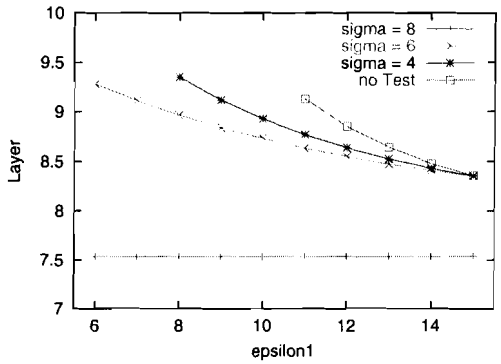


(b) Layer as a function of ϵ_1

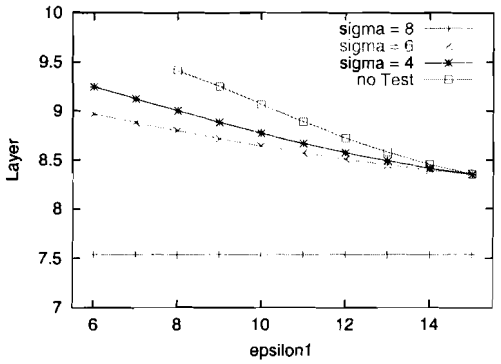


(c) Layer as a function of ϵ_i

Figure 3.9: AVERAGE Queries with different δ_1 values; no Test: QueryAvg only with δ and ϵ.



(a) Layer as a function of ϵ_1



(b) Layer as a function of ϵ_1

Figure 3.10: Effect of the standard deviation (σ) of the normal distribution.

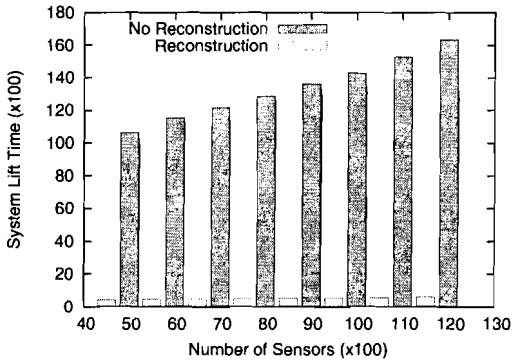


Figure 3.11: Energy consumption with and without reconstructions

source and all the intermediate sensors in the same layer just broadcast the query within their transmission range and collect the answers in a reverse fashion. Every data point presented in the figures is the average of 100 random experiments.

We consider the effect of the reconstruction of the layers in Fig. 3.11. Clearly, without reconstruction, the sensor network depletes significantly faster, usually less than 10% of the lifetime of the system with reconstructions. As the number of sensor nodes increases, the lifetime of the system also increases, since the distances between sensors are smaller, leading to less energy usage.

3.5 Conclusion

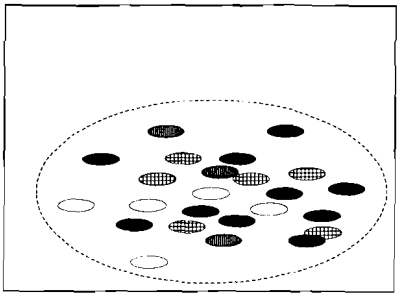
In this chapter we build a layered architecture for a sensor network where a small fraction of the sensors can be used for answering queries, so as to reduce the processing delay. We sacrifice the accuracy, however. We give provable bounds on the number of sensors queried and the overall precision of the queries for five different types of queries.

Chapter 4

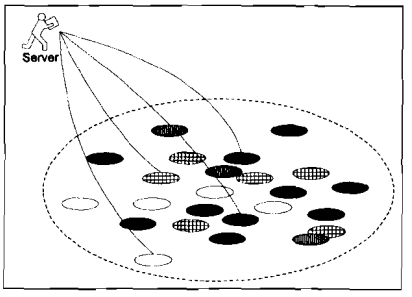
Data Collection in Extreme Environments

For applications in extreme environments, the demands on the network are different from simple field monitoring sensor systems. First, the delay requirement may be more restrictive. Second, as the data harvest is only performed occasionally, sensor nodes need to store the data temporarily when the server is absent. In contrast to aggregated values, raw data are more precious in the extreme environments as the raw data can be evaluated by the server in more refined way for multiple properties of the sensor network after collection. In this setting, the tree based routing structure is not suitable since without aggregation, the payload will not be reduced and the sensors close to the root of the tree will be heavily burdened. A ubiquitous access is thus suggested in [20]. In this solution, the server just randomly contacts a few sensor nodes to retrieve data. This server accessing (also known as *blind access*) is easy to implement. It inherently distributes the communication cost from the root sensor to multiple sensor nodes, and balances the load.

In this scheme, if the data can be retrieved accurately, it is also much faster. Unfortunately this straightforward approach may introduce large replication due to difficulties in cooperation of the sensors in data storage and scheduling in data retrieval. We see an example in Fig. 4.1. There are four different data segments distributed in the network. Each sensor (represented by a small circle) can store only one data segment (represented by the texture). The server randomly contacts several sensors to upload data. We see from Fig. 4.1 (b), server collection of right data segments is not always easy. In this example, the



(a) Server is absent.



(b) Server access.

Figure 4.1: An example of the problems for blind access.

server contacts four sensors, and unfortunately obtains only three different data segments.

Replication and redundancy management of similar problems have been studied by known coding algorithms, e.g., different types of erasure codes [8]. Most of these codes are generated at a central entity and then distributed to different storage locations. This is not realistic in our application, because no sensor is capable of storing all the data, let alone performing complicated encoding operations. A potential solution rises from *random network coding* [20][56][80], which distributively manipulates the data in each node. Such operations combine all (say, N) data segments, making the coded data segments equivalent to each other in decodeability, i.e., different combined data segments have no advantage to each other for decoding operations. A fast and load-balanced data collection is then realized.

A key deficiency of conventional network coding is the lack of support for removing obsolete data. In harsh environments, where the server may only approach the sensor network occasionally, the data is temporarily stored in each node. The sensors then have to remove obsolete data segments to be able to accommodate newly collected ones. To achieve this, conventional network coding has to first decode and then re-encode the combined data, which is time and resource-consuming [26]. Even worse, given that a sensor can only store a partial set of the combined data, it is impossible to carry out decoding operations in each individual sensor.

In this chapter, we present *Partial Network Coding* (PNC), which effectively solves the above problems. PNC inherits the blind access capability of NC, and yet achieves the following salient features: 1) it allows a higher degree of freedom in coded data management, in particular, decoding-free data removal; 2) its computation overhead for encoding and decoding is almost identical to the conventional network coding; and 3) we prove that its performance is quite close to that of the conventional network coding, except for a sub-linear overhead on storage and communications. We also address a set of practical concerns toward building and maintaining a PNC-based sensor network for continuous data collection and replacement. The feasibility and superiority of PNC are further demonstrated through simulation results.

4.1 Preliminaries

4.1.1 Model and Notations

We now give the a formal description of PNC. We assume that the total number of up-to-date events to be recorded in the whole system is N. Each event (e.g., that temperature goes beyond 100°F), upon its occurrence, is recorded by all the sensors in the sensor network. The event is represented by one data segment, denoted by c_j, and $c_{j'}$ is fresher than c_j if $j' > j$. Similar to existing studies on linear network coding, we use $\sum_{j=0}^{N-1} \beta_j \times c_j$ to generate a coded data segment f_i, where $\beta = (\beta_0, \beta_1, \cdots, \beta_{N-1})$ is a coefficient vector, each item of which is randomly generated from a finite field F_q, where q is the size of this finite field. Since the coding can be viewed as a combination process, f_i is also referred to as a *combined* data segment, and c_j as an *original* data segment. Notice that the size of f_i remains equal to c_j. We define the *cardinality* of f_i to be the number of original data segments it contains, and the *full cardinality* of the system is the highest possible number, i.e., N. Each sensor in our network has a buffer of size $B(< N)$ for storing the data segments. For each server access, W sensors are to be contacted and, without loss of generality, each sensor will upload one data segment from its buffer. A summary of the notations can be found in Table 4.1.

