

Electronic Theses and Dissertations, 2004-2019

2010

Query Processing In Location-based Services

Fuyu Liu
University of Central Florida

 Part of the [Electrical and Electronics Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Liu, Fuyu, "Query Processing In Location-based Services" (2010). *Electronic Theses and Dissertations, 2004-2019*. 1635.
<https://stars.library.ucf.edu/etd/1635>

QUERY PROCESSING IN LOCATION-BASED SERVICES

by

FUYU LIU

M.S. Georgia State University, 2003

M.S. University of Chicago, 2000

B.S. University of Science and Technology of China, 1999

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term

2010

Major Professor: Kien A. Hua

© 2010 Fuyu Liu

ABSTRACT

With the advances in wireless communication technology and advanced positioning systems, a variety of Location-Based Services (LBS) become available to the public. Mobile users can issue location-based queries to probe their surrounding environments. One important type of query in LBS is moving monitoring queries over mobile objects. Due to the high frequency in location updates and the expensive cost of continuous query processing, server computation capacity and wireless communication bandwidth are the two limiting factors for large-scale deployment of moving object database systems. To address both of the scalability factors, distributed computing has been considered. These schemes enable moving objects to participate as a peer in query processing to substantially reduce the demand on server computation, and wireless communications associated with location updates.

In the first part of this dissertation, we propose a distributed framework to process moving monitoring queries over moving objects in a spatial network environment. In the second part of this dissertation, in order to reduce the communication cost, we leverage both on-demand data access and periodic broadcast to design a new hybrid distributed solution for moving monitoring queries in an open space environment.

Location-based services make our daily life more convenient. However, to receive the services, one has to reveal his/her location and query information when issuing location-based queries. This could lead to privacy breach if these personal information are possessed by some untrusted parties.

In the third part of this dissertation, we introduce a new privacy protection measure called query l -diversity, and provide two cloaking algorithms to achieve both location k -anonymity and query l -diversity to better protect user privacy. In the fourth part of this dissertation, we design a hybrid three-tier architecture to help reduce privacy exposure. In the fifth part of this dissertation, we propose to use Road Network Embedding technique to process privacy protected queries.

ACKNOWLEDGMENTS

This work would not have been possible without the guidance of my advisor, Dr. Kien A. Hua. Throughout the years during my graduate study at UCF, Dr. Hua was very generous with his time and wisdom. Dr. Hua gave me numerous invaluable suggestions and spent a lot of precious time in discussing the dissertation with me and reviewing the dissertation. Special thanks to Dr. Hua. Furthermore, Dr. Hua set up a good role model for me in many aspects of life, which include but not just limited to doing research. His impact on me will be lifelong and I'm very grateful for that.

I would like to thank Dr. Charles E. Hughes, Dr. Cliff C. Zou, and Dr. Morgan C. Wang for their insightful suggestions during the work of this dissertation and for being part of my committee.

I must also acknowledge all of my fellow students at the Data Systems Group. Special thanks to Tai T. Do, Alex J. Aved, Danzhou Liu, Fei Xie, Hao Cheng, Ning Jiang, Yao Hua Ho, Ai Hua Ho, Ning Yu, and Rui Peng for their helpful discussions. In addition, I want to thank other fellow students who have helped me and made my time at UCF more enjoyable and delightful.

Finally, I must thank my family for their love and support behind me. Without their love and support, I would not have gone this far to complete this dissertation.

TABLE OF CONTENTS

LIST OF FIGURES	xii
LIST OF TABLES	xiii
LIST OF ACRONYMS	xiv
CHAPTER 1: INTRODUCTION	1
CHAPTER 2: MOVING QUERY MONITORING IN SPATIAL NETWORK ENVIRONMENTS	5
2.1 Introduction	5
2.2 Related Work	7
2.3 System Model	10
2.3.1 System Assumptions	10
2.3.2 Definitions	10
2.4 Proposed Solution for Moving Range Query	13
2.4.1 Definitions	13
2.4.2 Server Data Structure	15
2.4.3 Moving Object Data Structure	17
2.4.4 Initialization	17
2.4.5 Server Side Message Processing	18
2.4.5.1 Switch Segment Message	18
2.4.5.2 Enter/Exit Query Message	19
2.4.5.3 Other Messages	19
2.4.6 Moving Object Side Message Processing	19
2.4.7 An Example	21

2.5	Proposed Solution for Moving kNN Query	21
2.5.1	Definitions	22
2.5.2	Server Data Structure	23
2.5.3	Moving Object Data Structure	24
2.5.4	Initialization	25
2.5.5	Server Side Message Processing	26
2.5.5.1	Switch Segment Message	26
2.5.5.2	Enter Query Message	27
2.5.5.3	Exit Query Message	27
2.5.5.4	Other Messages	28
2.5.6	Moving Object Side Message Processing	28
2.5.7	An Example	28
2.6	Performance Study	29
2.6.1	Simulation Setup	30
2.6.2	Server Workload	31
2.6.3	Communication Cost	32
2.6.3.1	Analytical Study on Communication Cost	36
2.6.4	Client Workload	44
2.7	Extension to Cover Mixed Environments	46
2.8	Conclusion	48

CHAPTER 3: A HYBRID COMMUNICATION SOLUTION TO DISTRIBUTED MOVING QUERY MONITORING SYSTEMS 50

3.1	Introduction	50
3.2	Related Work	51
3.2.1	On Broadcast Channel	52
3.2.2	On Monitoring Queries over Moving Objects	52

3.3	Preliminary Model and Motivation	53
3.4	Broadcast Index Design	56
3.4.1	Grid Index	57
3.4.2	Direction Index	58
3.5	Data Structure and Query Processing	62
3.5.1	Data Structures	62
3.5.2	Query Installation	63
3.5.3	Changing Query Result Activity	63
3.5.4	Changing Velocity Activity	64
3.5.5	Changing Grid Cell Activity	64
3.5.6	Communication between Server and Objects	65
3.6	Performance Study	65
3.6.1	Simulation Setup	66
3.6.2	Selecting Cell Size	67
3.6.3	Communication Cost	67
3.6.3.1	Analytical Study on Communication Cost	72
3.6.4	Comparison of Proposed Indexing Schemes	77
3.7	Discussions	82
3.7.1	Communication Cost Revisited	82
3.7.2	Query Retrieval Latency	83
3.8	Conclusion	85

CHAPTER 4: PROTECTING USER PRIVACY BETTER WITH QUERY

	L-DIVERSITY	87
4.1	Introduction	87
4.2	Related Work	89
4.2.1	k -anonymity and l -diversity in relational databases	89

4.2.2	k -anonymity and l -diversity in location-based services	90
4.3	Preliminaries	91
4.3.1	System Architecture	91
4.3.2	Privacy-preserving Properties	92
4.3.3	Anonymization Goal	95
4.4	Cloaking Algorithms	96
4.4.1	Data Structure	96
4.4.2	Expand Cloak	97
4.4.3	Hilbert Cloak	101
4.4.4	Discussions	102
4.5	Performance Study	103
4.5.1	Performance Metrics	103
4.5.2	Experimental Setup	104
4.5.3	Experimental Results	105
4.5.3.1	Varying number of cells	105
4.5.3.2	Varying k and l	106
4.5.3.3	Scalability Study	110
4.5.3.4	Discussions	112
4.6	Conclusion	112
CHAPTER 5: USING BROADCAST TO PROTECT USER PRIVACY		113
5.1	Introduction	113
5.2	Proposed Solution	114
5.2.1	System Architecture	114
5.2.2	Anonymization Goals	114
5.2.3	Clustering Algorithms	115
5.2.4	Broadcast Channel	117

5.2.5	Hybrid Solution	118
5.3	Performance Study	119
5.3.1	Performance Metrics	119
5.3.2	Experimental Setup	119
5.3.3	Experimental Results	120
5.3.3.1	Varying k in k -anonymity	121
5.3.3.2	Varying number of objects	122
5.4	Conclusion	123
 CHAPTER 6: PRIVACY PROTECTED QUERY PROCESSING WITH ROAD NETWORK EMBEDDING		 124
6.1	Introduction	124
6.2	Road Network Embedding	125
6.3	Proposed Solution	126
6.3.1	Private kNN Query over Public Objects	127
6.3.2	Extensions	129
6.3.2.1	Private Range Query over Public Objects	129
6.3.2.2	Private Query over Private Objects	129
6.4	Performance Study	129
6.4.1	Performance Metrics	131
6.4.2	Experimental Setup	131
6.4.3	Experimental Results	132
6.5	Conclusion	135
 CHAPTER 7: CONCLUSION		 136
 REFERENCES		 145

LIST OF FIGURES

2.1	Range Query Examples	6
2.2	An example of calculating monitoring region	16
2.3	Data structure for DRQ processing	17
2.4	An example of calculating network distance region	22
2.5	Monitoring region for kNN query	24
2.6	Example for message processing	29
2.7	Effect of number of queries on server workload	32
2.8	Effect of number of objects on communication cost	33
2.9	Effect of maximum speed on communication cost	34
2.10	Effect of maximum segment length on communication cost	34
2.11	Effect of percentage of objects changing speed on communication cost	35
2.12	Effect of number of interested nearest neighbor on communication cost	35
2.13	Effect of maximum range query radius on communication cost	36
2.14	Monitoring regions for $n = 1$ and $n = 2$	39
2.15	Observations when $n = 2$	40
2.16	Monitoring region for $n = 3$	40
2.17	Effect of number of objects on communication cost based on analytical studies	45
2.18	Effect of number of queries on power consumption	46
2.19	Effect of number of queries on the average number of queries monitored	46
2.20	Mixed Environments	48
3.1	Examples	55
3.2	Layout for the Interleaving Technique	57
3.3	Grid Index structure	57
3.4	Query data bucket structure	57

3.5	Examples for query types	60
3.6	Scenarios when object changing cell	61
3.7	Direction Index structure	61
3.8	Effect of cell size on number of messages	68
3.9	Effect of cell size on number of queries monitored	68
3.10	Effect of cell size on the product of number of messages and number of queries monitored vs. α	68
3.11	Effect of number of objects on communication cost	70
3.12	Effect of number of queries on communication cost	70
3.13	Effect of percentage of objects changing velocity on communication cost	71
3.14	Effect of maximum velocity on communication cost	71
3.15	Effect of maximum query radius on communication cost	71
3.16	Effect of number of objects on communication cost based on analytical study	78
3.17	Effect of number of queries on communication cost based on analytical study	78
3.18	Effect of number of objects on access time	79
3.19	Effect of number of objects on tuning time	79
3.20	Effect of number of queries on access time	79
3.21	Effect of number of queries on tuning time	80
3.22	Effect of packet size on access time	80
3.23	Effect of packet size on tuning time	80
3.24	Effect of total number of channels on Normalized Communication Cost	83
3.25	Effect of number of queries on one broadcast cycle length based on analytical study	85
4.1	System Architecture	92
4.2	Illustrations for Expand Cloak and Hilbert Cloak	100
4.3	Varying number of cells	107

4.4	Varying k in k -anonymity	109
4.5	Varying l in l -diversity	110
4.6	Scalability Study by varying number of queries and query categories	111
5.1	System Architecture	115
5.2	Broadcast Index	118
5.3	Varying k in k -anonymity	120
5.4	Varying number of objects	121
6.1	Private query over Public Objects	128
6.2	Private query over Private Objects	130
6.3	Effect on Query Processing Time when varying k in k NN	133
6.4	Effect on Query Result Accuracy when varying k in k NN	133
6.5	Effect on Query Processing Time when varying size of MBR	133
6.6	Effect on Query Result Accuracy when varying size of MBR	134
6.7	Effect on Query Processing Time when varying percentage of dimensions used	134
6.8	Effect on Query Result Accuracy when varying percentage of dimensions used	134

LIST OF TABLES

2.1	Simulation Parameters	30
3.1	Simulation Parameters	66
4.1	Data in Cells	100
4.2	Simulation Parameters	105
5.1	Simulation Parameters	119
6.1	Simulation Parameters	132

LIST OF ACRONYMS

<i>LBS</i>	Location-Based Services
<i>P2P</i>	Peer-to-Peer
<i>GPS</i>	Global Positioning System
<i>DRQ</i>	Dynamic Range Query
<i>kNN</i>	k-Nearest Neighbor
<i>QOT</i>	Query Object Table
<i>QST</i>	Query Segment Table
<i>CQT</i>	Client Query Table
<i>SQT</i>	Server Query Table
<i>QBO</i>	Query Blind Optimal
<i>GI</i>	Grid Index
<i>DI</i>	Direction Index
<i>NCC</i>	Normalized Communication Cost
<i>QAT</i>	Query Anonymization Time
<i>QSR</i>	Query Success Rate
<i>RAA</i>	Relative Anonymization Area
<i>RAL</i>	Relative Anonymity Level
<i>RNE</i>	Road Network Embedding
<i>QET</i>	Query Execution Time
<i>NCM</i>	Number of Communication Messages
<i>MBR</i>	Minimal Bounding Rectangle

CHAPTER 1: INTRODUCTION

Mobile devices integrated with GPS have become ubiquitous, resulting in a multitude of services which allow users to query the environment around them. Since the services are typically based on locations, they are also known as Location-Based Services (LBS). Some examples are route planning, emergency road assistance, buddy search, and tourist information services, to name just a few. The location-based services also present good opportunity for mobile commerce. A mobile user could ask a service provider to suggest a restaurant or a garage. Businesses could also benefit from knowing the locations of mobile users by direct advertising. The spatial monitoring of moving objects is fundamental to these LBS applications. In particular, if the spatial query is anchored around a moving object, this query is referred to as a moving query. One example could be “Show me the addresses and brands of the 3 nearest gas stations while I’m driving for the next half an hour”. Since the query issuer is moving while the points of interest are static, this is a moving monitoring query over static objects. In a more complex environment where many objects are moving, we can also have moving monitoring queries over moving objects. For instance, while walking in a city, a tourist can issue a query like “Find the available taxis with company names within one mile of my current location in the next 20 minutes.” Since both the query issuer (the tourist) and the point of interest (taxi) are moving, this is a moving monitoring query over moving objects. Some other examples could be: “Find the friendly units that are within 2 miles from me”, issued by a soldier marching in a battle field; “Find two of my buddies within 1 mile from me” for someone who wants to find his/her buddies while driving, in which the buddies could be either moving or static.

A typical moving object database system consists of one or more servers and a large number of moving objects [50], [49], [74], [67]. Moving objects and the server communicate through base stations. To monitor moving queries over moving objects, there are three main challenges. First, the query results must be updated constantly until the query is explicitly

terminated by the user. Second, location updates are very expensive due to the constant movements of moving objects. Third, the region of interest (i.e., the query region) itself is also changing steadily, which adds significantly more complexity. For such a system to be successful, there are mainly three goals: reduce the server workload, reduce the communication cost, and maintain the query result quality.

In the first part of the dissertation, we propose a distributed framework to process moving queries over moving objects in a spatial network environment. Road segments are used as the building blocks for the framework. For each road segment, the server identifies a set of queries that need to be monitored for objects moving on that road segment. Moving objects also need to monitor their positions on the road segments and update their locations when they move into a new road segment. Upon receiving a location update, the server informs the moving object a new set of queries overlapping its current road segment. The moving object then monitors these relevant queries and updates the affected query results when it moves in or out of the corresponding queries. We also propose a novel concept called *Edge Distance* to facilitate the distance calculation on mobile clients. We discuss how to answer two typical spatial queries: range query and kNN query. The effectiveness of the technique is illustrated by simulation and analytical studies.

In the second part of the dissertation, in order to reduce the communication cost, we leverage a hybrid of on-demand data access and periodic broadcast to design a new distributed solution for moving monitoring queries on moving objects in an open space environment. The area of interest is first divided into grid cells. When a mobile object moves into a new cell, the object can tune into a broadcast channel to obtain a new set of queries that the object needs to monitor while moving within the new cell. To improve the tuning efficiency and access time, we propose two indexing schemes: Grid Index and Direction Index, for the broadcast technique. We also describe how to use the index to answer queries. Extensive simulation studies show the reduction in communication cost, which helps reduce energy consumption on mobile devices.

As of today, to get location-based query answered, one has to reveal his/her current location and the type of service to a service provider when launching location-based queries. If an adversary has access to the service provider's query logs, the adversary can easily collect location and requested service history of users and use the collected information to predict the user's next movement, which severely invades user privacy. To help protect user privacy, many techniques have been proposed [23], [18], [48], [12], [72], [70]. Most of these techniques assume a three-tier architecture, which consists of mobile clients, trusted anonymizer server, and service providers. The trusted anonymizer server collects users' locations and performs cloaking procedures using the k -anonymity concept [60] to blur the query issuer's location into a region, with the region defined by the bounding rectangle covering locations of other $(k-1)$ users. The query with the blurred region is then sent to the service provider. The service provider processes the query based on the cloaked area and returns the results to the anonymizer server. Finally, the anonymizer server filters out the unwanted query results and sends the refined query results to the query issuer.

In the third part of the dissertation, we propose to use both location k -anonymity and query l -diversity to better protect user privacy. A new property called $\langle k, l \rangle$ -sharing region is identified as the guide to design new cloaking algorithms. We use this property to design the Expand Cloak and Hilbert Cloak techniques to achieve both location k -anonymity and query l -diversity. To assess their performance, we also design an improved version of the original Interval Cloak technique [23] to handle query l -diversity. With simulation studies, we show that both techniques are significantly better than the improved Interval Cloak technique in providing user privacy protection.

