Energy-Efficient Data Dissemination and Data Gathering in Wireless Environments

Yuxia Yao

School of Computer Engineering

A thesis submitted to the Nanyang Technological University in fulfilment of the requirement for the degree of Doctor of Philosophy

2008
Abstract

Advances in wireless communication technology and portable devices have made data dissemination and gathering increasingly popular over wireless networks. In this thesis, we investigate energy-efficient data dissemination and data gathering in wireless environments.

Data broadcast has been recognized as an important method of wireless data dissemination. In the first part of the thesis, we investigate energy-efficient indexing techniques for push-based data broadcast. To cater for energy efficiency, existing air indexing schemes for data broadcast have focused on reducing tuning time only, i.e., the duration that a mobile client stays active in data accesses. On the other hand, existing broadcast scheduling schemes have aimed at reducing access latency through non-flat data broadcast to improve responsiveness only. Not much work has addressed the energy efficiency and responsiveness issues concurrently. We propose an energy-efficient indexing scheme called MHash that optimizes tuning time and access latency in an integrated fashion. MHash reduces tuning time by means of hash-based index and supports non-flat data broadcast to reduce access latency. The design of hash function and the optimization of bandwidth allocation are investigated in depth to refine MHash. Experimental results show that under non-uniform access distribution, MHash outperforms state-of-the-art air indexing schemes in energy efficiency and achieves access latency close to optimal broadcast scheduling.

Sensor networks provide us with the means of gathering data from the physical world.
Many sensor network systems allow users to specify their interests in the sensor data by issuing queries. In the second part of the thesis, we study the processing of kNN queries in object tracking sensor networks that request k nearest object locations to a given geographical point. Due to the limited power supply of sensor nodes, energy efficiency is a major performance concern in query processing. We propose a localized scheme for kNN query processing. A two-phase search mechanism is developed to process one-shot kNN queries. To support continuous monitoring of a kNN query, a monitoring area is established along with an initial evaluation of the query using the two-phase search mechanism. We develop methods to reevaluate the kNN query based on the location updates collected from the sensor nodes in the monitoring area. Experimental results show that establishing a monitoring area for continuous kNN query processing greatly reduces energy consumption. We also analyze the optimal maintenance of the monitoring area and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. The adaptive algorithm is experimentally shown to achieve close-to-optimal performance.
Acknowledgments

First, I would like to express my deepest gratitude to Prof. Tang Xueyan, my advisor, for his patience and guidance. He showed me the world beyond the shores of this island through research. He taught me a great deal about research and writing, and was always available for discussion. His thoughtful comments on every aspect of my writing style and research methodology have tremendously helped me improve my work. He has been an amazing advisor and I am grateful I had the chance to work with him.

I am also deeply grateful for having the opportunity to work with Prof. Lim Ee Peng, for showing me how a good researcher should think and work. The discussions with him have greatly enhanced the breadth of my knowledge about the topic. At the same time, many thanks for his precious suggestions and guidance in completing this thesis. Many thanks to Prof. Zhang Jun, member of our mobile database research group, for his suggestions and discussions in the weekly seminars. From his constructive questions, I found the direction for enhancing my research work.

I have also to thank my friends for their help and support. Yu Hai and Dr. Sun Aixin, for their generous sharing of their ideas, as well as the help given in reviewing the papers I wrote. Guo Jing and Zhou Bo, my friends, were always there sharing happiness with me.

I truly appreciate the school for their unlimited support in various matters related to my research and studies. I also thank Nanyang Technological University, Ministry
of Education, Singapore for offering me the research scholarship and the opportunity to pursue a higher degree.

Last but not least, my parents, for all the love, support and sacrifice that they have given me to pursue my interest in research. For, without the both of you, all this would not be possible.
## Contents

Abstract ................................................................. i  
Acknowledgments ....................................................... iii  
List of Figures .......................................................... viii  
List of Tables ............................................................ xi  

1 Introduction .......................................................... 1  
1.1 Overview .......................................................... 1  
1.2 Energy Constraints in Wireless Environments ................ 3  
1.3 Thesis Scope ...................................................... 5  
1.4 Thesis Contributions ............................................. 7  
1.5 Thesis Organization .............................................. 8  

2 Literature Review ....................................................... 9  
2.1 Wireless Data Broadcast ........................................ 9  
2.1.1 Push-based Data Broadcast .................................. 10  
2.1.2 Pull-based Data Broadcast .................................. 16  
2.2 Data Gathering in Wireless Sensor Networks ................ 20  
2.2.1 Routing Protocols ........................................... 20  
2.2.2 Data Collection ............................................... 25  
2.2.3 Spatial Queries and Object Tracking Sensor Networks .... 31
3 Energy-Efficient and Access Latency Optimized Indexing for Wireless Data Broadcast

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>MHash Indexing Scheme</td>
<td>40</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Overview</td>
<td>40</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Hole-Free Hash Function</td>
<td>46</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Spacing between Broadcast Instances</td>
<td>52</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Bandwidth Allocation</td>
<td>54</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance Evaluation</td>
<td>62</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Experimental Setup</td>
<td>62</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Comparison with LatOpt and FlatHash</td>
<td>66</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Comparison with DistTree, Exponential, UnbalanceTree and Hybrid</td>
<td>70</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>73</td>
</tr>
</tbody>
</table>

4 $k$NN Query Processing in Wireless Sensor Networks

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>74</td>
</tr>
<tr>
<td>4.2</td>
<td>Processing One-Shot $k$NN Queries</td>
<td>78</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Preliminary Search</td>
<td>81</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Expanded Search</td>
<td>83</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Cost Model for One-Shot $k$NN Query Processing</td>
<td>85</td>
</tr>
<tr>
<td>4.3</td>
<td>Processing Continuous $k$NN Queries</td>
<td>90</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Monitoring Area Setup</td>
<td>91</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Query Reevaluation</td>
<td>93</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Maintenance of Monitoring Area</td>
<td>95</td>
</tr>
<tr>
<td>4.4</td>
<td>Scheduling Strategy to Shrink the Monitoring Area</td>
<td>98</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Optimal Schedule to Shrink the Monitoring Area</td>
<td>98</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Data Dissemination and Data Gathering in Wireless Environments ........................................... 2

2.1 An example of Broadcast Disk [XLHL02] ...................................................................................... 10

2.2 Balanced Index Tree [IVB97] ..................................................................................................... 13

2.3 Imbalanced Index Tree [CWY03] ............................................................................................... 13

2.4 Hash-Based Index ...................................................................................................................... 15

2.5 Pull-based Data Broadcast System [AM98] ............................................................................... 16

2.6 SPIN protocol. (a) Node A starts advertising its data to node B. (b) Node B responds by sending a request to node A. (c) Node B receives the requested data. (d) Node B sends out advertisements to its neighbors and, (e) & (f) who in turn send requests back to B [HKB99]. .................................. 21

2.7 Directed Diffusion protocol: (a) Query propagation, (b) Data delivery [IGE00]. ................. 22

2.8 KNN Perimeter-tree [WL04] .................................................................................................... 32

2.9 Itinerary-based Propagation [XLXM06] .................................................................................... 33

2.10 Density-aware KNN Query Processing [WCCC07] ................................................................. 34

3.1 An Example of MHash .............................................................................................................. 42

3.2 An Example Hash Function $H(k, l)$ Producing Holes in Broadcast Cycle ......................... 47

3.3 An Example Hole-Free Hash Function $H'(k, l) = (H(k, l) - 7) \mod 11$ .......................... 47

3.4 Spacing between Broadcast Instances ...................................................................................... 53

3.5 Average Access Latency vs. Replication Bound ....................................................................... 66
4.18 Performance Comparison for Continuous kNN Query Processing (Multiple Queries, $R_t = 25m$) ................................................. 120

4.19 Impact of $k$ (Random WayPoint Model) ................................................. 121

4.20 Impact of $k$ (Random Walk Model) ................................................. 122

4.21 Impact of Number of Objects (Random WayPoint Model) ................. 123

4.22 Impact of Number of Objects (Random Walk Model) ......................... 124
List of Tables

2.1 Summary of Routing Protocols ........................................ 21
2.2 Summary of Data Collection Techniques ............................ 25
3.1 Summary of Notations .................................................... 40
3.2 System Parameters ....................................................... 63
4.1 Summary of Notations .................................................... 87
4.2 System Parameters and Settings ........................................ 107
4.3 Message Types and Contents ........................................... 109

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library
Chapter 1

Introduction

1.1 Overview

Wireless computing has seen unprecedented growth lately due to both technological advances and successful deployment of wireless communication and battery-powered portable devices (e.g., laptop computers, PDAs, handphones and sensor nodes). Using mobile devices, users can easily access or receive a wide variety of information anywhere anytime [Bar99]. The information can either be sourced from the Internet or be gathered by sensor networks that monitor physical phenomena in the real world. With the recent integration of sensing and wireless communication technologies, the collection of information from the real world using wireless sensor networks has become increasingly popular [CEE+01]. Figure 1.1 shows the architecture of data dissemination and data gathering in a wireless computing environment. In general, data dissemination [Fra96] refers to the delivery of data to the consumers. Data gathering, on the other hand, refers to the collection of data from the sources.

- Wireless Data Dissemination

Data dissemination is normally based on a client-server structure. In wireless environments, base stations are often deployed to act as servers disseminating data to mobile clients through wireless channels. There are three fundamental methods
for wireless data dissemination: point-to-point delivery, pull-based broadcast and push-based broadcast. In point-to-point delivery, a dedicated channel is established between a client and the base station. The client submits requests to the base station and the base station returns the requested data to the client through the dedicated channel. If multiple clients request for the same data, multiple copies of the data have to be sent to the clients separately. In pull-based broadcast, the clients also send requests to the base station, and the base station broadcasts data to the clients according to the outstanding requests from the clients. Unlike point-to-point delivery, broadcast allows simultaneous accesses to data by multiple clients. As a result, pull-based broadcast would satisfy all clients requesting the same data by transmitting the data only once. In push-based broadcast, the base station proactively sends data to all clients without the need for the clients to submit requests. It is up to the clients to filter and select the data they want. Push-based broadcast alleviates the load on the uplink channel from the clients to the base station. This is useful in wireless environments where the downlink communication capacity from the base station to the clients is often much higher than the uplink capacity. Wireless data broadcast has been widely used in commercial applications such as Starband [Sta] and DirecWay [DIR].
CHAPTER 1. INTRODUCTION

- **Wireless Data Gathering**

Wireless sensor networks are becoming increasingly popular for data gathering. A wireless sensor network is typically comprised of a base station and a large number of sensor nodes which are distributed in an area of interest. The sensor nodes are responsible for capturing the environmental data such as temperature and humidity. They are not only capable of sensing the physical phenomena, but are also equipped with storage, computation, and wireless communication capabilities [Pot00]. The sensor nodes can communicate with one another and the base station either directly or indirectly. The base station, on the other hand, serves as a gateway for the sensor network to exchange data with external applications. Wireless sensor networks have a wide range of applications, such as habitat monitoring [CEE+01, MPSC02], structural monitoring [XRC+04], and object tracking [SSW+05]. In object tracking applications, a network of acoustic, vibration or infrared sensor nodes are often deployed to collaboratively recognize target objects, determine their locations and track their movements [CHS04, BRS03].

1.2 **Energy Constraints in Wireless Environments**

Wireless computing environments have a number of salient characteristics such as the limited power supply of mobile devices, constrained bandwidth, and unreliable wireless communication [Bar99, IB94]. The limited power supply of mobile devices is one of the most prominent characteristics. This is because the mobile devices are usually battery powered for portability reason [VBH03]. According to a report from CNET [CNE], a SONY laptop powered by a Li-ion battery with 4400mAh capacity can serve only 5.5 hours under continuous use. When the energy is exhausted, the laptop user would not be able to perform any communication or computation until the battery is recharged. A MICA mote sensor node powered by two AA batteries with about 2000mAh capacity
can approximately last for one week under full load [YG03]. When a sensor node runs out of energy, its coverage is lost. The sensor network may fail to function if the loss of coverage is substantial. As we expect only a modest improvement in battery capacity (an average annual gain of about 6%) over the next decade [Bu06], energy efficiency becomes an important criteria for developing data dissemination and data gathering techniques in wireless environments.

Mobile devices consume energy in both local processing (e.g., CPU) and communication [IVB97]. The energy consumed in communication is however much higher than that consumed in local processing. For example, a laptop with a Wavelan card requires a power rate of 1.7W to receive packets from wireless channels while the power rate for computation by CPU is only 0.25W [IVB97]. For RockWell’s WIN sensor nodes, the energy consumed in transmitting one bit is equivalent to that of processing 1500–2700 instructions [RSPS02]. Hence, to save energy, we focus on reducing the communication costs.

In wireless data broadcast, mobile clients can operate in two modes: active mode and doze mode. They can only retrieve data from broadcast channels in the active mode which has much higher rate of energy consumption than that in the doze mode. Therefore, to save energy, it is desirable for mobile clients to switch to the doze mode as much as possible when waiting for the required data. The performance of a broadcast system is often characterized by two metrics: tuning time and access latency. Tuning time refers to the duration for which the mobile client stays active. It effectively measures the energy consumed by the client in the active mode. Access latency, on the other hand, refers to how fast the mobile client can access the required data. Access latency measures the responsiveness of the system. It also largely reflects the energy consumed by the mobile client in the doze mode. Therefore, to reduce the total energy consumption of data accesses for mobile clients, it is important to reduce both access latency and tuning time.
CHAPTER 1. INTRODUCTION

In wireless sensor networks, energy is mainly consumed in exchanging messages between the sensor nodes and the base station. To reduce energy consumption, it is desirable to reduce the amount of data transmitted in the network. One straightforward approach to gather data from the sensor network is to let every sensor node send all raw readings to the base station. It is up to the base station to process and filter the sensor data. Though simple, this approach would incur a significant number of message transmissions and hence consume large amount of energy. An alternative approach is to store the acquired data locally at the sensor nodes. The sensor nodes send the data to the base station only when requested by an application. In this way, only the relevant data are gathered and processed. Since sensor nodes often have the capability to perform computation locally, some processing functions such as aggregating data and eliminating redundant data can be pushed into the sensor network. This is known as in-network processing [IGE00, MFHH02, MF02]. In-network processing has the potential to reduce energy consumption by replacing more expensive communication operations with relatively cheaper computation operations.

1.3 Thesis Scope

In this thesis, we investigate two important issues in wireless data dissemination and data gathering. Our research goal is to design energy efficient methods to disseminate and gather data in wireless environments.

In the first part of the thesis, we investigate push-based data broadcast for wireless data dissemination. Our objective is to design energy efficient broadcast schemes to reduce the total energy consumption of data accesses for mobile clients. As mentioned earlier, to reduce total energy consumption, it is important to reduce both access latency and tuning time. The tuning time can be reduced by means of air indexing [XLHL02, ZXLL04, IVB97, IVB94]. The basic idea is to interleave index information with data
in the broadcast schedule to assist the clients in locating the required data. Following
the links in the index structure, the clients alternate between the active and doze modes
until the data arrive. Most existing air indexing schemes are based on the tree structures
that are originally designed for random-access media such as disks. To apply tree-based
index on wireless channels which are sequential-access media, data must be broadcast in
the order of key and with the same frequency [CWY03, IVB97, SV96]. This sacrifices
responsiveness (i.e., access latency) when client accesses are not uniformly distributed
among data items. In the presence of non-uniform access distribution, popular data
items should be broadcast more frequently than unpopular items to reduce average access
latency [XLHL02]. This is known as non-flat data broadcast. However, most existing
non-flat broadcast scheduling schemes do not consider air indexing [AAFZ95, VH99].
Without index, the clients have to continuously stay active and monitor the broadcast
channel until the required data arrive. This consumes significant amount of battery
power and compromises energy efficiency. Different from the existing work, we aim at
optimizing energy efficiency and responsiveness in an integrated fashion.

In the second part of the thesis, we study the processing of kNN queries in object
tracking sensor networks. Many sensor network systems allow users to specify their in-
terests in the sensor data by issuing queries. These queries can either be one-shot queries
which complete once the results are derived, or continuous queries which require con-
tinuous return of results over a period of time. Due to the geographically distributed
deployment of sensor nodes, spatial information plays an important role in the repre-
sentation of sensor data. Thus, it is natural to collect data from the sensor network by
specifying spatial conditions in the queries. Existing work on spatial query processing
in sensor networks has focused on one-shot queries that request the readings from the
sensor nodes based on the node locations, e.g., from the sensor nodes in a geographic
region [XLXM06]. In addition to the locations of sensor nodes, the data captured by
the sensor nodes may also include spatial information. For example, in object tracking sensor networks, the object locations detected by the sensor nodes are represented by geographical coordinates. Different from the existing work, we consider spatial queries that specify spatial conditions on the collection of object locations detected by object tracking sensor networks. Our objective is to collect a given number of $k$ detected object locations nearest to a specified geographical point in an energy-efficient manner. We investigate both one-shot and continuous queries.

1.4 Thesis Contributions

The main contributions of the thesis are summarized below:

- We propose a novel indexing structure called MHash for wireless data broadcast. MHash constructs the broadcast schedule using a two-argument hash function for indexing purpose. The two-argument nature of the hash function allows each data item to be mapped to an adjustable number of slots in the schedule, thereby enabling non-flat data broadcast. Under this framework, we further investigate the issues of generating hash functions that produce broadcast schedules free of unoccupied slots; improving access latency by properly spacing the broadcast instances of each data item in the schedule; and optimally allocating the bandwidth among data items. Experimental results show that under non-uniform access distribution, the proposed indexing scheme outperforms state-of-the-art air indexing schemes in energy efficiency and achieves access latency close to optimal broadcast scheduling.

- We propose a localized scheme for processing $k$NN queries in object tracking sensor networks. In our scheme, the sensor data are stored locally at the detecting sensor nodes and the queries are processed in-network. A two-phase search mechanism is developed to process one-shot $k$NN queries. To support continuous monitoring
of the $k$ nearest objects, the two-phase search mechanism is first used to conduct an initial evaluation of the query. A monitoring area is then setup in the network so that only the relevant location updates are collected to determine the $k$ nearest objects. We develop methods to reevaluate the $k$NN query based on the location updates collected from the sensor nodes in the monitoring area. Experimental results show that establishing a monitoring area for continuous $k$NN query processing greatly reduces energy consumption. Due to object movement, the monitoring area needs to be expanded and shrunk on the fly. We analyze the optimal maintenance of the monitoring area and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. The adaptive algorithm is experimentally shown to achieve close-to-optimal performance.

1.5 Thesis Organization

This thesis consists of five chapters. The remainder of the thesis is organized as follows. Chapter 2 reviews the existing data dissemination and data gathering methods in wireless environments. Chapter 3 presents the MHash indexing structure for wireless data broadcast. Chapter 4 studies the processing of $k$NN queries in object tracking sensor networks. Finally, Chapter 5 summarizes the research contributions and concludes the thesis.
Chapter 2

Literature Review

In this chapter, we review existing work on data dissemination and data gathering in wireless environments. Research on data dissemination using wireless data broadcast is described in Section 2.1. Data gathering techniques in wireless sensor networks are discussed in Section 2.2.

2.1 Wireless Data Broadcast

Wireless data broadcast can be categorized into two types of broadcast models: push-based data broadcast and pull-based data broadcast. In push-based data broadcast, the base station proactively and repeatedly broadcasts data items in cycles. In pull-based data broadcast, on the other hand, the base station disseminates data according to the outstanding requests submitted by the clients. The performance of wireless data broadcast is normally measured by two metrics: access latency and tuning time. Access latency refers to how fast the mobile client can access required data, and tuning time refers to the duration that the mobile client stays active. While the access latency is mainly determined by the scheduling of data items in the broadcast, the tuning time is primarily determined by air indexing techniques. In the following, we shall discuss the two broadcast models in detail.
2.1.1 Push-based Data Broadcast

The simplest push-based broadcast scheme is flat broadcast, in which all data items are scheduled in a round-robin manner. Flat broadcast, however, shows poor access latency when data accesses are not uniformly distributed among the data items. Acharya et al. [AAFZ95] proposed a hierarchical architecture called broadcast disk that considers non-uniform data access patterns. In this approach, data items with similar access probabilities are grouped together to form logical disks. Each disk is assigned a relative broadcast frequency: to reduce the average access latency over all data items, the disks with more popular items are assigned higher frequencies. The broadcast schedule is then constructed by circularly picking up items from the disks based on their relative broadcast frequencies. This is achieved by further dividing each disk into smaller, equal-size units called chunks.

![Figure 2.1: An example of Broadcast Disk [XLHL02]](image)

Figure 2.1 illustrates an example in which seven data items are divided into three groups \((a), (b, c), (d, e, f, g)\) based on their access probabilities and assigned to three separate disks in the broadcast. Suppose that item \(a\) is more popular than items \(b\) and \(c\)
while items $d$, $e$, $f$ and $g$ are the least popular items. The three groups of data items are assigned to three disks $D_1$, $D_2$ and $D_3$ respectively. Assume that the relative broadcast frequencies of $D_1$, $D_2$ and $D_3$ are $4:2:1$. Each disk is further divided into chunks, the number of which is inversely proportional to the relative broadcast frequency of the disk. In the example of Figure 2.1, there is only one chunk $C_{1,1}$ in disk $D_1$; disk $D_2$ is divided into two chunks $C_{2,1}$ and $C_{2,2}$; and disk $D_3$ is divided into four chunks: $C_{3,1}$, $C_{3,2}$, $C_{3,3}$ and $C_{3,4}$. To construct the broadcast cycle, one chunk is broadcast from each disk at a time, and all the chunks are cycled through sequentially over all the disks [XLHL02]. As a result, the broadcast cycle is organized as $C_{1,1}$, $C_{2,1}$, $C_{3,1}$, $C_{1,1}$, $C_{2,2}$, $C_{3,2}$, $C_{1,1}$, $C_{2,1}$, $C_{3,3}$, $C_{1,1}$, $C_{2,2}$, $C_{3,4}$. In the broadcast cycle, item $a$ is broadcast 4 times while items $b$ and $c$ are broadcast 2 times each and items $d$ to $g$ are broadcast only once each.

Vaidya et al. [VH99] studied optimal broadcast scheduling for non-uniform data accesses. It has been shown that, the average access latency of all data items is minimized when each item is allocated a broadcast frequency proportional to the square-root of its access probability, and when the broadcast instances of each item are equally spaced in the broadcast schedule.

The above scheduling algorithms do not consider air indexing. Without indexing, the clients have to continuously tune to the wireless channel for required data. Hence, the tuning time equals the access latency, resulting in high energy consumption for the clients. To cater for limited battery power, a number of air indexing techniques have been proposed to assist the clients in predicting the arrival times of required data and to support selective tuning to the wireless channel [LL96, IVB97, CWY03, SV96, XLT04]. Lee et al. [LL96] proposed a signature-based indexing method. In this approach, a broadcast cycle is divided into a number of frames. Each frame is preceded by a signature of its data items in the broadcast schedule. This allows the client to check whether a required item is in the frame by examining the signature only. However, the signature
does not indicate the arrival times of data items. Thus, when a match is found in the signature, the data items in the corresponding frame have to be searched sequentially. Moreover, since a signature does not contain global information about the broadcast, data accesses require sequential scans of the signatures in different frames.

Imielinski et al. [IVB97] applied the tree-based index designed for traditional disk storage to wireless data broadcast. In this approach, the data items are ordered based on their key values and divided into buckets — the smallest accessible unit of broadcast. A balanced index tree is built to index the buckets based on the key values. The fanout degree of an index node in the tree depends on the size of the bucket and the size of the key value. The index nodes are interleaved with the data buckets to construct the broadcast cycle. To access data, starting from the root of the index tree, the client follows the links in the index tree and tunes to selected index nodes to locate the required data. Figure 2.2(a) shows an example of a balanced index tree for 4 data buckets $R_1$ to $R_4$, assuming that each index node in the tree has a fanout degree of 2. The resulting broadcast cycle is shown in Figure 2.2(b). Consider a client that tunes to the broadcast channel for a data item in bucket $R_4$. Starting from the root node $I$, the client selectively tunes to index node $a_2$, and then data bucket $R_4$. Thus, the tuning time is 3 buckets. As can be seen, in the tree-based index, the tuning time is dependent on the height of the tree. However, the tree-based index may increase the access latency because the search always starts from the root node of the tree. To reduce the access latency, the index nodes at the upper levels of the tree can be replicated in a broadcast cycle so as to cut down the waiting time for the clients to access the index nodes [IVB97].

Chen et al. [CWY03] and Shivakumar et al. [SV96] further showed that the average tuning time can be reduced by an imbalanced index tree for non-uniform data access patterns. In their approach, popular data items are put closer to the root of the index tree than unpopular items. Consider again the example of Figure 2.2. Suppose the
access probabilities for $R_1$, $R_2$, $R_3$ and $R_4$ are 0.6, 0.3, 0.05 and 0.05 respectively. An imbalanced distributed index tree for these four data buckets can be constructed as shown in Figure 2.3(a). The corresponding broadcast cycle is shown in Figure 2.3(b). In this imbalanced index tree, starting from the root node $I$, the tuning time for a client to access $R_1$, $R_2$, $R_3$ and $R_4$ are 2, 3, 4, and 4 buckets respectively. Thus, the average tuning time is given as $(2 \times 0.6 + 3 \times 0.3 + 4 \times 0.05 + 4 \times 0.05) = 2.5$ buckets. On the other hand, in the balanced index tree of Figure 2.2, starting from the root node $I$, the tuning time for a client to access $R_1$ to $R_4$ are all 3 buckets. Hence, the average tuning
time is $3 \times (0.6 + 0.3 + 0.05 + 0.05) = 3$ buckets. Compared to the balanced index tree, the imbalanced index tree results in lower average tuning time.

Xu et al. [XLT04, XLT+06] extended tree-based indexes by constructing multiple index trees that share links. The resultant index structure allows searching to start from anywhere in the broadcast. However, most tree-based indexes are only applicable to flat broadcast because they require data items be ordered based on their key values in the broadcast schedule. Non-flat broadcast schedules generally do not have this property. Thus, when applied to non-flat broadcast, the indexes can only be built locally for short segments of broadcast, where each segment holds a sequence of data items with increasing key values. These segments have to be searched sequentially in data accesses [HLL01, HLL99, XLT04, XLT+06, YT97]. As a result, the effectiveness of indexing diminishes significantly.