Clearly, to obtain all the N original data segments, we must have $W \geq N$, and even so, not all the segments are necessarily obtained in one access. Consider a naive data storage and collection scheme with no coding. Assume that B segments (out of the N up-to-date original segments) are stored in each sensor's buffer and the distribution is uniform. Then the success ratio for this naive scheme is given by $\prod_{i=0}^{N-1} \frac{N-i}{N}$. Here, the success ratio serves as the major evaluation criterion in our study, and is defined as follows:

Definition 4.1.1 *(Success Ratio) The success ratio is the probability that a scheme successfully collects all the N original data segments. The default settings of W and B are $W = N$ and $B = 1$, which are their lower bounds for valid schemes.*

For the naive scheme, the success ratio is a decreasing function of N. As shown in Table 4.2, even for $N = 2$, the probability is barely 50%, and the performance is extremely poor for larger N.

Notation	Definition
N	Number of original data segments to be collected
B	Buffer size of each sensor
c_j	Original data segment with index j
β_j	Coding coefficient for c_j
f_i	Combined data segment with index i
f_i^k	f_i, with the oldest original segment k
$\hat{k}$	Cardinality of the combined data segments
W	Number of sensors queried by the server for each access
q	Size of finite field for coefficients

Table 4.1: List of Notations for Chapter 4.

N	Success Ratio	N	Success Ratio
2	0.5	6	0.0154321
3	0.222222	7	0.0061199
4	0.09375	8	0.00240326
5	0.0384	9	0.000936657

Table 4.2: Success ratio of the naive scheme ($W = N, B = 1$)

4.1.2 Network Coding based Collection: Pros and Cons

We now show that network coding can significantly increase the success ratio. With network coding, all data segments are stored in a combined fashion, and the N original data segments can be decoded by solving a set of linear equations after collecting any N combined data segments. A necessary condition here is that the coefficient vectors must be linearly independent. This is generally true for a large enough field size q [56]. As shown in Table 4.3, the probability of linear independency is over 99.6% for $q = 2^8$, and this is almost independent of N. As such, for the network coding based data storage and collection scheme, the success ratio with $W = N$ and $B = 1$ is close to 100%.

q	Probability	q	Probability	q	Probability
2^1	0.288788	2^5	0.967773	2^9	0.998043
2^2	0.688538	2^6	0.984131	2^{10}	0.999022
2^3	0.859406	2^7	0.992126	2^{11}	0.999511
2^4	0.933595	2^8	0.996078	2^{12}	0.999756

Table 4.3: Probability of Linear Independency as a Function of Finite Field Size (q).

$$\begin{aligned} f^0 &= [c_3, c_2, c_1, c_0] \\ f^1 &= [c_3, c_2, c_1] \\ f^2 &= [c_3, c_2] \\ f^3 &= [c_3] \end{aligned}$$

Figure 4.2: An example of PNC for $N = 4$.

So far we have focused on static data only. In network coding, it is easy to combine new data segments to existing data segments, which increases the cardinality. The reverse operation is difficult, however. Specifically, to remove a data segment, we have to first decode the data segments, delete the obsolete data segment, and combine the remaining data segments to a new one. This is time and resource-consuming for power limited sensors. Even worse, it is often impossible for $B < N$, because decoding requires N combined data segments. On the other hand, for continuously arriving data, if we keep obsolete data segments in the system, we have to upload more data segments than necessary to the server for a successful decoding of N needed data segments. Eventually, the buffer will overflow, and the system will simply crash. This becomes a key deficiency for applying network coding in continuous data collection.

4.2 Partial Network Coding based Data Storage and Replacement

In this section, we show a new coding scheme that conveniently solve the problem of data removal, thus facilitating continuous data management. Our coding scheme allows the combination of only part of the original data segments, and we refer to it as *Partial Network Coding (PNC)*, cf. *network coding (NC)* and no network coding at all *(Non-NC)*.

4.2.1 Overview of Partial Network Coding

In PNC, instead of having full cardinality of each combined data segment, we have varied cardinalities from 1 to N. Formally, for original data segments $c_0, c_1, \ldots, c_{N-1}$, we have a coding base $\mathcal{B} = \{f^k | f^k = \sum_{j=k}^{N-1} \beta_j \times c_j, k \in [0, \ldots, N-1], \beta_j \in F_q\}$. We omit β_j in this chapter and use $f^k = [c_{N-1}, c_{N-2}, \ldots, c_k]$ for ease of exposition. Notice that if $\hat{k}$ denote

the cardinality of a combined data segment, then the cardinality of f^k can be calculated by $\hat{k} = N - k$. The coding base for $N = 4$ is illustrated in Fig. 4.2. We may further drop the superscript k if the cardinality of the combined data segment is clear in its context.

In our application scenario, each sensor stores only a subset of these combined data segments given buffer size $B < N$. The storage for each sensor is $\mathcal{S} = \{f_i^k | f_i^k \in \mathcal{B}, 0 \leq i \leq B-1\}$. We use f_i provided that k is clear in the context to represent the ith combined data segment in this sensor. An illustrative example is shown in Fig. 4.3, which also includes the corresponding NC and Non-NC. In this example, data is distributed in a network of 6 sensors (s_0 through s_5) and each sensor has two units of storage. From Fig. 4.3 (c), we notice that, when a new c_4 is generated and c_0 becomes obsolete, sensors s_0 and s_4 can simply drop the longest combined data f_0 in their respective buffers. The buffers of s_0 and s_4 then become $\{f_0 = [c_4, c_3, c_2], f_1 = [c_4]\}$ and $\{f_0 = [c_4, c_3], f_1 = [c_4]\}$, respectively. This simple example demonstrates the salient feature of PNC, that is, removing the obsolete data without decoding.

4.2.2 Data Storage and Replacement in PNC

Distribution of Cardinality: Partial network coding intrinsically manages the shape (i.e., cardinality) of the combined data segments. It is not difficult to see that, for a server data collection, if the contacted sensors provide combined data segments with high cardinalities, then the success ratio will be higher. We summarize this in the following two observations.

Observation 4.2.1 *The success ratio is maximized if every sensor provides the server the combined data segment with the highest cardinality from its buffer.*

Observation 4.2.2 *Consider time instances t_1 and t_2. If at t_1, the probability for each sensor to provide high cardinality data segments is greater than t_2, then success ratio for a data collection at t_1 is higher than that at t_2.*

Generally speaking, in each particular time instance, it is ideal for the system to have combined data segments of cardinalities as high as possible. In an extreme case, if all the combined data segments in the system have cardinality N, the success ratio is 100% and the system is essentially reduced to the conventional network coding based. Once a new data segment arrives, however, all the combined data segments will have to be deleted to make

$$s_0 : \{c_3, c_1\}, \quad s_1 : \{c_1, c_0\}$$
$$s_2 : \{c_3, c_0\}, \quad s_3 : \{c_3, c_1\}$$
$$s_4 : \{c_3, c_2\}, \quad s_5 : \{c_2, c_0\}$$