In the fourth part of the dissertation, a hybrid three-tier architecture is proposed, where the trusted anonymizer server also serves as a broadcast server. The trusted anonymizer server takes a proactive approach. It first groups mobile users into clusters. Then for each cluster, it fetches the most popular query results from service providers. The query results are then broadcast through an air channel to reach all mobile users. As a result, to get a

query answered, a mobile user can first tune into the air channel to determine if the query result is available. If not, the mobile user just sends a traditional location-based query to the trusted anonymizer server. Other than the novel new three-tier architecture, we also propose two cell-based clustering algorithms, and a broadcast index to facilitate the download of query results. The proposed techniques are compared using simulation against the improved Interval Cloak technique under the traditional three-tier architecture. The extensive results show that our system is better in reducing communication cost and protecting user privacy.

In the fifth part of the dissertation, we propose to use Road Network Embedding (RNE) to answer cloaked queries in a road network environment. We first give an algorithm to answer k -nearest neighbor queries, then extend the algorithm to answer range queries and queries over private objects. Extensive simulation studies are performed to show the effectiveness of the proposed technique.

The rest of this dissertation is organized as follows. The distributed framework is presented in Chapter 2. The hybrid communication solution is discussed in Chapter 3. The Expand Cloak and Hilbert Cloak techniques are described in Chapter 4. Chapter 5 discusses the hybrid three-tier architecture. Chapter 6 presents how to use the RNE technique to answer privacy enabled queries. Finally, Chapter 7 concludes the dissertation.

CHAPTER 2: MOVING QUERY MONITORING IN SPATIAL NETWORK ENVIRONMENTS

2.1 Introduction

Advances in wireless network and positioning technology have enabled query processing over moving objects. One important query type is monitoring query over moving objects which, unlike traditional queries, requires real-time processing and has a relatively long lifetime. Moreover, object mobility entails frequent location updates during query execution. Many practical applications can benefit from an efficient processing of moving queries. As an example, a truck carrying sensitive material from location A to location B may want to continuously monitor the surrounding traffic. As another example, while one is driving in a city and wants to find all her/his nearby buddies, s/he might want to issue a query like “Find me all my buddies within 5 miles”. One commercial location-based application: Loopt [2], does provide this type of buddy-find service. This new type of query, demanding constant updates from moving objects to keep the query results accurate, raises a great challenge.

A typical mobile data management system has one or more central servers and a large number of moving objects. Two scalability issues for such a system are server side computation cost and wireless communication cost. Most existing solutions take a centralized approach where moving objects need to report their locations to a central server periodically [50], [4], [37], [49], [55], [61], [64], [74], [67]. The focus of these solutions is to efficiently compute query results without worrying about location updates. One interesting idea for reducing location updates is to use safe regions [54], [26], or thresholds [51], where an object moving within a safe region or a threshold does not need to update its location. More recently, some distributed techniques have been proposed [8], [9], [10], [17], [19], [63], [62]. In this distributed approach, moving objects utilize their own computing power to help process

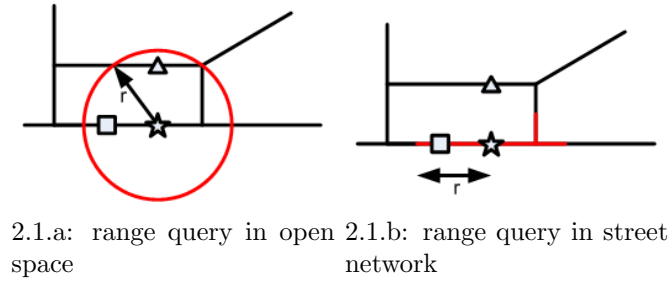


Figure 2.1: Range Query Examples

queries in order to reduce server load and avoid frequent location updates. We also notice that there have been some work, leveraging local ad-hoc network [14], [40].

Most distributed solutions mentioned above assume an open space environment, where the distance between two objects is the straight line distance between them. In real-life scenarios, many moving objects (e.g., cars) are restricted to move on a network (e.g., road network). Since the distance between two objects in a network is defined as the shortest network distance between them, techniques developed specifically for an open space environment cannot be easily extended to a road network. As illustrated in Fig. 2.1.a, in an open space environment, when an object (represented by the star) issues a range query with radius r , both the square object and the triangle object are included as the query result. However, if the same query is issued in a spatial network environment, as shown in Fig. 2.1.b, then only the square object belongs to the query result.

In this chapter, we propose a framework to process moving queries over moving objects in a spatial network environment. The road network is partitioned into road segments, and moving objects need only update their location when they move into a new segment. Upon a location update, the server informs the moving object the queries overlapping its current road segment. The moving object then monitors these relevant queries, and contacts the server to update the affected query results as it moves in or out of the corresponding queries. To enable the moving object to monitor its distance to other objects, we introduce a novel concept called *Edge Distance*. By storing *Edge Distance* locally, a moving object

can compute network distance efficiently without possessing a full network map. Under the proposed framework, we develop mechanisms to answer two typical spatial queries: range query and kNN query. The effectiveness of the proposed method is illustrated with detailed simulation study. Moreover, we present an analytical study to estimate the communication cost.

The remainder of this chapter is organized as follows. Related work is discussed in Section 2.2. Section 2.3 gives system overview and some formal definitions. In Section 2.4, we discuss how the system answers range queries, and in Section 2.5, we cover kNN queries. Detailed simulation studies are presented in Section 2.6. In Section 2.7, we extend the proposed model to answer queries in a mixed environment. Finally, Section 2.8 concludes the chapter.

2.2 Related Work

Quite a few works have been devoted on query processing in road networks. Papadias et al. propose two methods called Euclidean Restriction (ER) and Network Expansion (NE) in [53] to compute static range and nearest neighbor queries in network environment. ER uses the euclidean distance to prune the space, while NE starts from the query point, expands along the network until all results are found. A Voronoi diagram based algorithm [36] is introduced by Kolahdouzan and Shahabi to compute k nearest neighbors. In [25], Hu et al. first simplifies the road network by substituting the graph topology by some tree-based structures called SPIE's, then built index for each SPIE to facilitate answering kNN queries. Hu et al. [24] also propose a new index, called distance signature, which divided the distances between network nodes and objects into certain categories, and encoded these categories. Then they proposed a way to use the constructed distance signature to answer kNN queries. The ER and the NE methods are considered as a type of "online" approach, which does not rely on pre-computation. The second type of approach, usually considered as "pre-computation" type, such as the Voronoi diagram based algorithm, typically can

improve query performance but suffers when there are a lot of updates. Huang et al. [28] design a method called "island approach", which can control the amount of pre-computation and achieve a trade-off balance between query performance and update cost. Continuously kNN (CkNN) monitoring problems have also been studied in literature. Kolahdouzan and Shahabi [35] extend the Voronoi diagram based algorithm to answer the CkNN problem. Cho et al. [11] solve the same problem by observing that the union of the set of query results while the query point is moving along the path is equal to the union of the set of all data points on the path and the set of query results for all intersection nodes on the query path. A system demonstrated by Huang et al. [27] can answer k nearest neighbor queries effectively.

However, all the above methods are not directly applicable to continuously monitoring query on moving objects because they either consider a static query or only work with static interesting objects. Mouratidis et al. [52] study the kNN monitoring query problem in road networks, where query and data objects all move around. Two methods are proposed in the paper. The first method takes an incremental approach. Only updates that could potentially invalidate current query results are processed. The second method tries to reduce processing cost by grouping queries that fall together into the same sub-path. Both of their techniques only focused on reducing server workload without worrying about the update cost and the communication cost. As we pointed out in the introduction section, these costs will undermine the scalability of the proposed system.

To address both server computation and communication cost, some distributed solutions have been proposed to process monitoring queries. In [8], [9], [10], Cai and Hua propose a Monitoring Query Management (MQM) technique that can answer static monitoring queries over moving objects. Gedik and Liu introduce a MobiEyes system [17], [19], which is capable of answering moving queries over moving objects. Recently, Wu et al. [63] propose a distributed solution to answer moving kNN queries; nevertheless, the proposed solution is only applicable to open space environments. The above three techniques are all P2P based,

where moving objects participate in query processing. Another kind of distributed solutions relies on a concept called Safe Region [54], [26], [76]. In these solutions, each moving object is assigned with a Safe Region. As long as an object moves in its Safe Region, it does not need to send its location update to the server. Moreover, in [14], the authors use local-area wireless network to alleviate the high communication cost problem.

However, all these solutions mentioned above assume an open space as the underlying network and the extensions to road networks are not trivial. To the best of our knowledge, the work most related to ours is the research presented by Jensen et al. in [31], in which an algorithm was given for continuous kNN queries. This algorithm takes a client-server approach with the server keeps the location information of all the clients. For a given new query, the server performs a kNN search to identify a Nearest Neighbor Candidate set (NNC set) and a distance limit. This information is sent to the query object, which subsequently needs to repeatedly estimate distances between the clients in the NNC set and the query object to maintain the query result. When the number of clients in the NNC set with a distance to the query object greater than the distance limit exceeds a predefined certain threshold, the query object needs to contact the server to refresh the NNC set. A drawback of this approach is the potentially low accuracy in the kNN approximation because the criterion employed to refresh the NNC set does not consider the clients outside the NNC set, which could become the query's kNNs even when the criterion is still satisfied.

In summary, although there have been a tremendous amount of work in moving query processing, there is no existing distributed solution for such queries in a road network environment, which allows all objects to participate in query processing in order to reduce both server computation and communication costs.

2.3 System Model

The proposed system adopts a server-client architecture, where the server and clients communicate through a fixed wireless infrastructure, such as a cellular network. For each query, the server is responsible for identifying the potential clients that could possibly belong to the query result. The potential client needs to periodically check its location to determine if it is actually in some query's region. If it moves into or exits from a query's region, it will notify the server and the server just updates the query result.

2.3.1 System Assumptions

Our system has the following assumptions, which are widely accepted in the practice of wireless mobile environment.

- (i) Every moving object is equipped with some kind of positioning devices such as Global Positioning System (GPS) [3].
- (ii) Every moving object has a synchronized clock, for example, through using GPS.
- (iii) Every moving object has some computing power to perform data processing.

2.3.2 Definitions

Definition 2.1 (Network) *A network is an undirected graph $G = (N, E)$, where N is a set of nodes, and E is a set of edges. The distance between two nodes n_i and n_j is denoted by $d(n_i, n_j)$. If two nodes are directly connected by an edge, $d(n_i, n_j)$ is equal to the length of the edge. If the two nodes are not directly connected, $d(n_i, n_j)$ denotes the shortest network distance from n_i to n_j .*

Please note that for simplicity, we assume the network to be an undirected graph. However, our system can be easily extended to handle the directed graph case.

Definition 2.2 (Edge) An edge is expressed as $\langle n_i, n_j \rangle$, where n_i and n_j are two nodes. We assume that there is a universal labeling for nodes, and we express an edge in a way such that $n_i < n_j$ to avoid ambiguity. We refer to n_i and n_j as the start node and the end node of an edge, respectively.

In this chapter, road segment and edge are used interchangeably whenever there is no confusion. The next definition introduces a new concept, *edge distance*, which is not typical in conventional graph theory. *Edge distance* is utilized by mobile objects to compute network distances efficiently, even when the mobile objects do not have possession of the full network.

Definition 2.3 (Edge Distance)

The Edge Distance between any two different edges, say e_i and e_j , is defined as the distance from one node of e_i to another node of e_j . Since each edge has a start node and an end node, the edge distance between two edges can be one of the following four types: *SS*, *SE*, *ES*, *EE*, depending on which two nodes are used in calculating the distance. If both nodes are start (*S*) nodes, then the edge distance type is *SS*. If one node is a start (*S*) node and the other one is an end (*E*) node, then the edge distance type is *SE*. Similarly, there are distance types of *ES* and *EE*.

Formally, given $e_i = \langle n_{is}, n_{ie} \rangle$ and $e_j = \langle n_{js}, n_{je} \rangle$, $d_{xy}(e_i, e_j) = d(n_{ix}, n_{jy})$, where $x, y \in \{S, E\}$. Please note that this definition is not symmetric. For example, $d_{SE}(e_i, e_j) \neq d_{SE}(e_j, e_i)$. When the two edges are the same, we define an extra distance type called *SM* and the distance is zero. Formally, we have:

$$d_{SM}(e_i, e_j) = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

With the five types of edge distances all known, the shortest distance between any two edges can be calculated as follows by using edge distances:

$$d(e_i, e_j) = \min_{type \in \{SM, SS, SE, ES, EE\}} \{d_{type}(e_i, e_j)\}.$$

Definition 2.4 (Moving Object) A moving object is represented by a moving point in the road network. At any one time, an object o can be described as $\langle e, pos, direction, speed, reportTime \rangle$, where e is the edge that o is moving on and pos is the distance from o to the start node of e . Formally, if $e = \langle n_i, n_j \rangle$, then $d(o, n_i) = pos$ and $d(o, n_j) = d(n_i, n_j) - pos$. If o is moving from the start node of e to the end node of e , the direction is set to 1; otherwise, the direction is set to -1. $speed$ indicates the object's current moving speed. $reportTime$ records the time when the pos is reported. Distance between any two objects o_i and o_j , denoted as $d(o_i, o_j)$, is the shortest network distance from o_i to o_j . It can be calculated by using Property 1 below.

Property 1. Assume the positions of objects o_i and o_j are denoted as $\langle e_i, pos_i \rangle$ and $\langle e_j, pos_j \rangle$, where $e_i = \langle n_a, n_b \rangle$ and $e_j = \langle n_c, n_d \rangle$. The distance between o_i and o_j can be calculated as follows:

$$d(o_i, o_j) = \begin{cases} |pos_i - pos_j| & \text{if } e_i = e_j \\ \min_{x \in \{a, b\}, y \in \{c, d\}} \{d(o_i, n_x) \\ \quad + d(n_x, n_y) + d(n_y, o_j)\} & \text{if } e_i \neq e_j \end{cases}$$

Property 2. For a moving object, with pos , $direction$, $speed$, and $reportTime$ all known, and provided that the moving object still moves on the same edge, the new position of the moving object at current time $currentTime$ can be calculated as $(currentTime - reportTime) \times speed \times direction + pos$.

In order to handle long road segments, such as highway, we set a maximum road length allowed in the system. Any road segments longer than that maximum will be cut into

multiple shorter road segments. At each position where the road segment is cut, a virtual node is created and the resultant shorter road segments become virtual edges. In our system, virtual nodes and virtual edges are treated just like real nodes and real edges.

2.4 Proposed Solution for Moving Range Query

For a given query object, with a pre-specified range (defined by a network distance), a range query returns all moving objects inside that range. As the query object moves, the footprint (shape) of the query region also changes; hence, we name this class of queries as Dynamic Range Query (DRQ). This name also clearly distinguishes the studied query from the traditional rectangular-shaped or circle-shaped range queries. In this section, we discuss how the system handles DRQs.

2.4.1 Definitions

Definition 2.5 (Dynamic Range Query) *A Dynamic Range Query (DRQ) q can be denoted as $\langle o, length \rangle$, where o is the object issuing the query, and $length$ is the range (in network distance) of the query. Assume the set of all moving objects as O , $q.results = \{o_i | o_i \in O \wedge d(o, o_i) \leq length\}$.*

In the rest of this chapter, we refer to object o as *query object*, and moving object that has not issued monitoring query as *data object*.

Definition 2.6 (Monitoring Region of DRQ) *A monitoring region of a DRQ is a set of edges that can be reached by the query's range while the query object moves within its current edge. Formally, for a query $q = \langle o, length \rangle$ where o moves on edge e , its monitoring region $r = \{e_i | e_i \in E \wedge d(e, e_i) \leq length\}$. If an edge is included in a query's monitoring region, we say that this edge intersects with the query's monitoring region.*

Given a query $q = \langle o, length \rangle$, and o moving on edge $e = \langle n_s, n_e \rangle$, the monitoring region of the query can be determined by calling Algorithm 2.1. In Algorithm 2.1, we first add the edge where the query object is currently moving on to the monitoring region, then start to call Algorithm 2.2. Algorithm 2.2 utilizes a depth-first search to retrieve all edges included in the monitoring region starting from the input edge. The output of these two algorithms, denoted by MR , has the following format: $MR = \{\langle e_i, type, dist \rangle | e_i \in E \wedge type \in \{SM, SS, SE, ES, EE\} \wedge dist = d_{type}(e_i, e) \leq q.length\}$, where e_i is the edge included in the monitoring region, $type$ is the type of the *edge distance* from e_i to e defined by Definition 3 in Section 2.3, and $dist$ is the actual network distance. For an object moving on edge e_j in that monitoring region, it stores locally a subset of the above MR as $\{\langle e_i, type, dist \rangle | \langle e_i, type, dist \rangle \in MR \wedge e_i = e_j\}$, to facilitate computing its distance to the query object.