Besides tree-based indexes, hash functions can also be used for indexing purpose in wireless data broadcast. Imielinski et al. [IVB94] used a hash function to map a data item based on its key value to a slot in the broadcast schedule. In the case when the hash function is not collision-free, multiple data items would be hashed to the same slot. If one slot in a broadcast cycle can hold only one data item, an overflow situation may arise in the presence of collisions. Imielinski proposed to resolve collisions by pushing the overflow data items into succeeding slots and pushing forward the items originally hashed to the succeeding slots. To facilitate data accesses, each slot would include the hash function and a distance pointer which leads to the actual slot containing the data item hashed to the slot. Figure 2.4 shows an example of the hash-based index. The data items hashed to the slots are shown below the slots. Suppose that a client requiring item $E$ makes an initial probe at slot 1 and reads the hash function. According to the hash function, item $E$ is hashed to slot 2. The client then probes slot 2. However, since there is an overflow situation in slot 1, the items hashed to slot 2 are pushed forward
CHAPTER 2. LITERATURE REVIEW

by 2 slots (i.e., they are located in slot 4 onwards). So, the client reads the distance pointer recorded in slot 2, which is 2, and determines to tune to slot $2 + 2 = 4$. Starting from slot 4, the client sequentially scans the broadcast data items until it locates the required item $E$. As can be seen, the above collision resolution method, similar to linear probing, introduces an extra tuning time of one slot to all the items that are hashed to the succeeding slots and pushed forward.

![Diagram of Hash-Based Index]

Figure 2.4: Hash-Based Index

A salient feature of the hash-based index is that it eliminates the need to broadcast index structures (e.g., trees) — only a hash function is broadcast together with data. While the size of tree-based index structure and hence its broadcasting overhead normally increases with the number of data items, the broadcasting overhead of a hash function is largely independent of the number of data items. However, the above hash-based indexing scheme neither considered non-flat broadcast nor addressed the problem of producing broadcast schedules free of unoccupied slots. In this thesis, we propose a novel indexing scheme using a two-argument hash function. Our proposed scheme gracefully incorporates hash-based index with non-flat data broadcast. It naturally reduces both access latency and tuning time for popular items. We also propose a new chaining method for collision resolution to reduce the penalty in tuning time and investigate how to generate broadcast schedules without any unoccupied slot.
2.1.2 Pull-based Data Broadcast

A pull-based data broadcast system consists of a downlink broadcast channel and an uplink request channel as shown in Figure 2.5. The clients send their requests to the base station through the uplink channel and monitor the broadcast channel for the requested data items. Client requests are queued up at the base station upon arrival. The base station broadcasts the requested data items through the broadcast channel based on the outstanding requests. A key issue in pull-based data broadcast is the scheduling algorithm that selects the requested data items to broadcast on the broadcast channel [DAW86, AF99, AM98, XTL06].

A simple scheduling algorithm is First-Come-First-Served (FCFS), in which the base station broadcasts the data items according to the arrival order of their requests. Though simple, FCFS results in high average response time in case of non-uniform data access patterns. Three scheduling algorithms have been proposed to reduce the average response time [DAW86]:

- Most Request First (MRF): The data item with the maximum number of pending requests is broadcast first.
CHAPTER 2. LITERATURE REVIEW

- MRF Low (MRFL): MRFL is similar to MRF. However, under MRFL, the base station estimates the access probability for each data item based on the request history and breaks ties in favor of the data item with a lower access probability.

- Longest Wait First (LWF): The data item with the largest total waiting time of pending requests (i.e., the sum of the time that all pending requests for the item have been waiting) is broadcast first.

It has been observed that, when the number of requests is small, all the four schemes perform similarly [DAW86]. When the number of requests increases, and the access probabilities are the same for all items, the average response time for MRF is the lowest. When the access probability follows a non-uniform distribution, e.g., zipf distribution [Zip49], LWF has the best performance in terms of response time. However, the computational overhead of LWF algorithm is high when the size of the database at the base station is large. This is because the base station has to recalculate the total waiting time of pending requests for each data item in order to decide which item to broadcast next. Aksoy and Franklin proposed a low-overhead approach called $R \times W$ [AF99] as an approximation of LWF. The key idea is to first broadcast the item with the maximum value of $R \times W$, where $R$ is the number of outstanding requests for the data item, and $W$ is the time that the oldest outstanding request for that data item has been waiting. $R \times W$ yields a close performance to LWF and considerably reduces the computational overhead compared to LWF.

The above work assumes that all data items have the same size and uses the average response time as the performance metric for scheduling. To better evaluate the performance of scheduling algorithms when the data items have different sizes, a new metric called stretch is introduced in [AM98]. Stretch is defined as the ratio of the response time of a request to the service time of the requested data item, where the service time of a
data item is the time needed by the base station to broadcast it. A number of scheduling algorithms have been proposed to reduce stretch [AM98]. Longest Total Stretch First (LTSF) is such an algorithm which chooses to first broadcast the data item with the largest total current stretch over all of its pending requests. Here, the current stretch of a pending request is the ratio of the time the request has been waiting to the service time of the requested data item.

Xu et al. [XTL06] studied the pull-based broadcast scheduling when the requests have deadlines. They proposed a scheduling algorithm called SIN-α which integrates the urgency and the popularity of the requests. Their objective is to minimize the drop rate of the requests which is defined as the ratio of the number of requests missing their deadlines to the total number of requests.

Indexing techniques can be applied to pull-based data broadcast to reduce the energy consumption for mobile clients. Lee et al. [LCZ03] proposed to use index hints in the broadcast schedule. Specifically, the index hints, including the keys of the data items and their expected broadcast times, are interleaved with the data buckets in the broadcast schedule. In this way, the clients can predict the arrival times of the items when receiving the index hint. The index hints are constructed by estimating the broadcast schedule in the future based on the request history. While the above work allocates fixed sizes to both index hints and data buckets, Huang et al. [HP05] proposed an indexing scheme in which the sizes of index and data buckets are dynamically adjusted according to the change in the data access pattern.

Pull-based broadcast explicitly considers the requests made by the clients and dynamically adapts the broadcast schedule. However, in pull-based broadcast, sending requests would consume much energy of the mobile clients and also add demand on the scarce bandwidth of the uplink channel. When the number of requests increases, there might be congestion in the uplink channel that would lead to high response times of requests.
Push-based broadcast, on the other hand, removes the need for mobile clients to submit requests by proactively broadcasting all data items. Nevertheless, it may not adapt well to large databases. When the number of data items is large, the access latency for push-based broadcast would be high. To benefit from their advantages, push-based and pull-based broadcasts can be combined to form hybrid broadcast schemes.

Acharya et al. [AFZ97] proposed a hybrid approach called Interleaved Push and Pull (IPP) that combines the push-based broadcast disk method [AAFZ95] with a pull-based method. In this approach, the available bandwidth is divided into two portions with one portion used as broadcast channel for push-based broadcast and the other portion used as the uplink channel for the clients to submit their requests to the base station. All items in the database are proactively broadcast in the broadcast channel using the broadcast disk approach. A client wanting to access a data item first monitors the broadcast channel for a given period of time. If the wanted item is broadcast in the period, the client retrieves it. Otherwise, if the wanted item is not broadcast in the period, the client sends a request to the base station. Upon receiving the requests, the base station would adjust the broadcast schedule by putting the requested data items at the head of the broadcast cycle. While the above work broadcasts all data items through the broadcast channel, Stathatos et al. [SRB97] proposed to divide the items into two sets according to their access probabilities. Push-based broadcast is used for the set of data items with higher access probabilities and pull-based broadcast is used for the set of data items with lower access probabilities. The base station dynamically adjusts the sets of data items for push-based broadcast and pull-based broadcast based on their access probabilities. To reduce the energy consumption of mobile clients in hybrid broadcast, Datta et al. [DVCK99] applied the distributed tree structure proposed by Imielinski [IVB97] (as discussed in Section 2.1.1) to index the data items.
Chapter 2. Literature Review

2.2 Data Gathering in Wireless Sensor Networks

The most important characteristic of wireless sensor networks is the limited battery power of sensor nodes. Replacing/charging the batteries is inconvenient in many situations as sensor networks are often deployed to operate in an unattended manner. Hence, energy efficiency is the primary concern in the design of data gathering techniques in wireless sensor networks. In the following, we discuss energy efficient routing and data management strategies in wireless sensor networks.

2.2.1 Routing Protocols

Routing in wireless sensor networks is a challenging issue. Firstly, wireless sensor networks are normally comprised of a large number of sensor nodes. It is not feasible to assign global identifiers to all sensor nodes since the overhead of maintaining these identifiers is high. Hence, traditional address-based routing schemes (e.g., IP-based routing) are not applicable to sensor networks. Secondly, most sensor network applications require data to be transmitted from multiple nodes (called data sources) to a particular node (called data sink). The data generated by sensor nodes in proximity may be redundant. Such redundancy needs to be exploited by the routing protocols to reduce the network traffic. Finally, since the sensor nodes have limited power supply, the routing scheme must be energy efficient. A number of routing protocols have been proposed for sensor networks to tackle the above challenges. These routing protocols can be roughly classified into three categories: data-centric [HKB99, IGE00, BE02, YLC+02], hierarchical [HCB00, LR02] and location-based [KK00, YEC01]. Table 2.1 summarizes the features of these routing protocols.

In data-centric routing, the sink requests data by issuing queries. The objective of data-centric routing is to route the queries to meet the matching data and/or to route the data to meet the matching queries. SPIN is a data-centric routing protocol that
Table 2.1: Summary of Routing Protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-centric routing</td>
<td>Queries/data are flooded to look for data/queries with matching attributes.</td>
</tr>
<tr>
<td>Hierarchical routing</td>
<td>Sensor nodes are organized into clusters. A cluster head communicates with other cluster heads on behalf of the nodes within its cluster.</td>
</tr>
<tr>
<td>Location-based routing</td>
<td>The location information of the sensor nodes is used to make routing decisions.</td>
</tr>
</tbody>
</table>

Figure 2.6: SPIN protocol. (a) Node A starts advertising its data to node B. (b) Node B responds by sending a request to node A. (c) Node B receives the requested data. (d) Node B sends out advertisements to its neighbors and, (e) & (f) who in turn send requests back to B [HKB99].

routes the data to meet the matching queries [HKB99]. In this approach, a source node that acquires data advertises the attribute of the data to its immediate neighbors. If the neighbor has no interest in the data, it ignores the advertisement. Otherwise, it sends a request message back to the source node for the data and further advertises the attribute of the data to its neighbors. Figure 2.6 illustrates this process. The disadvantage of SPIN
is that, data delivery to remote nodes is not guaranteed when none of the neighbors of a
source node is interested in the advertised data.

![Directed Diffusion protocol](image)

Figure 2.7: Directed Diffusion protocol: (a) Query propagation, (b) Data delivery [IGE00].

Instead of advertising the data, Intanagowiwat *et al.* [IGE00] proposed a data-centric
routing mechanism called Directed Diffusion that advertises the queries to the entire
network. Starting from the sink, a query is flooded over the entire network. When a
source node having data matching the query is found, it sends the data back to the sink
through a reverse path of the query that has lower latency. Each sensor node along the
path is able to remove redundant data from different sources to reduce the amount of
network traffic. Figure 2.7 summarizes the Directed Diffusion approach.

A rumor routing mechanism is proposed in [BE02] to route both the queries and
the data over the network. Specifically, when a source node acquires any data, packets
containing information about the data are generated and randomly propagated in the
network to announce the existence of data. The sensor nodes visited by the agents cache
the routing paths to the source node. A query is routed randomly in the network until
it reaches a node that has cached the path for the requested data and is then directed to
the data source.

Instead of random propagation, a grid structure is constructed in the two-tier ap-
proach of [YLC+02] to assist data and query dissemination. When acquiring any data,
a source node proactively builds a grid structure in the sensing field and sets up the routing information at a set of dissemination nodes which are the nodes closest to grid points. To route a query to the source of the requested data, the query is first flooded within the local grid cell of the sink node to reach a dissemination node. Then, the dissemination node forwards the query to the data source, possibly through other dissemination nodes, based on the routing information. Finally, the requested data are sent back to the sink node along the reverse path of the query. Zhang et al. [ZCP03] proposed a similar approach to route the query to find the matching data based on a ring-based index structure.

Hierarchical routing protocols organize the sensor nodes into clusters. A cluster head communicates with other cluster heads on behalf of the nodes within its cluster. The cluster head may also remove redundant data to reduce the amount of data transmitted to the sink. LEACH (Low-Energy Adaptive Clustering Hierarchy) is a hierarchical routing protocol for wireless sensor networks [HCB00]. In LEACH, to construct the clusters, the sensor nodes periodically elect a subset of them to become the cluster heads which in turn advertise to the entire network to recruit cluster members. The non-cluster-head nodes determine which clusters to join based on the received signal strength of the advertisements. Since inter-cluster communication usually has longer transmission distance and hence higher energy consumption than intra-cluster communication, the sensor nodes in LEACH rotate the role of cluster head in order to balance the energy consumption.

PEGASIS [LR02] improves LEACH by organizing the sensor nodes into a chain instead of clusters. The transmission distance between neighboring sensor nodes on the chain is much shorter than the distance of intra-cluster transmission in LEACH. In PEGASIS, a sensor node only transmits data to and receives data from a neighbor node on the chain. The sensor data are eventually gathered at a selected node on the chain who
is responsible for sending the data to the sink. To balance the energy consumption, the
sensor nodes take turns to communicate with the sink.

Sensor nodes are spatially distributed in the network and are usually location-aware.
Location-based routing protocols make use of the location information of the sensor nodes
for routing purpose. GPSR is a location-based routing protocol that routes a message to
a destination node based on the location of the destination node [KK00]. At each hop,
the sensor node receiving the message makes a greedy forwarding decision based on the
locations of all immediate neighbors that are geographically closer to the destination node
than itself. The neighbor node that is closest to the destination node is selected as the
next hop. When the message reaches a region where greedy forwarding is impossible (i.e.,
a node receiving the message does not have any neighbor node closer to the destination
node than itself), GPSR recovers by routing around the perimeter of the region.

GPSR does not consider the energy levels of the sensor nodes when making routing de-
cisions. Yu et al. [YEG01] proposed energy-aware and geographically-informed neighbor
selection heuristics to route a message to the destination. In this approach, each sensor
node keeps an estimated cost and a learned cost of reaching the destination through its
neighbors. The estimated cost is a combination of the energy level of a sensor node and
its distance to the destination. The learned cost is a refinement of the estimated cost to
account for the cost of routing around the "holes" in the network. A "hole" exists when
all neighbors of a node have higher estimated costs than that of the node. If no "hole"
exists, the learned cost is equivalent to the estimated cost. In the process of routing, each
intermediate node forwards the message to the neighbor node with the smallest learned
cost. In our work of kNN query processing in wireless sensor networks, the location-based
routing protocol GPSR is employed as the underlying routing protocol.
2.2.2 Data Collection

Sensor networks are widely used for data gathering and monitoring applications, e.g., habitat and environmental monitoring [CEE+01, JRM]. These applications usually require the sensor nodes to continuously monitor the surroundings and send their readings to a sink node. The applications may either request the readings of individual sensor nodes or an aggregate form of the sensor readings over the network (e.g., average temperature reading). A simple approach of data collection is to let the sensor nodes periodically send their raw readings to the sink node. The data are then aggregated at the sink node if needed. Though simple, this approach induces high communication cost and network traffic. To reduce the network traffic, a number of approaches have been proposed. As shown in Table 2.2, these approaches can be roughly classified into three categories: (1) in-network data aggregation; (2) temporal compression; (3) spatial compression.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-network Aggregation</td>
<td>Aggregate data as they flow through the sensor nodes to reduce the network traffic.</td>
</tr>
<tr>
<td>Temporal compression</td>
<td>Suppress the transmission of a new reading from a sensor node if the new reading has not changed significantly compared to previously transmitted readings.</td>
</tr>
<tr>
<td>Spatial compression</td>
<td>Suppress the transmission of a new reading from a sensor node if the new reading can be inferred from the readings of the neighboring nodes.</td>
</tr>
</tbody>
</table>

In-network data aggregation has been recognized as an important way to reduce the amount of data transmitted in the network. The general idea is to aggregate data as they flow through the sensor nodes. TinyDB is a sensor database system that allows users to pose SQL-style queries at the sink node. It implements an in-network aggregation service called TAG that supports the five standard relational aggregations including...
SUM, COUNT, AVERAGE and MAX/MIN [MFHH02]. In TAG, the sensor nodes are first organized into a routing tree rooted at the sink node. To achieve this, the query is flooded over the network starting from the sink node. According to some criteria, each node chooses one of its neighbors from which it receives the query as its parent in the routing tree. On completing the tree construction, the sensor nodes would send data to the sink node via the routing tree at each sampling interval. Specifically, each leaf node sends the local reading to its parent. According to the given aggregate operator, each intermediate node aggregates its own reading with those received from its children and sends a partial aggregate result to its parent. This process continues until the partial aggregate results reach the sink node where the final aggregate result is computed. In this way, the amount of data sent by an intermediate node to its parent is kept constant independent of the size of the subtree rooted at the intermediate node. For example, to collect the maximum sensor reading over the network, the partial aggregate result derived at each intermediate node is the maximum reading among its own reading and those received from its children. To collect the average sensor reading over the network, the partial aggregate result derived at each intermediate node includes two values: the sum of the readings at all of its descendants and the total number of its descendants.

Median and percentile queries differ from the five standard relational aggregations. To derive the exact median sensor reading or a given percentile sensor reading, the readings of all sensor nodes must be sent to the sink node. Shrivastava et al. [SBAS04] proposed an approximate data aggregation scheme for median and percentile queries. They designed a data structure called q-digest to describe the distribution of a data set. Q-digest allows the median or any percentile value of the data set to be derived with a given precision guarantee. Q-digest can be used to represent the partial aggregate result to support in-network aggregation for median and percentile queries. That is, the partial aggregate result derived at each intermediate node is a q-digest for all the readings of its descendants.
CHAPTER 2. LITERATURE REVIEW

descendants. It can be computed by aggregating the local reading at the intermediate node and the q-digests received from its children.

In many applications, sensor readings show strong temporal correlations. Based on this observation, temporal compression can be used in data collection to reduce the network traffic in two scenarios. In the first scenario, if the change in the reading of a sensor node does not affect the query result derived at the sink node, the report of the new reading can be suppressed. An example of this scenario is exemplary aggregations like MAX and MIN. Different from summary aggregations like SUM and AVERAGE whose results involve readings from all sensor nodes, the results of MAX/MIN aggregations only include the reading of one sensor node. Hence, it is possible to suppress the updates of the readings captured by other sensor nodes without affecting the correctness of the aggregation result. For example, suppose that at one sampling interval, the maximum reading over the entire network is 10 which is captured by node A. At the following sampling interval, A's reading remains at 10 while the reading of another node B changes from 1 to 5. Then, B's reading does not have to be reported in order to evaluate the new maximum sensor reading. Silberstein et al. [SMY06] proposed a threshold-based scheme for monitoring MAX/MIN queries. They developed a number of algorithms to set thresholds at the sensor nodes to prevent those that are unlikely to have the maximum or minimum reading from reporting. A sensor node sends a new reading to its parent only if the new reading becomes higher (for MAX queries) or lower (for MIN queries) than the threshold. Wu et al. [WXTL06] proposed a similar approach for monitoring top-k queries which request the list of sensor nodes with the highest (or lowest) k readings. In this approach, a window-based filter is set at each sensor node to suppress unnecessary updates of sensor readings. A sensor node reports a new reading only if the new reading changes beyond the filtering window.

In the second scenario of temporal compression, if the application can tolerate some degree of inaccuracy in data collection, not all sensor nodes need to report their readings
Chapter 2. Literature Review

to the sink node at each sampling interval. Only the updates that are necessary to guarantee the desired level of accuracy need to be sent [SBLC03, DKR04, TX06]. Sharaf et al. [SBLC03] proposed a framework called TiNA which allows users to issue queries with precision requirements. The precision requirement can be specified by a quantitative error bound, e.g., the average temperature reading of all sensor nodes within an error bound of 1°C. In TiNA, the error bound is uniformly distributed to all the leaf nodes in the routing tree. At each sampling interval, a new reading at a leaf node is reported only if the reading differs from the last reported reading by more than the allocated error bound. Each intermediate node aggregates the new reading of its own and the latest readings reported by its children. It then sends the new partial aggregate result to the parent if the new result is different from the last reported result. Deligiannakis et al. [DKR04] optimized the error bound allocation to further reduce the updates of sensor readings in the network. The error bounds are adaptively redistributed to those nodes that would benefit most from error bound expansion. In addition to leaf nodes, the error bounds are also allocated to intermediate nodes to perform temporal compression at these nodes. Each intermediate node sends the partial aggregate result to its parent only if the result changes by more than the allocated error bound. While the above two studies focused on reducing the total network traffic, Tang et al. [TX06] considered the energy levels of individual sensor nodes and proposed an adaptive scheme to adjust their error bounds in order to balance the energy consumption of the sensor nodes and hence to extend the network lifetime.

Spatial compression is another approach to reduce network traffic. It is based on the observation that sensor readings captured by the sensor nodes in close proximity are usually correlated [Kot05, HL05]. For example, the sensor nodes deployed in the same room are likely to detect similar temperature readings. Thus, it is sufficient to let only a set of representative sensor nodes to report their readings. Spatial compression
CHAPTER 2. LITERATURE REVIEW

saves the message transmissions by the nodes whose readings can be predicted from the representative nodes. Kotidis [Kot05] proposed a scheme to elect the representative nodes in the network. To search for its representative node, each sensor node broadcasts a message containing its reading to the neighbors. A node A can be considered for representing another node B, if A can estimate B's reading within a given error bound. If A represents B, only A reports its reading at each sampling interval and B does not. The objective of Kotidis' scheme is to minimize the number of representative nodes in order to reduce the total network traffic. Hartl et al. [HL05] also explored spatial compression by selecting a set of representative nodes to report their readings to the sink node. A Bayesian inference approach is used at the sink node to infer the readings of the non-representative nodes from the readings of the representative nodes. To extend network lifetime, the selection of representative nodes attempts to balance the energy consumption among sensor nodes by including the energy levels of the sensor nodes in the selection criteria. As a result, the nodes whose readings are critical to enhancing the inference accuracy and have higher remaining energy are selected.

Spatial compression and temporal compression can be combined in sensor data collection [DGM+04, CDHH06, SBY06]. Desphande et al. [DGM+04] proposed an architecture called BBQ to support model-driven approximate data collection. In this approach, a statistical model is trained from the readings collected from all sensor nodes in an initial stage. The model characterizes both spatial and temporal correlations among the sensor readings. BBQ allows the users to pose queries with precision requirements and confidence levels. The query results are directly evaluated from the statistical model if such evaluation meets the precision requirement with a confidence level higher than the given one. Otherwise, a set of sensor nodes are selected according to the statistical model and probed by the sink node. The readings of these sensor nodes are collected to refine the query results derived by the statistical model [DGM+04, DGMH05]. The
Ken architecture, developed by Chu et al. [CDHH06], differs from BBQ in that it keeps the same statistical models at both the sink node and the nodes acquiring sensor data. Based on the statistical model, the reading of a sensor node is predicted by both the sensor node and the sink node. On acquiring a new sensor reading, the sensor node reports its reading to the sink node only if it differs from the predicted value by more than a given error bound. A similar approach has been proposed by Xu et al. [XWL04a] to reduce location updates in object tracking sensor networks.

While the effectiveness of the model-driven approach depends on whether the data observed in real time are similar to the training data used in constructing the statistical model, Silberstein et al. [SBY06] proposed a chain-based scheme that does not require building statistical models in advance. In their approach, a minimum spanning forest covering all sensor nodes is constructed over the network to capture the spatial correlations among the sensor readings. The weight of an edge in the forest indicates the difference between the readings of the two sensor nodes connected by the edge. As a result, the readings of all sensor nodes can be inferred from the readings of the nodes at the tree roots and the weights of the edges in the forest. These values are continuously monitored by the sink node. To take the advantage of the temporal correlations, the reading of each tree root is reported to the sink node only when it differs from the last reported reading. To leverage the spatial correlations, the weight of each edge is reported to the sink node only when the difference between the readings of the two relevant sensor nodes changes.

In our work of ANN query processing in wireless sensor networks, we apply temporal compression to reduce the location updates sent by the sensor nodes. This is achieved by setting up a monitoring area in the sensor network so that only the relevant location updates are collected for the purpose of query processing.
2.2.3 Spatial Queries and Object Tracking Sensor Networks

Due to the geographically distributed deployment of the sensor nodes, spatial information plays a key role in representing sensor data (e.g., a temperature sensor reading is usually associated with a location where it is acquired). Hence, it is natural to collect data from the sensor network by specifying spatial conditions in the queries. For example, users may be interested in the average temperature reading on NTU campus or the readings of $k$ nearest sensor nodes to a given geographical point. In these cases, the relevant geographical areas would be searched for the requested data. We refer to the queries including spatial conditions as spatial queries. There are two main approaches to process spatial queries: infrastructure-based [Dem03, WL04] and infrastructure-free [XLXMOG, WCCC07, FPL07]. The infrastructure-based approach relies on a distributed index structure for sensor locations to process spatial queries. In contrast, the infrastructure-free approach does not rely on any index structure.

Demirbas et al. [Dem03] proposed an infrastructure-based scheme called Peer-tree to process queries that request the reading of the sensor node nearest to a given query point. Peer-tree is a distributed version of R-tree [Gut84] where the index nodes of the tree are maintained at different sensor nodes. To construct the Peer-tree, the sensing field is partitioned into a hierarchy of Minimum Bounding Rectangles (MBR) for the sensor nodes. Each MBR at the lowest level covers one sensor node only. A higher level MBR covers all of its child MBRs. The MBR at the tree root covers the entire network. For each MBR, one sensor node in the MBR, called the cluster head, is designated to maintain the MBR. The cluster head also remembers the locations and identifiers of the cluster heads maintaining its parent MBR and child MBRs. As a result, the sensor nodes form an overlay tree following the logical structure of the R-tree. Queries are injected into the network from any node in the tree (called the query node). If the query point is within the MBR maintained by the query node, the query node directly forwards
the query downward the tree to its child MBRs to search for the nearest sensor node. Otherwise, the query node would first route the query upward the tree until an MBR covering the query point is found.