(a) Non-NC

$$s_0 : \{f_0 = 5c_3 + 2c_2 + 3c_1 + 4c_0, \quad f_1 = 7c_3 + 2c_2 + 3c_1 + 4c_0\}$$
$$s_1 : \{f_0 = 3c_3 + 2c_2 + 10c_1 + c_0, \quad f_1 = 10c_3 + 2c_2 + 5c_1 + c_0\}$$
$$s_2 : \{f_0 = 2c_3 + 5c_2 + 2c_1 + 4c_0, \quad f_1 = c_3 + 15c_2 + 6c_1 + 3c_0\}$$
$$s_3 : \{f_0 = c_3 + 18c_2 + 9c_1 + 4c_0, \quad f_1 = c_3 + 8c_2 + 9c_1 + 14c_0\}$$
$$s_4 : \{f_0 = 5c_3 + 2c_2 + 3c_1 + 4c_0, \quad f_1 = 2c_3 + 6c_2 + 3c_1 + 4c_0\}$$
$$s_5 : \{f_0 = 7c_3 + 7c_2 + 9c_1 + 5c_0, \quad f_1 = 8c_3 + 8c_2 + 8c_1 + 4c_0\}$$

(b) NC

$$\begin{aligned}
s_0 &: \{f_0 = [c_3, c_2, c_1, c_0], \quad f_1 = [c_3, c_2]\} \\
s_1 &: \{f_0 = [c_3, c_2, c_1], \quad f_1 = [c_3]\} \\
s_2 &: \{f_0 = [c_3, c_2, c_1], \quad f_1 = [c_3, c_2]\} \\
s_3 &: \{f_0 = [c_3, c_2], \quad f_1 = [c_3]\} \\
s_4 &: \{f_0 = [c_3, c_2, c_1, c_0], \quad f_1 = [c_3]\} \\
s_5 &: \{f_0 = [c_3, c_2, c_1], \quad f_1 = [c_3, c_2]\}
\end{aligned}$$

(c) PNC

Figure 4.3: Comparison of Non-NC, NC and PNC.

Algorithm Data Replacement(c_n)
  c_n: newly recorded data
  for $i = 1 \ldots B$
    if cardinality(f_i) $< N$,
      $f_i = \beta_n c_n + f_i$
    else
      $f_i = \beta_n c_n$

Figure 4.4: Data Replacement Algorithm.

room for the new segment. For subsequent data retrievals, no data segment but the newest one can be answered.

Since the data arrival and server collection times are not known in advance, an optimal choice of the cardinality distribution thus depends on the requirement of each particular application and how the overall profit function is defined. We now consider the performance of PNC with a uniform cardinality distribution throughout the lifetime of the system. This distribution does not favor data arrivals or server collection at any particular time, and is thus applicable to a broad spectrum of applications that have no specific arrival/retrieval patterns. It also serves as a baseline for further application-specific optimizations.

Data Replacement in PNC: In a uniform cardinality distribution, for each collected data segment, the probability of encountering any cardinality is $\frac{1}{N}$. However, a sensor does not have data segments of all the different cardinalities in its limited buffer. For example, as shown in Fig. 4.2, a sensor with a buffer size of two segments will never see all the four possible data segments. More importantly, it is impossible for the sensors to know exactly what other sensors store. As such, maintaining the uniformity of cardinalities becomes a great challenge in partial network coding.

We solve this problem by a *Data Replacement* algorithm locally executed at each sensor (Fig. 4.4). It translates the uniformity maintenance problem to a uniform configuration for the initial distribution; the latter is much easier to achieve.

Theorem 4.2.3 *If the cardinality is uniformly distributed, then after executing the Data Replacement algorithm (Fig. 4.4), the distribution of the cardinality remains uniform.*

Proof: If the distribution of the cardinality is uniform, then the probability that a combined data has cardinality $\hat{k}$ is $\frac{1}{N}$ for all $\hat{k} = 1 \ldots N$. After executing Data Replacement, the probability that a combined data segment has cardinality $\hat{k}$ is equal to the probability that

this segment previously has cardinality $\hat{k}-1$, $\hat{k}=1\ldots N-1$, and the probability for a combined data segment has cardinality 1 is equal to the probability it previously has cardinality N . Hence, the probability is still $\frac{1}{N}$, and the distribution remains uniform. □

The above theorem suggests that a uniformity is inherently maintained in data replacement, and the algorithm is fully distributed and localized. Therefore, before network deployment, we can uniformly assign the cardinalities to the sensors. Assume $B=1$; after the deployment, the sensor assigned with cardinality $\hat{k}$ can wait for $N-\hat{k}$ events and record and combine the $\hat{k}$ following events only. The initial cardinality distribution of the combined data in the sensors is then uniform. The above configuration can be easily generalized to larger buffer sizes.

4.2.3 Performance Analysis of PNC and Enhancements

We now analyze the performance of PNC, and identify its weakness. We then present two effective enhancements to improve its performance.

With the uniform distribution, we can focus on the success ratio of a single server data collection. Since the cardinality of each data segment is not necessarily N in PNC, it is possible that the success ratio is less than 100%. Let $B=1$ for each node and the success ratio for obtaining i data segments by collecting i data segments using partial network coding be $F(i)$. Define $F(0)=1$, we quantify the success ratio in this scenario as follows:

Theorem 4.2.4 *The success ratio* $F(N)=(\frac{1}{N})^N\sum_{i=0}^{N-1}\begin{pmatrix} N \\ N-i \end{pmatrix}F(i)i^i$.

Proof: The basic idea of our proof is to find the sets of combined data segments that are decode-able, referred to as *valid sets*. For example, a set of combined data segments with cardinality 1 through N is decode-able, and the set of combined data segments with all cardinality being 1 is not decode-able. The success ratio is given by the number of valid sets over the total number of possible sets of combined data segments; the latter is N^N.

A valid set can be constructed as as follows: pick $N-i$ combined data segments of cardinality N and i combined data segments with cardinality less or equal to i. These i combined data segments should be a valid set (decode-able) in terms of i. For example, if $N=4$, a valid set may consist of two combined data segments of cardinality 4 and two combined data segments with cardinality less than or equal to 2. The number of the latter is $F(i)i^i$. Since the retrieval can be of any sequence,we need to fit all these combined

data segments into N pickup sequences. For those $N-i$ combined data segments with cardinality N, there are $\begin{pmatrix} N \\ N-i \end{pmatrix}$ locations to fit in. Notice that we do not need to shuffle the i combined data segments with smaller cardinalities as $F(i)i^i$ has already contained all possible shuffles. Therefore, the total number of valid sets is $\sum_{i=0}^{N-1} \begin{pmatrix} N \\ N-i \end{pmatrix} F(i)i^i$ after summing up all $i \in [0, N-1]$, and the theorem follows. □

We further derive an upper bound and lower bound for PNC.

Theorem 4.2.5 *The upper and lower bounds of the success ratio are* $1 - (\frac{N-i}{N})^N$ *and* $\prod_{i=0}^{N-1} \frac{N-i}{N}$ *respectively.*

Proof: The success ratio is upper bounded by successfully obtaining the combined data segments with the highest cardinality. This probability is $\frac{1}{N}$ due to the uniform distribution. The probability of not getting it in N picks is $(\frac{N-i}{N})^N$, giving an upper bound $1 - (\frac{N-i}{N})^N$.

The success ratio of getting a set of combined data segment with cardinality 1 through N is $(\frac{1}{N})^N N!$, which is equal to $\prod_{i=0}^{N-1} \frac{N-i}{N}$. This is clearly a lower bound. □

As discussed before, the success ratio of Non-NC is:

Observation 4.2.6 *If the distribution of the data segments is uniform, the success ratio for obtaining all* N *data segments by randomly collecting* N *data segments (Non-NC) is* $\prod_{i=0}^{N-1} \frac{N-i}{N}$.