Algorithm 2.1 CalMonReg

CalMonReg[queryEdge $e_i = \langle n_s, n_e \rangle$, queryLength $length$]

- 1: $r = \{\langle e_i, SM, 0 \rangle\}$
 - 2: Find the start and end nodes of e_i
 - 3: $r = r \cup FindEdge(n_s, e_i, 0, start, length)$
 - 4: $r = r \cup FindEdge(n_e, e_i, 0, end, length)$
 - 5: Sort entries in r , remove duplicates. For entries with same e and $type$, keep the one with the shortest distance.
 - 6: RETURN r
-

Here we show an example using the above algorithms. Assume a simple road network as drawn in Fig. 2.2, where nodes are denoted as n_1, n_2 , etc. Each edge's length is indicated by the number close to that edge. Notations like n_1n_2, n_1n_3 , are used to represent edges. A query object A is moving on edge n_1n_6 , where n_1 is the start node and n_6 is the end node, and the query's range is 5. Then in the first line of Algorithm 2.1, we include the current edge into the result set as $\langle n_1n_6, SM, 0 \rangle$. Next, in line 3 of Algorithm 2.1, we call Algorithm 2.2 and add the following entries into the result set: $\langle n_1n_2, SS, 0 \rangle$. Because the length of edge n_1n_2 is less than the query's range (line 16 through line 19 in Algorithm 2.2), Algorithm 2.2 is called again.

Algorithm 2.2 FindEdge

FindEdge[startNode n , startEdge se , startDist $dist$, queryNodeType t , queryLength $length$]

```
1:  $r = \emptyset$ 
2: for each adjacent edge  $e$  of node  $n$  and  $e \neq se$  do
3:   if  $n$  is the start node of edge  $e$  then
4:     if  $t = start$  then
5:        $r = r \cup \langle e, SS, dist \rangle$ 
6:     else if  $t = end$  then
7:        $r = r \cup \langle e, SE, dist \rangle$ 
8:     end if
9:   else { $n$  is the end node of edge  $e$ }
10:    if  $t = start$  then
11:       $r = r \cup \langle e, ES, dist \rangle$ 
12:    else if  $t = end$  then
13:       $r = r \cup \langle e, EE, dist \rangle$ 
14:    end if
15:  end if
16:  if  $(dist + e.length) < length$  then
17:    Find the other node of  $e$ , denote as  $n'$ 
18:     $r = r \cup FindEdge(n', e, (dist + e.length), t, length)$ 
19:  end if
20: end for
21: RETURN  $r$ 
```

This time, $\langle n_2n_3, SS, 3 \rangle$ is added. Next, $\langle n_2n_{11}, SS, 3 \rangle$ is added. Similarly, $\langle n_1n_8, SS, 0 \rangle$, $\langle n_1n_9, SS, 0 \rangle$ are added. After executing line 4 and line 5 in Algorithm 2.1, we get the final result set as $\langle n_1n_2, SS, 0 \rangle$, $\langle n_1n_2, EE, 4 \rangle$, $\langle n_1n_6, SM, 0 \rangle$, $\langle n_1n_8, SS, 0 \rangle$, $\langle n_1n_9, SS, 0 \rangle$, $\langle n_2n_3, EE, 2 \rangle$, $\langle n_2n_3, SS, 3 \rangle$, $\langle n_2n_{11}, SS, 3 \rangle$, $\langle n_2n_{11}, SE, 4 \rangle$, $\langle n_3n_4, SE, 2 \rangle$, $\langle n_3n_6, EE, 0 \rangle$, $\langle n_5n_6, EE, 0 \rangle$, $\langle n_6n_7, SE, 0 \rangle$. In Fig. 2.2, all the thick edges are the edges that are included by the monitoring region.

2.4.2 Server Data Structure

A number of excellent disk-based storage structures have been proposed for road networks [58], [53]. Any of these techniques can be easily adapted for our network database to achieve

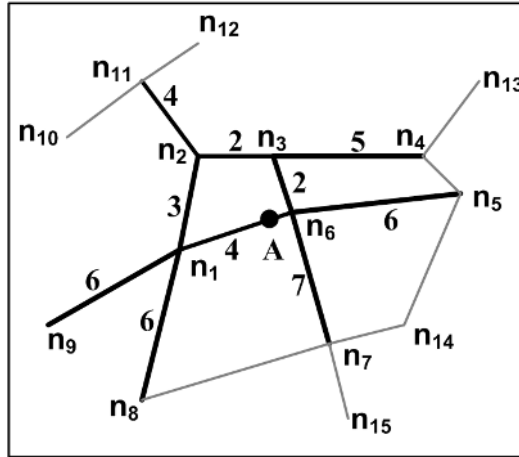


Figure 2.2: An example of calculating monitoring region

good access locality and therefore low I/O cost. As a result, in this section, we only focus on the data structures specific to our proposed technique.

First of all, we need to store the information about query objects and the associated monitoring queries. There are three tables used to meet this need. The first table is called Query Object Table (*QOT*), which stores query objects in the format of $\langle oid, eid, pos, direction, speed, reportTime \rangle$, where *oid* is the object ID for that query object, *eid* is the ID of the road segment where the query object is moving on, and *pos*, *direction*, and *speed* are recorded at time *reportTime*. The second table is named as Server Query Table (*SQT*), which stores information of queries. An entry in the *SQT* has the following format $\langle qid, oid, qLength, MR, results \rangle$, where *qid* is a universal ID assigned to that query, *oid* is the object ID for the query object, *qLength* is the range of the query, *MR* stores the monitoring region, and finally, *results* keeps the query result. The third table is a Query Segment Table (*QST*), where for each road segment, all queries intersecting with this road segments are stored. An entry in the *QST* has the format of $\langle eid, \{qid\} \rangle$. Fig. 2.3 illustrates the data structure employed on the server and client side.

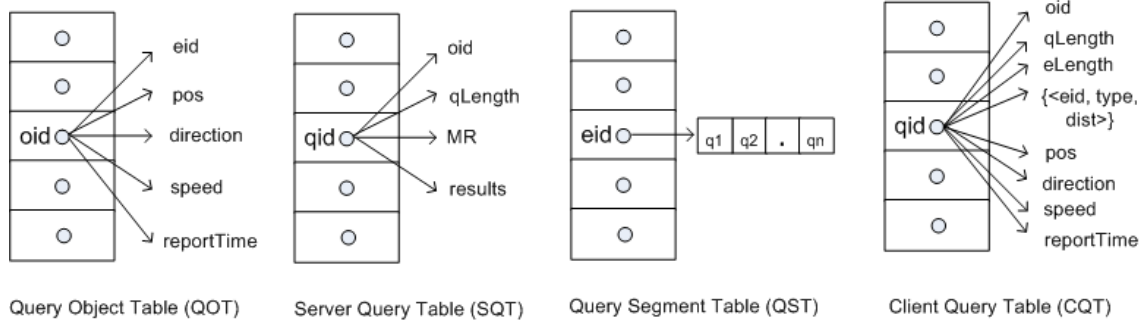


Figure 2.3: Data structure for DRQ processing

2.4.3 Moving Object Data Structure

On the moving object side, a table called Client Query Table (*CQT*) is used to store all queries whose monitoring regions intersect with the edge that the object is on. An entry in the *CQT* has a format of $\langle qid, oid, qLength, eLength, \{\langle eid, type, dist \rangle\}, pos, direction, speed, reportTime, \rangle$, where *qid* is the query ID, *oid* is the corresponding query object ID, *qLength* is the query's range, *eLength* is the length of the road segment that the query object is on, and $\{\langle eid, type, dist \rangle\}$ stores a set of tuples, where each tuple specifies the *edge distance*'s type and the actual distance from the query object's segment to the moving object's segment. Please note that *eid* should match the ID of the edge where this object is moving on. Moreover, for each stored query, we keep the *pos*, *direction*, *speed*, and the *reportTime* when these information were recorded for the corresponding query object.

2.4.4 Initialization

In the initialization phase, queries are submitted to the server by query objects. After the server receives a request from a query object, the server first saves the information about the query object and the query, and calculates the monitoring region for the query using Algorithm 2.1. After finding the monitoring region, the server updates the Server Query Table and notifies the moving objects on all segments of that monitoring region about this query using broadcast.

During the initialization phase, a data object also needs to send the server a message containing its location and heading. The server then determines the segment where the object is moving on, the object's relative position on the segment, and the object's moving direction (either -1 or 1) using Definition 4 given in Section 2.3. These information are sent back to the data object.

2.4.5 Server Side Message Processing

For a given range query, there are two different types of objects, namely, query object and data object. Below we list different types of messages sent out from moving objects, and discuss how the server responds to these messages.

2.4.5.1 Switch Segment Message

Every moving object needs to monitor its own position on the segment it is moving on. If its position on that segment is less than zero or greater than the segment's length, it knows that it has moved to a new segment. When a data object exits its current segment and enters a new segment, it provides the server with its new location and requests a new set of queries to be monitored. The server then first removes the data object from the current query results (if the data object is covered by any query), and sends the data object a new set of queries, the ID and the length of the new segment, the data object's relative position on the new segment, and its new moving direction.

However, if the object is a query object, the server needs to take the following steps.

- (i) update the QOT with the query object's new position.
- (ii) compute a new monitoring region for the query and update the SQT .
- (iii) send out messages to notify moving objects in the old monitoring region to stop monitoring the query, and notify moving objects in the new monitoring region to add this

query for monitoring. How moving objects respond to the broadcast message will be discussed in Section 2.4.6.

2.4.5.2 Enter/Exit Query Message

When a data object finds out that it enters or exits a query's region (the detail will be discussed in Section 2.4.6), it notifies the server. The server then updates the query result accordingly.

2.4.5.3 Other Messages

When a query object changes its speed, it notifies the server. The server then updates the *Query Object Table* and broadcasts the updated information. When a query object requests a different query range, the system treats it as the query object is moving to a new road segment.

2.4.6 Moving Object Side Message Processing

While a moving object is moving on a road segment, it needs to listen to broadcast messages from the server. If its road segment is covered by a new query's monitoring region, the moving object needs to add that query into its monitoring query list. If the message is for the update of an existing query, the moving object needs to update that query in its monitoring query list.

A moving object needs to periodically check queries in its *CQT* with Algorithm 2.3. If it enters a query region or exits a query region, it notifies the server, and the server updates the query result accordingly. Besides, a moving object also needs to periodically check if it is still in its current road segment. If it enters a new road segment, it first removes all old queries from its current monitoring query list and sends a message to the server with its new

location. After receiving a new set of queries and the new segment information, the moving object saves these data and starts the periodic checking routine again.

Algorithm 2.3 CalDist

CalDist[objPosition $oPos$, objEdgeLength $oeLength$, *Client Query Table*]

```

1:  $r = \emptyset$ 
2: for each query  $q$  in the Client Query Table do
3:    $d = \infty, minDist = \infty$ 
4:   Estimate query object's current position as  $qPos$ 
5:   for each  $\langle eid, type, dist \rangle$  tuple in query  $q$  do
6:     if  $type = SM$  then
7:        $d = |oPos - qPos|$ 
8:     else if  $type = SS$  then
9:        $d = oPos + qPos + dist$ 
10:    else if  $type = SE$  then
11:       $d = oPos + (eLength - qPos) + dist$ 
12:    else if  $type = ES$  then
13:       $d = (oeLength - oPos) + qPos + dist$ 
14:    else
15:       $d = (oeLength - oPos) + (eLength - qPos) + dist$ 
16:    end if
17:    if  $d < minDist$  then
18:       $minDist = d$ 
19:    end if
20:  end for
21:  if  $minDist \leq qLength$  then
22:     $r = r \cup \{qid\}$ 
23:  end if
24: end for
25: RETURN  $r$ 

```

The inputs for Algorithm 2.3 are the data object's position on the edge, the length of the edge where the data object is on, and the *CQT*. The output is the list of monitoring queries still containing the moving object in their results. Specifically, on line 4, we apply Property 2 in Section 2.3 to estimate the current position of the query object, given that the position and speed information about the query object are available. Then with the pre-calculated edge distances $\{\langle eid, type, dist \rangle\}$ between two edges in which the data object and query object are currently moving on, line 5-20 uses Property 1 in Section 2.3 to compute

the shortest distance between the data object and the query object. If the shortest distance is less than the query’s range, this query is included to the query result (line 21-23). Finally, line 25 returns all monitoring queries that contain the data object in their results. The interesting point of the algorithm is the utilization of pre-calculated edge distances. By using the server to carry out the computationally expensive computation of edge distances, we gain the following advantages: 1) there is no need to transfer the network map from the server to mobile objects for shortest network distance computation, and 2) we avoid the expensive shortest path computation on mobile objects. Even in the case in which each mobile object possesses a network map, pre-calculated edge distances can still be useful in realizing the second advantage.

2.4.7 An Example

We still use the example given in Section 2.4.1 to illustrate the idea. In Fig. 2.4, assume the query object A is at position 3.6 on edge n_1n_6 , and there is a data object B at position 0.5 on edge n_2n_{11} . From the discussion of Section 2.4.1, we know that the data object B has two entries for the query issued by the query object A stored as $\langle n_2n_{11}, SS, 3 \rangle$, $\langle n_2n_{11}, SE, 4 \rangle$. Using the first entry yields a distance as 7.1, which is out of the query’s range (the range is 5). With the second entry, the distance is 4.9, and the data object B should be included into the query’s result.

2.5 Proposed Solution for Moving kNN Query

In this section, we show that how the proposed framework can be used to answer moving kNN queries on moving objects. A moving kNN query, with query result including all objects that are the query object’s k nearest neighbors, is quite different from a Dynamic Range Query. For a moving object, to determine if itself is in the query result, the object not only needs to monitor its distance to the query object, but also needs to be concerned about the

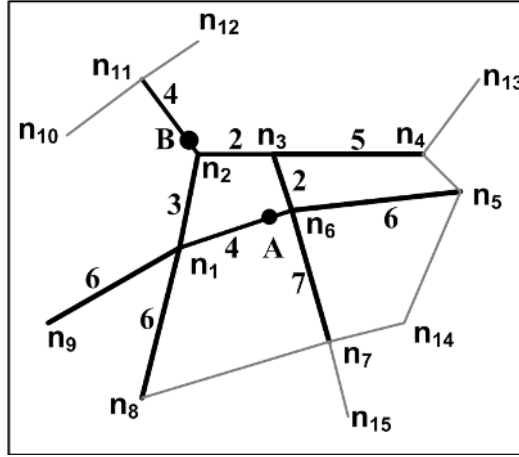


Figure 2.4: An example of calculating network distance region

distances of other objects to the query object. We address the problem by first defining a “range” for a kNN query. Using the “range”, we calculate the monitoring region for a kNN query. Then all objects in the monitoring region can monitor their distances to the query object, similar to handling Dynamic Range Queries. However, there are some key differences, which will be discussed in the following sections.

2.5.1 Definitions

Definition 2.7 (*k*-Nearest-Neighbor Query) A kNN query q can be denoted as $\langle o, k \rangle$, where o is the object issuing the query, and k is the number of nearest neighbors interested in. Denote the set of all other moving objects (i.e. excluding o) as O , a kNN query q returns a subset $O' \in O$ of k objects, such that for any object o_i in O' and any object o_j in $(O - O')$, $d(o_i, o) \leq d(o_j, o)$. For a given kNN query $\langle o, k \rangle$, we call the object o as the query object, all objects in the set $(O - O')$ as the data objects. Among all objects belonging to the query results O' , we name the object that has the largest distance to o as the kNN object, and all other objects in the set O' as the $(k-i)$ NN objects, with $i = 1, \dots, k - 1$.

Definition 2.8 (*Range of kNN Query*) Given a kNN query $q = \langle o, k \rangle$, with object o moving on edge e_o . Assume the kNN object for this query is object o_{NN} , which is moving on

edge e_{NN} , then the range of the kNN query, denoted as $q.range$, is defined as follows:

$$q.range = \begin{cases} e_o.length & \text{if } e_o = e_{NN} \\ e_o.length + e_{NN}.length \\ \quad + d(e_o, e_{NN}) & \text{if } e_o \neq e_{NN} \end{cases}$$

As shown in the following definition, this range concept is utilized to prune out objects that certainly can not be part of query result.

Definition 2.9 (Monitoring Region of kNN Query) *A monitoring region of a kNN query is a set of edges that can be reached within the query's range while the query object and the query's kNN object both move on their own current edges. Formally, for a query $q = \langle o, k \rangle$ where object o moves on edge e , its monitoring region $r = \{e_i | e_i \in E \wedge d(e, e_i) \leq q.range\}$.*

To illustrate the above definitions, we give an example in Fig. 2.5. Assume that there is one object A (represented by a star) moving on edge n_1n_4 , and we are interested in its 2-NNs, which have been determined to be B and C (represented by triangles). Based on Definition 7, A is the query object, B is the (k-i)NN object, C is the kNN object, and all other objects (represented by circles) are data objects. Since C is moving on edge n_2n_3 , and the shortest distance between edge n_1n_4 and n_2n_3 is 1 (through edge n_3n_4), based on Definition 8, the range of this query is computed as the sum of the lengths of edge n_1n_4 and edge n_2n_3 , then added by 1, which gives $(3 + 2 + 1) = 6$. The monitoring region is then computed by expanding from both nodes (n_1 and n_4) of edge n_1n_4 using Algorithm 2.1. The edges included in the monitoring region are shown in the figure as the thick edges. All objects moving in the monitoring region need to monitor this query.

2.5.2 Server Data Structure

Similar to the data structure used to answer Dynamic Range Queries, we also have three tables. The QOT and the QST are exactly the same as the ones used in Section 2.4.