In Peer-tree, the cluster heads maintaining high level MBRs can easily become system bottlenecks due to their greater responsibilities in query routing. Moreover, by routing queries along the hierarchy of cluster heads, it may create unnecessary hops in the route from a query node to the node nearest to the query point. To remedy these problems, Winter et al. [WL04, XFLW07] proposed a dynamic Perimeter-tree to request the readings from the $k$ nearest sensor nodes to a given query point. When a query is issued, it is first routed to a sensor node called the home node which is closest to the query point. A conservative boundary containing at least $k$ sensor nodes is estimated by the home node. Multiple trees rooted at the home node are then constructed within the boundary to propagate queries and to collect the locations and readings of the sensor nodes (see Figure 2.8). After collecting the data, the home node determines the $k$ sensor nodes nearest to the query point by sorting the locations of the sensor nodes, and reports their sensor readings as query results. The trees are destroyed once the query is completed.

![Figure 2.8: KNN Perimeter-tree [WL04]](image-url)
Infrastructure-based approaches are susceptible to network topology changes due to node failures and node movements. Constructing and maintaining a stable infrastructure in a sensor network often incur a large number of message exchanges. The performance of infrastructure-based query processing would deteriorate with increasing network dynamics. Xu et al. [XLXM06, FPL07] proposed an infrastructure-free approach to process spatial queries. They focused on window queries which allow users to collect data from the sensor nodes within a specified query window, i.e., a 2 or 3 dimensional geographical area. In this approach, the query is first routed towards the corner of the query window and then propagated within the window along a well-defined itinerary as shown in Figure 2.9. A set of sensor nodes called Q-nodes are chosen for query propagation. When a Q-node receives the query message, it broadcasts the query to all the nodes within its communication range and collects the sensor readings from these nodes. The Q-node then selects the next Q-node on the route of the itinerary from its neighboring nodes. This process continues until the query window has been fully covered.

![Diagram of itinerary-based propagation](image)

Figure 2.9: Itinerary-based Propagation [XLXM06]

Similarly, Wu et al. [WCCC07] proposed an infrastructure-free scheme to collect readings from $k$ nearest sensor nodes to a given query point. The data collection procedure is divided into two steps. In the first step, the query is routed from a query node towards
the home node which is the nearest node to the query point (see Figure 2.10(a)). In the mean time, the information about the node density in the network is gathered. Based on this information, the home node estimates a boundary that contains $k$ sensor nodes. In the second step, the query is propagated following a spiral-shaped itinerary as shown in Figure 2.10(a). Similar to the work in [XLXMOG], a set of Q-nodes are chosen to be responsible for query propagation and data collection. To reduce query latency, the search space may be divided into several sub-spaces and the query can be concurrently propagated in different sub-spaces following respective itineraries to support parallel search (see Figure 2.10(b)).

All the above work has focused on collecting sensor readings from the nodes based on the node locations. In addition to the locations of sensor nodes, the data captured by the sensor nodes may also include spatial information. For example, in object tracking sensor networks, the object locations detected by the sensor nodes are represented by geographical coordinates. Users may issue spatial queries that request the detected object locations satisfying certain spatial conditions. In the following, we briefly discuss object tracking sensor networks.
To improve energy efficiency in object tracking sensor networks, existing research efforts have concentrated on the energy efficiency in locating and tracking target objects [LLR+04, YS03, WCZ+03, BRS03, ZC04]. The general idea is to activate only a small group of sensor nodes around each target object to track its location while keeping most of the sensor nodes in the idle state that has low energy consumption. This is normally done with the assistance of a prediction model for the future locations of target objects.

Zhao et al. [ZSR02, LLR+04] proposed a leader-based tracking scheme in which only one sensor node is elected at a time to track the location of an object. The elected sensor node is called the leader. The leader, based on its local observation, computes an estimate of the object location. When the target object moves, the leader selects the next leader from its immediate neighbors. The selection of the next leader is based on the neighbors' positions and the communication costs with neighbors. The neighbor node that would provide more accurate estimate of the object location and has lower communication cost is selected as the next leader.

Though simple, the above scheme may not be able to achieve high accuracy. To improve tracking quality, several cluster-based approaches have been designed for object tracking. Instead of using only one sensor node to track an object, a group of sensor nodes collaboratively estimate the location of an object. The approach proposed by Yang et al. [YS03] organizes sensor nodes into static clusters. Each cluster has a fixed cluster head. When an object enters the sensing field, the clusters along the object's trajectory are sequentially activated to perform sensing, prediction, and communication tasks. Upon activation, the sensor nodes in a cluster report their detected signals to the cluster head for it to compute the object location. To estimate the future location of the object, a linear prediction model based on past object locations is used.
The static cluster structure may not be optimized for tracking objects at arbitrary locations. To further enhance the tracking quality, Chen et al. [WCZ+03, CHS04] proposed a hierarchical structure that designates high-capacity sensor nodes as fixed cluster heads and forms clusters dynamically. Upon detecting an object, a cluster head selects a number of low-end sensor nodes on the fly and triggers them to form a cluster. Brook et al. [BRS03, BFKP04] used a fully dynamic cluster structure without fixed cluster heads. The clusters are dynamically formed as the objects move in the sensing field. The sensor nodes in a cluster share the detected signals through flooding and the sensor node detecting the strongest signal is elected as the head of the cluster. The cluster head uses triangulation to compute the locations of an object. When the sensor nodes have equal energy capacities, Brook’s scheme helps to balance the energy consumption among the sensor nodes since every node has a chance to be elected as the cluster head. To reduce communication cost, Zhang et al. [ZC04] further proposed to structure the sensor nodes in a cluster into a minimum spanning tree rooted at the cluster head for communication purpose. The cluster head collects the signals detected by the sensor nodes in the cluster along the paths in the tree and computes the object location. As the object moves in the sensing field, the tree is dynamically reconfigured.

In addition to object tracking, the sensor network may operate in surveillance mode to detect the appearance of objects. To reduce energy consumption for surveillance, it is desirable to reduce the number of sensor nodes staying active at any time. Gui et al. [GM04] proposed a scheme called ALUL to schedule the sensor nodes in order to conserve energy while guaranteeing the quality of surveillance. Their objective is to minimize the average linear uncovered length which measures the quality of surveillance. While Gui studied the surveillance of sensor networks for detecting the appearance of moving objects, Liu et al. [LJW+07] proposed a scheduling algorithm to monitor a set of static objects within the sensing field. Their objective is to distribute the workload.
CHAPTER 2. LITERATURE REVIEW

evenly among the sensor nodes and hence to maximize the network lifetime. To do so, the sink node pre-computes a schedule and disseminates it to all sensor nodes. The sensor nodes then watch objects, turn off to sleep, receive and forward data according to the schedule.

Most work described above has concentrated on the energy efficiency in locating and tracking target objects. In contrast, we consider the energy efficiency in query processing. In our thesis research, we shall investigate kNN queries in an object tracking sensor network that request the locations of the k nearest objects to a given query point.
Chapter 3

Energy-Efficient and Access Latency Optimized Indexing for Wireless Data Broadcast

3.1 Introduction

In wireless environments, a base station is often deployed to disseminate data to mobile clients through wireless channels. As discussed in Chapter 1, push-based broadcast is an attractive dissemination method in such context. In push-based broadcast, data are disseminated proactively and the clients simply filter and pick the data they want, thereby alleviating the load on the uplink channel from the clients to the base station. Mobile clients are normally powered by batteries and can operate in two different modes: active mode and doze mode. They can retrieve data from broadcast channels in the active mode only. However, the clients have much higher rate of energy consumption in the active mode than that in the doze mode. Therefore, to save energy, it is desirable for mobile clients to switch to the doze mode as much as possible when waiting for the requested data.

The performance of broadcast systems is often characterized by two metrics: access latency and tuning time [IVB97, XLT04]. Access latency refers to how fast the client can access the requested data. It reflects the responsiveness of the system. Tuning time, on
the other hand, refers to the duration in which the client stays active. It measures the energy consumed by the client in the active mode. As discussed in Chapter 2, the tuning time can be reduced by means of air indexing [IVB97]. The basic idea is to interleave the index information with data in the broadcast schedule to assist the client in locating data. However, most existing air indexing schemes were designed for flat broadcast in which all data items are broadcast at the same frequency [CWY03, IVB94, IVB97, XLT04, XLT+06]. This sacrifices responsiveness when client accesses are not uniformly distributed among data items. To reduce average access latency under non-uniform access distribution, popular data items should be broadcast more frequently than unpopular items. This is known as non-flat data broadcast. Nevertheless, most existing schemes of non-flat broadcast scheduling did not consider air indexing [AAFZ95, STT99, VH99]. Without indexing, the client has to continuously stay active and monitor the broadcast channel until the requested data arrive. This consumes significant amount of battery power and sacrifices energy efficiency.

Different from the existing work, our objective is to optimize responsiveness and energy efficiency in an integrated fashion. In this chapter, we propose a novel indexing scheme called \textit{MHash} for wireless data broadcast. \textit{MHash} constructs the broadcast schedule using a two-argument hash function for indexing purpose. The two-argument nature of the hash function allows each data item to be mapped to an adjustable number of slots in the schedule, thereby enabling non-flat data broadcast. Under this framework, we further investigate the issues of generating hash functions that produce broadcast schedules free of unoccupied slots; improving access latency by properly spacing the broadcast instances of each data item in the schedule; and optimally allocating the bandwidth among data items. Experimental results show that under non-uniform access distribution, \textit{MHash} outperforms state-of-the-art air indexing schemes in energy efficiency and achieves access latency close to latency-optimal broadcast scheduling.
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

The rest of this chapter is organized as follows. Section 3.2 presents the MHash indexing scheme and investigates a variety of issues to refine MHash. Section 3.3 describes the experimental setup and discusses the experimental results. Finally, Section 3.4 summarizes the chapter.

3.2 MHash Indexing Scheme

3.2.1 Overview

Table 3.1: Summary of Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>length of a broadcast cycle</td>
</tr>
<tr>
<td>$n$</td>
<td>number of data items</td>
</tr>
<tr>
<td>$M$</td>
<td>maximum allowable number of times each item can be broadcast in one cycle</td>
</tr>
<tr>
<td>$d_i$</td>
<td>the $i$th data item</td>
</tr>
<tr>
<td>$p_i$</td>
<td>the access probability of item $d_i$</td>
</tr>
<tr>
<td>$r_i$</td>
<td>the fraction of bandwidth allocated to item $d_i$</td>
</tr>
<tr>
<td>$H(k, l)$</td>
<td>two argument hash function</td>
</tr>
<tr>
<td>$k$</td>
<td>the key of a data item</td>
</tr>
<tr>
<td>$l$</td>
<td>sequential identifier</td>
</tr>
<tr>
<td>$c$</td>
<td>actual number of broadcast instances of a data item in one cycle</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Zipf parameter of access distribution</td>
</tr>
<tr>
<td>$P$</td>
<td>bucket capacity</td>
</tr>
<tr>
<td>$S_b$</td>
<td>the size of a bucket</td>
</tr>
<tr>
<td>$S_d$</td>
<td>the size of a data item</td>
</tr>
</tbody>
</table>

The key idea of MHash is to construct an energy-efficient index that allows different items to be broadcast with different frequencies. The notations used in this chapter are summarized in Table 3.1. We consider a system that repeatedly broadcasts a set of data items in cycles. A broadcast cycle consists of a sequence of slots, each of which accommodates one item. All slots in the cycle are numbered sequentially: 0, 1, 2, ..., $N - 1$, where $N$ is called the cycle length. MHash first maps each item to a given number of $M$
slots. The item is then placed and broadcast in a subset of these slots. To reduce average access latency, popular data items are placed in more slots than unpopular items, thus enabling non-flat data broadcast. $M$ is a tunable parameter representing the maximum allowable number of times each item can be broadcast in one cycle. We refer to $M$ as the replication bound.

We use a two-argument hash function $H(k, l)$ to map a data item to a list of slots, where $k$ is the key of the item and $l$ is a sequential identifier. The function maps the key to the slots: $H(k, 1), H(k, 2), \ldots, H(k, M)$. If the item is to be broadcast $c \leq M$ times in a cycle, it is then placed and broadcast in the first $c$ slots on the list, i.e., $H(k, 1), H(k, 2), \ldots, H(k, c)$. They are called the hashed slots of the item and we say that the item is hashed to these slots. As will be discussed shortly, choosing a prefix of the list allows the tuning time to be further reduced by pruning in data accesses. The remaining slots, i.e., $H(k, c + 1), H(k, c + 2), \ldots, H(k, M)$, are called the cheating slots, since the item would not actually be broadcast in these slots.

It is likely that multiple items are hashed to the same slot. This is known as collision. Meanwhile, there might be some slots such that no item is hashed to them. We refer to these slots as empty slots. In the following, we propose a chaining method to resolve collisions. In Section 3.2.2, we shall investigate, under our collision resolution method, how to construct hash functions that produce broadcast schedules without any unoccupied slot where no item is placed.

To resolve collisions, all items hashed to the same slot are sorted in decreasing order of access probability. The first item (i.e., the most frequently accessed one) is placed in the hashed slot. The remaining items are sequentially placed in subsequent empty slots. To facilitate data accesses, a distance pointer is recorded in each slot to refer to the next slot accommodating an item with the same hashed slot. The distance pointer of the last item is set to 0. The purpose of broadcasting the items hashed to the same slot in decreasing order of access probability is to reduce the average access latency.
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

<table>
<thead>
<tr>
<th>k</th>
<th>H(k, 1)</th>
<th>H(k, 2)</th>
<th>H(k, 3)</th>
<th>H(k, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>C</td>
<td>16</td>
<td>8</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>12</td>
<td>3</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>F</td>
<td>9</td>
<td>1</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>H</td>
<td>6</td>
<td>14</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>11</td>
<td>3</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

(a) Hash Table

(b) Broadcast Cycle

Figure 3.1: An Example of MHash

Figure 3.1 shows an example of MHash indexing. The data items, listed in decreasing order of access probability, are A to J. Suppose the broadcast cycle consists of 17 slots. Item A is broadcast four times in the cycle, items B to E are broadcast twice each, and the remaining items are broadcast once each. Suppose the replication bound $M = 4$. Figure 3.1(a) shows a hypothetical hash table. Accordingly, the items hashed to each slot are shown below the slot in Figure 3.1(b). As can be seen, slots 2, 5, 7, 10, 13, 14 and 15 are empty slots where no item is hashed. Items B, G and J are all hashed to slot 1. To resolve the collision, B (the most popular item among B, G and J) is placed in its hashed slot 1, and G and J are placed in empty slots 2 and 5 respectively. Slot 1 has
a distance pointer 1 since the next item hashed to slot 1 (i.e., \( G \)) is broadcast one slot away in slot 2. Slot 2 has a distance pointer 3, indicating the next item hashed to slot 1 (i.e., \( J \)) is broadcast 3 slots away in slot 5. The distance pointer of slot 5 is 0 since \( J \) is the last item hashed to slot 1. As shown in Figure 3.1(b), the distance pointers link the items hashed to the same slot in a chain. Similarly, items \( D \) and \( E \) are both hashed to slot 3. So, \( D \) is placed in its hashed slot 3. Since empty slot 5 has been occupied, \( E \) is placed in the following empty slot 7.

**Algorithm 3.1 Data Access**

**Input:** Key \( k \)

**Output:** Target data item with key \( k \)

1. Calculate slot numbers \( H(k, 1), H(k, 2), ... , H(k, M) \);
2. Sort the slot numbers in increasing order of their distances ahead of the initial probing slot: \( H(k, l_1), H(k, l_2), ... , H(k, l_M) \), where \( l_1, l_2, ... , l_M \) is a permutation of \( 1, 2, ... , M \);
3. Remove all slot numbers \( H(k, l_i) \) where \( \min_{1 \leq j \leq i} l_j < l_i \);
4. Put remaining slot numbers in a searching set \( Q \);
5. **repeat**
6. Tune to the nearest slot \( q \in Q \) ahead;
7. if \( \text{Bcast}[q].data.key = k \) then
8. Probe success and return \( \text{Bcast}[q].data \);
9. else
10. Read the distance pointer \( d \) of \( q \);
11. **end if**
12. if \( d > 0 \) then
13. Insert \( q + d \) to \( Q \);
14. **end if**
15. Remove \( q \) from \( Q \);
16. **until** \( Q \) is empty
17. Probe failure;

Algorithm 3.1 describes the algorithm for data accesses. We illustrate the access process with the example in Figure 3.1. Note that item \( C \) is broadcast twice in the cycle at slots 10 and 16, which correspond to hashed slots \( H(C, 2) = 8 \) and \( H(C, 1) = 16 \) respectively.\(^1\) Suppose the client tunes to slot 5 at the initial probe and would like to

\(^1\) For convenience, we shall use the same letter to denote an item and its key.
access item $C$ given its key. The client first calculates the $M$ slot numbers where the
key is mapped (step 1 of Algorithm 3.1). Following the hash table in Figure 3.1(a), the
slot numbers where $C$ is mapped are 16, 8, 12, 3. They are the potential locations of the
target data item. These slot numbers are then sorted in increasing order of their distances
ahead of the initial probing slot (step 2). In our example, the sorted list of slot numbers
are 8, 12, 16 and 3. Intuitively, the client should tune to all of these slots sequentially to
look for item $C$. However, we note that the slot numbers can be shortlisted for searching
purpose. For example, since $H(C, 2) = 8$ precedes $H(C, 3) = 12$ and $H(C, 4) = 3$ on
the sorted list, $H(C, 3)$ and $H(C, 4)$ can be pruned. This is because if the target item
is broadcast at least twice in the cycle, $H(C, 2)$ is a hashed slot and the client would
be able to retrieve item $C$ from it (possibly by following the distance pointers). On the
other hand, if the item is broadcast fewer than twice in the cycle, all slots $H(C, l)$ where
$l \geq 2$ are cheating slots and the client would not be able to find item $C$ from any of
them. So, in either case, there is no need for the client to tune to slot $H(C, 3)$ or $H(C, 4)$
after checking $H(C, 2)$. In general, if the sorted list of slot numbers are $H(k, l_1), H(k, l_2),
..., H(k, l_M)$ where $k$ is the key, and $l_1, l_2, ..., l_M$ is a permutation of 1, 2, ..., $M$, the
client can get rid of all slots $H(k, l_i)$ where $\min_{1 \leq j \leq i-1} l_j < l_i$ (step 3). As will be shown
in Section 3.3, this pruning technique makes tuning time grow slowly (logarithmically) with
$M$.

The slot numbers left after pruning constitute a searching set $Q$ (step 4). The client
repeatedly tunes to the nearest slot $q \in Q$ ahead. If the target data item is not found in
slot $q$, the client reads the distance pointer $d$ from the slot. If $d > 0$, $Q$ is updated by
replacing $q$ with $q + d$. If $d = 0$, $q$ is removed from $Q$ (i.e., the item broadcast in slot $q$
is the last item hashed to its hashed slot). The process continues until the target data
item is found or $Q$ becomes empty$^2$ (steps 5 to 16). The latter case leads to a failure of

$^2$Note that the access process may extend to the next broadcast cycle if the initial probe is not at
the beginning of a cycle.
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

data access, i.e., there does not exist any item with the requested key in the broadcast schedule (step 17).

In our example, the initial searching set $Q$ includes $H(C, 2) = 8$ and $H(C, 1) = 16$. The client first tunes to slot 8 and finds that the item broadcast in slot 8 is not item $C$. It then reads the distance pointer 2 and replaces 8 by $8 + 2 = 10$ in $Q$. Now, the updated $Q$ includes slot numbers 10 and 16. The client tunes to slot 10 and retrieves item $C$. Thus, after the initial probe at slot 5, the client only tunes to slots 8 and 10 in the access process, and can switch to the doze mode in the other slots.

As seen from the example, a collision in hashing introduces some penalty to tuning time. In general, the performance of MHash indexing improves with decreasing collision rate of the hash function.\(^3\) This will be confirmed by the experimental results in Section 3.3. It is also intuitive that the tuning time increases with the number of cheating slots. Since an item that is broadcast $c$ times in the cycle has $M - c$ cheating slots, more frequently broadcast items would have lower tuning time. Therefore, if popular items are broadcast more frequently than unpopular items (to reduce average access latency), MHash naturally leads to less tuning time for popular items. This salient feature helps reduce the average tuning time.

The following parameters are needed by the client in data accesses: the hash function $H$, the cycle length $N$, and the replication bound $M$. These parameters can be recorded in the header of each bucket, which is the smallest accessible unit of broadcast [IVB97, XLT04]. The number of slots in a bucket depends on the size of a data item. If a bucket contains multiple slots, the accesses to the slots therein involve the same bucket access physically.

\(^3\)The selection of good hash function is application specific.
3.2.2 Hole-Free Hash Function

An empty slot may be left unoccupied in the broadcast schedule. Figure 3.2(b) shows the broadcast cycles derived from an example hash table shown in Figure 3.2(a). Slots 1 and 6 are left unoccupied in the first cycle because no item is hashed to or pushed to these slots due to collisions in the preceding slots. Meanwhile, items E and G, which are hashed to slots 9 and 10, have to be pushed forward to the second cycle. The existence of such unoccupied slots (referred to as holes) not only wastes bandwidth but also complicates the computation of distance pointers in subsequent cycles (e.g., slot 2 has different pointer values in the two cycles in Figure 3.2(b)). Therefore, it is desirable to look for hole-free hash functions that can produce broadcast schedules without unoccupied slots. In this section, we show that a hole-free hash function can be constructed by injecting an offset into an arbitrary hash function. For example, denote the hash function of Figure 3.2(a) by \( H(k, l) \). The holes in Figure 3.2(b) would be eliminated if we use a new hash function \( H'(k, l) = (H(k, l) - 7) \mod 11 \), where 11 is the cycle length. The new hash table is shown in Figure 3.3(a) and the derived broadcast cycles are shown in Figure 3.3(b).

In general, consider \( n \) data items \( d_1, d_2, \ldots, d_n \) to be broadcast \( c_1, c_2, \ldots, c_n \) times respectively in a cycle of length \( N = \sum_{i=1}^{n} c_i \). In MHash indexing, each item \( d_i \) has \( c_i \) hashed slots \( H(d_i, 1), H(d_i, 2), \ldots, H(d_i, c_i) \) between 0 and \( N - 1 \), and there are a total of \( N \) hashed slots for all items. Given a hash function \( H \), let \( f_i(H) \) be the number of items hashed to slot \( i \), then

\[
\sum_{j=0}^{N-1} f_j(H) = N.
\]

Let \( h_i(H) \) be the cumulative number of holes in slots 0 to \( i \). Obviously, \( h_0(H) \) depends on whether any item is hashed to slot 0, i.e.,

\[
h_0(H) = \begin{cases} 
0 & \text{if } f_0(H) > 0, \\
1 & \text{if } f_0(H) = 0.
\end{cases}
\]
Chapter 3. Energy-Efficient and Access Latency Optimized Indexing for Wireless Data Broadcast

<table>
<thead>
<tr>
<th>$k$</th>
<th>$H(k, 1)$</th>
<th>$H(k, 2)$</th>
<th>$H(k, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Original Hash Table

(b) Holes in Broadcast Cycle

Figure 3.2: An Example Hash Function $H(k, l)$ Producing Holes in Broadcast Cycle

<table>
<thead>
<tr>
<th>$k$</th>
<th>$H'(k, 1)$</th>
<th>$H'(k, 2)$</th>
<th>$H'(k, 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

(a) New Hash Table

(b) Hole-Free Broadcast Cycle

Figure 3.3: An Example Hole-Free Hash Function $H'(k, l) = (H(k, l) - 7) \mod 11$
For each $1 \leq i \leq N - 1$, since among the $i$ slots from 0 to $(i - 1)$, $i - h_{i-1}(H)$ slots are occupied by data items, we have
\[ \sum_{j=0}^{i-1} f_j(H) \geq i - h_{i-1}(H). \] (3.2)
If the equality holds, i.e., $\sum_{j=0}^{i-1} f_j(H) = i - h_{i-1}(H)$, no item hashed to slots 0 to $(i - 1)$ is pushed forward to slot $i$ for collision resolution. Otherwise, if $\sum_{j=0}^{i-1} f_j(H) > i - h_{i-1}(H)$, at least one item is pushed forward to slot $i$ for collision resolution. Note that slot $i$ ($i > 0$) is a hole if and only if (1) no item is hashed to it (i.e., $f_i(H) = 0$), and (2) no item is pushed forward to slot $i$ for collision resolution. Therefore, for each $1 \leq i \leq N - 1$,
\[ h_i(H) = \begin{cases} h_{i-1}(H) & \text{if } f_i(H) > 0 \text{ or } \sum_{j=0}^{i-1} f_j(H) > i - h_{i-1}(H), \\ h_{i-1}(H) + 1 & \text{if } f_i(H) = 0 \text{ and } \sum_{j=0}^{i-1} f_j(H) = i - h_{i-1}(H). \end{cases} \] (3.3)

Theorem 3.1 shows that a hole-free hash function can be constructed by injecting an offset into an arbitrary hash function.

**Theorem 3.1** Given any hash function $H(k, l)$, let $b$ be the smallest index such that slots $b$ to $N - 1$ are hole-free under $H$, then the new hash function $H'(k, l) = (H(k, l) - b) \mod N$ is hole-free, i.e., for each $0 \leq i \leq N - 1$, $h_i(H') = 0$.