We can see from Theorem 4.2.5 that the lower bound of the success ratio of PNC is the same as the success ratio of Non-NC. A preliminary comparison of the success ratio can be founded in Fig. 4.5 where we can see that PNC always perform better than Non-NC. Detailed comparison as well as practical enhancements which substantially improve the success ratio will be presented in the following sections.

It is also worth noting that, when the buffer size of a sensor increases, it can upload a data segment of higher cardinality when queried. The success ratio of PNC will thus be improved. On the contrary, for Non-NC, since each sensor can only randomly picks the data segment from its buffer for uploading, its performance remains unchanged.

We go on to compare PNC and NC. We know that, by ignoring the linear dependency of coefficients and data removal, NC achieves 100% success ratio when $W = N$ combined data segments are collected. An interesting question is thus whether PNC can achieve the

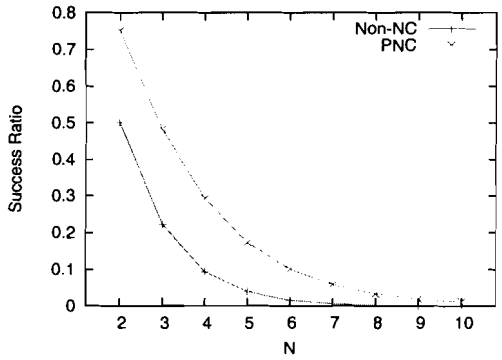


Figure 4.5: Success ratio as a function of N (in default values $M = N$ and $B = 1$).

same performance, or, if not, how can we improve the performance. To give some intuition, we see that in PNC, the chances for encountering c_{N-1} and c_0 are not identical: the most up-to-date data segment c_{N-1} is easier to collect because every combined data segment contains c_{N-1}; on the contrary, the oldest data segment c_0 (but not obsolete) exists in the combined data segment with cardinality of N only. As such, the decoding ratio with PNC after blindly accessing a subset of sensors could be lower than that with NC.

To address the above problem, we make two enhancements to the original PNC scheme. First, we extend the cardinality of the system from N to $N + \sqrt{N}$; that is, in addition to N required data segments, we store another $\sqrt{N}$ obsolete data segments in the system. These obsolete data segments makes the originally oldest data segment relatively "younger" and therefore more likely to be collected. Second, we expand the buffer size of a sensor to $B = \sqrt{N} + 1$, which facilitates the first enhancement. We see an illustration in Fig. 4.6; on the left hand side, the inner grey triangle denotes the original PNC with no extension. After extension, PNC-ext becomes the outer white triangle, with cardinality $N+\sqrt{N}$. Given buffer size $\sqrt{N}+1$ for each sensor, one data segment is picked in every $\sqrt{N}$ interval as shown by the lines. With these two modifications, the following lemma shows that there is a scheme such that each sensor can upload a data segment with cardinality at least N when queried.

Lemma 4.2.7 *By extending the cardinality of the system to $N + \sqrt{N}$ and the buffer size to $\sqrt{N}+1$, each sensor can have a combined data segment with cardinality at least N in its buffer.*

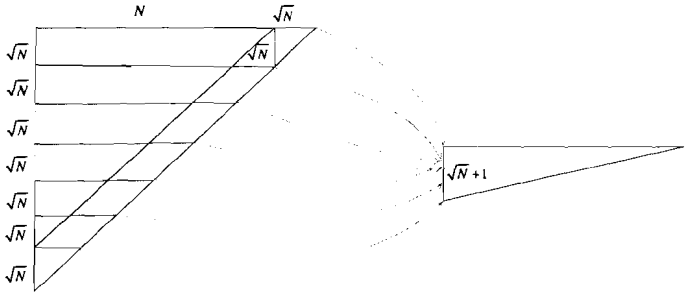


Figure 4.6: Cardinality extension and buffer storage in PNC.

$$
\begin{aligned}
f_0 &= [c_{N-1}, \ldots, c_{N-\sqrt{N}-1}, \ldots, c_{2\sqrt{N}-1}, \ldots, c_0, \ldots, c_{-\sqrt{N}}] \\
f_1 &= [c_{N-1}, \ldots, c_{N-\sqrt{N}-1}, \ldots, c_{2\sqrt{N}-1}, \ldots, c_0] \\
f_2 &= [c_{N-1}, \ldots, c_{N-\sqrt{N}-1}, \ldots, c_{2\sqrt{N}-1}] \\
&\ldots \\
f_{\sqrt{N}} &= [c_{N-1}, \ldots, c_{N-\sqrt{N}-1}]
\end{aligned}
$$

Figure 4.7: A snapshot of the buffer at a sensor in PNC.

Proof: Consider the following storage scheme for each sensor: A sensor picks a random number $k \in [-\sqrt{N}, 0]$ (we use a negative index to denote an obsolete data segment) and stores combined data segments $f_0 = [c_{N-1}, \ldots, c_k]$, $f_1 = [c_{N-1}, \ldots, c_{k+\sqrt{N}}]$, $f_2 = [c_{N-1}, \ldots, c_{k+2\sqrt{N}}]$, $\ldots, f_{\sqrt{N}} = [c_{N-1}, \ldots, c_{N-\sqrt{N}-k}]$. The difference of the cardinality between f_i and f_{i+1} is $\sqrt{N}$ for all $0 \leq i \leq (B-1)$. The buffer requirement of this scheme is $\sqrt{N}+1$, and for any k the sensor chooses, the cardinality of f_0 is greater than N. In addition, after executing the Data Replacement algorithm, the cardinality of f_0 remains greater than N until it is discarded upon the arrivals of $\sqrt{N}$ new data segments. After that, the cardinality of f_1 will be greater than N, and the iteration continues. □

A concrete example is shown in Fig. 4.7, where we denote the $\sqrt{N}$ obsolete data with negative indices. We can see that f_0 has a cardinality of $N+\sqrt{N}$ (with $\sqrt{N}$ obsolete data segments combined). When a new c_N is generated, according to Data Replacement algorithm, it will be combined to all f_i, and f_0 will be discarded. f_1 however will have a cardinality of $N+1$ (combined with one obsolete data segment c_0). We can guarantee that, at any given time, each sensor will have a data segment of cardinality at least N. We then have the following observation on the performance of PNC as compared to NC.

Theorem 4.2.8 *The success ratio of PNC with $B = \sqrt{N} + 1$ and $W = N + \sqrt{N}$ is 100% (neglecting linear dependency of the coefficients).*

Proof: From lemma 4.2.7, the server can collect $\sqrt{N} + N$ combined data segments with cardinality at least N. For decoding, we are trying to solve a set of linear equations, of which the coefficients form a $(N + \sqrt{N}) \times (N + \sqrt{N})$ matrix. Since the cardinality of each coefficient vector is at least N, then the rank of this matrix is at least N. Therefore, we can solve the first N variables (which contributes to the rank). □

Corollary 4.2.9 *The success ratio of PNC with $B = \sqrt{N}+1$ and $W = N+\sqrt{N}$ is identical to the success ratio of NC with $B = 1$ and $W = N$.*

In other words, after sacrificing a sublinear buffer overhead ($\sqrt{N}$) at each sensor and a sublinear communication overhead ($\sqrt{N}$), the PNC will be able to decode all the N original data segments in a blind access as NC does.

4.3 Protocol Design and Practical Issues

In this section, we address some major practical concerns, and present a collaborative and distributed protocol for continuous data collection with PNC.

4.3.1 Computation and Communication Overheads

Since the sensors are small and power constrained entities, the PNC operations must be light-weighted. It is known that the computational overhead for network coding lies mainly in the decoding process. This is however performed in the powerful servers. Each sensor just needs to randomly generate a set of coefficients, combine newly arrived data with those in the buffer, or remove an obsolete data segment. All of these operations are relatively simple with low costs. Another overhead is the transmission cost. For network coding based application, besides the combined data, the coefficient vectors have to be uploaded for decoding. Such overheads are generally much lower than the data volume, and our simulation results have shown that the benefits of PNC generally overcome these overheads.