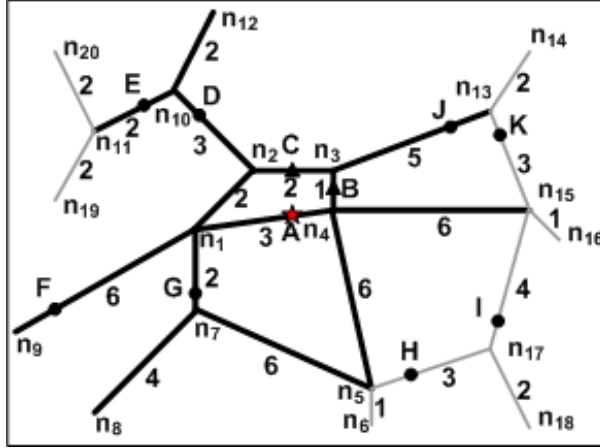


Figure 2.5: Monitoring region for kNN query

However, for kNN query, the system needs to monitor not only the position of query object, but also the information of the corresponding kNN object and $(k-i)$ NN objects. As a result, Each record in the *SQT* has the form of $\langle qid, oid, k, kNNobject, (k - i)NNobjects, MR \rangle$, where *MR* is the output from Algorithm 2.1 for monitoring region calculation.

Besides the three tables, to facilitate the initialization step (to be discussed in Section 2.5.4), we also keep track of how many objects currently moving on each edge.

2.5.3 Moving Object Data Structure

Similar to the data structure used to answer Dynamic Range Queries, there is a *CQT* on each moving object. For an object moving on edge e , each entry in the table has the following format: $\langle qid, oid, eLength, \{ \langle e, type, dist \rangle \}, oid_{nn}, eLength_{nn}, \{ \langle e_{nn}, type_{nn}, dist_{nn} \rangle \} \rangle$, where qid is the query id, oid is the corresponding query object's id, $eLength$ is the length of the edge that the query object is on, and $\{ \langle e, type, dist \rangle \}$ stores a subset of *MR* (the attribute inside the Server Query Table), where each tuple specifies the edge distance type and the actual edge distance from the moving object's segment to the query object's segment. With $eLength$ and the set $\{ \langle e, type, dist \rangle \}$, the moving object can calculate its distance to the query object. Similarly, oid_{nn} denotes the kNN object's object id, $eLength_{nn}$ is the length of

the edge where the kNN object is on, and $\{ \langle e_{nn}, type_{nn}, dist_{nn} \rangle \}$ stores a set of tuples which help to calculate the distance from the kNN object to the query object. Please note that e_{nn} is the edge where the kNN object is moving on.

In order to estimate the positions of the query object and the kNN object at different time units other than at the saved *reportTime*, we also store the position and speed information about the query object and the corresponding kNN object on moving objects.

2.5.4 Initialization

For every new moving object that enters the system, it needs to report its location, heading, and speed to the server. The server determines and sends the moving object a set of queries that should be monitored. If the new moving object is a query object, the server calculates the first set of k nearest neighbors in the following four steps:

- (i) Since the server knows how many objects are moving on each edge, by comparing with the requested number k , the server can decide the set of edges to send a probe message. The probe message has the format of $\langle qid, oid, pos, eLength, \{ \langle e, type, dist \rangle \} \rangle$, where pos is the position of the query object on its edge, and other parameters have the same meanings as those discussed in Section 2.5.3.
- (ii) After the probe message is received by all moving objects moving on the identified edges, based on Property 1, moving objects can calculate their distances to the query object and send the distances back to the server.
- (iii) The server compares all the returned distances and picks the k smallest ones. The moving objects with the k smallest distances are the initial k nearest neighbors. Among the k identified objects, the one with the largest distance is the kNN object.
- (iv) With the kNN object known, the server calculates the query's range using Definition 8, computes the query's monitoring region using Algorithm 2.1, and sends a message,

which contains the information of the query object and the kNN object, to all objects in the monitoring region.

2.5.5 Server Side Message Processing

For a kNN query, there are four different types of objects, namely, query object, data object, kNN object, and (k-i)NN object. Please note that since the system can support multiple concurrent kNN queries, for a moving object, it can assume multiple roles. Below we list different types of messages sent out from moving objects, and discuss how the server responds to these messages.

2.5.5.1 Switch Segment Message

If a moving object exits its current road segment, the moving object reports to the server with its current location and requests for the new segment's ID and length, and a new set of queries. For each query, the server sends the query object and the kNN object's information with the relevant edge distances, and the lengths of the edges where the query object and the kNN object are moving on, respectively. With the received information, later on, the moving object can estimate the position of the query object and the kNN object. Using saved edge distances, the moving object can calculate its distance to the query object and the distance from the kNN object to the query object.

However, if the moving object itself is also a query object (or a kNN object) for some query, the monitoring region for that query needs to be updated. The server performs the following three tasks:

- (i) if the object is a query object, then update the *QOT* with the query object's new position. If the object is a kNN object, then update the *SQT*.
- (ii) compute a new monitoring region for the query using updated information. Update the monitoring region in the *SQT*.

(iii) send out messages to notify moving objects in the old monitoring region to stop monitoring this query, and notify moving objects in the new monitoring region to add this query for monitoring. If the moving object is a (k-i)NN object for some query, although there is no need for monitoring region re-computation, the server needs to update the *SQT* accordingly.

2.5.5.2 Enter Query Message

For a data object, it periodically checks its distance to the monitored query object, and compares with the distance from the kNN object to the query object. If it is getting closer to the query object than the kNN object is, a message is sent to the server to indicate that it is currently part of the query result. The server first estimates the current positions of the saved kNN object and (k-i)NN objects to decide which object should be replaced by the new-coming object. Then the server updates the *QOT*. If the replaced object is the kNN object, the server also calculates a new monitoring region based on the new kNN object, and notifies all affected objects.

2.5.5.3 Exit Query Message

For a (k-i)NN object, since it could move further away from the query object and become the kNN object, it needs to periodically monitor its distance to the query object and compare with that of the kNN object. Once the distance is larger than the distance from the kNN object to the query object, the (k-i)NN object needs to report to the server. After the server receives this type of message, it replaces the current kNN object with the one sending out the message, re-calculates the monitoring region, and notifies all affected objects.

2.5.5.4 Other Messages

Other than the three types of messages described above, there are some other scenarios when a moving object needs to contact server. When a query object (or a kNN object) changes its speed, it sends the update to the server, and the server updates the QOT (or the SQT) and forwards the update to relevant moving objects. Similarly, when a (k-i)NN object changes its speed, it also notifies the server, and the server just updates the SQT (i.e. No need to send the update to moving objects).

2.5.6 Moving Object Side Message Processing

The operations on the moving object side is very similar to those required for handling Dynamic Range Query. However, there is one small difference. If the broadcast message is about a change from a kNN object. The moving object also needs to update its CQT with the updated kNN object information.

2.5.7 An Example

We use the same example used in Section 2.5.1 to show how a data object keeps monitoring its distance to the query object and the distance from the kNN object to the query object. At time t , as shown in Fig. 2.5, the query object A is at position 2.5 on edge n_1n_4 , the kNN object C is at position 1 on edge n_2n_3 , and a data object G is at position 1.5 on edge n_1n_7 . Since G is inside the query's monitoring region, it has A and C 's information saved locally. Besides, it stores the edge distance $\{\langle n_1n_7, SS, 0 \rangle\}$ and the length of edge n_1n_4 , to determine the distance from itself to A . To calculate the distance from C to A , it also has the edge distances $\{\langle n_2n_3, SS, 2 \rangle, \langle n_2n_3, SE, 3 \rangle, \langle n_2n_3, ES, 4 \rangle, \langle n_2n_3, EE, 1 \rangle\}$ and the length of edge n_2n_3 saved.

At time $(t+1)$, as shown in Fig. 2.6, data object G moves to position 1 on edge n_1n_7 . It estimates the new position of A on edge n_1n_4 using Property 2 and gets 1.5. Similarly, it

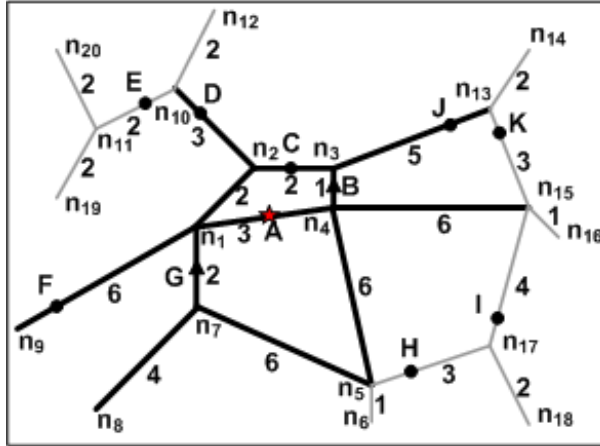


Figure 2.6: Example for message processing

estimates the new position of C on edge n_2n_3 as 1.1. Then with Property 2 in Section 2.3, it computes its distance to A as 2.5 (calculated as: $1 + 1.5 + 0$), while the distance from C to A is 3.4 (through the edge distance $\langle n_2n_3, EE, 1 \rangle$, calculated as: $(2 - 1.1) + 1.5 + 1$). Since its distance to the query object is less than the distance from the kNN object to the query object, it sends an *Enter Query Message* to the server. The server replaces C with G as the new kNN object, determines the new query range, and re-computes the monitoring region. In this example, the new query range is 5 (sum of the lengths of edge n_1n_4 and edge n_1n_7), and the new monitoring region is drawn as thick edges in Fig. 2.6. As we can see, data object E on edge $n_{10}n_{11}$ is no longer in the monitoring region.

2.6 Performance Study

We study the performance of the proposed technique using simulation. There are three performance metrics: server side workload, system communication cost, and client side workload. Additionally, we also present an analytical study on the system communication cost.

2.6.1 Simulation Setup

The interested area is a square shaped region of 50×50 square miles. We generate a synthetic road network by first placing nodes randomly on the region, then connecting nodes randomly to form road segments. Table 1 gives the definitions for the parameters used in the setup. We generate 2000 nodes and 4000 edges, with each edge’s length in the range of $(0, seg_len_{max}]$ miles. Moving objects are randomly placed on road segments with initial speeds and directions. The speeds are in the range of $[0.1, v_{max}]$ mile/min, following a Zipf distribution with a deviation of 0.7. The travel direction on a road segment is set to either 1 or -1. When a moving object approaches a road intersection, it moves to a new randomly selected road segment. At each time unit, a certain percentage (denoted as p) of the moving objects will change their speeds. The threshold for changing speed is set as 0.1 mile/min.

There are n_o number of moving objects and n_{mq} number of queries. Query objects are selected randomly from the moving objects. For range queries, the range is specified randomly by picking an integer value in the range of $[1, range_{max}]$ miles. For kNN queries, a pre-defined number (k) of interested nearest neighbors are specified, with k selected from the range $[1, k_{max}]$.

The time step parameter for the simulation is one minute. We run simulation for 100 times and compute the average as the final output. Each simulation lasts for 200 time units. The simulation was run on a Pentium 4 2.6GHz desktop PC with 2GB memory.

Table 2.1: Simulation Parameters

Parameter	Description	Value Range	Default
n_o	Number of moving objects	[50000, 100000]	100000
n_{mq}	Number of queries	[100, 1000]	1000
seg_len_{max}	The maximum length of road segment (mile)	[3, 8]	3
v_{max}	Maximum speed (mile/min)	[1, 2]	1
$range_{max}$	Maximum length of query range (mile)	[1, 10]	2
k_{max}	Maximum number of Nearest Neighbors	[1, 20]	5
p	PCT of objects changing speed / time unit	[2, 50]	10

In the experiments, we vary different parameters, as listed in Table 1, to study the scalability of the proposed system. If not otherwise specified, the experiments take the default values.

2.6.2 Server Workload

Since all existing distributed approaches are not applicable to a spatial network environment or extensions are non-trivial, we compare our technique with one centralized approach, named as *Query Index* [54], which is originally designed for an open space environment. To make the comparison fair, the *Query Index* scheme is adapted to a road network environment. In the adapted scheme, queries are indexed by a Query Segment Table (similar to the table used in our technique), where for each segment, all queries whose query object can reach that segment within the query’s range are saved. Every time when a moving object sends its updated location to the server, based on the segment where the moving object is on, the server retrieves all queries for that segment from the Query Segment Table. Then the server computes the distances from the moving object to query objects to determine if the moving object belongs to any query result. Also, every time when a query object’s location is updated or its kNN object’s location is updated, the server updates that Segment Query Table.

The server workload can be characterized by I/O time and CPU time, of which I/O time is more dominant. In a centralized approach, to answer a query, the server needs to find all objects (or objects) residing on the affected road segments. Therefore, we measure the server side workload as the total number of road segments accessed. We note that this is not a very accurate measurement, nevertheless, it serves as a good reference.

Fig. 2.7 shows the server loads for our technique and the *query index* method when the number of queries increases from 100 to 1000. Please note that the y-axis is in logarithmic scale. The plot shows that both the proposed technique and the *Query Index* method incur

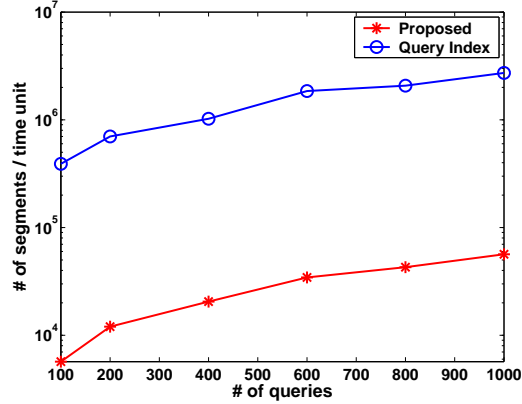


Figure 2.7: Effect of number of queries on server workload

more server workload with the increases in the number of concurrent queries. The figure also shows that the proposed approach is about 50 times better than the *Query Index* method. This huge savings can be attributed to the computations carried out on moving objects, which greatly reduce server workload.

2.6.3 Communication Cost

We also compare communication cost to a centralized approach, which we name it as Query Blind Optimal (*QBO*) method. In the *QBO* method, moving objects only need to contact the server when they switch segments or change speeds. When an object moves to a new segment, the server sends back the new segment’s length to help the object to determine when it moves out of that segment. At each time unit, the server estimates all moving objects’ locations and answers all queries. This method is optimal on communication cost if we assume that moving objects do not have any knowledge about queries, which is why we call it Query Blind Optimal method. Besides this *QBO* method, we also have a naive method which serves as a basis for comparison. In this naive method, all moving objects report to the server when their locations change, as a result, there is no need for the server to send messages back to the clients.

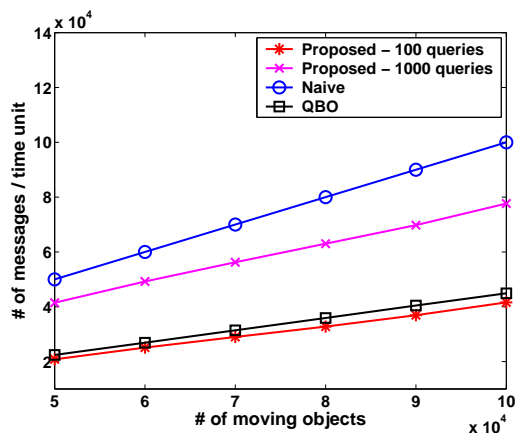


Figure 2.8: Effect of number of objects on communication cost

The communication cost is measured by summing up the total number of messages sent by the clients and the total number of messages sent by the server to reflect bandwidth consumption.

In Fig. 2.8, we vary the number of moving objects to study its effect on communication cost. The number of objects is varied from 50000 to 100000. The plot shows that when the number of moving objects increases, all three methods experience an increased number of messages. Among them, the naive method requires the largest number of messages, which is due to its periodical reporting mechanism. The *QBO* method incurs relatively lower communication cost. Interestingly, when the number of moving queries is small (i.e., 100 queries), the proposed technique demands fewer messages when compared to the *QBO* method. This is because in the *QBO* method, every significant move by an object has to be reported, however, in the proposed technique, only if a move can affect some query’s result, the object will report to the server. Nevertheless, when the number of queries is larger (i.e., 1000 queries), the amount of messages is large. That is because, the more queries, the more activities associated with objects crossing query’s boundaries.

Fig. 2.9 studies the effect of maximum speed. The allowed maximum speed is increased from 1 mile/min to 2 miles/min. The plot shows that when the speed increases, both the proposed technique and the *QBO* technique incur higher communication cost. That is

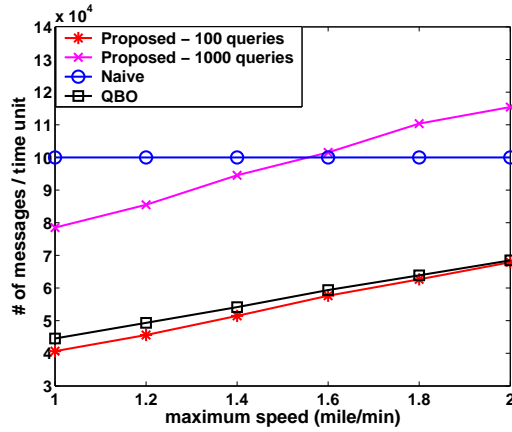


Figure 2.9: Effect of maximum speed on communication cost

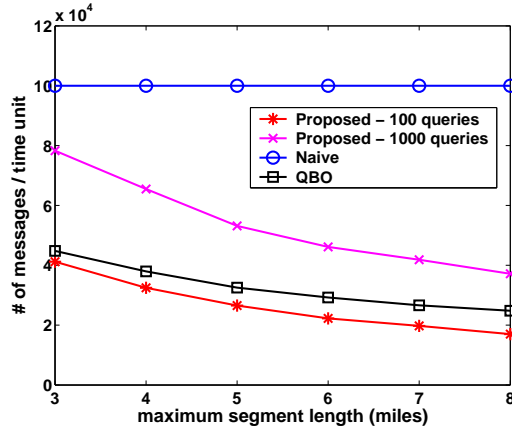


Figure 2.10: Effect of maximum segment length on communication cost

because with increased speed, objects will switch segments more frequently, which lead to more messages. For the naive method, varying the maximum speed has no effect. This is as expected, since objects just report to the server periodically.