**Proof:** We first show that $h_{N-1}(H) = h_{N-2}(H)$. This is because if $f_{N-1}(H) > 0$, it follows from Equation (3.3) that $h_{N-1}(H) = h_{N-2}(H)$. Otherwise, if $f_{N-1}(H) = 0$, since $\sum_{j=0}^{N-1} f_j(H) = N$, we have
\[ \sum_{j=0}^{N-2} f_j(H) = \sum_{j=0}^{N-1} f_j(H) = N > N - 1 \geq N - 1 - h_{N-2}(H). \]
Therefore, based on Equation (3.3), $h_{N-1}(H) = h_{N-2}(H)$. This implies the last slot in the cycle cannot be a hole, i.e., $b$ exists and $0 \leq b \leq N - 1$.

If all slots from 0 to $N - 1$ are hole-free under $H(k, l)$, then $b = 0$ and the conclusion is trivial. Otherwise, if $1 \leq b \leq N - 1$, two properties follow from the definition of $b$: 48
(i) slot \( b - 1 \) is a hole, i.e., \( h_{b-1}(H) = h_{b-2}(H) + 1 \), and

(ii) for each \( b \leq i \leq N - 1 \), \( h_i(H) = h_{i-1}(H) \).

Next, we show that the hash function \( H'(k, l) = (H(k, l) - b) \mod N \) is hole-free, i.e., for each \( 0 \leq i \leq N - 1 \), \( h_i(H') = 0 \). Note that the definition of \( H'(k, l) \) indicates \( f_i(H') = f_{(b+i) \mod N}(H) \).

Since \( h_{b-1}(H) = h_{b-2}(H) + 1 \), based on Equation (3.3), we have \( f_{b-1}(H) = 0 \) and \( \sum_{j=0}^{b-2} f_j(H) = b - 1 - h_{b-2}(H) \). Thus,

\[
\sum_{j=0}^{b-1} f_j(H) = \sum_{j=0}^{b-2} f_j(H) = b - 1 - h_{b-2}(H) = b - h_{b-1}(H).
\] (3.4)

According to Property (ii), \( h_b(H) = h_{b-1}(H) \). Combining Equations (3.3) and (3.4), we must have \( f_0(H) > 0 \). Therefore, \( f_0(H') = f_0(H) > 0 \). It follows from Equation (3.3) that \( h_0(H') = 0 \).

To prove that for each \( 1 \leq i \leq N - 1 \), \( h_i(H') = 0 \), we consider three cases separately.

**Case I:** \( 1 \leq i \leq N - b - 1 \).

Prove by induction. Suppose \( h_{i-1}(H') = h_{i-2}(H') = \ldots = h_0(H') = 0 \) for some \( 1 \leq i \leq N - b - 1 \), we show that \( h_i(H') = 0 \).

Since \( 1 \leq i \leq N - b - 1 \), we have \( f_i(H') = f_{b+i}(H) \). According to Property (ii), \( h_{b+i}(H) = h_{b+i-1}(H) \). It follows from Equation (3.3) that either \( f_{b+i}(H) > 0 \) or \( \sum_{j=0}^{b+i-1} f_j(H) > b+i-h_{b+i-1}(H) \). If \( f_{b+i}(H) > 0 \), we have \( f_i(H') = f_{b+i}(H) > 0 \) and hence, based on Equation (3.3), \( h_i(H') = h_{i-1}(H') \). Otherwise, if \( \sum_{j=0}^{b+i-1} f_j(H) > \).
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

$b + i - h_{b+i-1}(H)$, based on Equation (3.4),

\[
\sum_{j=0}^{i-1} f_j(H') = \sum_{j=0}^{i-1} f_{b+j}(H) \\
= \sum_{j=k}^{b+i-1} f_j(H) \\
= \sum_{j=0}^{b+i-1} f_j(H) - \sum_{j=0}^{b-1} f_j(H) \\
= \sum_{j=0}^{b+i-1} f_j(H) - (b - h_{b-1}(H)) \\
> b + i - h_{b+i-1}(H) - (b - h_{b-1}(H)) \\
= i - h_{b+i-1}(H) + h_{b-1}(H).
\]

Note that Property (ii) implies $h_{b+i-1}(H) = h_{b+i-2}(H) = \ldots = h_{b-1}(H)$. Therefore, 
\[
\sum_{j=0}^{i-1} f_j(H') > i = i - h_{i-1}(H').
\]

It follows from Equation (3.3) that $h_i(H') = h_{i-1}(H') = 0$.

Case II: $i = N - b$.

Based on Equation (3.4) and $\sum_{j=0}^{N-1} f_j(H) = N$,

\[
\sum_{j=0}^{N-b-1} f_j(H') = \sum_{j=0}^{N-b-1} f_{b+j}(H) \\
= \sum_{j=b}^{N-1} f_j(H) \\
= \sum_{j=0}^{N-1} f_j(H) - \sum_{j=0}^{b-1} f_j(H) \\
= N - (b - h_{b-1}(H)).
\]

According to Property (i), $h_{b-1}(H) = h_{b-2}(H) + 1 > 0$. Also, it has been proven
in case I that \( h_{N-b-1}(H') = 0 \). Therefore,

\[
\sum_{j=0}^{N-b-1} f_j(H') = N - b + h_{b-1}(H) > N - b = N - b - h_{N-b-1}(H').
\]

It follows from Equation (3.3) that \( h_{N-b}(H') = h_{N-b-1}(H') = 0 \).

**Case III:** \( N - b + 1 \leq i \leq N - 1 \).

Again, prove by induction. Note that it has been proven in case II that \( h_{N-b}(H') = 0 \). Suppose \( h_{i-1}(H') = h_{i-2}(H') = \ldots = h_{N-b}(H') = 0 \) for some \( N - b + 1 \leq i \leq N - 1 \), we show that \( h_i(H') = 0 \).

For each \( N - b + 1 \leq j \leq N - 1 \), \( f_j(H') = f_{j+b-N}(H) \). Based on Equation (3.4) and \( \sum_{j=0}^{N-1} f_j(H) = N \), we have

\[
\sum_{j=0}^{i-1} f_j(H') = \sum_{j=0}^{N-b-1} f_j(H') + \sum_{j=N-b}^{i-1} f_j(H') = \sum_{j=b}^{N-1} f_j(H) + \sum_{j=0}^{N-b-1} f_j(H') = \sum_{j=0}^{i+b-N-1} f_j(H) + \sum_{j=0}^{i+b-N-1} f_j(H) = N - (b - h_{b-1}(H)) + \sum_{j=0}^{i+b-N-1} f_j(H).
\]

Recall from Equation (3.2) that for each \( 1 \leq m \leq N - 1 \), \( \sum_{j=0}^{m-1} f_j(H) \geq m - h_{m-1}(H) \). Thus, \( \sum_{j=0}^{i+b-N-1} f_j(H) \geq (i + b - N) - h_{i+b-N-1}(H) \). On the other hand, it follows from Property (i) that \( h_{b-1}(H) > h_m(H) \) for any \( 0 \leq m \leq b - 2 \). Since
0 \leq i + b - N - 1 \leq b - 2$, we have $h_{b-1}(H) > h_{i+b-N-1}(H)$. Therefore,

$$
\sum_{j=0}^{i-1} f_j(H') \geq N - b + h_{b-1}(H) + (i + b - N) - h_{i+b-N-1}(H)
= i + h_{b-1}(H) - h_{i+b-N-1}(H)
> i
= i - h_{i-1}(H').
$$

Based on Equation (3.3), $h_i(H') = h_{i-1}(H') = 0$.

Therefore, for each $0 \leq i \leq N - 1$, $h_i(H') = 0$. Hence, the theorem is proven.

### 3.2.3 Spacing between Broadcast Instances

If an item is broadcast multiple times in a cycle, its access latency, to a large extent, depends on how the slots broadcasting the item (called broadcast instances) are located in the cycle. Intuitively, if multiple broadcast instances are clustered in a short segment of the cycle, they would not reduce access latency much compared to a single broadcast instance. In general, the access latency of an item can be reduced by equalizing the spaces between successive broadcast instances of the item [VH99].

In this section, we construct hash functions that approximately equalize the spaces between broadcast instances in MHash indexing. Recall that each item is broadcast in the slots whose numbers are a prefix of the list $H(k, 1), H(k, 2), ..., H(k, M)$, where $k$ is the item key. Our idea is to let any prefix of length $2^m$ ($1 \leq m \leq M$) be a uniform partition of the broadcast cycle. When $m = 1$, the prefix consists of $H(k, 1)$ and $H(k, 2)$. To produce a uniform partition, the $H(k, 2)$-to-$H(k, 1)$ space must be set at $\frac{1}{2}N$, where $N$ is the cycle length. When $m = 2$, the prefix consists of $H(k, 1)$, $H(k, 2)$, $H(k, 3)$ and $H(k, 4)$. To produce a uniform partition, the $H(k, l)$-to-$H(k, 1)$ spaces ($2 \leq l \leq 4$) must constitute a set $\{\frac{1}{4}N, \frac{1}{2}N, \frac{3}{4}N\}$. Since the $H(k, 2)$-to-$H(k, 1)$ space is set at $\frac{1}{2}N$,}
the $H(k, l)$-to-$H(k, 1)$ spaces (3 ≤ $l$ ≤ 4) must constitute a set \{1/4 N, 3/4 N\}. We propose to set the $H(k, 3)$-to-$H(k, 1)$ space at 1/4 N, and the $H(k, 4)$-to-$H(k, 1)$ space at 3/4 N. In general, for any $m$, the $H(k, l)$-to-$H(k, 1)$ spaces (2 ≤ $l$ ≤ $2^m$) must constitute a set \{i/2 N | 1 ≤ $i$ ≤ $2^m$ - 1\}. It follows that the $H(k, l)$-to-$H(k, 1)$ spaces ($2^{m-1}$ + 1 ≤ $l$ ≤ $2^m$) must constitute a set \{2i/2^m | 1 ≤ $i$ ≤ $2^{m-1}$\}. For each 1 ≤ $i$ ≤ $2^{m-1}$, we propose to set the $H(k, 2^{m-1} + i)$-to-$H(k, 1)$ space at $2i N / 2^m$. Therefore, given $H(k, 1)$, we have

$$H(k, l) = (H(k, 1) + \frac{2l - 2\lceil\log_2 l\rceil - 1}{2\lceil\log_2 l\rceil} N) \mod N. \quad (3.5)$$

It can be inferred that a hash function satisfying Equation (3.5) has the following property: for any prefix of the list $H(k, 1)$, $H(k, 2)$, ..., $H(k, M)$, the spaces between neighboring slots in the cycle differ by at most a factor of 2. For example, a prefix of 5 slots have the spaces 0, $1/2 N$, $1/4 N$, $3/4 N$, and $1 N$ with respect to $H(k, 1)$. Thus, as shown in Figure 3.4, the spaces between neighboring slots in the cycle are $1/8 N$, $1/8 N$, $1/4 N$, $1/4 N$, and $1 N$. As will be shown in Section 3.3.2, hash functions satisfying Equation (3.5) lead to much lower access latency compared to randomly chosen two-argument hash functions.

![Figure 3.4: Spacing between Broadcast Instances](image)

Note that the techniques proposed in this section and Section 3.2.2 are orthogonal. To construct a hole-free hash function that satisfies Equation (3.5), we can first pick an arbitrary one-argument hash function $H(k, 1)$, then extend it to a two-argument function according to Equation (3.5), and finally inject an offset to make it hole-free based on Theorem 3.1.
3.2.4 Bandwidth Allocation

So far, we have assumed the number of times each item should be broadcast in a cycle is given. In this section, we discuss the bandwidth allocation problem in MHash indexing. Since the objective of non-flat data broadcast is to reduce access latency, we consider average access latency as the performance metric in bandwidth allocation. Given \( n \) items \( d_1, d_2, \ldots, d_n \), let \( p_i \) be the access probability\(^4\) of \( d_i \), where \( \sum_{i=1}^{n} p_i = 1 \). Without loss of generality, assume that \( p_1 \leq p_2 \leq \cdots \leq p_n \). Let \( r_i \) be the fraction of bandwidth allocated to \( d_i \), where \( \sum_{i=1}^{n} r_i = 1 \). If the broadcast instances of each item are equally spaced, the space between neighboring broadcast instances of \( d_i \) is proportional to \( \frac{1}{r_i} \) and thus, the average access latency of \( d_i \) is proportional to \( \frac{1}{2r_i} \). Therefore, the overall access latency is proportional to

\[
\sum_{i=1}^{n} p_i \cdot \left( \frac{1}{2r_i} \right) = \frac{1}{2} \sum_{i=1}^{n} \frac{p_i}{r_i}.
\]

It has been proven that the latency is minimized when \( r_i \propto \sqrt{p_i} \), i.e., \( r_i = \frac{\sqrt{p_i}}{\sum_{j=1}^{n} \sqrt{q_j}} \) for each item \( d_i \) [VH99]. However, this solution is not directly applicable to the MHash bandwidth allocation problem due to the following constraint. Note that with MHash indexing, each item is broadcast at least once and at most \( M \) times per cycle, where \( M \) is the replication bound. Thus, the bandwidth fractions allocated to the items can differ by at most a factor of \( M \). The bandwidth allocation problem in MHash indexing is formally defined as follows.

**Definition 3.1 [MHash Bandwidth Allocation Problem]**

*Given the access probabilities \( p_1 \leq p_2 \leq \cdots \leq p_n \) of data items \( d_1, d_2, \ldots, d_n \) respectively, and the replication bound \( M \), the objective of the MHash bandwidth allocation problem is to find an allocation \( R = (r_1, r_2, \ldots, r_n) \) where \( 0 < r_i < 1 \), \( \sum_{i=1}^{n} r_i = 1 \) and \( \forall i, j, \frac{r_i}{r_j} \leq M \),

---

\(^4\)The server can estimate the access probabilities by a number of methods [SY03].
such that \( T = \frac{1}{2} \sum_{i=1}^{n} \frac{p_i}{r_i} \) is minimized. Here, \( \forall i, j, \frac{r_i}{r_j} \leq M \) is called the differentiation constraint.

If \( \frac{p_i}{p_j} \leq M^2 \), the settings of \( r_i = \frac{\sqrt[p_i]{p_i}}{\sum_{j=1}^{n} \sqrt[p_j]{p_j}} \) \((i = 1, 2, ..., n)\) satisfy the differentiation constraint because \( \frac{r_i}{r_j} \leq M \) for any \( i \) and \( j \). Since these settings are the optimal bandwidth allocation in the absence of the differentiation constraint [VH99], they are also the optimal solution to the MHash bandwidth allocation problem. If \( \frac{p_i}{p_j} > M^2 \), the optimal bandwidth allocation in MHash indexing is less obvious. We start by showing that more frequently accessed items must be allocated higher bandwidth in the optimal allocation.

**Theorem 3.2** The optimal solution \( R = (r_1, r_2, \ldots, r_n) \) to the MHash bandwidth allocation problem satisfies \( r_1 \leq r_2 \leq r_3 \leq \cdots \leq r_n \).

**Proof:** Assume on the contrary that there exist \( 1 \leq i < j \leq n \) such that \( r_i > r_j \) in \( R = (r_1, r_2, \ldots, r_n) \). We consider two cases \( p_i < p_j \) and \( p_i = p_j \) separately.

If \( p_i < p_j \), we construct a new allocation \( R' \) by swapping the bandwidth fractions allocated to \( d_i \) and \( d_j \) in \( R \), i.e.,

\[
R' = (r_1, \ldots, r_{i-1}, r_j, r_{i+1}, \ldots, r_{j-1}, r_i, r_{j+1}, \ldots, r_n).
\]

The access latency of \( R' \) is given by

\[
T' = \frac{1}{2} \left( \frac{p_i}{r_j} + \frac{p_j}{r_i} + \sum_{k=1, k \neq i, k \neq j}^{n} \frac{p_k}{r_k} \right).
\]

Note that the access latency of \( R = (r_1, \ldots, r_n) \) is

\[
T = \frac{1}{2} \sum_{k=1}^{n} \frac{p_k}{r_k}.
\]
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

Since

\[ T - T' = \frac{1}{2} \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} \right) - \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} \right) \]
\[ = \frac{(r_j - r_i)(p_i - p_j)}{2r_i r_j} \]
\[ > 0, \]

it follows that \( T > T' \), which contradicts with the optimality of \( R \).

If \( p_i = p_j \), we construct a new allocation \( R'' \) by equalizing the bandwidth fractions allocated to \( d_i \) and \( d_j \) in \( R \), i.e.,

\[ R'' = (r_1, \ldots, r_{i-1}, \frac{1}{2}(r_i + r_j), r_{i+1}, \ldots, r_{j-1}, \frac{1}{2}(r_i + r_j), r_{j+1}, \ldots, r_n). \]

The access latency of \( R'' \) is given by

\[ T'' = \frac{1}{2} \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} \right) + \frac{1}{2}(r_i + r_j) + \sum_{k=1,k\neq i,k\neq j}^{n} \frac{p_k}{r_k}. \]

Since

\[ T - T'' = \frac{1}{2} \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} \right) - \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} \right) \]
\[ = \frac{(r_j - r_i)(p_i - p_j)}{2r_i r_j} \]
\[ = \frac{p_i(r_j - r_i)^2}{2r_i r_j(r_i + r_j)} \]
\[ > 0, \]

it follows that \( T > T'' \), which also contradicts with the optimality of \( R \).

Hence, the theorem is proven.

Theorem 3.3 shows that if \( \frac{\alpha}{p_i} > M^2 \), \( \frac{\alpha}{r_i} \) must be \( M \) in the optimal bandwidth allocation.
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

**Theorem 3.3** If $\frac{p_n}{p_1} > M^2$, the optimal solution $R = (r_1, r_2, \ldots, r_n)$ to the MHash bandwidth allocation problem satisfies $\frac{r_n}{r_1} = M$.

**Proof:** Assume on the contrary that $\frac{r_n}{r_1} < M$. We construct a new allocation $R'$ by reallocating the bandwidth between $d_1$ and $d_n$ such that their bandwidth fractions differ by a factor of $M$, i.e.,

$$R' = (r'_1, r_2, r_3, \ldots, r_{k-1}, r'_n)$$

where $r'_1 = \frac{r_1 + r_n}{M + 1}$, and $r'_n = \frac{M(r_1 + r_n)}{M + 1}$. The access latency of $R'$ is given by

$$T' = \frac{1}{2}\left(\frac{p_1}{r'_1} + \frac{p_n}{r'_n} + \sum_{k=2}^{n-1} \frac{p_k}{r'_k}\right).$$

The access latency of $R = (r_1, r_2, \ldots, r_n)$ is

$$T = \frac{1}{2}\sum_{k=1}^{n} \frac{p_k}{r_k}.$$

Thus,

$$T - T' = \frac{1}{2}\left(\left(\frac{p_1}{r_1} + \frac{p_n}{r_n}\right) - \left(\frac{p_1}{r'_1} + \frac{p_n}{r'_n}\right)\right)$$

$$= \frac{1}{2}\left(\left(\frac{p_1}{r_1} + \frac{p_n}{r_n}\right) - \left(\frac{p_1}{r_1 + r_n} + \frac{p_n}{M(r_1 + r_n)}\right)\right)$$

$$= \frac{(r_n - Mr_1)(Mp_1 r_n - p_n r_1)}{2Mr_1 r_n (r_1 + r_n)}.$$

Since $\frac{p_n}{p_1} > M^2$ and $\frac{r_n}{r_1} < M$, we have $r_n - Mr_1 < 0$ and $Mp_1 r_n - p_n r_1 < 0$. Therefore, $T - T' > 0$, which contradicts with the optimality of $R$.

Hence, the theorem is proven. \(\blacksquare\)

Theorems 3.2 and 3.3 imply that if $\frac{p_n}{p_1} > M^2$, the optimal bandwidth allocation must take one of the following two forms: (A) $r_1 = r_2 = \cdots = r_{i-1} < r_i = r_{i+1} = \cdots = r_n$ for some $1 < i \leq n$ (we shall call $i$ the single separator), where $\frac{r_n}{r_1} = M$; or (B) $r_1 = \cdots =
Chapter 3. Energy-Efficient and Access Latency Optimized Indexing for Wireless Data Broadcast

\[ r_{i-1} < r_i \leq r_{i+1} \leq \cdots \leq r_{j-1} \leq r_j < r_{j+1} = \cdots = r_n \] for some \( 1 < i < j < n \) (we shall call \( i \) and \( j \) the lower and upper separators respectively), where \( \frac{r_n}{r_1} = M \).

Given the single separator \( i \), the allocation in form \( A \) is given by \( r_1 = r_2 = \cdots = r_{i-1} = \frac{1}{i-1+M(n-i+1)} \) and \( r_i = r_{i+1} = \cdots = r_n = \frac{M}{i-1+M(n-i-1)} \). If we denote the associated \( T \) value by \( T_A(i) \), the best allocation in form \( A \) is then given by \( \min_{1<i<n} T_A(i) \).

For the allocations in form \( B \), Theorem 3.4 presents some properties of the lower and upper separators.

**Theorem 3.4** If \( \frac{p_i}{p_j} > M^2 \) and the optimal solution \( R = (r_1, r_2, \ldots, r_n) \) to the MHash bandwidth allocation problem has form \( B \), the lower and upper boundary indexes \( i \) and \( j \) satisfy: (i) \( \frac{p_i}{p_j} < M^2 \); and (ii) \( \frac{p_{i+1}}{p_{i-1}} \geq M^2 \).

**Proof:** We first prove claim (i) by contradiction. Assume on the contrary that \( \frac{p_i}{p_j} \geq M^2 \).

We construct a new allocation \( R' \) by moving some bandwidth fraction from \( d_i \) to \( d_j \) under the constraint that the relative order of bandwidth fractions does not change, i.e.,

\[ R' = (r_1, \ldots, r_{i-1}, r'_i, r_{i+1}, \ldots, r_{j-1}, r'_j, r_{j+1}, \ldots, r_n), \]

where \( r'_i = r_i - \Delta \), \( r'_j = r_j + \Delta \), and \( \Delta = \min(r_{j+1} - r_j, r_i - r_{i-1}) > 0 \). It is easy to see that \( R' \) satisfies the differentiation constraint of the MHash bandwidth allocation problem. The access latency of \( R' \) is given by

\[ T' = \frac{1}{2} \left( \frac{p_i}{r_i - \Delta} + \frac{p_j}{r_j + \Delta} + \sum_{k=1 \atop k \neq i \atop k \neq j}^{n} \frac{p_k}{r_k} \right). \]

Note that the access latency of \( R = (r_1, r_2, \ldots, r_n) \) is

\[ T = \frac{1}{2} \sum_{k=1}^{n} \frac{p_k}{r_k}. \]
Thus,

\[
T - T' = \frac{1}{2} \left( \frac{p_i}{r_i} + \frac{p_j}{r_j} - \left( \frac{p_i}{r_i - \Delta} + \frac{p_j}{r_j + \Delta} \right) \right)
\]

\[
= \frac{1}{2} \left( \frac{p_j \Delta}{(r_j + \Delta) r_j} - \frac{p_i \Delta}{(r_i - \Delta) r_i} \right)
\]

\[
= \frac{(p_j r_i (r_i - \Delta) - p_i r_j (r_j + \Delta))}{2 r_i r_j (r_i - \Delta) (r_j + \Delta)}.
\]

Since \( r_i - \Delta \geq r_{i-1} \) and \( r_j + \Delta \leq r_{j+1} \), we have

\[
\frac{r_j + \Delta}{r_i - \Delta} \leq \frac{r_{j+1}}{r_{i-1}} = M.
\]

Note that \( \frac{p_j}{r_i} < M \) and \( \frac{p_j}{r_i} \geq M^2 \). Therefore,

\[
\frac{r_j + \Delta}{r_i - \Delta} < M^2 \leq \frac{p_j}{p_i}
\]

and

\[
p_j r_i (r_i - \Delta) > p_i r_j (r_j + \Delta).
\]

Thus, it follows that \( T > T' \) which contradicts with the optimality of \( R \).

Next, we prove claim (ii) by contradiction. Assume on the contrary that \( \frac{p_{j+1}}{p_{i-1}} < M^2 \).

We construct a new allocation \( R'' \) by reallocating the bandwidth between \( d_{i-1} \) and \( d_{j+1} \) proportionally to the square-root of their access probabilities, i.e.,

\[
R'' = (r_1, \ldots, r_{i-2}, r''_{i-1}, r_i, \ldots, r_j, r''_{j+1}, r_{j+2}, \ldots, r_n),
\]

where

\[
r''_{j+1} = \left( r_{i-1} + r_{j+1} \right) \cdot \frac{\sqrt{P_{j+1}}}{\sqrt{P_{j+1}} + \sqrt{P_{i-1}}}
\]

and

\[
r''_{i-1} = \left( r_{i-1} + r_{j+1} \right) \cdot \frac{\sqrt{P_{i-1}}}{\sqrt{P_{j+1}} + \sqrt{P_{i-1}}}.
\]

It is easy to see that \( R'' \) satisfies the differentiation constraint of the MHash bandwidth allocation problem. The access latency of \( R'' \) is given by

\[
T'' = \frac{1}{2} \left( \frac{p_{i-1}}{r''_{i-1}} + \frac{p_{j+1}}{r''_{j+1}} + \sum_{k=1,k\neq i-1,k\neq j+1}^{n} \frac{p_k}{r_k} \right).
\]
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

Since \( r_{j+1} = M r_{i-1} \),

\[
T - T'' = \frac{1}{2} \left( \frac{p_{i-1}}{r_{i-1}} + \frac{p_{j+1}}{r_{j+1}} - \frac{p_i}{r_i} - \frac{p_{j+1}}{r_{j+1}} \right) 
= \frac{1}{2} \left( \frac{p_i}{r_i} - \frac{p_{j+1}}{r_{j+1}} \right) 
= \frac{(M \sqrt{p_{i-1}} - \sqrt{p_{j+1}})^2}{2r_{i-1}(M + 1)M} > 0.
\]

Therefore, \( T > T'' \) which contradicts with the optimality of \( R \).

Hence, the theorem is proven.