An alternative way to reduce the computation overhead is using polynomial interpolation [73]. We change the coding base as $\mathcal{B} = \{f^k | f^k = \sum_{j=k}^{N-1} \beta^j \times c_j, k \in [0, \ldots, N-1], \beta \in F_q\}$. Notice that the major difference in this setting is that an integer β is chosen and the

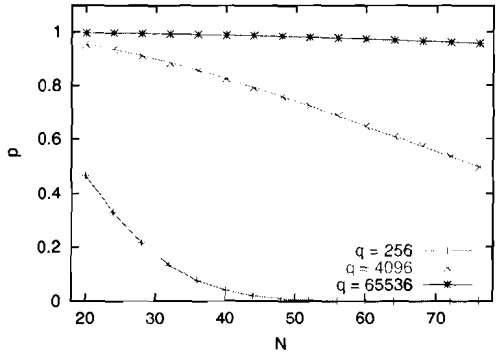


Figure 4.8: Probability of linear independency as a function of the number of data segments.

coefficient vector is $(1, \beta, \beta^2, \ldots, \beta^{N-1})$ (for a combined data segment of cardinality N). Therefore, only β, a constant in respect to N, is needed to be uploaded. If all β uploaded are different from each other, then the coefficient vectors are guaranteed to be linearly independent [73]. Therefore, the overhead is reduced substantially. This, however, does not come for free. If two sensors choose the same β, then these two combined data segments are linear dependent. The linear dependency is thus higher than the case where all items of the coefficient vectors are chosen randomly. Formally, the linear independency probability is $p = \prod_{i=0}^{N-1} \frac{q-i}{q}$, where q is the size of the finite field F_q. Fig. 4.8 illustrates the relationship of p, N and q. To migrate this effect, we can use larger finite field F_q. Comparing $q = 2^8$ and $q = 2^{16}$, the probability of linear independency has been remarkably improved. For different applications, this is worthwhile as the increase of F_q costs only logarithmic additional bits.

4.3.2 Multiple Data Patterns

In many applications, the sensor network is required to collect multiple data ranges or patterns. For example, the sensor network may need to track the temperature of multiple critical levels. Therefore, the sensors need to be invoked at different times to record different data sets. The problem here is whether to use a mixed storage with each sensor splitting its buffer to store different temperature levels, or just assign different subset of sensors to record different levels. The tradeoff is obvious: the former might record certain temperature levels incompletely if the buffer is too small, i.e., smaller than N; the latter, while fully recording certain levels of temperature, will risk the incapability of decoding an entire level.

The above *all or nothing* effect is also considered in [16]. Yet, for PNC based collection, we can see that a larger buffer might provide data with higher cardinalities, and it is easy to add an importance parameter in our system. That is, for important data patterns, we can use more sensors to maintain them, and thus have higher probability to successfully collect them. We will further investigate the impact of the importance parameter in the next section through simulations.

4.3.3 Collaborative and Distributed Implementation

To guarantee success, our PNC suffers only a sublinear overhead ($\sqrt{N}$) in buffer storage and communication cost. In practice, if N is too big, even a buffer of size $\sqrt{N}+1$ might not be available at a tiny sensor. In addition, the buffer sizes of the sensors might not be identical. To overcome these problems, the sensors can work collaboratively to provide combined data segments when queried. Specifically, they can form clusters in advance, where the members of a cluster maintain different cardinalities. A cluster can then upload one highest cardinality data segment upon accessing.

We thus suggest the following collaborative and distributed implementation. We assume that the server is interested in m data patterns and, for each pattern, N_i recent data segments, $1 \leq i \leq m$. After deployment, each sensor will send a probe message to its surrounding area to form a cluster, where the number of sensors in this cluster, n, is greater than $\sum_{i=1}^{m} N_i$. Presumably, the sensors in one cluster can reside in 1 or 2 hops from each other. If n is too large, a two tier structure can be built, where each cluster in the first tier stores the data for a single pattern. A cluster head is then selected for each cluster, which distributes a storage schedule to the sensors in its cluster. When a sensor receives a server query, it first forwards this message to the cluster head; the cluster head checks whether this query is to search a data pattern associated with its own cluster. If so, the head will notify the sensor that currently has the combined data segment of the highest cardinality to upload the data; otherwise, it will forward the query to an appropriate head that is associated with the pattern for further processing.

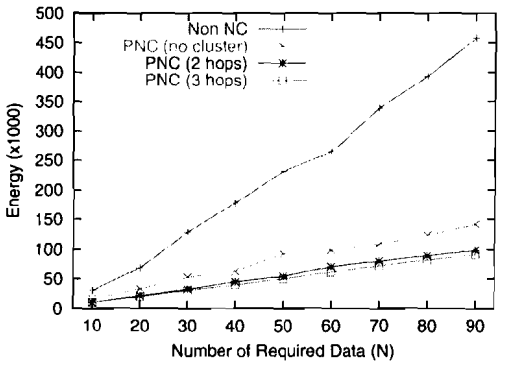


Figure 4.9: Energy consumption as a function of N for different cluster radiuses.

4.4 Performance Evaluation

4.4.1 Simulation Settings

In this section, we present our simulation results for PNC-based sensor data collection. We deploy 1000 sensors randomly into a field of 10m $\times$ 10m. The distance between the server and the sensor nodes is much larger than the distance between the sensors, and, as suggested in [49], we assume that there is a 10-fold difference. The server can thus access the data without necessarily entering deep into the sensor field, which is useful for data collection from a dangerous area. The default number of data segments that the server collects is the most recent 50 data segments ($N = 50$) and the default buffer size B is 1. We examine other possible values in our simulation as well. The linear equations in network coding are solved using the Gaussian Elimination [25], and the coefficient field is $q = 2^8$, which can be efficiently implemented in a 8-bit or more advanced microprocessor [80]. To mitigate randomness, each data point in a figure is an average of 1000 independent experiments.

4.4.2 Energy Consumption

Since NC does not have the capability of data removal, it will eventually lead to a crash of the system in continuous data collection. Therefore, in our simulations, we only study the performance of PNC and compare PNC with Non-NC.

We first compare the energy consumption of PNC with Non-NC. We use an energy consumption model of $E = d^4$ [81], where d is the transmission range. The field will

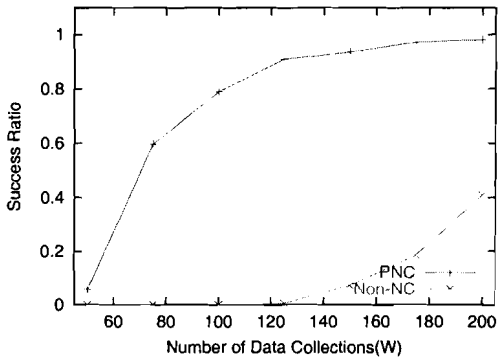


Figure 4.10: Success ratio as a function of W for PNC and Non-NC.

generate events which are of interest in an hourly basis and the sensors will record these events. Server will randomly and occasionally approach with an expected interval of 20 hours. The server is interested in the most recent $N = 50$ data pieces. It will first randomly collect 50 data segments and if some original data segments are missing (for Non-NC) or the combined data segments can not be decoded (for PNC), then the server will send additional requests one by one, until all 50 data segments are obtained.