In Fig. 2.10, we vary the allowed maximum segment length from 3 miles to 8 miles to study its effect on communication cost. The plot shows that for the proposed method and the *QBO* method, the number of messages decreases as the segment length increases. This is mainly due to the fact that longer segment leads to longer travel time on each segment, which can reduce the frequency of object switching segment. Also, as expected, this variable does not have effect on the naive method.

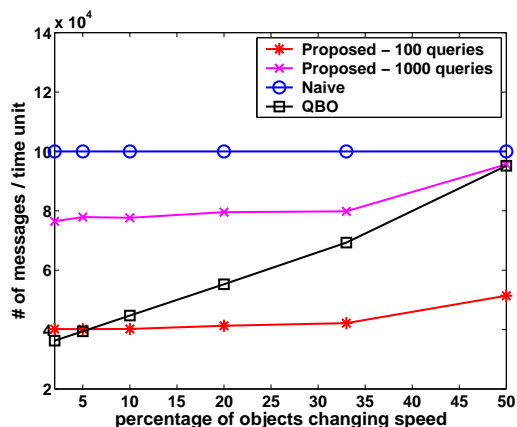


Figure 2.11: Effect of percentage of objects changing speed on communication cost

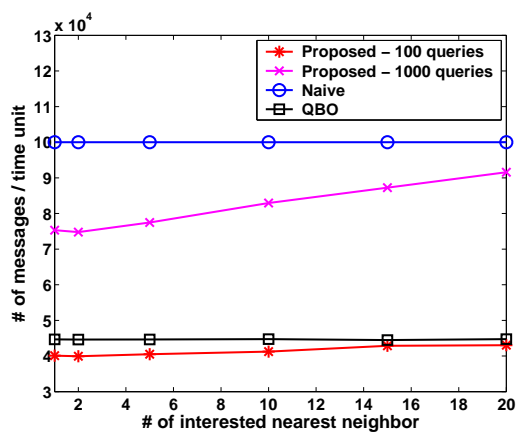


Figure 2.12: Effect of number of interested nearest neighbor on communication cost

Fig. 2.11 studies the effect of increasing the percentage of objects changing speed per time unit from 2 percent to 50 percent. The result shows that for the naive method, the curve is a flat line. Both our technique and the *QBO* technique incur more communication cost when there are more objects changing speeds at every time step. Compared to the *QBO* technique, our technique has a much less steeper curve because among all objects that change speeds, only query objects, kNN objects, and (k-i)NN objects, which combined account for a small fraction of the total number of objects, need to contact the server, however, in the *QBO* technique, all objects changing speeds have to report to the server.

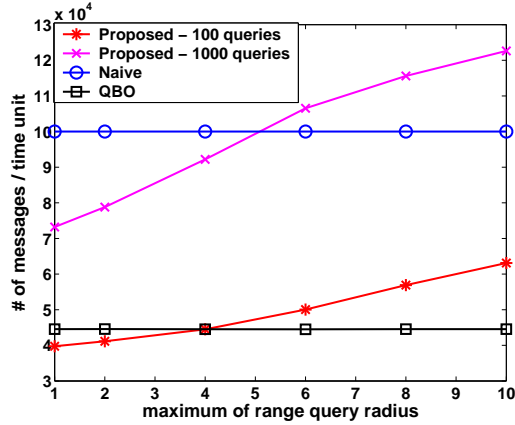


Figure 2.13: Effect of maximum range query radius on communication cost

Fig. 2.12 studies the effect of the number of interested nearest neighbor on communication cost. The plot shows that this variable does not have any impact on the naive method and the *QBO* method, as expected. Regarding our techniques, the communication cost increases gradually as there are more nearest neighbors interested. This is expected since a bigger k will make more objects into $(k-i)$ NN objects, and quite probably, larger monitoring regions are demanded. Consequently, higher communication cost is necessary.

In Fig. 2.13, we focus on the range queries by enlarging the range query radius from 1 mile to 10 miles. Similar to Fig. 2.12, no effect is observed on the naive and *QBO* method. For the proposed technique, the communication cost increases linearly as the radius increases. This is because with a larger query radius, the monitoring region is larger, which leads to more objects covered by the region. Consequently, there is a higher chance of changing query result activities.

2.6.3.1 Analytical Study on Communication Cost

In this section, we study the communication cost of the system using analysis. The study can help us to better understand the result of the experimental studies. To simplify the study, we assume that we only have circular range queries in the system. Let $com_cost(T)$

denote the communication cost (measured as the number of messages) for an object during a given time period of T .

The communication cost consists of the following three components: seg_switch_cost , the cost associated with the object switching segment; spd_change_cost , the cost associated with the object changing speed; and $result_update_cost$, the cost associated to the messages sent from the object to the server when the object enters or exits some query's region. Denote avg_seg as the average length of road segments, avg_r as the average radius of query's range, and avg_spd as the average speed of an object, t as the basic time unit for system evaluation (which means how frequently an object checks for speed changes, result updates, or segment switches), p as the percentage of an object changing velocity at each time unit, $prob_change_result$ as the probability of an object entering or exiting some query's region at each time unit (How to estimate this probability will be discussed in detail later). Then $com_cost(T)$ can be described as follows:

$$\begin{aligned}
com_cost(T) = & \frac{T}{\frac{avg_seg}{avg_spd}} \times seg_switch_cost + \\
& \frac{T}{t} \times p \times \frac{n_{mq}}{n_o} \times spd_change_cost + \\
& \frac{T}{t} \times prob_change_result \times result_update_cost
\end{aligned} \tag{2.1}$$

The expression before seg_switch_cost estimates the number of times for a given object switching segments in the time interval T . Since the spd_change_cost only applies to query object, the expression before spd_change_cost is the probability that a given object is a query object times the percentage of an object changing speed in the time interval T . The expression before $result_update_cost$ is an estimate of the number of times an object affecting some query result in the time interval T .

Also, we have

$$seg_switch_cost = 1 + 1 + \frac{n_{mq}}{n_o} \times 2 = 2 + \frac{2n_{mq}}{n_o} \quad (2.2)$$

$$spd_change_cost = 1 + 1 = 2 \quad (2.3)$$

$$result_update_cost = 1 \quad (2.4)$$

The equation for *seg_switch_cost* can be explained as follows. When an object switches segment, first it needs to send one message to the server to request for a new set of queries, and the server sends one message back to the object. The third component means that, should the object be a query object (with probability of $\frac{n_{mq}}{n_o}$), the sever needs to broadcast two message, one message to notify objects to stop monitoring this query, and another message to notify object to start monitoring this query. The *spd_change_cost* only applies to query object, and is equal to 2, due to the following reason. When a query object changes speed, it first sends a message to the sever, then the server broadcasts a message to notify the affected data objects. Moreover, the *query_update_cost* is 1, since the object only needs to send one message to the server to update the query result.

To estimate *prob_change_result*, we first need to find out the average number of queries to be monitored per object, denoted as *avg_q_mon*. To estimate *avg_q_mon*, we have to know the average number of segments covered in a query's monitoring region. To simplify the estimation, we assume that the road network is like a grid, which means all segments have the same length, and each node is connected to four segments.

Theorem 2.1 *Denote*

$$n = \lceil avg_r / avg_seg \rceil \quad (2.5)$$

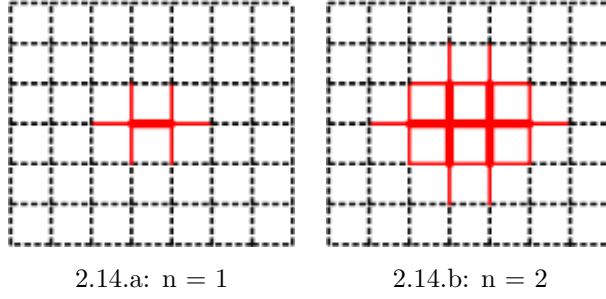


Figure 2.14: Monitoring regions for $n = 1$ and $n = 2$

then the total number of segments covered by a query's monitoring region, denoted as $f(n)$, is:

$$f(n) = 4n^2 + 4n - 1 \quad (2.6)$$

Proof: When $n = 1$, as shown in Fig. 2.14.a, there are 7 segments in the monitoring region. The solid thick segment in the center is the segment where a query object is on. All the surrounding 6 segments are included in the monitoring region.

This means that Eq. 2.6 holds when $n = 1$:

$$f(1) = 7 = 4 \times 1^2 + 4 \times 1 - 1 \quad (2.7)$$

Now, assume that when $n = k - 1$, with $k \geq 2$, we have

$$f(k - 1) = 4(k - 1)^2 + 4(k - 1) - 1 \quad (2.8)$$

The next step is to identify the relationship between $f(k)$ and $f(k - 1)$. When $n = 2$, the monitoring region is drawn in Fig. 2.14.b. As we can see, the same segments inherited from when $n = 1$ are still included in the monitoring region, represented with solid thick segments. Moreover, there are new ones added into the monitoring region represented in the solid thin segments.

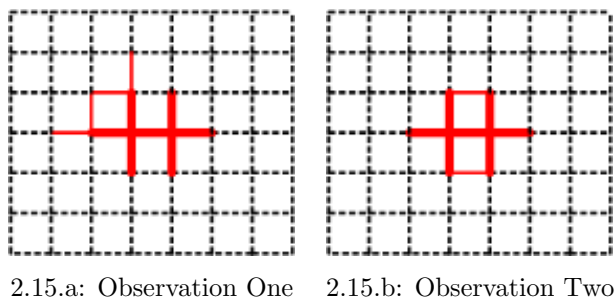


Figure 2.15: Observations when $n = 2$

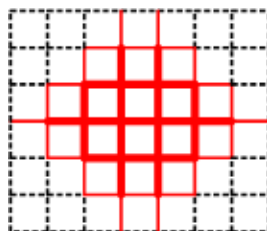


Figure 2.16: Monitoring region for $n = 3$

By comparing Fig. 2.14.b and Fig. 2.14.a, we can tell that when n is increased from 1 to 2, there are two groups of new segments added into the monitoring region, as depicted by the solid thin segments in Fig. 2.15.a and Fig. 2.15.b, respectively. The thin solid segments in Fig. 2.15.a are the corner segments to be added into the new monitoring region. When $n = 2$, there are four such corner segments in each corner. When $n = k$, there will be $2k$ of such segments in the corner. Since there are four corners, there are $2k * 4 = 8k$ such segments, except two segments that are counted twice (the segments on the far left and right). So the number of segments belonging to the first set is $8k - 2$. The second group of new segments is depicted in Fig. 2.15.b. There is one solid thin segment that bridges the left-top corner and the right-top corner, similarly, there is another one in the bottom. When n is increased by one, there are always two new segments to be added into the new monitoring region.

To help verify these two observations, we draw the monitoring region for $n = 3$ in Fig. 2.16, which shows that there are 22 segments belonging to the first group and 2 segments from the second group.

As a conclusion, when n is increased from $k - 1$ to k , there are two groups of segments added into the new monitoring region. The first group has $(8k - 2)$ segments, while the second group has 2 segments. Then we have

$$\begin{aligned} f(k) &= f(k - 1) + (8k - 2) + 2 \\ &= f(k - 1) + 8k \end{aligned} \tag{2.9}$$

After plugging Eq. 2.8 into Eq. 2.9, we get

$$\begin{aligned} f(k) &= 4(k - 1)^2 + 4(k - 1) - 1 + 8k \\ &= 4k^2 + 4k - 1 \end{aligned} \tag{2.10}$$

By combining Eq. 2.7, 2.8, and 2.10, we prove the correctness of Eq. 2.6 using mathematical induction.

□

Based on Theorem 2.1, the average number of queries monitored by an object is

$$avg_q_mon = \frac{n_{mq}(4n^2 + 4n - 1)}{n_{seg}} \tag{2.11}$$

For all segments in a monitoring region, there are two types of segments: boundary segments and inner segments. A boundary segment has one or more adjacent segments not residing in the same monitoring region, and an inner segment has all of its adjacent segments belonging to the same monitoring region. Given a monitoring region with $(4n^2 + 4n - 1)$ number of segments, $8n$ of them are the boundary segments, and $8(n - 1)$ are secondary boundary segments, which means if we delete all boundary segments from a monitoring region, there are $8(n - 1)$ inner segments becoming boundary segments¹.

¹Please see the proof of Theorem 2.1 for more details.

For all objects residing in a query's monitoring region, we assume that only objects residing on boundary segment are possible to move into the query's region; similarly, only objects on the secondary boundary segment are possible to move out of the query's region.

Given an object, at time t_1 , record that the distance between the object and the query object is d_1 . After time t , at $t_2 = t_1 + t$, the distance between the two objects as d_2 . To simplify the problem, we assume that the object and the query object can move either in the same direction or the opposite direction. Then if the object and query object move toward the same direction, then d_2 stills equals to d_1 , and we have $|d_2 - d_1| = 0$, however, if they move in the opposite direction, we have $|d_2 - d_1| = 2d$, with $d = avg_spd \times t$. Since the two events have same probability, the expected value of $|d_2 - d_1|$ is d .

After one time unit t , an object can either move closer to the query object by d or further away from the query object by d . Therefore, for an object residing on boundary segments, if it moves closer to the query object, the chance of moving into the query's region is the minimum of $\frac{d}{avg_seg}$ and one. If d is greater than avg_seg , the object will move into the query's region for sure, otherwise, the probability is $\frac{d}{avg_seg}$. Since the object only has a half chance of moving closer to the query object, the overall probability of moving into query's region is $\frac{1}{2}min(\frac{d}{avg_seg}, 1)$. Similarly, for objects residing on secondary boundary segments, the probability of moving out of query's region is $\frac{1}{2}min(\frac{d}{avg_seg}, 1)$.

Considering the fact that an object has $\frac{8n}{4n^2+4n-1}$ of possibility of being on a boundary segment and $\frac{8(n-1)}{4n^2+4n-1}$ of possibility of being on a secondary boundary segment, the total possibility of an object affecting query result by either entering query's region or exiting query's region is

$$\begin{aligned}
& \frac{1}{2} \times min(\frac{d}{avg_seg}, 1) \times \frac{8n}{4n^2 + 4n - 1} + \\
& \frac{1}{2} \times min(\frac{d}{avg_seg}, 1) \times \frac{8(n-1)}{4n^2 + 4n - 1} \\
& = min(\frac{d}{avg_seg}, 1) \times \frac{8n - 4}{4n^2 + 4n - 1}
\end{aligned} \tag{2.12}$$

The above equation multiplied by the average number of queries monitored by an object gives the *prob_change_result* as

$$\begin{aligned}
& \text{prob_change_result} = \\
& \min\left(\frac{d}{\text{avg_seg}}, 1\right) \times \frac{8n - 4}{4n^2 + 4n - 1} \times \frac{n_{mq}(4n^2 + 4n - 1)}{n_{seg}} = \\
& \min\left(\frac{d}{\text{avg_seg}}, 1\right) \times \frac{n_{mq}(8n - 4)}{n_{seg}} \tag{2.13}
\end{aligned}$$

Finally, we put all these together to get *com_cost(T)* as

$$\begin{aligned}
& \text{com_cost}(T) = \\
& \frac{T}{\frac{\text{avg_seg}}{\text{avg_spd}}} \times \left(2 + \frac{2n_{mq}}{n_o}\right) + \frac{T}{t} \times \frac{2pn_{mq}}{n_o} + \\
& \frac{T}{t} \times \min\left(\frac{\text{avg_spd} \times t}{\text{avg_seg}}, 1\right) \times \frac{n_{mq}(8n - 4)}{n_{seg}} \tag{2.14}
\end{aligned}$$

If we set the length of T equal to the length of one time unit and equal to one minute, namely $T = t = 1\text{min}$, and multiply *com_cost(T)* by the total number of objects n_o , we can get the total number of messages in one time unit as

$$\begin{aligned}
& \text{total_com_cost} = \\
& \frac{2\text{avg_spd}}{\text{avg_seg}} n_o + 2\left(\frac{\text{avg_spd}}{\text{avg_seg}} + p\right) n_{mq} + \\
& \min\left(\frac{\text{avg_spd}}{\text{avg_seg}}, 1\right) \frac{8n - 4}{n_{seg}} n_o n_{mq} \tag{2.15}
\end{aligned}$$

From this equation, we learn that the total communication cost increases as the average speed increases, and decreases as the average segment length increases. The increases of the number of moving objects and moving queries also lead to the increase of the message cost. All of these observations have been demonstrated by our simulation studies.

We can compare with the results obtained from simulation studies. To ensure a fair comparison, we need to choose the parameters close to the ones specified in Section 2.6.1. We first set $n_{seg} = 4000$ and $p = 0.1$, which are the same numbers as used in the simulation. Secondly, we set $avg_seg = 2$ miles and $avg_r = 1.5$ miles, such that $n = 1$. Finally, we set $avg_spd = 0.3$ mile/min, which is close to the median of the speeds generated in the simulation (Recall that the speeds are in the range of $[0.1, v_{max}]$ miles/min following Zipf distribution with a deviation factor of 0.7, and the default v_{max} value is 1 mile/min).