Given the lower and upper separators \( i \) and \( j \), let \( r_1 + r_{i+1} + \cdots + r_j = X \). Based on Lagrange multiplier theorem, the access latency is minimized when \( r_k = \frac{\sqrt{p_k}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X \) for each \( i \leq k \leq j \). Note that these settings satisfy the differentiation constraint because Theorem 3.4 shows that \( \sum p_i < M^2 \). The remaining fractions in the allocation are given by

\[
r_1 = r_2 = \cdots = r_{i-1} = \frac{1 - X}{i - 1 + M(n - j)},
\]

and

\[
r_{j+1} = r_{j+2} = \cdots = r_n = \frac{M(1 - X)}{i - 1 + M(n - j)}.
\]

Therefore,

\[
T = \frac{1}{2} \left( \frac{(\sum_{m=i}^j \sqrt{p_m})^2}{X} + \frac{M \cdot \sum_{m=i}^{j-1} p_m + \sum_{m=j+1}^n \sqrt{p_m} \cdot (i - 1 + M(n - j))}{M(1 - X)} \right).
\]

Since \( r_{i-1} < r_i \) and \( r_j < r_{j+1} \), we have

\[
\frac{1 - X}{i - 1 + M(n - j)} < \frac{\sqrt{p_i}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X,
\]

and

\[
\frac{\sqrt{p_j}}{\sum_{m=i}^j \sqrt{p_m}} \cdot X < \frac{M(1 - X)}{i - 1 + M(n - j)}.
\]
It follows that
\[
\frac{1}{\sum_{m=1}^{j} \sqrt{p_m}} + 1 < X < \frac{M}{\sum_{m=1}^{j} \sqrt{p_m} + M}.
\]  
(3.6)

We now show how to compute the minimum value of \( T \) in the range of \( X \) given by Equation (3.6). To simplify the presentation, we denote the average access latency by
\[
T = \frac{C_A}{X} + \frac{C_B}{1 - X},
\]
where
\[
C_A = \frac{1}{2} \left( \sum_{m=1}^{j} \sqrt{p_m} \right)^2 > 0
\]
and
\[
C_B = \frac{1}{2} \frac{(M \cdot \sum_{m=1}^{i-1} p_m + \sum_{m=j+1}^{n} p_m) \cdot (i - 1 + M(n - j))}{M} > 0.
\]

Then, the derivative of \( T \) with respect to \( X \) is given by
\[
\frac{dT}{dX} = -\frac{C_A}{X^2} + \frac{C_B}{(1 - X)^2}.
\]  
(3.7)

It is easy to see that: when \( X = \frac{\sqrt{C_A}}{\sqrt{C_A + \sqrt{C_B}}} \), \( \frac{dT}{dX} = 0 \); when \( X < \frac{\sqrt{C_A}}{\sqrt{C_A + \sqrt{C_B}}} \), \( \frac{dT}{dX} < 0 \); and when \( X > \frac{\sqrt{C_A}}{\sqrt{C_A + \sqrt{C_B}}} \), \( \frac{dT}{dX} > 0 \). Thus, without restricting the range of \( X \), the minimum value of \( T \) is reached when \( X = \frac{\sqrt{C_A}}{\sqrt{C_A + \sqrt{C_B}}} \). Following Equation (3.6), \( X \) is restricted to the range of \((L, R)\) in our problem, where

\[
L = \frac{1}{\sum_{m=1}^{i} \sqrt{p_m} + 1}
\]
and
\[
R = \frac{M}{\sum_{m=1}^{j} \sqrt{p_m} + M}.
\]

Depending on the relative order of \( L, R \) and \( \frac{\sqrt{C_A}}{\sqrt{C_A + \sqrt{C_B}}} \), the minimum value of \( T \) for \( X \)’s range \((L, R)\) is computed as follows:
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

- If \( L < \frac{\sqrt{C_A}}{\sqrt{C_A} + \sqrt{C_B}} < R \), the minimum value of \( T \) is reached when \( X = \frac{\sqrt{C_A}}{\sqrt{C_A} + \sqrt{C_B}} \), and the minimum value is given by \((\sqrt{C_A} + \sqrt{C_B})^2\).

- If \( L < R \leq \frac{\sqrt{C_A}}{\sqrt{C_A} + \sqrt{C_B}} \), the minimum value of \( T \) is reached when \( X = R \), and the minimum value is given by \( \frac{C_A}{R} + \frac{C_B}{1-R} \).

- If \( \frac{\sqrt{C_A}}{\sqrt{C_A} + \sqrt{C_B}} \leq L < R \), the minimum value of \( T \) is reached when \( X = L \), and the minimum value is given by \( \frac{C_A}{L} + \frac{C_B}{1-L} \).

Given the lower and upper separators \( i \) and \( j \), if we denote the minimum value of \( T \) by \( T_B(i, j) \), the best allocation in form \( \mathcal{B} \) is then given by

\[
\min_{\sum_{l=1}^{M^2} p_l \geq M^2} T_B(i, j).
\]

Therefore, if \( \frac{p_n}{p_1} > M^2 \), the optimal MHash bandwidth allocation produces the minimum \( T \) value of

\[
\min \left( \min_{1 \leq i \leq n} T_A(i), \min_{\sum_{l=1}^{M^2} p_l \geq M^2} T_B(i, j) \right).
\]

Assume that the least frequently accessed item is broadcast once per cycle. Having obtained the optimal allocation \((r_1, r_2, \ldots, r_n)\), the number of times each item \( d_i \) should be broadcast in a cycle is given by

\[
c_i = \begin{cases} 
1 & i = 1, \\
\left\lfloor \frac{p_i}{r_i} - \frac{1}{2} \right\rfloor & 2 \leq i \leq n.
\end{cases}
\]

3.3 Performance Evaluation

3.3.1 Experimental Setup

We have conducted simulation experiments to compare MHash indexing with a wide range of existing schemes. Table 3.2 summarizes the system parameters and their settings in the experiments. We simulate a set of \( n \) data items whose access probabilities are
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

assumed to follow a Zipf-like distribution. Specifically, the access probability \( p_i \) of the \( i \)th most popular item follows \( p_i \propto \frac{(1/i)^\theta}{\sum_{i=1}^n(1/i)^\theta} \), where \( \theta > 0 \) is the Zipf parameter [Zip49]. It is obvious that the higher the value of \( \theta \), the more skewed the access distribution. A \( \theta \) value of 0 degenerates the Zipf-like distribution to a uniform distribution. The default Zipf parameter was set at 1.0. The default replication bound \( M \) was set at 8. The default bucket size and item size were set at 1KB and 128 bytes respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>number of data items</td>
<td>5000</td>
</tr>
<tr>
<td>( M )</td>
<td>replication bound</td>
<td>8</td>
</tr>
<tr>
<td>( \theta )</td>
<td>Zipf parameter of access distribution</td>
<td>1.0</td>
</tr>
<tr>
<td>( S_b )</td>
<td>size of a bucket</td>
<td>1KB</td>
</tr>
<tr>
<td>( S_d )</td>
<td>size of a data item</td>
<td>128 bytes</td>
</tr>
</tbody>
</table>

We refer to the number of slots that can be accommodated in a bucket as the bucket capacity \( P \). It is given by \( P = \lceil \frac{S_b}{S_d + S_{sh}} \rceil \), where \( S_b \) and \( S_d \) are the bucket size and item size respectively, and \( S_{sh} \) and \( S_{ph} \) are the per-bucket and per-slot indexing overheads respectively. For MHash indexing, as mentioned in Section 3.2.1, the following information is recorded in the header of each bucket: the hash function \( H \), the cycle length \( N \) and the replication bound \( M \). We assume the hash function is characterized by two constants (e.g., see \( H_i(k,1) \) below) and an offset (Theorem 3.1). The above information together with the bucket number and the bucket capacity\(^5\) results in an \( S_{bh} \) of 28 bytes. In addition to the broadcast item, the distance pointer is also broadcast in each slot. Therefore, \( S_{sh} \) was set at 4 bytes.

The keys of data items are assumed to be integers and were randomly generated in our experiments. The key values are uniformly distributed in the range of \([0, 2^{32} - 1]\).

\(^5\)The bucket number and bucket capacity are included for the purpose of deriving slot numbers.
We used the following hash functions:

\[ \mathcal{H}_1(k, 1) = (A \cdot ((A + B) \oplus k) + B) \mod 2^{31}, \]

where \( k \) is the key value, \( A = 1103515245 \), \( B = 12345 \) and \( \oplus \) is a bitwise exclusive-or operation [TR98], and a universal hash function

\[ \mathcal{H}_2(k, 1) = (x_1m_1 \oplus x_2m_2 \oplus \cdots \oplus x_{32}m_{32}), \]

where \( x_1x_2 \ldots x_n \) is the binary representation of key value, and \( m_i \)'s are 0-1 bit vectors of length 32 [CW79]. These hash functions were extended based on the techniques described in Sections 3.2.2 and 3.2.3 for MHash indexing. In addition, we also tested MHash without attempting to equalize the spaces between the hashed slots of a data item (i.e., without applying the techniques presented in Section 3.2.3). To do so, we used the following two-argument hash function:

\[ \mathcal{H}_3(k, l) = (A \cdot ((A + B) \oplus k) + A \cdot ((A + B) \oplus l) + B) \mod 2^{31} \]

where \( k \) is the key value and \( l \) is the sequential identifier. Furthermore, we tested MHash with a synthetic collision-free hash table in which no pair of items are hashed to the same slot. To generate a collision-free hash table, we first extend the hash function \( \mathcal{H}_1(k, 1) \) to a hole-free two-argument hash function based on the techniques described in Sections 3.2.2 and 3.2.3. The collisions in item-to-slot mappings are then resolved by the chaining method described in Section 3.2.1. The collision-free hash table is generated following the final mappings between the items and their broadcast slots after the collisions are resolved. On constructing the broadcast schedule, the expected access latency and tuning time of each data item were first calculated by taking an average over all possible locations of the initial probe. The overall access latency and tuning time were then computed by taking a weighted average over all data items based on their access probabilities. Both
Chapter 3. Energy-Efficient and Access Latency Optimized Indexing for Wireless Data Broadcast

Metrics are measured in the unit of bucket, which is the smallest accessible unit of broadcast. We have tested a wide range of parameter settings. For each setting, we randomly generated 100 sets of key values. Each set of keys were randomly ordered into a list, where the $i$th key on the list was assumed to be the $i$th most popular item. The average performance of these 100 simulation runs is plotted for performance comparison. The 95% confidence interval of the experimental results was calculated to be within 1.5 percent of the mean.

In addition to MHash, the following existing schemes were included in the experiments for comparison purposes: latency-optimal broadcast (abbreviated as LatOpt) [VH99], one-argument hash index (FlatHash) [IVB94], distributed tree index (DistTree) [IVB97], exponential index (Exponential) [XLT04, XLT+06], unbalanced tree index (Unbalance-Tree) [CNY03, SV96], and hybrid index (Hybrid) [HLL01]. These schemes have been briefly described in Chapter 2. The latency-optimal broadcast differentiates the broadcast frequencies of data items based on the square-root rule of bandwidth allocation (i.e., $r_i \propto \sqrt{p_i}$) [VH99]. The broadcast instances of each item are assumed to be equally spaced in the broadcast schedule. Thus, LatOpt is used as a yardstick (lower bound) on access latency. However, LatOpt does not include any index in the broadcast. This leads to a tuning time equal to the access latency. The remaining five schemes all build index on broadcast data. All these schemes, except the Hybrid index, construct a flat schedule where each item is broadcast exactly once in a cycle regardless of the access distribution. The replicated levels of index nodes in DistTree and the index base and chunk size in Exponential were tuned to optimize access latency. The Hybrid scheme used three disks with relative broadcast frequencies of 3, 2 and 1 in broadcast disk scheduling. In all tree-based indexes, each key value and offset in the index tables was assumed to take up 4 bytes.
3.3.2 Comparison with LatOpt and FlatHash

In this section, we investigate the performance of MHash under different replication bounds and hash functions. We tested MHash with four different hash functions: \( H_1 \), \( H_2 \), \( H_3 \), and a synthetic collision-free hash table. Their results are labeled MHash\( (H_1) \), MHash\( (H_2) \), MHash\( (H_3) \), and MHash(Ideal) respectively. We also compare MHash against LatOpt and FlatHash.

![Figure 3.5: Average Access Latency vs. Replication Bound](image)

Figure 3.5 shows the access latency as a function of replication bound \( M \). Note that FlatHash and LatOpt do not rely on \( M \), so their performance is independent of \( M \). When \( M = 1 \), MHash broadcasts all items exactly once in each cycle. So, its access latency is similar to that of FlatHash which also employs flat broadcast. When \( M \) increases, MHash becomes more flexible in bandwidth allocation. Therefore, the access latency of MHash decreases rapidly with increasing \( M \). As shown in Figure 3.5, an \( M \) value of 2 improves the access latencies of MHash\( (H_1) \) and MHash\( (H_2) \) by 30% compared to \( M = 1 \). Comparing with FlatHash, their access latencies are 50% lower when \( M = 8 \).
We conducted two sets of experiments for LatOpt. In the first set, the entire bucket was fully used to accommodate data items, i.e., bucket capacity \( P = \left\lfloor \frac{S_p}{S_d} \right\rfloor \). This represents the lower bound access latency without any indexing overhead. The second set of experiments assumed a bucket capacity equal to that of MHash. This represents the lower bound access latency in the presence of MHash’s indexing overhead. The results of these two sets of experiments are labeled LatOpt(NoOverhead) and LatOpt(MHashOverhead) respectively. As shown in Figure 3.5, MHash approaches LatOpt(MHashOverhead) in access latency when \( M \) grows beyond 8. This implies our spacing and bandwidth allocation techniques presented in Sections 3.2.3 and 3.2.4 succeed in optimizing the access latency by means of non-flat broadcast. We also note that the difference between LatOpt(MHashOverhead) and LatOpt(NoOverhead) is small in access latency. This shows MHash has little indexing overhead.

![Figure 3.6: Average Tuning Time vs. Replication Bound](image)

Figure 3.6 shows the tuning time as a function of \( M \). Comparing MHash with FlatHash, we note that when \( M = 1 \), MHash has a lower tuning time than FlatHash. This is because they use different methods to resolve collisions. By using chaining, MHash...
guarantees that for each slot, one of the items hashed to it is broadcast in the slot. On the other hand, FlatHash uses linear probing. If a collision pushes an item forward to a slot, the slot would not be used to accommodate any item hashed to it. This has the effect of increasing the tuning time to all the items hashed to the slot by 1.

In general, the tuning time of MHash increases with $M$. This is because a larger replication bound increases the number of cheating slots. Due to pruning in data accesses, MHash's tuning time increases almost logarithmically\(^6\) with $M$ (i.e., very slowly). As shown in Figure 3.6, the tuning times of $\text{MHash}(H_1)$ and $\text{MHash}(H_2)$ are lower than that of FlatHash up to an $M$ value of 8 and are only 1.5 buckets higher than that of FlatHash when $M$ rises to 128. On the other hand, since LatOpt($\text{MHashOverhead}$) and LatOpt($\text{NoOverhead}$) do not use air indexing, their tuning times are identical to the access latencies which are not shown in Figure 3.6 due to high values.

![Figure 3.7: Normalized Energy Cost vs. Replication Bound](image)

The total energy cost of data accesses is the sum of that consumed in the active and doze modes. While tuning time measures the energy consumed in the active mode,

\(^6\)Note that the axis of $M$ is in logscale.
access latency largely reflects the energy consumed in the doze mode. In the absence of concurrent accesses to multiple items by a client, the total energy cost of a data access can be approximated by

\[ E = \text{access\_latency} - \text{tuning\_time} \cdot r_{\text{doze}} + \text{tuning\_time} \cdot r_{\text{active}}, \]

where \( r_{\text{doze}} \) and \( r_{\text{active}} \) are the energy consumption rates of the doze and active modes respectively. Figure 3.7 plots the energy cost normalized by that of \( \text{MHash}(\mathcal{H}_1) \) at \( M = 8 \) when \( r_{\text{doze}} \) and \( r_{\text{active}} \) are respectively set at 60mW and 950mW, representing a typical wireless PC card ORiNOCO [VBH03]. The energy costs of LatOpt(MHashOverhead) and LatOpt(NoOverhead) are too high to be shown in the figure. It is clearly seen that MHash significantly improves energy efficiency. When \( M = 8 \), MHash(\mathcal{H}_1) and MHash(\mathcal{H}_2) outperform FlatHash by over 45% and LatOpt(MHashOverhead) by over 90% (not shown in Figure 3.7) in energy cost.

Comparing different hash functions, we note that the tuning time of MHash improves with decreasing collision rate. As shown in Figure 3.6, MHash(Ideal) outperforms MHash(\mathcal{H}_1) and MHash(\mathcal{H}_2) in tuning time. However, the improvement is not significant. This implies the hash functions \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) have low collision rates. MHash(\mathcal{H}_3), on the other hand, performs worse than MHash(\mathcal{H}_1) and MHash(\mathcal{H}_2) in terms of the tuning time. This is because \( \mathcal{H}_3 \) does not control the spaces between the hashed slots of a data item and has a higher collision rate. It is also seen that the access latency of MHash is greatly affected by the spacing between the broadcast instances. As shown in Figure 3.5, MHash(\mathcal{H}_1), MHash(\mathcal{H}_2) and MHash(Ideal) result in similar access latencies which are significantly lower than that of MHash(\mathcal{H}_3) (by about 20%). This is because in \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \), the spaces between the hashed slots of a data item follow Equation (3.5). In contrast, \( \mathcal{H}_3 \) does not make effort to equalize the spaces and simply allocates broadcast instances through random hashing.

69
In the rest of this chapter, we shall report only the results of MHash(\(\mathcal{H}_I\)). Since the access latency and energy cost of MHash indexing remains similar when \(M\) grows beyond 8, the default value of \(M\) was set at 8 in the remaining experiments.

### 3.3.3 Comparison with DistTree, Exponential, UnbalanceTree and Hybrid

In this section, we study the performance of MHash under different access distributions and compare it against DistTree, Exponential, UnbalanceTree and Hybrid indexes. Figures 3.8 and 3.9 show the performance results for various Zipf parameters.

**Figure 3.8: Average Access Latency vs. Zipf Parameter**

DistTree and Exponential both construct flat schedules regardless of the access distribution. Each item is broadcast exactly once in a cycle and in the order of key value. So, their access latencies remain similar over a wide range of access distribution (see Figure 3.8). With multiple index trees that share links, Exponential allows searching to start at anywhere in the broadcast. Thus, its access latency is lower than that of DistTree. In contrast, MHash uses non-flat broadcast. Its access latency is similar to that of Exponential under uniform access distribution and reduces substantially when \(\theta\)
increases. When $\theta = 1.5$, MHash's access latency is over 80% and 75% lower than those of DistTree and Exponential respectively. Similar performance trends are observed for tuning time. Figure 3.9 shows that the tuning time of MHash decreases with increasing access skewness. This demonstrates MHash's nature that popular items have less tuning time than unpopular items (see Section 3.2.1). When the skewness in access distribution increases, popular items take up a larger portion of all requests, thereby decreasing the average tuning time. MHash's tuning time is much lower than those of DistTree and Exponential at $\theta$ values higher than 1.

In UnbalanceTree, more frequently accessed items are placed closer to the root of the index tree. Therefore, its tuning time decreases with increasing access skewness. However, like other tree-based indexes, the searching must start from the root index node in UnbalanceTree. Normally, the client needs to tune to the broadcast channel at least twice (an initial probe and the root index node) before it accesses the data item. As a result, UnbalanceTree's tuning time cannot be further reduced below 3. In contrast, hash-based indexes allow searching to start from anywhere. As shown in Figure 3.9, MHash reduces the tuning time by over 18% and 30% compared to UnbalanceTree at $\theta$ values.
of 1.0 and 1.5 respectively. When constructing the broadcast schedule, UnbalanceTree places popular items closer to the root index node than unpopular items to reduce access latency. Thus, UnbalanceTree’s access latency decreases with increasing access skewness. However, since UnbalanceTree uses flat broadcast, Figure 3.8 shows that its access latency is substantially higher than that of MHash. Moreover, unlike DistTree, UnbalanceTree does not use replication to reduce the latency to access the root index node. Therefore, UnbalanceTree’s access latency is even higher than that of DistTree under uniform access distribution.

The Hybrid scheme incorporates broadcast disk scheduling [AAFZ95], tree-based index [IVB97] and signature-based index [LL96]. Broadcast disk scheduling differentiates the broadcast frequencies of data items based on their popularity. However, the differentiation is not as fine grained as that in MHash. Therefore, as seen from Figure 3.8, the access latency of Hybrid reduces with increasing access skewness but is still much higher than MHash under non-uniform access distribution. Due to non-flat broadcast, Hybrid constructs index only for short segments of broadcast, where each segment holds a sequence of items with increasing key values. So, the effectiveness of indexing diminishes. Moreover, since a signature does not provide the arrival times of data items, when a match is found, the buckets indexed by the signature have to be searched sequentially. As a result, Hybrid has considerably higher tuning time than the other schemes (see Figure 3.9).

Figure 3.10 summarizes the total energy cost for different schemes. It shows that the energy cost of MHash decreases with increasing access skewness and is much lower than that of other schemes under non-uniform access distribution. MHash saves over 40% and 60% of the energy compared to the other schemes at \( \theta \) values of 1.0 and 1.5 respectively.
CHAPTER 3. ENERGY-EFFICIENT AND ACCESS LATENCY OPTIMIZED INDEXING FOR WIRELESS DATA BROADCAST

Figure 3.10: Normalized Energy Cost vs. Zipf Parameter

3.4 Summary

In this chapter, we have presented an indexing scheme called MHash that optimizes access latency and tuning time in an integrated fashion for wireless data broadcast. MHash reduces tuning time by mapping data items to the slots in the broadcast schedule via a hash function. We have shown that a hole-free hash function for the purpose of broadcast scheduling can be constructed by injecting an offset into an arbitrary hash function. Meanwhile, the two-argument nature of the hash function allows each data item to be broadcast for an adjustable number of times in a cycle. Popular items are broadcast more frequently than unpopular ones, thereby enabling non-flat data broadcast to reduce access latency. We have derived an optimal bandwidth allocation for MHash indexing. Experimental results show that under non-uniform access distribution, MHash outperforms state-of-the-art air indexing schemes in energy efficiency and achieves access latency close to optimal broadcast scheduling.
Chapter 4

\textbf{kNN Query Processing in Wireless Sensor Networks}

\section{Introduction}

Sensor networks provide us with the means of gathering data from the physical world. As discussed in Chapter 1, allowing users to issue queries to sensor networks is an important way to perform data gathering. Due to geographically distributed deployment of sensor nodes, spatial information plays an important role in the representation of sensor data. Thus, it is natural to collect data from the sensor network by specifying spatial conditions in the queries. Existing work on spatial query processing in sensor networks has focused on one-shot queries that request the readings from the sensor nodes based on the node locations, e.g., from the sensor nodes in a window based area [XLXM06]. In addition to the locations of sensor nodes, the data captured by the sensor nodes may also include spatial information. For example, in object tracking sensor networks, the object locations detected by the sensor nodes are represented by geographical coordinates. Different from the existing work, we consider spatial queries that specify spatial conditions on the collection of object locations detected by object tracking sensor networks. We focus on \textit{kNN} queries that request a given number of \textit{k} detected object locations nearest to a specified geographical point called the \textit{query point}. 

74
Energy efficiency is a critical design consideration of wireless sensor networks. To prolong network lifetime, it is important to reduce network-wide energy consumption in data gathering. In wireless sensor networks, energy is mainly consumed in exchanging messages between the sensor nodes [Pot00]. Thus, to reduce energy consumption, we need to reduce the number of message transmissions. A straightforward centralized scheme for processing kNN queries in object tracking sensor networks is to continuously send all sampled object locations to a base station and route the queries to the base station for processing. However, the centralized scheme is likely to suffer from unnecessary message exchanges. This is because kNN queries are usually localized in that only the locations of the objects close to the query points are needed for query processing. The objects that are far away from the query points are normally not relevant to the query results and thus, their locations do not have to be reported. To improve energy efficiency, it is desirable to store the acquired data locally at the sensor nodes in a distributed manner and process the queries in-network [IGE00, XTL05, YTL06]. Recent technology advances have substantially improved the capacities and energy efficiency of local storage for sensor networks [DGMS07].

Motivated by the localized property of kNN queries, in this chapter, we propose a localized scheme to evaluate one-shot and continuous kNN queries in object tracking sensor networks. A two-phase search mechanism is developed to process one-shot kNN queries. To support continuous monitoring of the k nearest objects, the two-phase search mechanism is first used to conduct an initial evaluation of the query. A monitoring area is then setup in the network so that only the relevant location updates are collected to maintain the k nearest objects. We develop methods to reevaluate the kNN query based on the location updates collected from the sensor nodes in the monitoring area. Experimental results show that establishing a monitoring area for continuous kNN query processing greatly reduces energy consumption. Due to object movement, the monitoring
area needs to be expanded and shrunk on the fly. We analyze the optimal maintenance of the monitoring area and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. The adaptive algorithm is experimentally shown to achieve close-to-optimal performance.

Some grid-based methods have been explored in continuous monitoring of $k$NN queries in the context of spatial databases [GL04, MHP05, MPBT05, PXK+02, XMA05, YPK05]. Yu et al. [YPK05] proposed a method called YPK-CNN, assuming a centralized repository storing the locations of all objects which are indexed by a grid structure. The location updates are continuously sent to the centralized repository. $k$NN queries are reevaluated periodically according to the new locations of the objects. YPK-CNN achieves low computation time and memory usage. The CPM scheme proposed by Mouratidis et al. [MHP05] also uses a grid structure for indexing. It further reduces the computation time by optimizing the visiting order of the grid cells to handle the location updates more effectively. Similar to YPK-CNN, the SEA-CNN scheme proposed by Xiong et al. [XMA05] also stores the object locations in a centralized repository and indexes them by a grid structure. Each query is assigned a circle called the answer region centered at the query point and with a radius equal to the distance between the query point and the $k$th nearest object in the $k$NN result. When location updates are collected, the query is reevaluated only if the updates affect its answer region. SEA-CNN achieves high scalability in terms of computation time when there are multiple queries. The above methods assume that there is a centralized repository to store all object locations and all location updates are simply reported to the centralized repository. However, such centralized storage is costly for object tracking sensor networks due to their energy constraints. Therefore, these methods are not appropriate for $k$NN monitoring in sensor networks.