The results of the experiments are shown in Fig. 4.9. It is clear that PNC performs better than Non-NC for different N. It can be seen that the energy consumptions are linear with respect to the number of required data (N), but the slope for PNC is much smaller. As a result, when N is greater than 50, the energy consumption with Non-NC is 3 to 4 times higher than that with PNC. The energy consumption with PNC is further reduced when clusters are employed (the cluster radius is set to 2 or 3 hops, respectively), because data segments with higher cardinalities could be uploaded from a larger aggregated buffer.

4.4.3 Performance of PNC

In our applications, when server approaches, a fast data collection is always desired. Therefore, it is better for the server to estimate the number of sensors it should query and send the query simultaneously, instead of in an incrementally fashion as previous section. Success ratio is thus a good indicator of how many sensors should be queried at once. Therefore, we use success ratio to evaluate the performance of PNC starting from this section.

PNC vs Non-NC: We revisit the performance of PNC and Non-NC using success ratio.

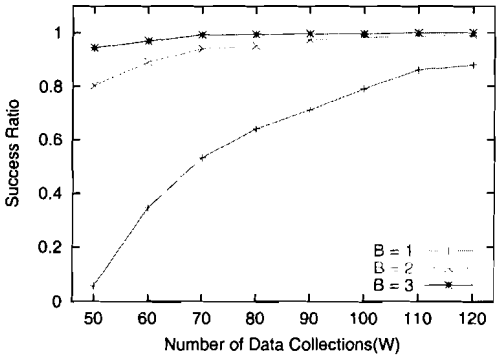


Figure 4.11: Success ratio as a function of W with different buffer size.

Fig. 4.10 shows the success ratio as a function of the number of data segments collected (W). Not surprisingly, the success ratio increases when W increases for both PNC and Non-NC, but the improvement PNC is more substantial. For example, if 100 data segments are collected, the success ratio is about 80% for PNC; for Non-NC, after collecting 200 data segments, the success ratio is still 40% only.

Effect of Buffer Size: We then increase the buffer size from $B = 1$ to 2 and 3 to investigate its impact. We require the sensors to upload the data segment of the highest cardinality for each server access. The results are shown in Fig. 4.11, where a buffer increase from 1 to 2 has a notice improvement in success ratio, and a buffer of 3 segments delivers almost optimal performance. This is not surprising because there is a higher degree of freedom for storing and uploading data in a larger buffer space.

In our analysis, we show that given $W = N + \sqrt{N}$, and $B = \sqrt{N} + 1$ the system guaranteed to decode N original data segments (ignoring linear dependency). In this case, the server has to decode $N + \sqrt{N}$ data segments, among which $\sqrt{N}$ are obsolete. An interesting question is thus: *Can we reduce the overhead for W but still guarantee an optimal success ratio?* Unfortunately, from Fig. 4.12, we can see that this unlikely happens. Among the 1000 experiments, only in 4 experiments the server successfully decodes before collecting all the $N + \sqrt{N}$ data segments. We conjecture that $\sqrt{N}$ could be a lower bound of the overhead, though it has yet to be proved.

The above two sets of results suggest that PNC works quite well for a reasonable buffer size even without extension to include obsolete data segments. Therefore, unless a guarantee

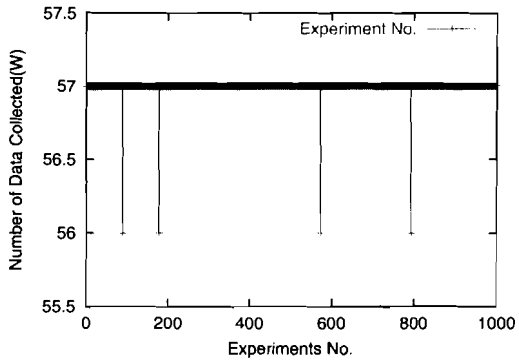


Figure 4.12: Number of communication needed (W) to successfully decode N original data segments. $N = 50$ and $N + \sqrt{N} = 57$.

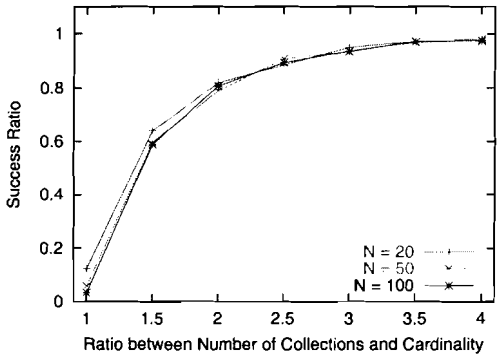


Figure 4.13: Success ratio as a function of $\lambda = \frac{W}{N}$.

is desired, the straightforward PNC is enough for most applications.

Impact of N: We then explore the impact of the cardinality N. In Fig. 4.13, we depict the decoding ratio for different number of original data segments (N=20, 50, and 100). The x-axis denotes the ratio between the number of data collected and the cardinality, i.e. $\lambda = \frac{W}{N}$. We can see from Fig. 4.13 that their differences are insignificant, and general reduce when W increases. Recall that, the performance of Non-NC decreases sharply when N increases, as shown in Table 4.2, while NC is marginally affected by N only. These simulation results thus reaffirm that PNC inherits the good scalability of NC.

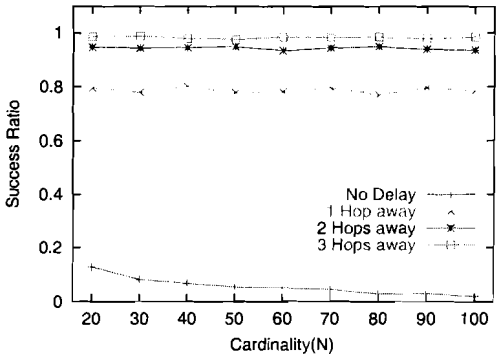


Figure 4.14: Success ratio as a function of cardinality for different cluster radiuses.

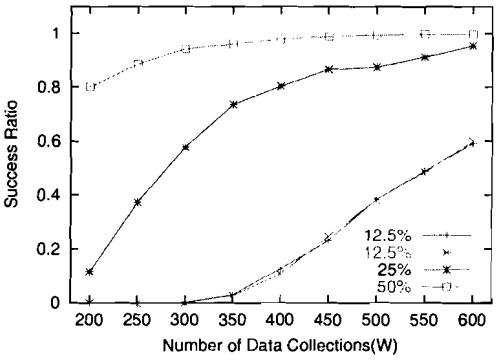


Figure 4.15: Success ratio as a function of W for multiple patterns.

4.4.4 Effect of Clustering

As discussed in section 4.3, to surpass the limited buffer size, the sensors can form clusters to achieve a larger aggregated buffer space. In Fig. 4.14, we show the success ratio for a buffer limited sensor network with different cluster radiuses, i.e., the number of hops to reach the farthest sensors, ranging from 0 to 3. When the radius is 0, there is basically no cluster and a contacted sensor has to respond to the server immediately using data from its local buffer. We can see that the success ratio significantly increase when the clustering algorithm is enabled. For a cluster radius of 2, it is already quite close to 100%.

4.4.5 Impact of Multiple Patterns

We next investigate the impact of requiring the sensor network to maintain multiple data patterns, e.g., to record more than one event of interest. Fig. 4.15 shows the success ratio for a 4-pattern scenario. To differentiate the importance of the patterns, we have assigned different number of sensors to each pattern (in this example, 12.5%, 12.5%, 25%, and 50% of the total number of sensors).

Not surprisingly, the success ratio favors data pattern with more sensors assigned. An interesting observation is that the improvement is not uniform for all the four patterns, either. It favors first for the data pattern with the largest number of assigned sensors (50%), then the pattern with the second largest number of assigned sensors (25%), and so forth. This is clearly desirable given that we want to differentiate the patterns.