After plugging these numbers, we get $total_com_cost$ as:

$$total_com_cost = 0.3 n_o + 0.5 n_{mq} + \frac{3}{20000} n_o n_{mq} \quad (2.16)$$

Fig. 2.17 shows the results based on the above equation when the number of moving objects are varied from 50000 to 100000. For the ease of comparison, the communication cost obtained using simulation as in Figure 2.8 are re-plotted here.

The plot shows that the trend obtained from analytical studies is similar to that from simulation studies, but the communication cost in simulation studies is higher, especially when there are 1000 queries. That is mainly because we treat all queries as range queries in the analytical studies. kNN queries incur higher communication cost, because other than query object, kNN object and (k-i)NN objects also need to report their activities to the server.

2.6.4 Client Workload

In the previous section, we study the total communication cost in the system by examining the total number of messages sent by the client and by the server. If we focus on one individual object, the object receives messages and sends out messages. These two activities are very expensive operations in terms of power consumption. Thus, we show the study on the power consumption in terms of number of messages in Fig. 2.18. Realizing that receiving

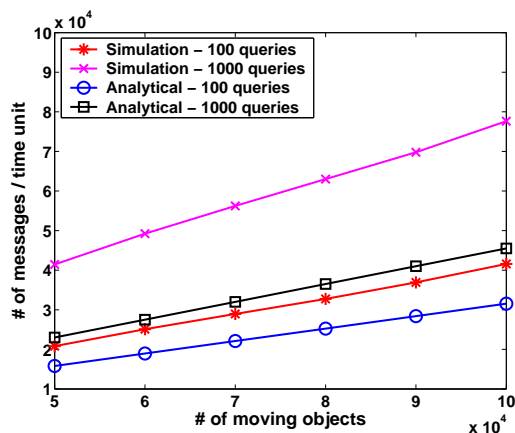


Figure 2.17: Effect of number of objects on communication cost based on analytical studies

a message only costs about half of the energy required for sending a message, we treat each received message as one half message. The plot shows that when we increase the number of queries, the proposed technique performs worse than the *QBO* and the naive method. This is mainly because if the number of queries is large, there are increased activities due to the query object movement, which leads to a lot of message received by data objects in the corresponding monitoring region.

We also study the average number of queries monitored per object, while varying the number of queries, as shown in Fig. 2.19. There are three curves in the plot, with each corresponding to a different maximum segment length. The figure shows that with the increase of the number of queries, more queries are required to be monitored on each object, which is as expected. Also, since an object needs to monitor queries whose monitoring regions intersect with its current segment, a longer segment means it has a higher probability to be covered by queries, which explains why there are an increased number of queries to be monitored when the maximum segment length increases.

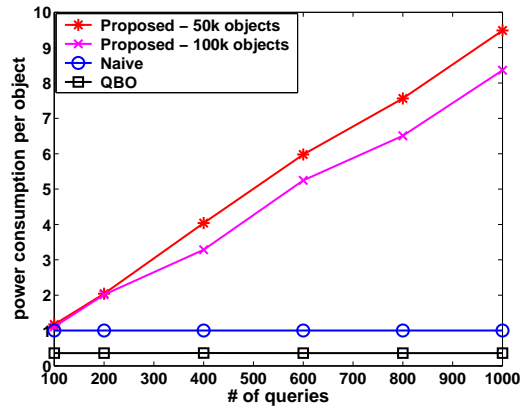


Figure 2.18: Effect of number of queries on power consumption

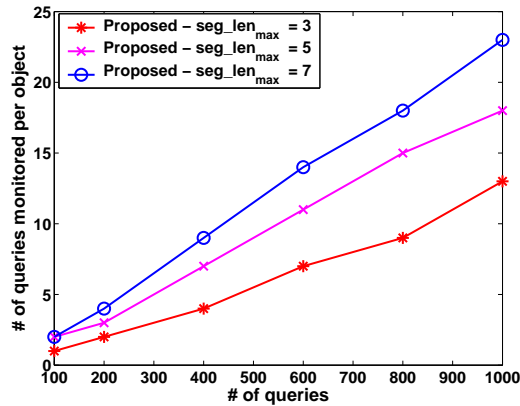


Figure 2.19: Effect of number of queries on the average number of queries monitored

2.7 Extension to Cover Mixed Environments

Till now, we always assume that mobile objects only move on road segments. However, in certain scenarios, for example, when a group of soldiers moves in a city to finish a mission, they may not always stick to a pre-defined road network and could break down barriers and walk on an open field. In this section, we discuss how to extend the proposed model to cover this new scenario, where mobile objects could move on both road segments and open space.

For the ease of discussion, we break the scenario into three different cases: data object moving off road segment but query object moving on road segment; data object moving

on road segment but query object moving off road segment; both data and query objects moving off road segment.

Let's look at the first case first, as shown in Fig. 2.20. Object A is a query object and moves on a road segment, while Object B is a data object and moves on an open space. To enable Object B to calculate its distance to Object A , we modify the communication protocol as follows: When a data object moves off a road segment, the object sends a message to the server (Similar to the Switch Segment Message). The server first determines the polygon (consisting of edges) which the object moves within, then sends the object the absolute positions of all the nodes of the polygon (Nodes n_1, n_2, n_7, n_8 for Object B). The road segment that the object moves off from is chosen as the *home edge* of the polygon (Edge n_1n_2). All the edge distances associated with the home edge are sent to the data object. In addition, the absolute positions of the home edge's two nodes (Node n_1 and n_2) are also sent to the data object.

On the mobile object side, at each time interval, it first computes its distances to the two nodes of the home edge (since the positions of these nodes are known), then uses a modified² Algorithm 2.3 to calculate its distance to the query object. In addition, the object needs to monitor its position against the polygon. Should it move out of the polygon, it sends a message to the sever and receives a new polygon, a new home edge, and a new set of edge distances. Note that when an object moves inside the polygon, we can't guarantee that the path of the shortest distance to the query object is always through the home edge's nodes. Nevertheless, we argue that the proposed method provides a good heuristic without adding significant computation cost to the server.

For the second case, as illustrated in Fig. 2.20, Object C is a query object moving on an open space, and Object D is a data object moving on a road segment. Similarly, we need to modify the communication protocol. When a query object moves off a segment, the server

²The modification is quite simple. Basically, $oPos$ and $(oeLength - oPos)$ should be replaced by the data object's distances to the home edge's nodes when appropriately.

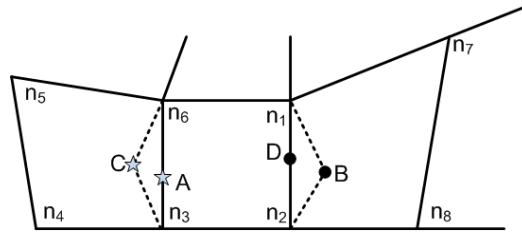


Figure 2.20: Mixed Environments

finds a polygon and a home edge for the query object (For Object C , the polygon consists of nodes n_3, n_4, n_5, n_6 , and the home edge is n_3n_6). Then the server needs to notify all data objects in the query’s monitoring region (calculated by assuming that the query object still moves on the home edge) of the query object’s position and velocity, along with the absolute positions of the home edge’s two nodes. With these information, a data object could first estimate the query object’s distances to the home edge’s two nodes, and use a modified³ Algorithm 2.3 to calculate its distance to the query object.

The third case is shown by query object C and data object B in Fig. 2.20. This is a simple combination of the first two cases and details are omitted.

2.8 Conclusion

In this chapter, we propose a framework to answer moving queries over moving objects in spatial environments. Our solution uses road segments as building blocks for monitoring regions of moving queries. Moving objects utilize their own computing power to help reduce server workload and save wireless bandwidth. We discuss how to use this framework to support two typical spatial queries: range queries and kNN queries. Detailed simulation results show that our method can save considerably in terms of server workload and has lower communication cost when there are light or medium amount of moving queries. When there are too many concurrent queries, our technique does incur some communication overhead

³The modification is quite simple. Basically, $qPos$ and $(eLength - qPos)$ should be replaced by the query object’s distances to the home edge’s nodes when appropriately.

when compared to the Query Blind Optimal (QBO) method. But considering the huge savings on the server side, we think our method is still favored. We also present an analytical study on communication cost, with the result matching the observations obtained from the simulation study.

CHAPTER 3: A HYBRID COMMUNICATION SOLUTION TO DISTRIBUTED MOVING QUERY MONITORING SYSTEMS

3.1 Introduction

Most moving object database systems employ a centralized approach [50, 52, 11, 74], in which one assumption is that the centralized server knows the updated locations of all moving objects. As a result, the focus of a centralized approach is to design efficient data structures and algorithms. The limitations of this approach are twofold. First, the server, with finite computation and communication capability, may not be able to cope with the high location-update frequency. Second, frequent location update can quickly drain the batteries of energy-constrained mobile devices.

One obvious solution to the aforementioned issues associated with the centralized approach is to reduce the location-update rate. However, this simple solution decreases the quality of the query results. To address the issue without sacrificing the query result's quality, some recent techniques have focused on distributed solutions [63, 17, 19]. The distributed framework proposed in Chapter 2 also belongs to this category. In a distributed solution, mobile objects monitor the query, and need to update the server only when the query result changes. This approach requires much fewer location updates to the server. Nevertheless, the server needs to supply relevant information, from time to time, to support the query monitoring processes at the mobile devices. When the number of mobile objects increases or the object mobility increases, this communication cost can become quite high. As a result, a more scalable communication technique between the server and mobile clients, suitable for distributed monitoring query, is desirable.

In a wireless environment, there are two ways of communicating between the server and clients. One way is on-demand access, in which clients and the server send messages to each other whenever necessary. Another way is using broadcast, where the server periodically

broadcasts data on an open wireless channel. Periodic broadcast is ideally suited for very popular data, while on-demand access is better for less demanded data.

In this chapter, we design a new distributed solution for moving monitoring queries on moving objects with a focus on an efficient communication technique between the server and mobile clients. At the heart of our proposed technique is a hybrid communication scheme leveraging both on-demand data access and periodic broadcast. The server sets aside a broadcast channel to repeatedly broadcast query information. The area of interest is mapped into grid cells, and when a mobile client moves from one cell into another cell, it tunes into the broadcast channel to download information on relevant queries instead of contacting the server for the information. To improve the tuning efficiency, we propose two indexing schemes for the broadcast technique. Our simulation results indicate that the proposed solution achieves from 30% to 60% savings in communication cost when compared to MobiEyes [17]. The reduction in communications also significantly improves energy conservation on the mobile devices.

The remainder of this chapter is organized as follows. We first discuss some related work in Section 3.2. In Section 3.3, we present an overview of the existing distributed system, which motivates us to employ a broadcasting technique to reduce the overall system communication cost. We discuss the two proposed indexing schemes in Section 3.4, and the query processing technique in Section 3.5. The simulation and analytical studies are presented in Section 3.6. In Section 3.7, we discuss a number of issues suggested by our results. Section 3.8 concludes this chapter.

3.2 Related Work

We first briefly discuss the use of a broadcast channel in mobile environment, then give an overview of existing solutions on monitoring queries on moving objects.

3.2.1 On Broadcast Channel

In a wireless environment, the mobile device’s limited battery power is a critical concern. Because sending a message consumes more energy than receiving a message, there are proposals for using broadcast channels to replace on-demand data requests to save energy. In the existing proposals [75, 42, 68, 69], the focus is on how to design an effective index to facilitate data access. Since a mobile object has the ability to switch between an active mode and a doze mode, with an effective index, it can first tune into the broadcast channel to get the predicted arrival time of the desired data, then goes back into doze mode, and returns to the active mode to download the data as it becomes available.

Cai et al. [7] propose a Hybrid Wireless Broadcast (HWB) model, where the broadcast server broadcasts data in two distinct channels: one broadcast main channel and the other relatively narrow-band on-demand subchannel; in the mean time, the base stations enable point-to-point communications among the server and moving objects. To address the long access latency problem with the use of broadcast channel, Ku et al. [41] suggest to leverage results cached in nearby moving objects to reduce access latency.

3.2.2 On Monitoring Queries over Moving Objects

Monitoring queries over moving objects have been studied extensively. One direction in this area is to reduce the server side workload by proposing efficient server side data structures and index. Yu et al. [74] propose two algorithms using grid indices to monitor k-nearest neighbor queries over moving objects: one is based on indexing objects, and the other on queries. Mouratidis et al. [50] propose Conceptual Partitioning Monitoring (CPM) algorithm for continuous nearest neighbor monitoring. CPM conceptually partitions the space around each query q , such that the nearest neighbor retrieval and the query result maintenance are only needed for objects close to q . Cho and Chung [11] propose UNICONS (a UNIQue CONTinuous Search algorithm) for Nearest Neighbor (NN) and Continuous Nearest Neighbor

(CNN) queries on road network. The algorithm use the precomputed nearest neighbor lists to facilitate the query processing.

Another direction is to use distributed computing to reduce both server side workload and communication cost, which has been covered in detail in Section 2.2. However, even with distributed computing, due to the mobility of mobile objects and mobile queries, these techniques still suffer from high communication cost. Noticing the benefits of using broadcast channel, in this chapter, we proposed to apply a hybrid communication model, which combines the use of on-demand access with broadcast channel, to answer the moving monitoring query.

3.3 Preliminary Model and Motivation

To better understand the motivation of the proposed solution, we first need to understand the limitations of the existing solutions. Therefore, in this section, we start with an overview of the existing work, which also serves as the foundation of the proposed work, then present the motivation of this work. In the next section, we will discuss the proposed solution in detail.

As introduced in Section 3.1, existing solutions can be divided into two categories: centralized solutions and distributed solutions. Since our technique is focused on reducing the communication cost for a distributed solution, we give a brief overview of the distributed approach here.

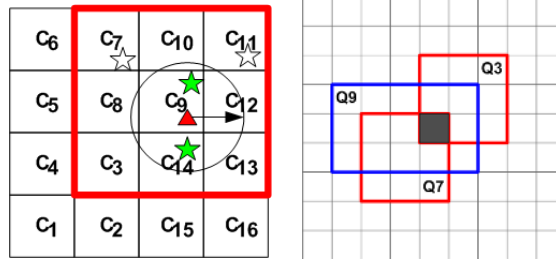
The common system architecture consists of a centralized server, a number of stationary base stations and a large population of moving objects. A moving object and the server communicate with each other through base stations, and each has a GPS-like device to determine its location and has some computing capability.

The terrain (the geographical area of interest) is a large square area, mapped into grid of cells of $\alpha \times \alpha$ square miles. Every moving object is assigned with a unique object ID. When

a moving object reports its location to the server, the message has the following format $\langle i, p_i, v_i, t_i \rangle$, where i is the object ID, p_i is the object's position, v_i is the object's velocity, and t_i is the time when the object's position and velocity are recorded.

A moving object can issue moving range queries to monitor its surroundings. An object that has issued a moving query is a *query object*; otherwise, it is a *data object*. A moving range query is modeled as a tuple $\langle qid, oid, range \rangle$, where qid is the query's ID, oid is the ID of the associated query object, and $range$ defines the search area around the query object. A query area can be a circle (specified by a radius) or a rectangle (specified by width and length). The result of a query is a set of identifiers of the moving objects currently residing in the query's region. An example is given in Fig. 3.1.a, where the terrain is divided into 16 cells, labeled according to the Hilbert curve order as C_1, C_2, \dots, C_{16} . A query object is drawn as a triangle, and the query region is the area covered by the circle. Two data objects (drawn as the stars) residing in that circle are included as the query result.

In a distributed solution, one important concept is the *monitoring region* of a moving range query. Given a query object and its current grid cell, the query region can overlap with several neighboring grid cells. For example, in Fig. 3.1.a, the query region (drawn as the circle) overlaps with $C_3, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}$, and C_{14} . The union of all grid cells this query region may overlap with as the query point moves inside its current grid cell is referred to as the *monitoring region* for the given query. In the distributed solution, all moving objects within the monitoring region of a query must monitor their distance to the query point, and update the query result by sending messages to the server if they fall within the query region (i.e., the distance is less than the range field specified in the query). To calculate the distance to a query object, a moving object first estimates the query object's current position using its knowledge of the query object's movement information. Therefore, each query object needs to report its significant velocity change to the server, and the server relays the change to affected moving objects to assure a good estimation of the query object's position.



3.1.a: Example of moving query 3.1.b: Example of monitoring region

Figure 3.1: Examples

Back to the example as shown in Fig. 3.1.a, as the query object moves within C_9 , all the nine grid cells in the upper right corner could overlap with the query region. Based on the definition of *monitoring region*, these cells form the monitoring region of the query. There are two data objects in cells C_7 and C_{11} , respectively. They are currently not part of the query result, but still need to monitor their distance to the query object. Note that the shape of a monitoring region is rectangular if the query region is a rectangle.

If a monitoring region of a query overlaps with a given grid cell, we say this query *intersects* the cell. When an object moves around, it could leave its current cell and enter a new cell. When this happens, the object needs to obtain a new set of queries intersecting with the new cell to continue the monitoring task. In Fig. 3.1.b, the monitoring regions of three queries Q_3 , Q_7 , and Q_9 are represented by the three rectangular areas. The shaded cell in the center intersects with all three of these queries. Any object moving into this shaded cell should monitor these three queries. Obviously, the cost associated with obtaining these relevant queries through on-demand access increases with more number of data objects or higher object mobility.