Other relevant work on spatial query monitoring includes the MobiEyes algorithm proposed by Gedik et al. [GL04] and a threshold-based algorithm proposed by Mouratidis.
et al. [MPBT05]. Similar to YPK-CNN, SEA-CNN and CPM, these studies also assume a centralized server in the system. But differently, the MobiEyes and threshold-based algorithms assume *smart* objects that have some storing and processing capabilities. When an object moves away from its current position, the object can decide whether to send a location update to the server or not. Both the MobiEyes and threshold-based algorithms aim at reducing the communication cost between the objects and the server by eliminating unnecessary location updates. MobiEyes [GL04] focuses on monitoring window queries by assigning a safe region to each query. The objects within the safe region periodically check whether they are in the query windows. Only the objects within the query windows report their locations to the server. The threshold-based algorithm [MPBT05] focuses on monitoring $k$NN queries and assigns a distance range to each object in the $k$NN result set based on its distance to the query point. The distance range for the $i$th nearest object is delineated by the midpoint between the $i$th and $(i-1)$th nearest objects and the midpoint between the $i$th and $(i+1)$th nearest objects. Only when an object moves out of its distance range to the query point is the location update of the object sent to the server. The approaches of [GL04] and [MPBT05] are similar to many studies on continuous monitoring in sensor networks [SBLC03, SMY06, WXTL06, SBY06]. The general idea of these studies is to set some constraints at the sensor nodes to prevent them from reporting all sensor data to the base station. Only when the constraint is violated should the sensor node report the updates of sensor data. The detailed setting of the constraints at relevant sensor nodes depends on specific query types. The above studies have focused on monitoring stationary phenomena (e.g., temperature and humidity). A stationary phenomenon is always captured by the same sensor node over time. In object tracking sensor networks, however, the object location is detected by different sensor nodes as the object moves. In this chapter, we set up monitoring areas in the object tracking sensor network to eliminate unnecessary updates from the sensor nodes.
The rest of this chapter is organized as follows. Section 4.2 presents one-shot \( k \)NN query processing. Section 4.3 describes continuous monitoring of \( k \)NN queries. The maintenance strategies of the monitoring area are presented in Section 4.4. Section 4.5 describes the experimental setup and discusses the experimental results. Finally, Section 4.6 summaries the chapter.

### 4.2 Processing One-Shot \( k \)NN Queries

We consider a sensor network with the sensor nodes distributed on a 2-D field. The sensor nodes are aware of their locations through GPS [HWLC97] or other localization algorithms [NN03]. Each sensor node can communicate directly with the nodes (called neighbors) within a transmission range \( R_t \). Through message exchanges, each sensor node is aware of the geographical locations of its neighbors. We assume the network is connected, i.e., any sensor node can communicate with any other node either directly or indirectly through a routing protocol.

We assume a dense sensor network in which the geographical area of interest is fully covered by the sensing ranges of the sensor nodes. At each sampling interval, the location of each object is detected by a sensor node in the network.\(^1\) Suppose the sensing range of each sensor node is \( R_s \). Then, the detecting sensor node of an object must be located within distance \( R_s \) to the object. Instead of sending all detected object locations to a central repository, we propose to store them locally at the detecting sensor nodes. \( k \)NN queries can be made via sensor nodes from anywhere in the network (e.g., through the hand-held devices of the users). Recall that each \( k \)NN query specifies a query point \( q \). The sensor node receiving a query first forwards it towards \( q \) through GPSR routing [KK00]. Given a destination location, GPSR routes the message to the node closest

\(^1\)Although the sensor nodes may work collaboratively to determine the location of an object in their vicinity [LWH02], we assume that for each object, only one sensor node (the sensing leader or cluster head) is responsible for storing the computed object location at each sampling interval [XTL05, XWL04b, ZSR02]. For simplicity, the detecting sensor node in the rest of this chapter refers to this node.
CHAPTER 4. ANN Query Processing in Wireless Sensor Networks

to the destination location [RKL+02]. Thus, the node closest to the query point \( q \) would receive the query. We shall refer to this node as the query sink.

One simple approach to process one-shot \( k \text{NN} \) queries is to flood the query over the entire sensor network starting from the query sink. The sensor nodes, on receiving the query, send all detected object locations to the query sink for it to compute the \( k \text{NN} \) result set. Though simple, the flooding scheme would incur high message complexity. This is because the query is disseminated to all sensor nodes. Moreover, the flooding scheme does not take advantage of the localized property of the \( k \text{NN} \) query — all detected object locations are simply sent to the query sink.

![Figure 4.1: Grid Structure](image)

In this section, we present the localized scheme to process one-shot \( k \text{NN} \) queries. The localized scheme also forms the basis for initial and continuous evaluations of long running \( k \text{NN} \) queries. For each \( k \text{NN} \) query, we define a grid structure to conceptually partition the sensor network into a set of grid cells. As shown in Figure 4.1, each grid cell is a square of size \( \alpha \times \alpha \) and the query point \( q \) is set as the centroid of a grid cell. Each sensor node can autonomously compute the grid cell in which it is located provided that it knows the location \((q.x, q.y)\) of the query point \(q\). The centroid of the grid cell containing a sensor node located at \((x, y)\) is given by \((q.x + \left\lfloor \frac{x-(q.x-\frac{\alpha}{2})}{\alpha} \right\rfloor \cdot \alpha, q.y + \left\lfloor \frac{y-(q.y-\frac{\alpha}{2})}{\alpha} \right\rfloor \cdot \alpha)\).
shall be discussed later, the knowledge of \( q \) is made known to relevant sensor nodes in query processing.

Starting from the query sink, the query message is passed along the sensor nodes to search for nearby objects in the neighboring grid cells. When a grid cell is visited, the object locations detected by the sensor nodes in the cell are collected by one sensor node called the R-node. To do so, the R-node broadcasts a one-hop probe message to the sensor nodes in the cell. To guarantee that all nodes within the cell can hear the probe message, the diameter of the grid cell (i.e., \( \sqrt{2} \alpha \)) should be less than the transmission range \( R_t \). Therefore, we set the cell size at \( \alpha = \frac{1}{\sqrt{2}} \cdot R_t \). To cover the entire cell with one-hop broadcast, the R-node of a grid cell must be located within distance \( \frac{1}{2} R_t \) to the centroid of the cell (as shown in Figure 4.1).

The visit to a grid cell \( G \) is divided into two steps.

- In the first step, the query message is routed to the R-node of \( G \). To do so, the sensor node currently holding the query message first checks whether any of its neighbors is within distance \( \frac{1}{2} R_t \) to \( G \)'s centroid. If such a neighbor exists, it is selected as the R-node of \( G \) and the query message is sent to it in one hop. In case multiple neighbors are within distance \( \frac{1}{2} R_t \) to \( G \)'s centroid, the one closest to \( G \)'s centroid is selected as the R-node of \( G \). Otherwise, if no such neighbor exists, the query message is routed towards \( G \)'s centroid by GPSR routing. In this case, the sensor node closest to \( G \)'s centroid would receive the query message [RKL+02]. If the receiving sensor node is within distance \( \frac{1}{2} R_t \) to \( G \)'s centroid, it assumes the role of \( G \)'s R-node. Otherwise, \( G \) must be an empty grid cell without any sensor node in it. In this case, no data need to be collected from \( G \) and the query message will be routed to the next grid cell. Note that if \( G \) is non-empty, the R-node of \( G \) would be successfully selected in this step.
In the second step, the R-node broadcasts a probe message to the sensor nodes in $G$. The probe message contains the locations of the query point $q$ and $G$'s centroid. By including the query point in the probe message, each sensor node receiving the probe message can compute the centroid of the grid cell containing it. The node then compares the centroid with $G$'s centroid to determine whether it is in $G$. Only the nodes in $G$ would reply to the R-node with detected object locations if any. A number of scheduling methods exist to avoid the collisions between the replies from different nodes in $G$ [XLXM06].

The evaluation of a one-shot $k$NN query proceeds in two phases: preliminary search and expanded search. The purpose of the preliminary search is to find a boundary object and define the search space. In this step, the grid cells surrounding the query sink are visited by message passing until at least $k$ objects are found. Among these objects, the $k$th nearest object to the query point is selected as the boundary object. A search space is then defined based on the location of the boundary object to guarantee the inclusion of all sensor nodes possibly detecting an object closer to the query point than the boundary object. During the expanded search, the grid cells in the search space that are not yet visited in the preliminary search are visited to locate the $k$ nearest objects to the query point. Finally, the query result is routed back to the query sink. We now present the preliminary search and the expanded search in detail.

### 4.2.1 Preliminary Search

In the preliminary search, we need a rule to determine the order of the grid cells visited by the query message. Since the location of the boundary object determines the search space for the expanded search, to reduce search cost, we would like the boundary object to be as close to the query point $q$ as possible. Thus, it is intuitive to visit the grid cells based on their distances to $q$. We propose a circular approach to determine the
visiting order of grid cells. Specifically, the search starts from the cell centered at $q$ and is divided into rounds as shown in Figure 4.2. In each round $i$ ($i \geq 1$), the unvisited grid cells whose minimum distances to $q$ are shorter than $i \cdot \alpha$ are visited in clockwise order. Figure 4.3 shows the visiting order of grid cells in the preliminary search. Initially, the
query message contains the location of the query point \( q \). When a grid cell is visited, its R-node collects the object locations detected by the sensor nodes in the cell and records them in the query message. Given the location of \( q \), the R-node autonomously determines the next cell to visit and routes the query message to it.

The preliminary search completes when the number of objects recorded in the query message is no less than \( k \). Among these objects, the \( k \)th nearest object to the query point \( q \) is chosen as the boundary object. Denote the boundary object by \( o_b \) and its distance to \( q \) by \( d(o_b, q) \). The search space is then defined as the set of grid cells whose minimum distances to \( q \) are shorter than \( d(o_b, q) + R_s \), where \( R_s \) is the sensing range. Intuitively, if a grid cell is in the search space, the sensor nodes in the grid cell are likely to detect objects closer to the query point \( q \) than the boundary object.

### 4.2.2 Expanded Search

We refer to the grid cells to be visited in the expanded search as a *search list*. It consists of the set of grid cells in the search space that are not yet visited in the preliminary search. Note that these grid cells must be included in rounds \( i, i + 1, \ldots, j \) of the circular visiting order, where \( i \) is the round where the preliminary search ends and \( j = \left\lceil \frac{d(o_b, q) + R_s}{a} \right\rceil \). All grid cells in these rounds can be arranged in a sequence following their visiting order in the circular approach. We shall use a bit sequence of equal length to represent the search list, where a bit ‘1’ means the corresponding grid cell is in the search list and a bit ‘0’ means otherwise. To assist the mapping of bits to grid cells, the round number \( i \) is also included in the representation.

The query message in the expanded search contains the search list, the \( k \) recorded object locations, and the query point \( q \). To visit a grid cell \( G \) in the search list, the query message is again routed to the R-node of \( G \). In the expanded search, the R-node first removes \( G \) from the search list (by setting the corresponding bit in the bit sequence to
0). After the R-node collects the object locations detected by the sensor nodes in \( G \), one of the following three cases can occur: (i) no object is detected by any node in \( G \); (ii) all objects detected are further away from the query point \( q \) than the boundary object; (iii) at least one object detected is closer to \( q \) than the boundary object. In cases (i) and (ii), the search list and the object locations recorded in the message remain unchanged. In case (iii), the detected object locations closer to \( q \) are used to update the \( k \) object locations recorded in the message. Meanwhile, the updated \( k \)th nearest object is chosen as the new boundary object \( o_b \) and the search space is shrunk accordingly. The search list is then updated by removing all grid cells outside the new search space by setting their bits to 0. After visiting grid cell \( G \), the query message visits the next cell on the search list that is closest to \( G \). The expanded search continues until the search list becomes empty. On completion of the expanded search, the message is routed to the query sink and the \( k \) object locations recorded in the message form the \( k \)NN result.

Figure 4.4 shows an example of 1-NN query processing. The dark grey grid cells are visited in the preliminary search. Suppose a boundary object \( o_b \) is found when cell
A is visited. A's R-node determines the search space (defined based on the solid circle in Figure 4.4) and derives the search list (the light grey grid cells in Figure 4.4). In the expanded search, the query message starts visiting the grid cells on the search list. Suppose that when cell B is visited, an object $o_y$ closer to q than $o_x$ is found. Then, the search space is shrunk accordingly. The new search space is defined based on the dashed circle in Figure 4.4. Cells C, D, E and F are now the only four grid cells left in the updated search list. The expanded search completes after visiting these cells.

Algorithm 4.1 summarizes the algorithm executed at each R-node when a query message is received.

### 4.2.3 Cost Model for One-Shot $k$NN Query Processing

In this section, we derive a cost model to estimate the message complexity of processing one-shot $k$NN queries. The notations used in this chapter are summarized in Table 4.1. We consider a total number of $N$ sensor nodes deployed in the sensing field and assume that the sensing field is divided into a number of $M$ grid cells. Suppose a total number of $n$ objects are tracked in the sensing field. We focus on deriving the message complexity of the preliminary and expanded searches. We start by calculating the average number of grid cells visited during these two phases. Then, by estimating the message complexity of visiting one grid cell, we derive the total message complexity of query processing.

Let $p(i, j)$ be the probability of detecting an exact number of $j$ ($0 \leq j \leq n$) objects after $i$ grid cells are visited. Assume that the objects are randomly distributed in the sensing field, i.e., each object has equal probability $\frac{1}{M}$ of being detected in any grid cell. For simplicity, in our analysis, we assume that the objects in a cell are detected by the sensor nodes in the cell. There are a number of $M^n$ different ways to place $n$ objects in $M$ grid cells. Given a subset of $j$ objects out of $n$ objects, there are $i^j$ different ways to place these $j$ objects in the $i$ grid cells. For the remaining $(n - j)$ objects, there are
Algorithm 4.1 Algorithm executed at R-nodes

1: if a query message $p$ is received at a R-node $i$ responsible for a grid cell $G$ then
2:   Let $\mathcal{X}$ be the set of object locations recorded in message $p$;
3:   Collect object locations by the sensor nodes in cell $G$, and let $\mathcal{Y}$ be the set of object locations collected;
4:   if $|\mathcal{X}| < k$ and $|\mathcal{X} \cup \mathcal{Y}| < k$ then
5:     Replace $\mathcal{X}$ by $\mathcal{X} \cup \mathcal{Y}$ in message $p$;
6:     Determine the next grid cell $G'$ to visit according to the circular approach;
7:     Send out message $p$ to $G'$ to continue the preliminary search;
8:   else if $|\mathcal{X}| < k$ and $|\mathcal{X} \cup \mathcal{Y}| \geq k$ then
9:     Replace $\mathcal{X}$ in message $p$ by $k$ object locations in $\mathcal{X} \cup \mathcal{Y}$ nearest to the query point $q$ and let $o_b$ be the $k$th nearest object to $q$ in $\mathcal{X} \cup \mathcal{Y}$;
10:    Initialize the search list based on the location of $o_b$;
11:    Record the search list in $p$;
12:    Select cell $G'$ from the search list closest to cell $G$;
13:    Send out message $p$ to $G'$ to start the expanded search;
14:   else if $|\mathcal{X}| = k$ then
15:      Remove cell $G$ from the search list in message $p$;
16:      Let $o_b$ be the $k$th nearest object to the query point $q$ in $\mathcal{X}$;
17:      if $\exists o_b \in \mathcal{Y}, d(o_b, q) < d(o_b, q)$ then
18:         Replace $\mathcal{X}$ in message $p$ by $k$ object locations in $\mathcal{X} \cup \mathcal{Y}$ nearest to $q$;
19:      end if
20:     if the search list is not empty then
21:        Select cell $G'$ from the search list closest to cell $G$;
22:        Send out message $p$ to $G'$ to continue the expanded search;
23:     else
24:        Send out a result message $p'$ including $\mathcal{X}$ to the query sink;
25:     end if
26:   end if
27: end if

$(M - i)^{n-j}$ different ways to place them in the rest $(M - i)$ grid cells. Let $C(n, j)$ be the number of possible combinations for selecting $j$ objects out of $n$ objects. It follows that

$$p(i, j) = \frac{C(n, j) \cdot i^j \cdot (M - i)^{n-j}}{M^n}. \quad (4.1)$$

Let $k$ be the number of nearest objects requested by a $k$NN query. Then, the probability of detecting at least $k$ objects after $i$ grid cells are visited is given by:

$$1 - \sum_{b=0}^{k-1} p(i, b).$$
Table 4.1: Summary of Notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>the number of sensor nodes</td>
</tr>
<tr>
<td>$n$</td>
<td>the number of objects</td>
</tr>
<tr>
<td>$R_t$</td>
<td>transmission range</td>
</tr>
<tr>
<td>$R_s$</td>
<td>sensing range</td>
</tr>
<tr>
<td>$V_{max}$</td>
<td>maximum object moving velocity</td>
</tr>
<tr>
<td>$q$</td>
<td>query point</td>
</tr>
<tr>
<td>$k$</td>
<td>the number of nearest neighbors required</td>
</tr>
<tr>
<td>$\alpha \times \alpha$</td>
<td>the size of a grid cell</td>
</tr>
<tr>
<td>$s \times s$</td>
<td>the size of a sensing field</td>
</tr>
<tr>
<td>$f$</td>
<td>sensor node density</td>
</tr>
<tr>
<td>$T$</td>
<td>sampling interval</td>
</tr>
<tr>
<td>$P(i, j)$</td>
<td>the probability of detecting at least $j$ objects after $i$ grid cells are visited</td>
</tr>
<tr>
<td>$p(i, j)$</td>
<td>the probability of detecting an exact number of $j$ objects after $i$ grid cells are visited</td>
</tr>
</tbody>
</table>

Similarly, the probability of detecting at least $k$ objects after $i - 1$ grid cells are visited is

$$1 - \sum_{b=0}^{k-1} p(i - 1, b).$$

Let $P(i, k)$ be the probability that $i$ grid cells are visited in the preliminary search, i.e., exactly $i$ grid cells are visited in order to find at least $k$ objects. Then, $P(i, k)$ is given by

$$P(i, k) = (1 - \sum_{b=0}^{k-1} p(i, b)) - (1 - \sum_{b=0}^{k-1} p(i - 1, b)). \quad (4.2)$$

As described in Section 4.2.1, at the end of the preliminary search, an initial search space is derived as the set of grid cells whose minimum distances to $q$ are shorter than $d(o_b, q) + R_s$, where $R_s$ is the sensing range and $o_b$ is the boundary object. The number of grid cells in the initial search space gives an upper bound estimate for the total number of grid cells visited in the preliminary and expanded searches. In the following, we derive the average number of grid cells in the initial search space given that a number of $i$ grid cells are visited in the preliminary search.
Recall that in the preliminary search, the grid cells are visited in rounds by the circular approach. Given the number of rounds \(x\), we first derive the relationship between \(x\) and the total number of grid cells visited in rounds \(1, 2, ..., x\) of the circular approach. From Figure 4.2, it is intuitive that the number of grid cells visited is roughly proportional to the area of the circle with radius \(x \cdot \alpha\). Therefore, we use a quadratic model \(ax^2 + bx + c\) to approximate the number of grid cells visited in rounds \(1, 2, ..., x\), where \(a\), \(b\) and \(c\) are constants. We count the actual numbers of grid cells visited in rounds \(1, 2, ..., x\) for different \(x\)'s and then determine the values of \(a\), \(b\) and \(c\) by means of quadratic regression. That is, the values of \(a\), \(b\) and \(c\) are given by those generating the least square error between the empirical result (the actual number of grid cells visited) and the regression result (the number of grid cells computed by \(ax^2 + bx + c\)).

The regression result indicates \(a = 3.1417, b = 4.1178, c = 2.3241\). Figure 4.5 shows that the regression result matches the empirical result quite well. The difference between the regression result and the empirical result is within 3%.

We now estimate \(d(o_b, q) + R_s\), which is the radius of the initial search space. Let \(i\) be the number of grid cells visited in the preliminary search. Suppose that the last grid cell visited in the preliminary search is in round \(x_i\). Then, \(d(o_b, q)\) can be approximated by \(\alpha \cdot x_i\). To derive \(x_i\) from \(i\), we note that \(a(x_i - 1)^2 + b(x_i - 1) + c\) indicates the number of grid cells visited in the first \(x_i - 1\) rounds, and \(ax_i^2 + bx_i + c\) is the number of grid cells visited if round \(x_i\) completes. It follows that

\[
a(x_i - 1)^2 + b(x_i - 1) + c \leq i \leq ax_i^2 + bx_i + c.
\]

Therefore,

\[
\frac{-b + \sqrt{b^2 - 4a(c - i)}}{2a} \leq x_i \leq \frac{-b + \sqrt{b^2 - 4a(c - i)}}{2a} + 1.
\]

\(^2\)The number of grid cells derived from regression is larger than \(\pi x^2\) because the total area of cells overlapping with a circle is larger than the area of a circle.
So, we approximate \( x_i \) by
\[
x_i = \left[ -b + \sqrt{b^2 - 4a(c - i)} \right] / 2a.
\] (4.3)

As a result,
\[
d(o_b, q) + R_s = \alpha \cdot x_i + R_s = \alpha \cdot \left[ -b + \sqrt{b^2 - 4a(c - i)} \right] / 2a + R_s.
\]

Hence, the grid cells in the initial search space are all located within round \( \left\lfloor \frac{\alpha \cdot x_i + R_s}{\alpha} \right\rfloor \) of the circular approach. Therefore, the number of grid cells in the initial search space is bounded by
\[
a \cdot \left( \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} \right] \right)^2 + b \cdot \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} \right] + c.
\]

Averaging over different values of \( i \), the expected number of grid cells visited in the preliminary and expanded searches is derived as
\[
\sum_{i=1}^{M} P(i, k) \cdot \left( a \cdot \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} \right] \right)^2 + b \cdot \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} \right] + c,
\] (4.4)

where \( P(i, k) \) is the probability that \( i \) grid cells are visited in the preliminary search.
CHAPTER 4. \textit{kNN Query Processing in Wireless Sensor Networks}

Now we consider the message complexity of visiting one grid cell. There are three types of messages in the preliminary and expanded searches: the \textit{query} messages transmitted between the R-nodes, the \textit{probe} messages broadcast by the R-nodes, and the \textit{probe reply} messages sent by the sensor nodes detecting objects. The number of probe messages in query processing is equivalent to the number of grid cells visited. The complexity of query messages, on the other hand, depends on the hop count between the R-nodes of two successively visited grid cells. Since the maximum distance between two neighboring grid cells is about twice the transmission range, we approximate that an average of two hops are needed to transmit a query message between the R-nodes of two successively visited cells. Since the number of objects tracked is often much smaller than the number of sensor nodes in the sensing field [JZMK04], only a small portion of sensor nodes are expected to detect objects and reply to the R-nodes. Hence, we ignore the number of probe reply messages. Therefore, the message complexity of visiting one grid cell is approximated by 3. Hence, the total message complexity of processing a one-shot \textit{kNN} query with the localized scheme is given by three times the total number of grid cells visited in the preliminary and expanded searches\footnote{For simplicity, the message complexity is analyzed under the assumption of a dense sensor network that has no empty cell.}, i.e.,

\[ 3 \cdot \left( \sum_{i=1}^{M} P(i, k) \cdot (a \cdot \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} \right] + b \cdot \left[ \frac{\alpha \cdot x_i + R_s}{\alpha} + c \right] + c) \right). \] (4.5)

We shall experimentally compare the message complexities for the localized and flooding schemes in Section 4.5.2.

4.3 Processing Continuous \textit{kNN} Queries

The set of \textit{kNNs} and their locations may change over time as the objects move. In this section, we propose a localized scheme to continuously derive the \textit{kNN} result at the
query sink. The basic idea is to collect only the relevant data from the sensor nodes near the query point for $k$NN monitoring. A straightforward strategy for $k$NN monitoring is to reevaluate the query from scratch at each sampling interval using the one-shot $k$NN query processing algorithm described in Section 4.2. However, this may incur high message complexity. We propose to set up a monitoring area in the sensor network to facilitate continuous $k$NN query processing. The sensor nodes in the monitoring area proactively report to the query sink the location updates that may potentially affect the $k$NN result. In our approach, there are two stages in the processing of a continuous $k$NN query. In the first stage, the $k$NN query is initially evaluated at the first sampling interval using the scheme described in Section 4.2. The monitoring area is established along with the initial query evaluation as will be discussed in Section 4.3.1. In the second stage, the query sink continuously collects the location updates, reevaluates the query results, and maintains the monitoring area at each subsequent sampling interval. The query reevaluation will be discussed in Section 4.3.2 and the maintenance of monitoring area will be discussed in Section 4.3.3.

4.3.1 Monitoring Area Setup

A monitoring area is defined as the set of grid cells whose minimum distances to $q$ are shorter than $d(o_k, q) + R_s$, where $d(o_k, q)$ is the distance between the $k$th nearest object $o_k$ and $q$, and $R_s$ is the sensing range. The radius is set in this way to guarantee that all sensor nodes possibly detecting objects closer to $q$ than $o_k$ are included in the monitoring area. To collect all object locations closer to $q$ than $o_k$, the sensor nodes in the monitoring area may simply update all detected object locations with the query sink at each sampling interval. However, this may result in unnecessary location updates for objects that are further away from $q$ than $o_k$. To reduce location updates, we divide the grid cells in the monitoring area into two groups: all-report cells and partial-report cells. The sensor
nodes in the all-report cells update all detected object locations with the query sink; the sensor nodes in the partial-report cells keep a distance threshold and only update with the query sink the detected object locations within the distance threshold from \( q \). To guarantee that all object locations closer to \( q \) than \( o_k \) are reported, the distance thresholds at the sensor nodes in the partial-report cells should be set at no less than \( d(o_k, q) \). Lastly, the sensor nodes in the grid cells beyond the monitoring area do not update any detected object location with the query sink.