4.5 Conclusion

In this chapter, we developed partial network coding (PNC), for continuous data collection in an extreme network environment. PNC allows efficient storage replacement for continuous data, where the conventional network coding is not able achieve. We proved that the performance of PNC is quite close to NC, except for a sublinear overhead on storage and communications. We also addressed a set of practical concerns toward PNC-based continuous sensor data collection.

In network coding research, it is known that the higher the cardinality is, the more the benefits we could expect. Therefore, many existing schemes have focused on achieving a full cardinality in data combination; For example, the proposals in [18][20][26][56] generally increase the cardinality by combining as much data as possible in intermediate nodes and then forward to others. Our work on partial network coding (PNC), however, shows that the opposite direction is worth consideration as well.

Chapter 5

Future Work

The area of wireless sensor networks is new and growing fast. There are diverse applications with numerous challenges and opportunities. We consider the following areas for our future efforts.

5.1 Data Filters in Sensor Networks

Many sensor applications require long term data collection but may tolerate approximate results. We will further explore the possible transmission reduction and energy saving techniques by using historical information (e.g., in previous rounds). We would like to design an adaptive filtering scheme that can provide a trade-off between system lifetime and precision guarantees. In contrast to chapter two where we use fewer sensors, the adaptive filters mainly try to reduce the number of transmissions by exploring the "similarities" of the current data value and previous data value. The data value with small enough changes will be filtered out. We will further illustrate the impact of different underlying data routing architecture. For a better understanding of our scheme, we will also validate our data filtering schemes by real world sensor deployment on mica-2 sensors. We will then draw an abstraction of our adaptive filtering scheme and build it as a separate sub-layer that is uniformly applicable to a wide range of applications with continuous queries.

5.2 Network Coding

Network coding is a new and rapidly developing field, with numerous theoretical and practical unsolved questions. Our work on network coding provides a new coding paradigm; yet better understanding is needed. Beyond the practical issues toward implementing a real PNC-based sensor network, we are interested in the following two questions: First, based on our simulations, we observe that the performance of PNC is very close to NC. We therefore suspect whether the overhead of $\sqrt{N}$ reaches the potential limit of PNC. Second, our PNC is only used for data collection. We expect that an enhancement could facilitate more complicated queries, such as Max/Min, Quantile, and Average/Summation. Given its flexibility in data management, we believe that PNC can be applied in many other applications, especially considering the recent advances of data streaming in numerous fields.

5.3 Cross Layer Interaction of Sensor Coverage and Sensor Data Collection

Sensors can provide quality coverage of the sensor field as well as valuable topological information of the sensors. We expect to explore the correlation of the sensor data in a close neighborhood. The first step will be a study of the impact of the correlation of the readings on the reduction of the number of data packets submitted. A cross layer optimization of the topological control layer and application layer is then expected; where the interaction between the two layers will be carefully designed and engineered.

Bibliography

[1] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4), July 2000.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[3] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications*, 11(6):6–28, December 2004.

[4] J. Albowitz, A. Chen, and L. Zhang. Recursive position estimation in sensor networks. In *Proc. IEEE ICNP'01*, Riverside, CA, November 2001.

[5] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *Proc. IEEE INFOCOM'01*, Anchorage, AK, April 2001.

[6] I. Beichl and F. Sullivan. The metropolis algorithm. *Computing in Science and Engineering*, 2(1):65–69, 2000.

[7] S. Bhadra and S. Shakkottai. Looking at large networks: Coding vs queueing. In *Proc. IEEE INFOCOM'06*, Bacellona, Spain, April 2006.

[8] R. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, Reading, MA, 1983.

[9] A. Boukerche, R. Pazzi, and R. Araujo. A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications. In *Proc. ACM MSWiM'04*, Venice, Italy, October 2004.

[10] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. WSNA'02*, Atlanta, GA, September 2002.

[11] N. Bulusu, J. Heidemann, and D. Estrin. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.

[12] C. Buragohain, D. Agrawal, and S. Suri. Power aware routing for sensor databases. In *Proc. IEEE INFOCOM'05*, Miami, FL, March 2005.

[13] M. Cardei, M. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. In *Proc. IEEE INFOCOM'05*, Miami, FL, March 2005.

[14] V. Cerf et al. Delay tolerant network architecture. Internet draft, http://www.ipnsig.org/reports/draft-irtf-ipnrg-arch-01.txt, July 2004.

[15] J. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Proc. IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.

[16] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. Allerton Conference on Communication, Control and Computing'03*, Monticello, IL, October 2003.

[17] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3):293–313, August 2002.

[18] S. Deb and M. Medard. Algebraic gossip: A network coding approach to optimal multiple rumor mongering. In *Proc. Allerton Conference on Communication, Control and Computing 2004*, Urbana, IL, October 2004.

[19] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Proc. 9th International Conference on Extending DataBase Technology (EDBT'04)*, Heraklion-Grete, Greece, March 2004.

[20] A. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *Proc. IPSN'05*, Los Angeles, CA, April 2005.

[21] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. ACM MOBICOM'99*, Seattle, WA, August 1999.

[22] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. ACM SIGCOMM'03*, Karlsruhe, Germany, August 2003.

[23] R. Gallager. *Low-Density Parity-Check Codes.* MIT Press, Cambridge, MA, 1963.

[24] J. Gao, L. Guibas, J. Hershburger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proc. ACM MOBIHOC'01*, Long Beach, CA, October 2001.

[25] J. Gentle. *Numerical Linear Algebra for Applications in Statistics.* Springer, 1998.

[26] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. IEEE INFOCOM'05*, Miami, FL, March 2005.

[27] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. ACM PODS'04*, Paris, France, June 2004.

[28] C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *Proc. ACM MOBICOM'04*, Philadelphia, PA, September 2004.

[29] T. He, S. Krishnamurthy, J. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proc. ACM MOBISYS'04*, Boston, MA, June 2004.

[30] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. Hawaaian International Conference on Systems Science (HICSS'00)*, Wailea Maui, HI, January 2000.

[31] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. ACM MOBICOM'99*, Seattle, WA, August 1999.

[32] M. Ho and K. Fall. Poster: Delay tolerant networking for sensor networks. In *Proc. IEEE SECON'04*, Santa Clara, CA, October 2004.

[33] T. Ho, M. Medard, J. Shi, M. Effros, and D. Karger. On randomized network coding. In *Proc. 41st Allerton Annual Conference on Communication, Control, and Computing*, Monticello, IL, October 2003.

[34] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[35] A. Howard, M. Mataric, and G. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots Special Issue on Intelligent Embedded Systems*, 13(2):113–126, 2002.

[36] A. Howard, M. Mataric, and G. Sukhatme. Mobile sensor network deployment using potential field: a distributed scalable solution to the area coverage problem. In *Proc. DARS'02*, Fukuoka, Japan, June 2002.

[37] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proc. IEEE ICDCS'02*, Vienna, Austria, July 2002.

[38] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. ACM MOBICOM'00*, Boston, MA, August 2000.

[39] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proc. ACM SIGCOMM'04*, Portland, OR, August 2004.

[40] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proc. ACM ASPLOS'02*, San Jose, CA, October 2002.

[41] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, and D. Estrin. Intelligent fluid infrustructure for embedded networks. In *Proc. ACM MOBISYS'04*, Boston, MA, June 2004.

[42] S. Karlin and H. Taylor. *An Introduction to Stochastic Modeling.* Academic Press, Orlando, FL, third edition, 1998.

[43] B. Karp and H. Kung. Greedy perimeter stateless routing for wireless networks. In *Proc. ACM MOBICOM'00*, Boston, MA, August 2000.

[44] V. Kawadia and P. Kumar. the power control and clustering in ad-hoc networks. In *Proc. IEEE INFOCOM'03*, San Francisco, CA, March 2003.