When information pertaining to these relevant queries is in high demand, periodic broadcast is a more effective communication method than on-demand access, because the cost of broadcasting is independent of the number of data objects as well as object mobility. As mentioned in Section 3.2.1, there have been proposals on how to utilize broadcast techniques

to answer queries issued from mobile objects. This inspires us to propose a hybrid communication technique to reduce the overall communication cost for the distributed query monitoring approach.

3.4 Broadcast Index Design

In existing research works [63, 17, 19], when an object moves to a new cell (or a new road segment), it contacts the server to request a new set of queries, which can lead to a large number of messages. In our hybrid solution, we continuously broadcast query information on a wireless channel. When an object needs to determine if relevant queries exist, instead of sending a request to the server, it just tunes into the wireless channel and downloads them. For other types of messages, such as velocity change and query result updates, the object still contacts the server directly through on-demand access.

To facilitate the downloading of queries from the broadcast channel, we need to have an effective broadcast index. Two performance metrics are typically used to evaluate an index: tuning time and access latency (time). Tuning time means the total time that a mobile object needs to stay in the active mode to get data, which includes the time spent on searching the index and downloading data. The access latency refers to the total time elapsed from the moment a mobile object tuning into the broadcast channel until the time the desired data has been obtained. One popular technique to reduce access latency is the $(1, m)$ interleaving technique [29], as shown in Fig. 3.2. In this technique, a complete index is broadcast preceding every $\frac{1}{m}$ fraction of the full broadcast cycle. By duplicating the index for m times, the waiting time to reach an index can be shortened, thus access latency is reduced. Please note that this technique is orthogonal to any proposed wireless index, and thus can be applied to any index.

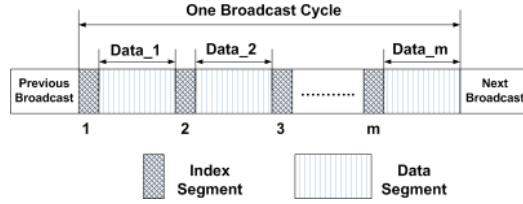


Figure 3.2: Layout for the Interleaving Technique

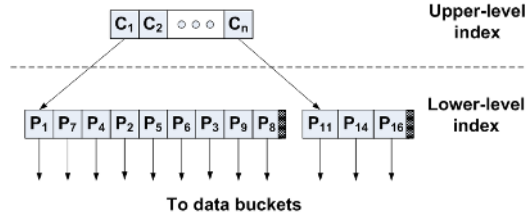


Figure 3.3: Grid Index structure

A good index should enable a short tuning time and incur little overhead in terms of access time. We proposed two indexes: the Grid Index and the Direction Index in the following sections.

3.4.1 Grid Index

The first proposed index is named Grid Index (GI), with the overall structure illustrated in Fig. 3.3. This index consists of two levels. The upper-level index is built on top of grid cells, which can be any type of index supporting a quick identification of the grid cell given a geographical location, for example, a quad-tree [16]. In our case, since the cell has a known fixed size, no tree index is needed. A simple mapping function is sufficient to determine the desired cell.

qid	Pos_x	Pos_y	Vel_x	Vel_y	Time	Range_x	Range_y
-----	-------	-------	-------	-------	------	---------	---------

Figure 3.4: Query data bucket structure

The low-level index consists of many blocks, with each block corresponding to a cell in the upper-level index. Inside each block, we store the pointers to all the queries intersecting with that corresponding cell. For example, two blocks are shown in Fig. 3.3. The first block stores all the pointers to the queries intersecting with the cell C_1 , where P_1, P_2 , etc. are the pointers to query Q_1, Q_2 , etc. At the end of each block, we put a special tag to indicate the end of that block. The data structure of a query data bucket is as depicted in Fig. 3.4, which stores a query object's movement information, including its position, velocity, the time when that position and velocity were reported, and the interested query range. If the interested query region happens to be a circle, only the *Range_x* attribute is used and the *Range_y* attribute is set to zero.

With the upper-level index, a moving object can map its location to the corresponding grid cell. Following the pointer in that cell, the object is directed to the lower-level index, where the object can download the pointers to the queries it should monitor. When it sees the tag for the end of a block, it stops downloading and enters doze mode. It returns to the active mode when the desired query data buckets arrive.

3.4.2 Direction Index

One drawback of the Grid Index scheme is that when a mobile object moves from one cell to another cell, it has to download from the broadcast channel a complete new set of queries that it should monitor in the new cell.

We observe that since the old cell and the new cell are adjacent, there are some queries required to be monitored by objects in both cells. When a mobile object moves from the old cell to the new cell, only a subset of the queries need to be downloaded. This inspires us to design an index that can facilitate the downloading of only the missing queries. Notice that a mobile object can move into a new cell from four directions: west, south, east, and north. We can classify the queries whose monitoring regions intersect with the new cell into

different types, such that for a specific direction from which the mobile object enters the new cell only certain types of queries need to be retrieved. An algorithm that categorizes such queries into nine types is given in Algorithm 3.1.

Algorithm 3.1 FindQueryType

FindQueryType[cell c]

```

1: Get all the queries intersecting with cell  $c$ 
2: for each query  $q$  intersecting with cell  $c$  do
3:   Get the monitoring region ( $MR$ ) of  $q$ 
4:   if  $c$  a west side boundary cell of  $MR$  then
5:     if  $c$  is the north-west corner cell of  $MR$  then
6:        $q.type = 1$ 
7:     else if  $c$  is the south-west corner cell of  $MR$  then
8:        $q.type = 3$ 
9:     else
10:       $q.type = 2$ 
11:    end if
12:   else if  $c$  is a south side boundary cell of  $MR$  then
13:     if  $c$  is the south-east corner cell of  $MR$  then
14:        $q.type = 5$ 
15:     else
16:        $q.type = 4$ 
17:     end if
18:   else if  $c$  is an east side boundary cell of  $MR$  then
19:     if  $c$  is the north-east corner cell of  $MR$  then
20:        $q.type = 7$ 
21:     else
22:        $q.type = 6$ 
23:     end if
24:   else if  $c$  is a north side boundary cell of  $MR$  then
25:      $q.type = 8$ 
26:   else {not a boundary cell}
27:      $q.type = 9$ 
28:   end if
29: end for

```

The algorithm checks the spatial relation between the input cell and a given query's monitoring region. In Fig. 3.5, we show examples for queries belonging to type 1 and 2, respectively. In Fig. 3.5.a, there are three queries: m_1 , m_2 , and m_3 , with their monitoring regions drawn as rectangles (or squares). As we can see, for the dark-shaded cell, all of

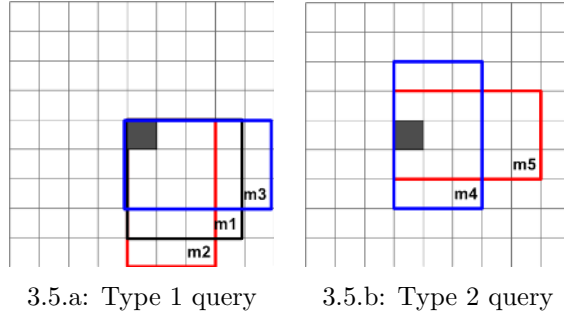


Figure 3.5: Examples for query types

three queries are classified as type 1 queries, and that is because the dark-shaded cell is the north-west corner cell for the queries’ monitoring regions. Similarly, in Fig. 3.5.b, the two queries: m_4 and m_5 , are type 2 queries for the dark-shaded cell.

After using Algorithm 3.1 to classify queries, we can achieve the following: queries belonging to type 1, 2, or 3 should be added for monitoring when an object moves into this cell from the west side; queries belonging to type 3, 4, or 5 should be added for monitoring when an object enters from the south side. Similarly, when an object enters from the east side, queries of type 5, 6, or 7 should be added; and queries of type 7, 8, or 1 should be added when an object enters from the north side. Fig. 3.6 illustrates the four scenarios when an object moves into a new cell from four different directions. In the diagram, $Q_1, Q_2, \dots,$ and Q_8 are examples of queries which belong to type 1, type 2, , and type 8, respectively. As we can see, in the Fig. 3.6.a, when an object moves from the lightly shaded cell (the old cell) into the darkly shaded cell (the new cell), only Q_1, Q_2 and Q_3 need to be added. Fig. 3.6.b, 3.6.c, and 3.6.d are examples for the other three scenarios.

Based on the above observations, we propose a three-level index structure, called Direction Index (DI). Compared to the Grid Index, we include one more level to represent the direction from which an object enters the new cell, as shown in Fig. 3.7. We use letters “W”, “S”, “E”, and “N” to represent the directions “West”, “South”, “East”, and “North”, respectively. Inside a “W” cell, there is a pointer pointing to the beginning of the list of

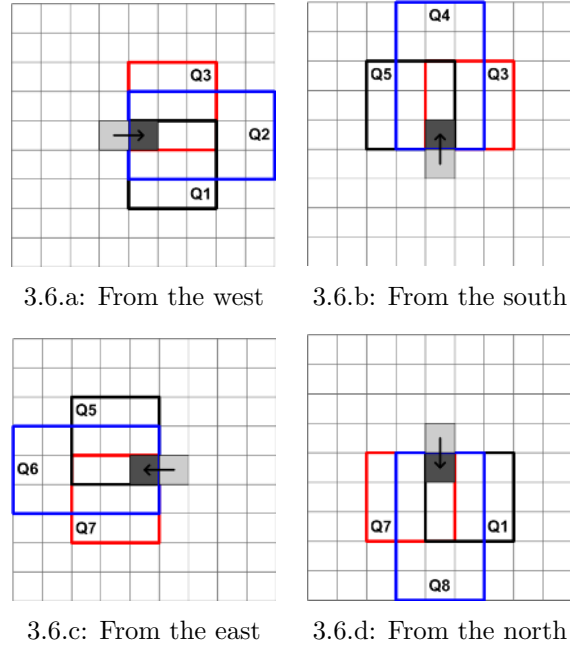


Figure 3.6: Scenarios when object changing cell

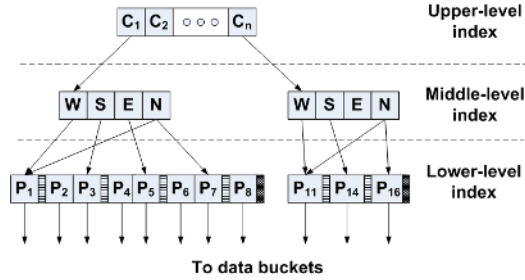


Figure 3.7: Direction Index structure

type 1 queries in the lower-level index. It is similar for the “S” and “E” cell. For an “N” cell, it is a little more complex. Since an object entering a new cell from the north needs to get queries of type 7, 8 and 1, we provide two pointers in an “N” cell, with the first one pointing to the beginning of type 1 queries, and the second one pointing to the beginning of type 7 queries. Besides the addition of the middle-level index, there are two differences in the lower level. The first difference is that there is no need to include pointers for type 9 queries because they are already in the object’s monitoring list when it is in the old cell. This feature reduces the index size, and leads to better access time and tuning time. The

second difference is that we need to sort the pointers using the associated query types, and use a tag to mark the end of types 1, 3, 5, and 7. This feature enables an object to know when to stop downloading the index packets.

As an example in Fig. 3.7, if an object is interested in downloading queries of types 1, 2, and 3, it first follows the pointer from the “W” cell and gets to the start of type 1 queries. It then downloads the queries until it recognizes the end tag of type 3 queries. As another example, if an object enters cell C_n from the east direction, since there is no link originated from the “E” cell, the object knows that there is no query that needs to be downloaded and it can reenter doze mode.

3.5 Data Structure and Query Processing

In the proposed solution, query processing is distributed among moving objects and the server. After a query is initiated, the server calculates the monitoring region for that query and notifies objects in that monitoring region to monitor this query. In this section, we first present the data structures which reside on the server and moving objects, and then discuss how the system handles various activities.

3.5.1 Data Structures

Three main data structures are used by the server. The Query Object Table stores the list of query objects and their parameters including velocity, position, and the time stamp when the velocity and position are recorded. The Server Query Table maintains the list of moving queries. Each query contains the query object’s ID, the specified range, and the monitoring region. Query result is also saved in this table. The third table is the Reverse Query Table, where all queries intersecting with each cell are saved. To facilitate the Direction Index, the query’s type is stored in this table.

On the client side, each client needs to maintain a Client Query Table to track all the queries it should monitor. In this table, we store for each query its ID, velocity, position, the time when the velocity and position were reported, and the specified range.

3.5.2 Query Installation

A moving query can be initiated by a moving object itself, or an administrator. In either case, the movement information on that query object, which includes the query object's ID, position, velocity, and the query's search range, should be submitted to the server. Upon receiving such a message, the server first updates the Query Object Table and the Server Query Table, then calculates the monitoring region for this new query and saves the result into the Server Query Table, and finally updates the Reverse Query Table. After installing the query on the server, the server forwards the query's information to all moving objects. Moving objects in the monitoring region save the message and start to monitor their distance to the query object. For objects outside of the query's monitoring region, they can simply ignore that message.

3.5.3 Changing Query Result Activity

For all queries in its monitoring list, an object needs to periodically predict the current positions of the query objects using the saved velocity, time, and position information in the Client Query Table. It then calculates its distance to the query object to determine if it is in the query's range. If the result is different from the previous result computed in the last time unit, the object sends a message to request the server to either adding it to the query result or removing it from the query result. After receiving the message from the moving object, the server locates the query from the Server Query Table, and updates the result accordingly.

3.5.4 Changing Velocity Activity

Query objects need to report their updated velocities to the server if there are significant changes. Once the server receives an updated velocity, it broadcasts a message with the updated velocity information to moving objects in that query’s monitoring region. Upon receiving the message, a moving object first updates its Client Query Table, and then applies the updated velocity to calculate the distance under monitoring. Please note that for a data object, velocity change does not entail any communication message.

3.5.5 Changing Grid Cell Activity

When an object moves from one cell to another cell, the object first needs to determine which queries should not be monitored anymore. With the saved query’s information, an object computes its minimum distance to the query object given that the object itself is moving within the new cell. If the minimum distance is greater than the specified query range, that query is dropped from the monitoring list. Also, the moving object needs to get new queries to monitor. In existing solutions, such as MobiEyes [17], moving objects always send messages to the server to request new queries. In our system, with the help of a broadcast channel containing query information, moving objects do not need to send messages to the server anymore. Instead, they just tune into the broadcast channel and download the necessary new queries. We study the effectiveness of the two proposed indexing schemes in Section 3.6.

When the moving object changing cells happens to be a query object, it still needs to send a message to the server. In response, the server performs the following three tasks. First, it updates the tables on the server, computes a new monitoring region for the query, and updates the corresponding data bucket in the broadcast channel. Second, the server broadcasts a message to all moving objects residing in the old monitoring region to stop

monitoring this query. Finally, the server notifies all moving objects in the new monitoring region to monitor this query.

3.5.6 Communication between Server and Objects

As we have discussed in Section 3.4, objects need to switch to doze mode when they are waiting for query data buckets to arrive. However, objects have to stay awake to receive broadcast messages from the server, as suggested in Section 3.5.4 and 3.5.5. This presents a dilemma. To solve this problem, we can make use of synchronized time between the server and moving objects [59, 38]. With synchronized time, the server sends out messages to moving objects only at pre-scheduled time slots, on the other hand, moving objects just need to wake up periodically during doze mode at these pre-scheduled time slots. If a moving object does not receive anything from the server when it wakes up, it goes back to sleep and wakes up again at the next pre-scheduled wake up time.

3.6 Performance Study

We implement a simulator in Java to evaluate the performance of the proposed solution. The system in the simulation consists of a base station, a broadcast channel, and a number of moving objects. The available bandwidth for the broadcast channel is set to 100 Kbps. The packet size is varied from 64 bytes to 1024 bytes. In each packet, two bytes are used for the packet ID. Two bytes are allocated to a pointer. One coordinate is represented with eight bytes. The size of a query data bucket is set to 50 bytes (to hold the query ID, position, velocity, time, and range, as illustrated in Fig. 3.4). In the broadcast channel, queries are ordered using the Hilbert curve order. For each query, we first identify the cell where the query's query object is located, then calculate the Hilbert curve value for that cell, and use that value for ordering.

3.6.1 Simulation Setup

Our simulation is set up as follows. The area of interest is a square-shaped region of 64×64 square miles. The whole region is divided into grid cells, where each cell is a square of $\alpha \times \alpha$ square miles. Moving objects are randomly generated and placed in the region. The velocities are in the range of $[0.1, v_{max}]$ mile per time unit, with random directions, following a Zipf distribution with a deviation of 0.7. In the simulation, v_{max} is varied in the range of $[1, 2]$ miles per time unit (equivalent to $[60, 120]$ miles per hour). Some of the moving objects are randomly selected as query objects. The query regions have circular shape with a radius randomly chosen in the range of $[1, r_{max}]$ miles, where r_{max} is an integer in the range of $[1, 9]$ miles. At each time unit, 10% of the moving objects change their velocities. The threshold for changing velocity is set as 0.1 mile per time unit. We run each simulation setting 10 times with different seeds and compute the average as the final simulation results. Each simulation lasts for 200 time units. In the experiments, we vary different parameters to study the performance. The parameters are listed in Table 3.1. If not otherwise stated, the experiment takes the default values.