We now show how to set up the monitoring area in the initial query evaluation. Recall that during the preliminary and expanded searches, all grid cells whose minimum distances to \( q \) are shorter than \( d(o_k, q) + R_s \) are visited. When a grid cell is visited, its R-node broadcasts a probe message to all nodes in the cell. To establish the monitoring area, a new parameter is included in the probe message to indicate whether the grid cell is classified as an all-report cell or a partial-report cell. All grid cells visited during the preliminary search are classified as all-report cells. For each grid cell visited during the expanded search, if its minimum distance to the query point \( q \) is shorter than the distance from the boundary object recorded in the query message to \( q \), the grid cell is classified as an all-report cell. Otherwise, the grid cell is classified as a partial-report cell and the distance thresholds at the sensor nodes in the grid cell are set to the distance from the boundary object to \( q \). Note that the boundary object recorded in the query message at any time in the expanded search cannot be closer to \( q \) than the \( k \)th nearest object \( o_k \) in the final \( k \text{NN} \) result. Therefore, the distance thresholds set must be larger than \( d(o_k, q) \).

Figure 4.6 illustrates an example of monitoring area setup following Figure 4.4 in Section 4.2. The dark grey grid cells are visited in the preliminary search and the light grey grid cells \( G_1 \) to \( G_9 \) are visited in the expanded search. The dark grey grid cells are classified as all-report cells. When grid cells \( G_1 \) to \( G_4 \) are visited, \( o_z \) is the boundary
cells visited in the preliminary search

Cells visited in the expanded search

Figure 4.6: Setup of Monitoring Area

Object and $G_1$ to $G_4$ are classified as all-report cells since their minimum distances to $q$ are all shorter than $d(o_x, q)$. At cell $G_5$, a new boundary object $o_y$ replaces $o_x$ in the query message. When $G_6$ to $G_9$ are visited, the boundary object recorded in the query message is $o_y$. As the minimum distances from $G_5$ and $G_6$ to $q$ are shorter than $d(o_y, q)$, $G_5$ and $G_6$ are classified as all-report cells. On the other hand, $G_7$ to $G_9$ are classified as partial-report cells because their minimum distances to $q$ are longer than $d(o_y, q)$. As a result, the distance thresholds at the sensor nodes in $G_7$ to $G_9$ are set at $d(o_y, q)$.

4.3.2 Query Reevaluation

After the initial query evaluation at the first sampling interval, the monitoring area is set up accordingly. At subsequent sampling intervals, the query sink continuously collects
the location updates from the sensor nodes in the monitoring area and reevaluates the kNN result.

Consider a sampling interval $i$. Suppose the monitoring area at the beginning of interval $i$ includes the set of grid cells whose minimum distances to $q$ are shorter than $d(o_{k}^{i-1}, q) + R_s$, where $o_{k}^{i-1}$ is the $k$th nearest object to $q$ at interval $i - 1$. At interval $i$, denote by $k'$ the number of object locations reported by the sensor nodes in the monitoring area. If $k' < k$ (case A), the query sink needs to search for $k - k'$ more objects. Similar to the scheme described in Section 4.2, the search involves two phases: the preliminary search and the expanded search. The only difference is that the all-report cells are exempted from the preliminary search. This is because the sensor nodes in these grid cells have already reported all detected object locations to the query sink. Note that the grid cells whose minimum distances to $q$ are shorter than $d(o_{k}^{i-1}, q)$ must have been classified as all-report cells. Thus, the query message, containing the $k'$ object locations collected, is sent from the query sink to the first grid cell in round $\lceil \frac{d(o_{k}^{i-1}, q)}{\alpha} \rceil$ of the circular visiting order to start the preliminary search.

If $k' \geq k$, a list of $k$ object locations closest to $q$ are selected from the $k'$ object locations collected. Let $o_b$ be the $k$th nearest object to $q$ among these $k$ objects. If $d(o_0, q) \leq d(o_{k}^{i-1}, q)$ (case B), the $k$ objects selected are the new kNN result since all object locations nearer to $q$ than $o_{k}^{i-1}$ are included in the $k'$ object locations collected. Neither the preliminary search nor the expanded search is needed in this case. Otherwise, if $d(o_0, q) > d(o_{k}^{i-1}, q)$ (case C), the expanded search is carried out to refine the new kNN result. $o_b$ is set as the initial boundary object for the expanded search, and the search list includes the grid cells whose minimum distances to $q$ are shorter than $d(o_b, q) + R_s$ and longer than $d(o_{k}^{i-1}, q)$ (note that all grid cells whose minimum distances to $q$ are shorter than $d(o_{k}^{i-1}, q)$ are all-report cells).
4.3.3 Maintenance of Monitoring Area

The monitoring area, initially set up at the first sampling interval, may need to be updated later due to the change in the kNN result upon query reevaluation. Let \( o_i^k \) and \( o_i^{k-1} \) be the \( k \)th nearest objects to \( q \) at intervals \( i \) and \( i - 1 \) respectively. Then, the monitoring area at the beginning of interval \( i \) includes the grid cells whose minimum distances to \( q \) are shorter than \( d(o_i^k, q) + R_s \). If \( d(o_i^k, q) > d(o_i^{k-1}, q) \), the monitoring area should be expanded at interval \( i \) to include all grid cells whose minimum distances to \( q \) are shorter than \( d(o_i^k, q) + R_s \). Thus, a set of new grid cells should be added to the monitoring area. The sensor nodes in these grid cells need to be notified to update detected object locations with the query sink starting from the next sampling interval. Note that \( o_i^k \) is further away from \( q \) than \( o_i^{k-1} \) only in cases \( A \) and \( C \) discussed in Section 4.3.2. In both cases, the preliminary search and/or expanded search are needed to reevaluate the query. The new grid cells to be added to the monitoring area would be visited in these searches.

Similar to the initial query evaluation, the sensor nodes in these cells are notified along with the searches.

On the other hand, if \( d(o_i^k, q) \leq d(o_i^{k-1}, q) \), the monitoring area can be shrunk to reduce the number of sensor nodes updating object locations with the query sink. As shown in Figure 4.7, the old monitoring area includes the grid cells whose minimum distances to \( q \) are shorter than \( r_{old} = d(o_i^{k-1}, q) + R_s \). The new monitoring area includes the set of grid cells whose minimum distances to \( q \) are shorter than \( r_{new} = d(o_i^k, q) + R_s \). We divide the grid cells in the old monitoring area into three categories:

(i) The grid cells whose minimum distances to \( q \) lie in \([r_{new}, r_{old}]\) (shown by the dark grey grid cells in Figure 4.7). The sensor nodes in these grid cells need to be informed to stop updating detected object locations with the query sink;
Figure 4.7: Shrinking of the Monitoring Area

(ii) The grid cells whose minimum distances to $q$ lie in $[d(o_k^l, q), r_{new}]$ (shown by the light grey grid cells in Figure 4.7). These grid cells would be classified as partial-report cells in the new monitoring area. The sensor nodes in these grid cells need to be informed about the new distance threshold $d(o_k^l, q)$;

(iii) The grid cells whose minimum distances to $q$ are shorter than $d(o_k^l, q)$ (shown by the white grid cells in Figure 4.7). These grid cells are all-report cells in the old monitoring area and would remain all-report cells in the new monitoring area.

To inform the sensor nodes in the grid cells of the first two categories, a notification message is sent to them through geocast [NI97]. To describe the target area of geocast, the notification message contains the location of the query point $q$, the new distance threshold $d(o_k^l, q)$, and the radii of the old and new monitoring areas $r_{old}$ and $r_{new}$. In the geocast, the notification message is first sent to a sensor node in the target area by unicast. Starting from this sensor node, the message is flooded to all sensor nodes.
in the target area. For each node in a grid cell whose minimum distance to \( q \) falls in \([r_{new}, r_{old}]\), it stops reporting location updates to the query sink upon receiving the notification message. For each node in a grid cell whose minimum distance to \( q \) falls in \([d(o_k, q), r_{new}]\), it records the new distance threshold upon receiving the notification message.

**Figure 4.8: Example of Geocast \((m = 4)\)**

Traditional geocast does not guarantee that all sensor nodes in the target area would receive the notification message. This is because some nodes in the target area may only be reachable via the nodes outside the target area [Sto04]. To increase the delivery rate, in the unicast step of geocast, the notification message can be sent to a number of \( m \) nodes in the target area instead of one node only [Sto04]. The flooding then starts from these nodes concurrently. As shown in Figure 4.8, the target area in our case has a ring shape. We propose to equally divide the ring into \( m \) sub-areas. A notification message is sent to the center of each sub-area in the unicast step of geocast. In the flooding step, each sensor node receiving the notification message broadcasts the message to its neighbors, unless it has received the notification message before or it is in a grid cell.
whose minimum distance to $q$ is longer than $r_{old}$ or shorter than $d(o_k^i, q)$. We shall study the impact of $m$ through simulation experiments in Section 4.5.3.

There is in fact a tradeoff between the overhead of shrinking the monitoring area and the saving in location updates sent by the sensor nodes in the monitoring area. If the monitoring area is shrunk whenever the $k$th nearest object moves nearer to $q$ (called the AggressiveShrink strategy), unnecessary location updates are aggressively eliminated. However, geocasting the notification message to shrink the monitoring area incurs communication overhead. Moreover, if the $k$th nearest object moves away from $q$ later, the monitoring area would have to be expanded again. As a result, the overhead of updating the monitoring area may exceed the saving in location updates, thereby increasing the total message complexity. On the other hand, if the monitoring area is never shrunk (called the NoShrink strategy), a large number of unnecessary location updates may be sent to the query sink leading to high total message complexity. In the following section, we first analyze an offline optimal schedule to shrink the monitoring area. Then, we propose an adaptive strategy that dynamically decides when to shrink the monitoring area on the fly based on the shrinking overhead relative to the saving in location updates.

Algorithm 4.2 highlights the algorithm executed at the query sink including the initial query evaluation at the first time interval (steps 1-3), and query reevaluations at subsequent sampling intervals (steps 4-28).

4.4 Scheduling Strategy to Shrink the Monitoring Area

4.4.1 Optimal Schedule to Shrink the Monitoring Area

Given all object locations and their detecting sensor nodes at each sampling interval, we would like to compute the optimal schedule to shrink the monitoring area, i.e., to find a
Algorithm 4.2 Algorithm executed at the query sink

1: At the first sampling interval, conduct the initial query evaluation by sending out a query message $p$ to the first grid cell $G$ according to the circular approach;
2: Wait for the result message $p'$;
3: Extract from $p'$ the $k$ nearest object locations to the query point $q$ and let $o_k^1$ be the $k$th nearest object location;
4: for each subsequent sampling interval $i$ before the query expires do
5: Record in $U$ the location updates received from the sensor nodes in the monitoring area;
6: if $|U| < k$ then
7: Record $U$ in a query message $p$;
8: Select the grid cell $G$ to visit for the preliminary search;
9: Send out message $p$ to $G$ to start the preliminary search;
10: Wait for the result message $p'$;
11: Extract from $p'$ the $k$ nearest object locations to the query point $q$ and let $o_k^1$ be the $k$th nearest object location;
12: else
13: Derive from $U$ the $k$ nearest object locations to $q$ (called $U'$) and let $o_b$ be the $k$th nearest object location in $U'$;
14: if $d(o_b, q) > d(o_k^{i-1}, q)$ then
15: Initialize the search list for the expanded search;
16: Record $U'$ and the search list in a query message $p$;
17: Select cell $G$ from the search list which is nearest to the query sink;
18: Send out message $p$ to $G$ to start the expanded search;
19: Wait for the result message $p'$;
20: Extract from $p'$ the $k$ nearest object locations to the query point $q$ and let $o_k^i$ be the $k$th nearest object location;
21: else if $d(o_b, q) \leq d(o_k^{i-1}, q)$ then
22: The $k$ nearest object locations to $q$ are those in $U'$ and let $o_k^i = o_b$;
23: end if
24: end if
25: if the maintenance strategy determines to shrink the monitoring area then
26: Send out the shrink message to a relevant ring-shaped area;
27: end if
28: end for

set of intervals such that the total message complexity is minimized if the monitoring area is shrunk at these intervals. The total message complexity includes those of shrinking the monitoring area, query reevaluation, and location updates at each sampling interval.

We start by providing some basic definitions. Suppose that at an interval $i$, the
monitoring area consists of all grid cells whose minimum distances to \( q \) are shorter than 
\[
d(o^i_k, q) + R_s
\]
where \( o^i_k \) is the \( k \)th nearest object at interval \( i \). We consider the message complexity at a sampling interval \( v > i \) assuming that the monitoring area does not shrink at intervals \( i + 1, i + 2, \ldots, v - 1 \). At the beginning of interval \( v \), the monitoring area would include all grid cells whose minimum distances to \( q \) are shorter than 
\[
\max_{i \leq x < v} d(o^x_k, q) + R_s
\]
where \( o^x_k \) is the \( k \)th nearest object at interval \( x \). We define \( c_r(i, v) \) as the message complexity for the sensor nodes in the monitoring area to report location updates to the query sink at interval \( v \). Let \( o^v_k \) be the \( k \)th nearest object at interval \( v \). If 
\[
d(o^v_k, q) > \max_{i \leq x < v} d(o^x_k, q)
\]
the query reevaluation at the query sink involves the preliminary search and/or expanded search. As a result, the monitoring area must be expanded. Otherwise, if 
\[
d(o^v_k, q) \leq \max_{i \leq x < v} d(o^x_k, q)
\]
the query reevaluation does not involve any message transmission, and it is possible to shrink the monitoring area at interval \( v \). In this case, we say that interval \( v \) is shrinkable with respect to interval \( i \). Given \( i \), we shall denote all intervals \( v > i \) that are shrinkable with respect to interval \( i \) by a set \( S(i) \). Without loss of generality, for any \( v > i \), we define \( c_q(i, v) \) as the message complexity for query reevaluation at interval \( v \); for any \( v \in S(i) \), we define \( c_s(i, v) \) as the message complexity for shrinking the monitoring area at interval \( v \). The complexities \( c_r(i, v), c_q(i, v), c_s(i, v) \) can be derived from the object locations and their detecting sensor nodes at the sampling intervals.

We now formulate the optimal shrinking schedule problem. Assume that at a sampling interval \( I \), the monitoring area consists of all grid cells whose minimum distances to \( q \) are shorter than 
\[
d(o^I_k, q) + R_s
\]
We consider a period of sampling intervals \( I + 1, I + 2, \ldots, H \). Suppose the monitoring area is shrunk at intervals \( x_1, x_2, \ldots, x_m \), where \( I < x_1 < x_2 < \ldots < x_m < H \), \( x_i \in S(x_{i-1}) \) for each \( 1 < i \leq m \), and \( x_1 \in S(I) \). Then, the total message
complexity over intervals $I + 1, I + 2, \ldots, H$ is given by

$$
\text{cost}(I, H : x_1, x_2, \ldots, x_m) = c_s(I, x_1) + \sum_{v=I+1}^{x_1} \left( c_r(I, v) + c_q(I, v) \right) \\
+ \sum_{i=1}^{m-1} \left( c_s(x_i, x_{i+1}) + \sum_{v=x_{i+1}}^{x_{i+1}} \left( c_r(x_i, v) + c_q(x_i, v) \right) \right) \\
+ \sum_{v=x_m+1}^{H} \left( c_r(x_m, v) + c_q(x_m, v) \right).
$$

(4.6)

Given $c_r(i, v)$ and $c_q(i, v)$ for all $I \leq i < v \leq H$, and $c_s(i, v)$ for any $i$ and $v$ where $v \in S(i)$, the offline optimal shrinking schedule problem is to find a set of intervals $I < x_1 < x_2 < \ldots < x_m < H$ that minimize (4.6). For convenience, we shall call it the $(I, H)$-optimization problem. We show that the problem can be solved by a dynamic programming algorithm since the optimal schedule to the $(I, H)$-optimization problem contains optimal solutions to some subproblems. Let $(x_1, x_2, \ldots, x_m)$ be an optimal shrinking schedule to the $(I, H)$-optimization problem. Then, $(x_2, x_3, \ldots, x_m)$ must be an optimal shrinking schedule to the $(x_1, H)$-optimization problem. This is because if there exists another shrinking schedule $(y_1, y_2, \ldots, y_l)$ (where $x_1 < y_1 < y_2 < \ldots < y_l < H$) that results in a lower message complexity than $(x_2, x_3, \ldots, x_m)$, i.e.,

$$
\text{cost}(x_1, H : y_1, y_2, \ldots, y_l) < \text{cost}(x_1, H : x_2, x_3, \ldots, x_m),
$$

(4.7)

it follows that

$$
\text{cost}(I, H : x_1, y_1, y_2, \ldots, y_l) \\
= c_s(I, x_1) + \sum_{v=I+1}^{x_1} \left( c_r(I, v) + c_q(I, v) \right) + \text{cost}(x_1, H : y_1, y_2, \ldots, y_l) \\
< c_s(I, x_1) + \sum_{v=I+1}^{x_1} \left( c_r(I, v) + c_q(I, v) \right) + \text{cost}(x_1, H : x_2, x_3, \ldots, x_m) \\
= \text{cost}(I, H : x_1, x_2, \ldots, x_m),
$$

which contradicts the optimality of $(x_1, x_2, \ldots, x_m)$.
CHAPTER 4. \textit{kNN QUERY PROCESSING IN WIRELESS SENSOR NETWORKS}

For any \( I \leq j < H \), let \( C[j] \) be the minimum achievable message complexity in the \((j, H)\)-optimization problem, and \( B[j] \) be the first shrinking interval in the optimal schedule to the \((j, H)\)-optimization problem. Note that if the monitoring area does not shrink at any interval, the message complexity over intervals \( j+1, j+2, \ldots, H \) is \( \sum_{v=j+1}^{H} (c_r(j,v) + c_q(j,v)) \). Hence, the recurrences for dynamic programming are given by

\[
C[j] = \begin{cases}
  c_r(H-1,H) + c_q(H-1,H) & \text{if } j = H - 1, \\
  \min \left( \min_{j < x < H, x \in S(j)} (c_r(j,x) + \sum_{v=j+1}^{x} (c_r(j,v) + c_q(j,v))) + C[x] \right) & \text{if } j < H - 1,
\end{cases}
\]

and

\[
B[j] = \begin{cases}
  \emptyset & \text{if } j = H - 1, \\
  \arg \min_{j < x < H, x \in S(j)} (c_r(j,x) + \sum_{v=j+1}^{x} (c_r(j,v) + c_q(j,v))) + C[x] & \text{if } j < H - 1 \text{ and } C[j] = \min_{j < x < H, x \in S(j)} \left( c_r(j,x) + \sum_{v=j+1}^{x} (c_r(j,v) + c_q(j,v)) \right) + C[x] ,
\end{cases}
\]

(4.8)

(4.9)

Starting from \( C[H-1] \) and \( B[H-1] \), we can compute all \( C[j] \)'s and \( B[j] \)'s in decreasing order of \( j \). On obtaining all \( C[j] \)'s and \( B[j] \)'s, the optimal shrinking intervals can be derived by tracing back the \( B \)-entries. Starting from \( x_1 = B[I] \), we can obtain the shrinking intervals in the optimal schedule by setting \( x_{v+1} = B[x_v] \) iteratively until \( B[x_v] = \emptyset \).

Now we analyze the time complexity of the dynamic programming algorithm. Let \( P = H-I \) be the length of the period under consideration. Given any \( w \), the computation complexity of \( \sum_{u=w+1}^{u} (c_r(u,v) + c_q(u,v)) \) for all different \( u \)'s is \( O(P) \). Hence, \( \sum_{u=w+1}^{u} (c_r(u,v) + c_q(u,v)) \) for all pairs of \( u \) and \( w \) can be computed in a pre-processing stage in \( O(P^2) \) time. Then, the time complexity to compute \( C[j] \) is given by \( O(P) \). Thus,
the time complexity to compute all \(C[j]\)'s is given by \(O(P^2)\). Therefore, the total time complexity of the dynamic programming algorithm is \(O(P^2)\). The computed optimal schedule shall be referred to as the \textit{OptimalShrink} strategy.

4.4.2 Adaptive Schedule to Shrink the Monitoring Area

The \textit{OptimalShrink} strategy provides the minimal total message complexity over a designated period. However, it is computed in an offline manner where object locations at all sampling intervals are assumed known a priori. In practice, the object locations are not known beforehand. Thus, in this section, we propose an adaptive strategy (called \textit{AdaptiveShrink}) to dynamically decide when to shrink the monitoring area on the fly. The general idea is to compare the saving in location updates with the overhead of shrinking the monitoring area.

Consider a sampling interval \(i\). Suppose prior to interval \(i\), the monitoring area was last updated at interval \(j < i\) (see Figure 4.9). It could be either shrunk by geocast or be expanded due to query reevaluation. Then, the monitoring area at the beginning of interval \(i\) includes the grid cells whose minimum distances to \(q\) are shorter than \(r_{old} = d(o_i^j, q) + R_a\). The monitoring area can be shrunk at interval \(i\) only if \(d(o_i^j, q) < d(o_i^j, q)\). If the monitoring area is shrunk at interval \(i\), the new monitoring area would include
the grid cells whose minimum distances to \( q \) are shorter than \( r_{\text{new}} = d(o_k^i, q) + R_s \). We notice that, if the monitoring area is shrunk at interval \( i \), it saves not only the location updates at the next sampling interval \( i+1 \), but also the location updates at subsequent sampling intervals as long as the sizes of the monitoring areas at these sampling intervals are less than \( r_{\text{old}} \). Let \( h > i \) be the first interval at which the size of the monitoring area first exceeds \( r_{\text{old}} \) since interval \( i \) (i.e., \( d(o_k^h, q) \geq d(o_k^i, q) \)). Then, the shrinking of the monitoring area at interval \( i \) would save the location updates at intervals \( i+1, i+2, \ldots, h \).

Denote the total saving of location updates by \( c_{\text{saving}} \). On the other hand, the overhead of shrinking the monitoring area not only includes the cost of shrinking the monitoring area at interval \( i \) (denoted as \( c_{\text{shrink}} \)), but also includes the cost of expanding the monitoring area from size \( r_{\text{new}} \) to size \( r_{\text{old}} \) due to query reevaluation at later sampling intervals. Let \( g > i \) be the first interval at which the size of the monitoring area exceeds \( r_{\text{new}} \) (i.e., \( d(o_k^g, q) > d(o_k^i, q) \)). It follows that \( h \geq g > i \). Then, the cost of expanding the monitoring area from size \( r_{\text{new}} \) to \( r_{\text{old}} \) is given by the total query reevaluation cost at intervals \( g, g+1, g+2, \ldots, h \) (denote by \( c_{\text{query}} \)). In our adaptive strategy, the new monitoring area is kept unchanged at interval \( i \) if \( c_{\text{saving}} \leq c_{\text{shrink}} + c_{\text{query}} \). Otherwise, if \( c_{\text{saving}} > c_{\text{shrink}} + c_{\text{query}} \), the monitoring area is shrunken. The problem remains to predict \( c_{\text{saving}}, c_{\text{shrink}} \) and \( c_{\text{query}} \) at interval \( i \).

We start by estimating intervals \( g \) and \( h \). Note that \( g > i \) is the first interval since \( i \) such that \( d(o_k^g, q) > d(o_k^i, q) \). For clarity of presentation, we shall replace \( g \) by \( g(i) \). We estimate \( g(i) \) by computing \( g(i) - i \) based on the historical durations of \( g(v) - v \) for each interval \( v \) prior to \( i \). That is, \( g(i) - i = \frac{1}{i-1} \sum_{v=1}^{i-1} (g(v) - v) \), i.e., \( g(i) = i + \frac{1}{i-1} \sum_{v=1}^{i-1} (g(v) - v) \). To do so, the query sink keeps the history of the \( k \)th nearest object locations. For each \( v < i \), \( g(v) \) can be computed straightforwardly if \( \max_{v \leq u < i} d(o_k^u, q) > d(o_k^v, q) \). Otherwise, if \( \max_{v \leq u < i} d(o_k^u, q) \leq d(o_k^v, q) \), \( g(v) \) is simply set to \( i \). To compute \( h \), we estimate \( h - g \) by \( i - j \). Thus, given \( g \) and \( j \), \( h \) is computed as \( i - j + g \).
After estimating the intervals $g$ and $h$, we first compute $c_{\text{saving}}$, the saving in location updates. Let $\mathcal{N}$ be the number of objects whose distances to the query point $q$ are between $r_{\text{old}} = d(o_i^k, q) + R_s$ and $r_{\text{new}} = d(o_i^k, q) + R_s$ at interval $i$. $\mathcal{N}$ can be derived at the query sink. For each sampling interval from $i+1$ to $g$, we estimate that $\mathcal{N}$ objects would be exempted from location updates if the monitoring area is shrunk at interval $i$. For the sampling intervals from $g+1$ to $h$, the monitoring area is gradually expanded from size $r_{\text{new}}$ to $r_{\text{old}}$. We approximate that on average $\frac{1}{2} \mathcal{N}$ objects are exempted from location updates at each interval from $g+1$ to $h$. The message complexity to send one location update of an object to the query sink is approximated by $\lceil \frac{1}{2} (r_{\text{old}} + r_{\text{new}}) / R_t \rceil$. Thus, the total saving in location updates from interval $i+1$ to $h$ is estimated by

$$c_{\text{saving}} = (g - i + \frac{1}{2} \cdot (h - g)) \cdot \mathcal{N} \cdot \lceil \frac{1}{2} (r_{\text{old}} + r_{\text{new}}) / R_t \rceil.$$

Next, we compute $c_{\text{shrink}}$, the cost of shrinking the monitoring area at interval $i$. Following the two steps of geocast, the cost of shrinking the monitoring area includes that of sending the unicast messages to the target area and that of flooding the messages within the target area. For simplicity, we ignore the cost of sending the unicast messages in the estimation of $c_{\text{shrink}}$ since it is usually much smaller than the cost of flooding. The message complexity of flooding is approximated by the number of sensor nodes in the target area. The target area includes all grid cells whose minimum distances to $q$ lie between $d(o_i^k, q)$ and $r_{\text{old}} = d(o_i^k, q) + R_s$.