[45] B. Krishnamachari, D. Estrin, and S. Wicker. the impact of data aggregation in wireless sensor networks. In *Proc. IEEE ICDCS Workshop on Distributed Event-based System (DEBS'02)*, Vienna, Austria, July 2002.

[46] S. Kumar, T. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proc. ACM MOBICOM'05*, Cologne, Germany, August 2005.

[47] S. Kumar, T. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proc. ACM MOBICOM'04*, Philadelphia, PA, September 2004.

[48] S. Lin and D. Costello. *Error Control Coding: Fundamentals and Applications.* Printice Hall, Upper Saddle River, NJ, 2004.

[49] S. Lindsey and C. Raghavendra. Pegasis: Power-efficient gathering in sensor networks. In *IEEE Aerospace Conference Proceedings*, volume 3, pages 1125–1130, 2002.

[50] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley. Mobility improves coverage of sensor networks. In *Proc. ACM MOBIHOC'05*, Urbana-Champaign, IL, May 2005.

[51] M. Luby. Lt codes. In *Proc. IEEE FOCS'02*, Vancouver, Canada, November 2002.

[52] S. Madden and M. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. IEEE ICDE'02*, San Jose, California, February 2002.

[53] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad hoc sensor networks. In *Proc. USENIX OSDI'02*, Boston, MA, December 2002.

[54] S. Madden, R. Szewczyk, M. Franklin, and W. Hong. Supporting aggregate queries over ad-hoc wireless sensor networks. In *Proc. IEEE International Workshop on Mobile Computing Systems and Application (WMCSA'02)*, Callicon, NY, June 2002.

[55] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. ACM WSNA'02*, Atlanta, GA, September 2002.

[56] M. Medard, S. Acedanski, S. Deb, and R. Koetter. How good is random linear coding based distributed networked storage? In *Proc. NETCOD'05*, Riva del Garda, Italy, April 2005.

[57] S. Meguerdichian, F. Koushanfar, G. Gu, and M. Potkonjak. Exposure in wireless ad-hoc sensor networks. In *Proc. ACM MOBICOM'01*, Rome, Italy, July 2001.

[58] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proc. IEEE INFOCOM'01*, Anchorage, AK, April 2001.

[59] E. Miluzzo, N. Lane, and A. Campbell. Virtual sensing range. In *Proc. ACM Sensys'06*, Boulder, CO, November 2006.

[60] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturisation: a tool for investigation in control algorithms.

[61] et al N. Metropolis. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[62] S. Nath, H. Yu, P. Gibbons, and S. Seshan. Synopsis diffusion for robust aggregation in sensor networks. In *Proc. ACM SENSYS'04*, Baltimore, Maryland, November 2004.

[63] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. ACM SIGMOD'03*, San Diego, CA, June 2003.

[64] J. Plank and M. Thomason. A practical analysis of low-density parity-check erasure codes for wide area storage applications. In *Proc. International Conference on Dependable Systems and Networks (DSN)'04*, Florence, Italy, June 2004.

[65] S. Ross. *Introduction to Probability Models.* Academic Press, Boston MA, fourth edition, 1989.

[66] N. Sadagopan, B. Krishnamachari, and A. Helmy. the acquire mechanism mechanism for efficient querying in sensor networks. In *Proc. the IEEE International Workshop on Sensor Network Protocol and Applications (SNPA'03)*, Seattle, WA, May 2003.

[67] C. Schindelhauer. Mobility in wireless networks. In *invited talk of SOFSEM'06*, Merin, Czech Republic, January 2006.

[68] S. Shakkottai, R. Srikant, and N. Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In *Proc. IEEE INFOCOM'03*, San Francisco, CA, March 2003.

[69] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. In *Proc. ACM HotNets'02*, Princeton, NJ, October 2002.

[70] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: New aggregation techniques for sensor networks. In *Proc. ACM SENSYS'04*, Baltimore, MD, November 2004.

[71] G. Sibley, M. Rahimi, and G. Sukhatme. Robomote: A tiny mobile robot platform for large-scale sensor networks. In *Proc. IEEE ICRA '02*, Washington DC, May 2002.

[72] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Proc. IEEE ICC'01*, Helsinki, Finland, June 2001.

[73] E. Suli and D. Mayers. *An Introduction to Numerical Analysis.* Cambridge University Press, 2003.

[74] X. Tang and J. Xu. Extending network lifetime for precision-constrained data aggregation in wireless sensor networks. In *Proc. IEEE INFOCOM'06*, Barcelona, Spain, April 2006.

[75] G. Veltri, Q. Huang, G. Qu, and M. Potkonjak. Minimal and maximal exposure path algorithms for wireless embedded sensor networks. In *Proc. ACM SENSYS'03*, Los Angeles, CA, November 2003.

[76] D. Wang, Y. Long, and F. Ergun. A layered architecture for delay sensitive sensor networks. In *Proc. IEEE SECON'05*, Santa Clara, CA, September 2005.

[77] D. Wang, Q. Zhang, and J. Liu. Partial network coding: Theory and application in continuous sensor data collection. In *Proc. IEEE IWQoS'06*, New Haven, CT, June 2006.

[78] G. Wang, G. Cao, and T. LaPorta. A bidding protocol for deploying mobile sensors. In *Proc. IEEE ICNP'03*, Atlanta, GA, November 2003.

[79] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure-coding based routing for opportunistic networks. In *Proc. ACM SIGCOMM Workshop WTDN'05*, Philadelphia, PN, August 2005.

[80] J. Widmer and J. Boudec. Network coding for efficient communication in extreme networks. In *Proc. ACM SIGCOMM WorkShop WTDN'05*, Philadelphia, PN, August 2005.

[81] J. Wieselthier, G. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proc. IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.

[82] C. Wu and B. Li. Echelon: Peer-to-peer network diagnosis with network coding. In *Proc. IEEE IWQoS'06*, New Haven, CT, June 2006.

[83] Y. Wu, P. Chou, and S.-Y. Kung. Minimum-energy multicast in mobile ad hoc networks using network coding. *IEEE Transactions on Communications*, 53(11):1906–1918, November 2005.

[84] G. Xing, C. Lu, R. Pless, and J. O'Sullivan. Co-grid: an efficient coverage maintenance protocol for distributed sensor networks. In *Proc. ACM IPSN'04*, Berkeley, CA, April 2004.

[85] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad-hoc routing. In *Proc. ACM MOBICOM'01*, Rome, Italy, July 2001.

[86] T. Yan, T. He, and J. Stankovic. Differentiated surveillance of sensor networks. In *Proc. ACM SENSYS'03*, Los Angeles, CA, November 2003.

[87] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Proc. CIDR'03*, Asilomar, CA, January 2003.

[88] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In *Proc. IEEE INFOCOM'04*, Hong Kong, China, March 2004.

[89] Y. Yu, B. Krishnamachari, and V. Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *Proc. IEEE INFOCOM'04*, Hong Kong, China, March 2004.

[90] W. Zhang and G. Cao. Optimizing tree reconfiguration for mobile target tracking in sensor networks. In *Proc. IEEE INFOCOM'04*, Hong Kong, China, March 2004.

[91] Y. Zhu, B. Li, and J. Guo. Multicast with network coding in application layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):107–120, January 2004.

[92] Y. Zou and K. Chakrabarty. Sensor deployment and target localizatoin based on virtual forces. In *Proc. IEEE INFOCOM'03*, San Francisco, CA, March 2003.

[93] Y. Zou and K. Chakrabarty. A distributed coverage- and connectivity-centric technique for selecting active nodes in wireless sensor networks. *IEEE Transactions on Computer*, (8):978–991, August 2005.