According to the derivation in Section 4.2.3, the number of grid cells visited in rounds $1, 2, \ldots, x$ of the circular approach is given by $a \cdot x^2 + b \cdot x + c$, where $a = 3.1417, b = 4.1178, c = 2.3241$. In the circular approach, a grid cell visited in round $x$ has a minimum distance to $q$ within $\alpha \cdot x$. Thus, we approximate the number of grid cells in the target area by that between rounds $\lceil \frac{d(o_i^k, q)}{\alpha} \rceil$ and $\lceil \frac{d(o_i^k, q) + R_s}{\alpha} \rceil$ of the circular approach, i.e.,

$$\left( a \cdot \left( \lceil \frac{d(o_i^k, q) + R_s}{\alpha} \rceil \right) \right)^2 + b \cdot \left( \lceil \frac{d(o_i^k, q) + R_s}{\alpha} \rceil \right) + c - \left( a \cdot \left( \lceil \frac{d(o_i^k, q)}{\alpha} \rceil \right) \right)^2 + b \cdot \left( \lceil \frac{d(o_i^k, q)}{\alpha} \rceil \right) + c.$$
CHAPTER 4. \( k \)NN QUERY PROCESSING IN WIRELESS SENSOR NETWORKS

Suppose \( f \) is the mean sensor node density. Then, on average, each grid cell would contain \( f \cdot \alpha^2 \) sensor nodes. Therefore, the expected cost of shrinking the monitoring area at interval \( i \) is estimated by

\[
c_{\text{shrink}} = f \cdot \alpha^2 \cdot \left( (a \cdot \left( \left\lceil \frac{d(o_k^i, q) + R_s}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q) + R_s}{\alpha} + c \right)
- (a \cdot \left( \left\lceil \frac{d(o_k^i, q)}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q)}{\alpha} + c).
\]

Finally, we compute \( c_{\text{query}} \), the total message complexity of query reevaluation from interval \( g \) to \( h \). We first estimate the number of grid cells visited during query reevaluation. Then, by estimating the message complexity of visiting a grid cell, we can derive the message complexity of query reevaluation. The grid cells visited in query reevaluation include those whose minimum distances to \( q \) lie between \( d(o_k^i, q) \) and \( r_{\text{old}} = d(o_k^i) + R_s \). As discussed above, the number of these grid cells is approximated by that between rounds \( \left\lceil \frac{d(o_k^i, q)}{\alpha} \right\rceil \) and \( \left\lceil \frac{d(o_k^i, q) + R_s}{\alpha} \right\rceil \) of the circular approach, i.e.,

\[
(a \cdot \left( \left\lceil \frac{d(o_k^i, q) + R_s}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q) + R_s}{\alpha} + c) - (a \cdot \left( \left\lceil \frac{d(o_k^i, q)}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q)}{\alpha} + c).
\]

Following the discussion in Section 4.2.3, the message complexity of visiting one grid cell is approximated by \( 3 \). Hence, the total message complexity of query reevaluation is estimated by

\[
c_{\text{query}} = 3 \cdot \left( (a \cdot \left( \left\lceil \frac{d(o_k^i, q) + R_s}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q) + R_s}{\alpha} + c \right)
- (a \cdot \left( \left\lceil \frac{d(o_k^i, q)}{\alpha} \right\rceil \right) + \frac{d(o_k^i, q)}{\alpha} + c).
\]

4.5 Performance Evaluation

4.5.1 Experimental Setup

We conducted a wide range of experiments to evaluate the performance of the localized scheme for one-shot and continuous \( k \)NN query processing. We used a simulator called
Table 4.2: System Parameters and Settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of sensor nodes</td>
<td>2500</td>
</tr>
<tr>
<td>( R_t )</td>
<td>Transmission range</td>
<td>12.5m</td>
</tr>
<tr>
<td>( R_s )</td>
<td>Sensing range</td>
<td>10m</td>
</tr>
<tr>
<td>f</td>
<td>Sensor node density</td>
<td>1 node/50m²</td>
</tr>
<tr>
<td>s ( \times ) s</td>
<td>Size of sensing field</td>
<td>360m ( \times ) 360m</td>
</tr>
<tr>
<td>( \alpha \times \alpha )</td>
<td>Size of a grid cell</td>
<td>9m ( \times ) 9m</td>
</tr>
<tr>
<td>n</td>
<td>Number of objects tracked</td>
<td>[50, 100, 200, 400, 600]</td>
</tr>
<tr>
<td>T</td>
<td>Sampling interval</td>
<td>30s</td>
</tr>
<tr>
<td>( V_{max} )</td>
<td>Maximum object moving velocity</td>
<td>5m/s</td>
</tr>
<tr>
<td>k</td>
<td>Number of NNs requested by a query</td>
<td>[1, 2, 4, 6, 8, 10]</td>
</tr>
</tbody>
</table>

J-Sim [SHK+06]. Table 4.2 summarizes the system parameters and their settings. We simulated a sensor network covering a 360m \( \times \) 360m sensing field. A total of 2500 sensor nodes were randomly deployed in the field, implying that on average, there was one sensor node in an area of 50m². Similar to other studies [HVY+07], the default transmission range and the sensing range for each sensor node were set at 12.5m and 10m respectively. Each sensor node had an average of 9 neighbors. A given number of \( n \) objects were randomly distributed and tracked in the sensing field. The object locations were sampled by the sensor nodes at every 30 seconds. In our experiments, we assumed that the detecting sensor node of an object is the one closest to the object [ZC04].

We simulated both one-shot and continuous \( k \)NN query processing in object tracking sensor networks. Each query was made via a randomly selected sensor node. The query point of the query was assumed to be the location of the sensor node. For one-shot \( k \)NN query processing, we implemented the proposed localized scheme and the simple flooding scheme. In the localized scheme, when visiting a grid cell, we used the contention-based scheduling scheme [XLXM06] to avoid collisions between the probe reply messages from the sensor nodes detecting objects. For each experimental setting, we randomly generated 2000 sets of object locations. The average performance of these 2000 simulation runs
is plotted for performance comparison. The 95% confidence interval of the experimental results was calculated to be within 2 percent of the mean. For continuous \( k \text{NN} \) query processing, we studied the proposed localized monitoring scheme in depth by simulating five different strategies: No-Monitoring-Area, OptimalShrink, AdaptiveShrink, NoShrink and AggressiveShrink. The query is initially evaluated in the first sampling interval and reevaluated at each subsequent sampling interval. In the No-Monitoring-Area strategy, no monitoring area is established in the network. Hence, no location update is sent to the query sink. The object locations are all stored locally at the detecting sensor nodes. At each sampling interval, the query is reevaluated from scratch using the two-phase search mechanism described in Section 4.2. On the other hand, in the OptimalShrink, AdaptiveShrink, NoShrink and AggressiveShrink strategies, a monitoring area is established in the network. These four strategies use the same methods of query evaluation, monitoring area setup and query reevaluation as described in Section 4.3. Their difference lies in the maintenance of the monitoring area. The OptimalShrink strategy derives the optimal shrinking schedule in an offline manner as described in Section 4.4.1. The AdaptiveShrink strategy dynamically determines when to shrink the monitoring area on the fly as described in Section 4.4.2. The NoShrink strategy only expands the monitoring area at query reevaluation and never shrinks the monitoring area. The AggressiveShrink strategy, on the other hand, shrinks the monitoring area whenever the \( k \)th nearest object computed in the current sampling interval is closer to the query point than the \( k \)th nearest object in the previous sampling interval. For each experimental setting in continuous \( k \text{NN} \) query processing, we randomly generated 5 sets of object locations and their movements. Each simulation run of the query was performed for 2000 sampling intervals. The averaged performance of these 5 simulation runs is plotted for comparison. The 95% confidence interval of the experimental results was calculated to be within 3 percent of the mean.
Table 4.3: Message Types and Contents

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>query-related</td>
<td>query</td>
<td>query point ( q + k ) + search list + (0 to ( k )) object locations</td>
</tr>
<tr>
<td>query result</td>
<td></td>
<td>( k ) object locations</td>
</tr>
<tr>
<td>probe</td>
<td></td>
<td>centroid of a grid cell + ( q ) + distance threshold</td>
</tr>
<tr>
<td>probe reply</td>
<td></td>
<td>object location</td>
</tr>
<tr>
<td>shrink</td>
<td></td>
<td>parameters of the target area + ( q )</td>
</tr>
<tr>
<td>location-update</td>
<td>location update</td>
<td>object location(s)</td>
</tr>
</tbody>
</table>

The performance is evaluated by two metrics: energy consumption and message complexity. Specifically, energy consumption is the average amount of energy consumed by all sensor nodes in the simulation. We focus on the energy consumption of message exchanges.\(^4\) The energy for sending and receiving one byte was set at \( 24.9 \times 10^{-6} J \) and \( 18.72 \times 10^{-6} J \) respectively [PHC04]. Message complexity refers to the total number of messages transmitted in the network. There are six types of messages in the proposed localized scheme for \( k \)NN monitoring. As shown in Table 4.3, these messages can be divided into two categories: query-related messages and location-update messages. In our experiments, we assumed that each integer data value takes up 4 bytes in the message and each floating-point value takes up 8 bytes.

### 4.5.2 Performance Study for One-Shot \( k \)NN Query Processing

In this section, we investigate the performance of one-shot \( k \)NN query processing. Figure 4.10(a) shows the average energy consumption over all sensor nodes with different \( k \)'s for the localized scheme and the flooding scheme. The number of objects tracked was set at 50. It is seen that the energy consumption for the flooding scheme is much higher

---

\(^4\)Since we focus on query processing, we do not include the energy consumed in tracking objects. As is the practice in other studies [XLXM06, RKS\(^4\)03], we do not measure the routing protocol load placed by GPSR since GPSR generates a constant volume of routing protocol traffic (beacon messages) that is independent of the query processing scheme and is usually of lower order than the application data traffic.
Figure 4.10: Impact of $k$ for One-Shot $k$NN Query Processing ($R_t = 12.5m$)

than that of the localized scheme. This is because in the flooding scheme, the query is flooded to all sensor nodes and all detected object locations are sent to the query sink. In contrast, the localized scheme explores the localized property of a $k$NN query and routes the query only to the sensor nodes in the proximity of the query point so that
only the relevant object locations are collected. In general, the energy consumption of the localized scheme increases with \( k \) since more grid cells have to be visited to find a larger number of objects. Figure 4.10(b) shows the network-wide message complexities for the localized and flooding schemes. For the localized scheme, we plot both the experimental results from the simulation and the analytical results derived from Equation (4.5). As can be seen, the flooding scheme incurs much higher message complexity than the localized scheme. The experimental results for the localized scheme match the analytical results quite well.

Figure 4.11(a) shows the average energy consumption over all sensor nodes for different numbers of objects tracked. The number of nearest objects requested was set at 8. It is observed that the average energy consumption of the localized scheme decreases with increasing number of objects tracked. This is because when there are more objects, the \( k \)th nearest object is normally closer to the query point. As a result, the number of grid cells visited during the preliminary search and the expanded search becomes smaller. On the other hand, since all object locations are sent to the query sink, the average energy consumption for the flooding scheme increases with the number of objects tracked. Figure 4.11(b) shows the network-wide message complexities for the localized and flooding schemes. The experimental results for the localized scheme match the analytical results well. It is also seen that the flooding scheme incurs many more message transmissions than the localized scheme.

We have also conducted experiments with a larger transmission range of 25m — about 2.5 times of the sensing range [XTL05, GM04]. The results, as shown in Figures 4.12 and 4.13, reflect similar performance trends.
Chapter 4. kNN Query Processing in Wireless Sensor Networks

Figure 4.11: Impact of Number of Objects for One-Shot kNN Query Processing ($R_c = 12.5m$)

4.5.3 Performance Study for Continuous kNN Query Processing

In this section, we investigate the processing of continuous kNN queries and compare the performance of the five strategies: OptimalShrink, AdaptiveShrink, NoShrink, Ag-
Figure 4.12: Impact of $k$ for One-Shot $k$NN Query Processing ($R_t = 25m$)

gressiveShrink and No-Monitoring-Area. We first investigate the impact of $m$ on the
delivery rate of geocast, where the delivery rate is defined as the number of sensor nodes
successfully receiving the message over the total number of nodes in the target area of
geocast. We tested a large number of target areas that are rings centered at a query
Figure 4.13: Impact of Number of Objects for One-Shot $k$NN Query Processing ($R_t = 25m$)

point with a width of $R_s$ since $R_s$ is the narrowest possible width of target areas in our proposed localized monitoring scheme (representing the worst-case delivery rate). Figure 4.14 shows the average delivery rate as a function of $m$. As can be seen, the delivery rate generally increases with $m$. It becomes rather steady when $m$ increases beyond 4.
Hence, the default value of \( m \) was set at 4 in our experiments.

![Figure 4.14: Delivery Rate v.s. Number of Unicast Messages in Geocast (m)](image)

To study continuous \( k \)NN query processing, the objects were initially placed at random in the sensing field. We simulated two different object mobility models: random waypoint and random walk. In the random waypoint model (abbreviated as RWP), each object repeatedly picks a random destination in the sensing field and moves to the destination at a speed randomly chosen from a range \((0, V_{\text{max}}]\). After reaching a destination, the object immediately chooses the next destination and moves towards it. In the random walk model (abbreviated as RW), each object periodically changes its moving speed by randomly choosing the velocity from a range \((0, V_{\text{max}}]\) and the moving direction from a range \([0, 2\pi]\). The changing period in random walk model was set at 30s. In our experiments, the default number of objects was set at 200 and the default \( k \) was set at 8. For the OptimalShrink strategy, we first conducted a set of simulation experiments to compute the costs \( c_r(i, v) \), \( c_q(i, v) \) and \( c_d(i, v) \) for all pairs of \( i \) and \( v \). These costs were then used to derive the optimal shrinking schedule as described in Section 4.4.1.

We first simulated the processing of a single continuous \( k \)NN query. In this set of experiments, the number of objects was set at 200 and \( k \) was set at 8. The maximum
moving speed for the objects was set at 5m/s. The transmission range was set at 12.5m.

Figure 4.15(a) shows the average energy consumption of the five strategies. It is seen that: (1) the energy consumption in the No-Monitoring-Area strategy is much higher than those strategies establishing a monitoring area in the network; (2) the NoShrink
and AggressiveShrink strategies consume much more energy than the AdaptiveShrink strategy; (3) the average energy consumption of the AdaptiveShrink strategy is close to the OptimalShrink strategy.

To better understand these observations, we present the message breakdown for the five strategies in Figure 4.15(b). We observe that the No-Monitoring-Area strategy has the highest number of query-related messages and involves no location-update message. This is because in the No-Monitoring-Area strategy, no monitoring area is established to monitor the location updates. Hence, the query has to be reevaluated from scratch at each interval. For the other four strategies, a monitoring area is set up to monitor the location updates that contribute to query reevaluation. Thus, the number of query-related messages is greatly reduced.

Among the four strategies establishing monitoring areas, the AggressiveShrink strategy has the lowest number of location-update messages but the highest number of query-related messages. This is because the monitoring area in AggressiveShrink is always shrunk to a minimum circle covering the $k$ nearest objects and their detecting sensor nodes. In this way, the location updates are kept at minimum. However, aggressive shrinking of monitoring area may lead to frequent query reevaluation when the objects move away from the query point. Thus, the AggressiveShrink strategy has the highest number of query-related messages among the four strategies. In contrast, the NoShrink strategy has the lowest number of query-related messages. This is because the monitoring area is never shrunk in NoShrink. Thus, the possibility of carrying out preliminary and expanded searches in query reevaluation is low. However, the NoShrink strategy leads to a large number of unnecessary location updates. Hence, it has the highest number of location-update messages among the four strategies. The AdaptiveShrink strategy makes a good balance between query-related and location-update messages by considering the tradeoff between the shrinking overhead and the saving in location updates. It
shrinks the monitoring area only when it is beneficial. As a result, the AdaptiveShrink strategy achieves lower total message complexity than AggressiveShrink and NoShrink. Figure 4.15 shows that its performance is close to the offline OptimalShrink strategy.

![Figure 4.15: Average Energy Consumption](image)

![Figure 4.16: Message Breakdown](image)

Figure 4.16: Performance of Localized Scheme (Multiple Queries, $R_t = 12.5m$)

We also simulated the scenario where 20 kNN queries were monitored concurrently.
Figure 4.16 shows the performance results of the four strategies AdaptiveShrink, NoShrink, AggressiveShrink and No-Monitoring-Area. It is seen that the performance trends of the four strategies are similar to those of monitoring a single $k$NN query in the network (Figure 4.15).

![Graph](image)

(a): Average Energy Consumption

![Graph](image)

(b): Breakdown of Message Complexity

Figure 4.17: Performance Comparison for Continuous $k$NN Query Processing (Single Query, $R_t = 25m$)
We have also conducted experiments with a larger transmission range of 25m. The results, as shown in Figures 4.17 and 4.18, reflect similar performance trends. In the following experiments of the localized scheme, only the results using the default transmission range (i.e., 12.5m) are reported.
Figure 4.19: Impact of $k$: (Random WayPoint Model)

Figures 4.19 and 4.20 show the average energy consumption and the message breakdown for different numbers of nearest objects monitored. In this set of experiments, the number of objects was set at 200 and $V_{\text{max}}$ was set at $5m/s$. A total number of 20 $k$NN queries were concurrently monitored in the network. It is seen from Figures 4.19(a) and
4.20(a) that the average energy consumption increases with \( k \) for all four strategies. This is because when the requested number of nearest objects increases, more sensor nodes are visited in query evaluation and reevaluation. Thus, as shown in Figures 4.19(b) and 4.20(b), the number of query-related messages increases with \( k \). Moreover, for the
Figure 4.21: Impact of Number of Objects (Random WayPoint Model)

AdaptiveShrink, NoShrink and AggressiveShrink strategies, a larger monitoring area is established when more nearest objects are requested. As a result, more location updates are sent to the query-sink at each sampling interval. Figures 4.19(b) and 4.20(b) show that the number of location-update messages in these four strategies also increases
with $k$. As $k$ increases, the energy consumption of NoShrink grows rapidly due to large number of location updates. Similarly, with increasing $k$, the energy consumption of AggressiveShrink rises quickly due to high cost of query reevaluation. The AdaptiveShrink strategy balances the costs in query reevaluation and location updates. Hence, its energy
consumption increases mildly when \( k \) becomes larger.

Figures 4.21 and 4.22 show the average energy consumption and the message breakdown for different numbers of objects tracked in the network. In this set of experiments, \( V_{\text{max}} \) was set at 5\( m/s \) and \( k \) was set at 8. Again, a total number of 20 \( kNN \) queries were concurrently monitored in the network. It is observed that the average energy consumption decreases with increasing number of objects in the network. This is because when there are more objects in the network, the \( k \)th nearest object becomes closer to the query point. Hence, the message complexity of query reevaluation decreases (see Figures 4.21(b) and 4.22(b)). The size of the monitoring area in the AdaptiveShrink, NoShrink and AggressiveShrink strategies also reduces with increasing number of objects tracked. The relative performance of the four strategies remains similar over a wide range of object numbers. Under both mobility models, the AdaptiveShrink strategy outperforms the AggressiveShrink and NoShrink strategies.

### 4.6 Summary

In this chapter, we have proposed a localized scheme for continuous \( kNN \) query processing in object tracking sensor networks. In the localized scheme, the object locations are stored locally at the detecting sensor nodes. A monitoring area is set up when the \( kNN \) query is initially evaluated. Only the location updates from sensor nodes in the monitoring area are collected to reevaluate the query. The monitoring area is expanded and shrunk on the fly upon object movement. Experimental results show that the cost of query reevaluation is greatly reduced by establishing the monitoring area in the localized scheme. We have further studied the maintenance of the monitoring area. We analyze the optimal maintenance and develop an adaptive algorithm to dynamically decide when to shrink the monitoring area. Experimental results show that the AdaptiveShrink strategy
achieves close-to-optimal performance and significantly outperforms the naive NoShrink and AggressiveShrink strategies in terms of energy consumption and message complexity.
Chapter 5

Conclusion

The rapid development of wireless technologies has made wireless computing an increasingly important research topic. In this thesis, we have investigated two important issues of data dissemination and data gathering in wireless computing environments. In this chapter, we summarize our research and contributions as well as discuss some future research directions.

In the first part of the thesis, we have investigated push-based data broadcast for wireless data dissemination. In wireless data broadcast, mobile clients can operate in two modes: active mode and doze mode. They can only retrieve data from broadcast channels in the active mode which has much higher rate of energy consumption than that in the doze mode. Therefore, to save energy, it is desirable for mobile clients to switch to the doze mode as much as possible when waiting for the required data, i.e., to cut down the tuning time. The tuning time can be reduced by means of air indexing that interleaves index information with data in the broadcast schedule to assist the clients in locating the required data. Following the links in the index structure, the clients alternate between the active and doze modes until the data arrives. Most existing air indexing schemes are based on the tree structures that are originally designed for random-access media such as disks. To apply tree-based index on wireless channels which are sequential-access media, data must be broadcast in the order of key and with the same frequency. This sacrifices
responsiveness (i.e., access latency) when client accesses are not uniformly distributed among data items. In the presence of non-uniform access distribution, popular data items should be broadcast more frequently than unpopular items to reduce average access latency. However, most existing non-flat broadcast scheduling schemes do not consider air indexing. Without index, the clients have to continuously stay active and monitor the broadcast channel until the required data arrive. This consumes significant amount of battery power and sacrifices energy efficiency. Different from the existing studies, in our work, we have aimed at optimizing energy efficiency and responsiveness in an integrated fashion.

In the second part of the thesis, we have studied the processing of \textit{kNN} queries in object tracking sensor networks that request \textit{k} nearest object locations to a given geographical point. In wireless sensor networks, energy is mainly consumed in exchanging messages between the sensor nodes and the base station. To reduce energy consumption, it is desirable to reduce the amount of data transmitted in the network. Due to the geographically distributed deployment of sensor nodes, spatial information plays an important role in the representation of sensor data. Thus, it is natural to collect data from the sensor network by specifying spatial conditions in the queries. Existing work on spatial query processing in sensor networks has focused on one-shot queries that request the readings from the sensor nodes based on the node locations. In addition to the locations of sensor nodes, the data captured by the sensor nodes may also include spatial information. For example, in object tracking sensor networks, the object locations detected by the sensor nodes are represented by geographical coordinates. Different from the existing studies, in our work, we have investigated collecting a given number of \textit{k} detected object locations nearest to a specified geographical point in an energy-efficient manner. Both one-shot and continuous queries have been investigated.
5.1 Contributions

The major contributions of the thesis are summarized as follows:

- We have proposed a new indexing scheme called MHash to enhance energy efficiency and responsiveness in an integrated fashion for push-based data broadcast. MHash reduces tuning time by mapping data items to the slots in the broadcast schedule via a two-argument hash function. We have shown that a hole-free hash function for broadcast scheduling purpose can be constructed by injecting an offset into an arbitrary hash function. Meanwhile, the two-argument nature of the hash function allows each data item to be mapped to an adjustable number of slots in the schedule, thereby enabling non-flat data broadcast. Popular items can be broadcast more frequently than the unpopular ones in order to reduce access latency. Furthermore, we have derived an optimal bandwidth allocation for MHash indexing. Experimental results show that under non-uniform access distribution, MHash outperforms state-of-the-art air indexing schemes in energy efficiency and achieves access latency close to optimal broadcast scheduling.

- We have proposed a localized scheme for processing $k$NN queries in object tracking sensor networks. A two-phase search mechanism has been designed to evaluate one-shot $k$NN queries. To support continuous monitoring of a $k$NN query, a monitoring area is established along with an initial evaluation of the query using the two-phase search mechanism. We have developed methods to reevaluate the $k$NN query based on the location updates collected from the sensor nodes in the monitoring area. Experimental results show that the energy cost of query reevaluation is greatly reduced by establishing the monitoring area. Due to object movement, the monitoring area needs to be expanded and shrunk on the fly. Hence, we have further studied the maintenance of the monitoring area. We have analyzed the
optimal maintenance and developed an adaptive strategy to dynamically decide when to shrink the monitoring area. Experimental results show that the adaptive strategy achieves close-to-optimal performance and significantly outperforms two naive strategies in terms of energy consumption.

5.2 Future Research Directions

In a wireless environment, energy efficient data dissemination and data gathering are two major tasks that require much research. Many future directions can be explored. For example,

- Real-time data dissemination with time constraints. With the development of time-critical information services and business-oriented applications, clients are likely to issue requests that have time constraints. Hence, there is an increasing demand to support real-time data dissemination based on the client requests [XTL06]. Building index structure in such a broadcast system to achieve energy efficiency is a non-trivial research problem.

- Privacy-preservation in sensor networks. As sensor network applications expand to include increasingly sensitive measurements of daily life (e.g., patients’ information collected by a health-care sensor network), preserving data privacy while gathering high quality data has become an important issue that would lead to the social acceptance of sensor network applications. To improve energy efficiency, sensor data are often processed within the sensor network, e.g., through in-network aggregation. As a result, the data would be manipulated by multiple untrustful sensor nodes before they are made available to the applications. Since sensor networks are usually deployed in an unattended manner, each individual sensor node is prone to attacks and is not trustworthy. Hence, it is challenging to design privacy-preserving schemes to achieve in-network data processing with network-wide trustworthy [HLN+07].
Sensor networks as a Web. Till now, the research on sensor networks has mainly focused on routing, data aggregation and energy efficiency within a single sensor network. It is expected that a large number of sensor networks will be deployed for various application purposes in the future. In addition to operating independently, these sensor networks can be connected via the well established Internet infrastructure to form a sensor web. A sensor web supports a variety of services that query, as a single unit, large quantities of data from numerous widely distributed sensor nodes [BDF+07]. The large amount of data provided by multiple sensor networks poses new challenges to data management. For example, where should the sensor data be stored? Where and when should the data be aggregated considering the granularity specified by the user? How to perform query planning and optimization? Furthermore, building indexing structures to support efficient sensor data retrieval from multiple sensor networks is also an important research problem.
References


REFERENCES


REFERENCES


134
REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


Appendix A

List of Publications

During the thesis research, I have published the following articles in referred journals and conference proceedings:

Articles in Referred Journals


Articles in Referred Conference Proceedings