Table 3.1: Simulation Parameters

Parameter	Description	Value Range (or Set)	Default
α	Cell size (mile)	1, 2, 4, 8, 16	2
n_o	Number of moving objects	[2000, 10000]	10000
n_{mq}	Number of queries	[200, 1000]	200
$size_{packet}$	Packet size (byte)	64,128,256,512, 1024	64
v_{max}	Maximum velocity (mile/min)	[1, 2]	1
r_{max}	Maximum length of query radius (mile)	[1, 9]	3
p	PCT of objects changing speed / time unit	[2, 50]	10

In the remainder of this section, we first determine a good cell size for the simulation study. We then compare our system with MobiEyes in terms of communication cost, followed by the evaluation of the proposed indexing schemes. Finally, we discuss the cost of the broadcast channel and the query latency issue.

3.6.2 Selecting Cell Size

One important parameter in our system is the cell size α . An optimal α value should reduce the number of messages as much as possible. To make sure that the comparison on communication cost to MobiEyes is fair, we try to determine a good α value for MobiEyes.

Fig. 3.8 shows that as the value of α increases, fewer messages are communicated. This can be explained as follows. When the cell size is increased, there are fewer messages resulting from objects changing cells. We note that when the whole terrain consists of only one cell, the number of messages drops to the minimum. However, in this situation, all objects need to monitor all queries in the system, which increases the computation workload on the client side considerably. To study this effect, we use the number of queries monitored by each moving object as a metric to gauge the level of computation workload. Fig. 3.9 shows the average number of queries monitored by each moving object with the increases in cell size. As we can see, the curve increases sharply as expected. To find a good α value, we compute the product of the number of messages and the number of queries monitored, and then plot the curve against the cell size as shown in Fig. 3.10. This figure shows that the product is at a minimum when the cell size is smallest (equal to 1). Since the curve increases only slightly when the cell size is set to 2, we select “ $\alpha = 2$ ” as a good setting for MobiEyes. This value gives a good balance between communication cost and computation cost. We note that this α value is selected to give MobiEyes the best performance in our simulation study. It is not a universally good value.

3.6.3 Communication Cost

The communication cost of the system is measured in the number of messages among the server and moving objects. Our system is compared against MobiEyes, where an on-demand communication mechanism is used between the server and moving objects. We study the sensitivity of our system on a number of parameters.

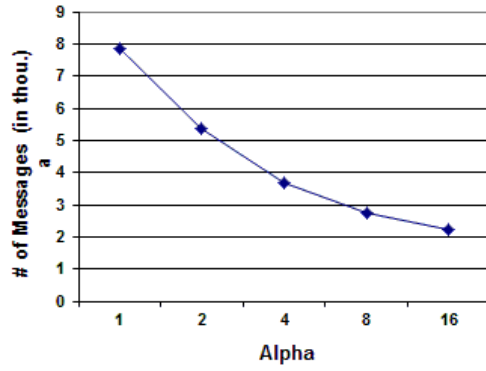


Figure 3.8: Effect of cell size on number of messages

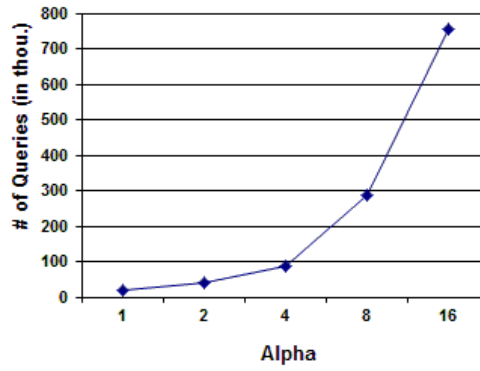


Figure 3.9: Effect of cell size on number of queries monitored

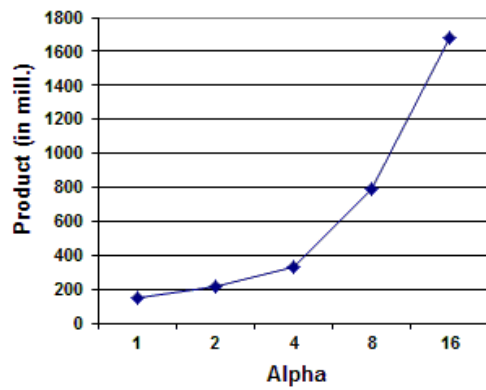


Figure 3.10: Effect of cell size on the product of number of messages and number of queries monitored vs. α

In Fig. 3.11, we vary the number of moving objects from 2000 to 10000, and measure the average number of messages per time unit. The figure shows that our system requires less than half the number of messages required by MobiEyes, which equals about 60% in savings in terms of communication cost. The reason is that there are a lot of messages due to changing-cell activities in MobiEyes. In contrast, in our system, unless the object that changes cell is a query object, the changing-cell activity does not entail any message.

In Fig. 3.12, we study the effect of the number of queries on communication cost. As we can see, when the number of queries is increased from 200 to 1000, more messages are needed. This is reasonable since more queries mean more updates for query result change and query object velocity change, which require more messages. Our method saves about 60% of the messages when the number of queries is small, and still saves about 30% when the number of queries is large.

In Fig. 3.13, the percentage of objects changing velocity is varied from 2% to 50% in each simulation time unit. The plot shows that when the percentage increases from 2% to 10%, the number of messages is only increased slightly, however, if the percentage keeps increasing, many more messages are necessary. This can be explained as follows. A query object needs to report to the server when its velocity changes, and in turn the server has to notify all affected moving objects. When few objects change velocities (i.e., the percentage is small), only a very small number of them belong to query objects. That is why the number of messages only increases very slightly. But when the percentage is large, quite a few query objects' velocities are changed, which lead to a lot of communication messages. Moreover, from this plot, we also learn that our solution incurs much less communication cost than MobiEyes.

In Fig. 3.14, we increase the maximum velocity for moving objects from 1 mile per minute to 2 miles per minutes. The results show that with the increase of the maximum velocity, more messages are needed for both techniques and our technique always demands much fewer messages than MobiEyes. This is expected since with higher velocities, objects

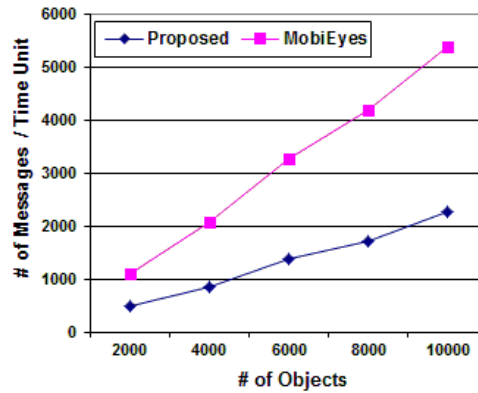


Figure 3.11: Effect of number of objects on communication cost

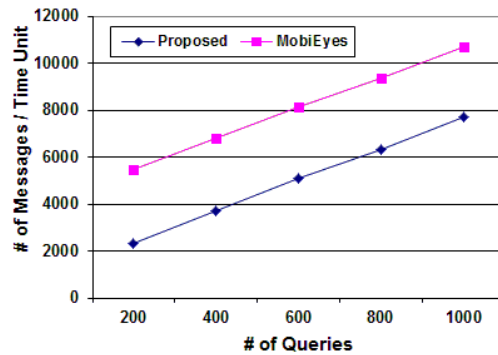


Figure 3.12: Effect of number of queries on communication cost

will have higher probability switching cells, as well as entering and leaving query regions. All these activities lead to more communication messages.

Fig. 3.15 studies the effect of maximum query radius on the communication cost. The plot shows that more messages are needed for both techniques when the maximum query radius is increased from 1 mile to 9 miles. This is because with a larger radius, more objects are affected by the query region, which consequently incurs higher communication cost due to the increased number of entering/exiting query region events.

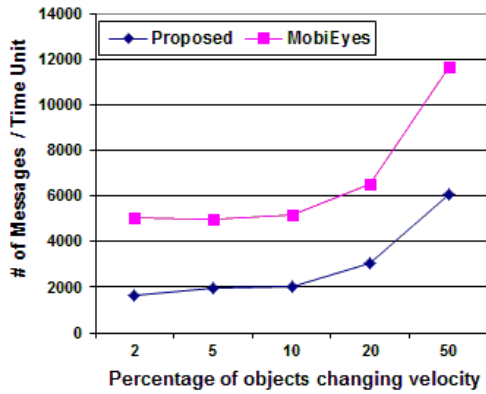


Figure 3.13: Effect of percentage of objects changing velocity on communication cost

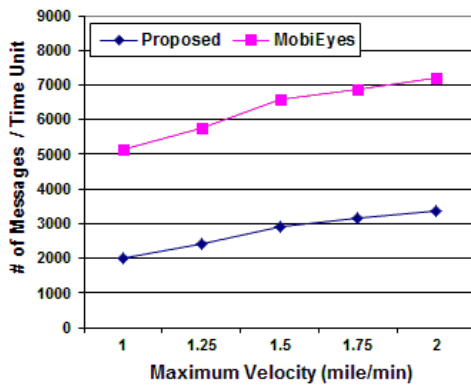


Figure 3.14: Effect of maximum velocity on communication cost

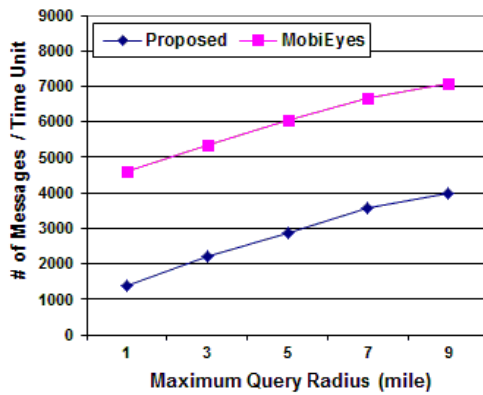


Figure 3.15: Effect of maximum query radius on communication cost

3.6.3.1 Analytical Study on Communication Cost

We first analyze the communication cost of the existing system (which does not use broadcast channel), then present the analytical study on the communication cost for the proposed system.

Let $com_cost(T)$ denote the communication cost (measured in the number of messages) for an object during a given time period of T . The communication cost consists of the following three components: $cell_switch_cost$, the cost associated with object switching cells; vel_change_cost , the cost associated with query object changing velocity; and $result_update_cost$, the cost for updating query result when an object enters or exits some query's region. Denote α as the cell size, n_o as the number of objects, n_{mq} as the number of moving queries, avg_r as the average radius of query's range ¹, avg_v as the average velocity of an object, t as the basic time unit that an object needs to wait before contacting the server, p as the percentage of an object changing velocity at each time unit, $prob_change_result$ as the probability of an object entering or exiting some query's region at each time unit, avg_q_mon as the average number of queries to be monitored per object. (We will discuss how to estimate the last two variables in detail later.) Then $com_cost(T)$ can be described as follows:

$$\begin{aligned}
com_cost(T) = & \\
& \frac{T}{\frac{\alpha}{avg_v}} \times cell_switch_cost + \\
& \frac{T}{t} \times p \times \frac{n_{mq}}{n_o} \times vel_change_cost + \\
& \frac{T}{t} \times avg_q_mon \times prob_change_result \times result_update_cost \tag{3.1}
\end{aligned}$$

The expression before $cell_switch_cost$ estimates the number of times for a given object to switch cells during the time interval T . Since the vel_change_cost only applies to query object, the expression before vel_change_cost gives the probability that a given object is a

¹For simplicity, we only consider queries defined by a radius.

query object times the percentage of an object changing velocity during the time interval T . The expression before $result_update_cost$ estimates the number of times an object affecting some query result during the time interval T . Also, we have:

$$\begin{aligned} cell_switch_cost &= 1 + 1 + \frac{n_{mq}}{n_o} \times 2 \\ &= 2 + \frac{2n_{mq}}{n_o} \end{aligned} \quad (3.2)$$

$$vel_change_cost = 1 + 1 = 2 \quad (3.3)$$

$$result_update_cost = 1 \quad (3.4)$$

The equation for $cell_switch_cost$ can be explained as follows. When an object switches cell, first it needs to send one message to the server to request for a new set of queries, and the server sends one message back to the object. The third component means that, should the object be a query object (with probability of $\frac{n_{mq}}{n_o}$), the server needs to broadcast two messages, one message to notify objects in the old monitoring region to stop monitoring this query, and another message to notify objects in the new monitoring region to start monitoring this query. The vel_change_cost only applies to query object, and is equal to 2. Because when a query object changes velocity, it first sends a message to the sever, then the server broadcasts a message to notify the affected moving objects. Moreover, the $result_update_cost$ is 1, since an object only needs to send one message to the server to update the query result.

To estimate avg_q_mon , we need to know the average number of cells covered in a query's monitoring region. Denote

$$n = \lceil avg_r/\alpha \rceil \quad (3.5)$$

The total number of cells covered by this query's monitoring region can be approximated² as $(2n+1)^2$. Assume that the queries are uniformly distributed, and denote the total number of cells as n_{cell} , the average number of queries monitored by an object per cell is

$$avg_q_mon = \frac{n_{mq} (2n + 1)^2}{n_{cell}} \quad (3.6)$$

To estimate *prob_change_result* is more challenging. Objects can change query results by either exiting some query's region or entering some query's region. In the first event, when an object is inside a query's region, we assume the object could be in any location inside the query's region. After t time, the object moves a distance of $avg_v \times t$. To simplify, we only consider two extreme cases: object moving in the same direction or the opposite direction with query object. If the object moves in the same direction as the query object, this object remains in the query's region after t given that they have the same velocity. Should the object move in the opposite direction as the query object, the distance between the object and the query object after t is $(2 avg_v \times t)$. Therefore, for all moving objects spreading in the query's region (at the width of $2avg_r$), if $2avg_v \times t \geq 2avg_r$, all objects exit the query's region; otherwise, only a ratio of the objects, that is, $\frac{2avg_v \times t}{2avg_r}$ exits the query's region. Based on the two extreme cases, the probability that an object moves out a query region can be approximated as follows (where the min function means the minimum of the two input numbers):

$$\frac{1}{2} \min\left(\frac{2 avg_v \times t}{2avg_r}, 1\right) + \frac{1}{2} \times 0 = \min\left(\frac{avg_v \times t}{2avg_r}, \frac{1}{2}\right) \quad (3.7)$$

²This value is only an approximation and serves as a loose upper bound. To obtain the exact number is very complex, so we argue that the approximated value is sufficient for our analytical study. The approximation is done as follows: assume the monitoring region and cells are all projected onto an one-dimension line, then the monitoring region could possibly reach $(2n+1)$ cells while the query object moves within its cell. Therefore, in a two-dimension space, the monitoring region could reach $(2n+1)^2$ cells potentially.

Similarly, if the object is outside of a query's region, we first assume that the object is within α distance from the query's region. Then if the distance traveled by an object in time t is greater than α , and provided that the object is moving toward the query object, the object becomes part of the query result. Therefore, the probability that an object will move into a query's region is:

$$\frac{1}{2} \min\left(\frac{2 \text{avg}_v \times t}{\alpha}, 1\right) + \frac{1}{2} \times 0 = \min\left(\frac{\text{avg}_v \times t}{\alpha}, \frac{1}{2}\right) \quad (3.8)$$

Since all objects move randomly, it is fair to assume the two events (entering a query's region and exiting a query's region) have equal probability. Then for each query an object is monitoring, the probability of changing that query's result, *prob_change_result*, is:

$$\begin{aligned} & \frac{1}{2} \min\left(\frac{\text{avg}_v \times t}{2 \text{avg}_r}, \frac{1}{2}\right) + \frac{1}{2} \min\left(\frac{\text{avg}_v \times t}{\alpha}, \frac{1}{2}\right) \\ & = \min\left(\frac{\text{avg}_v \times t}{4 \text{avg}_r}, \frac{1}{4}\right) + \min\left(\frac{\text{avg}_v \times t}{2\alpha}, \frac{1}{4}\right) \end{aligned} \quad (3.9)$$

Finally, we put all these together to get *com_cost*(T) as:

$$\begin{aligned} & \text{com_cost}(T) = \\ & \frac{T}{\frac{\alpha}{\text{avg}_v}} \left(2 + \frac{2 n_{mq}}{n_o}\right) + \frac{T}{t} \times \frac{2 p n_{mq}}{n_o} + \\ & \frac{T}{t} \left(\min\left(\frac{\text{avg}_v \times t}{4 \text{avg}_r}, \frac{1}{4}\right) + \min\left(\frac{\text{avg}_v \times t}{2\alpha}, \frac{1}{4}\right)\right) \times \\ & \frac{n_{mq} (2n + 1)^2}{n_{cell}} \end{aligned} \quad (3.10)$$

If we set the length of T equal to the length of one time unit and equal to one, namely $T = t = 1$ minute, and multiply *com_cost*(T) by the total number of objects n_o , we can get

the total number of messages in one time unit as

$$\begin{aligned}
total_com_cost = & \\
& \frac{2\ avg_v}{\alpha} (n_o + n_{mq}) + 2p n_{mq} + \\
& (\min(\frac{avg_v}{4\ avg_r}, \frac{1}{4}) + \min(\frac{avg_v}{2\alpha}, \frac{1}{4})) \times \\
& \frac{(2n + 1)^2}{n_{cell}} n_o n_{mq} \tag{3.11}
\end{aligned}$$

In this equation, the first component is due to the cell switching activities, with the first subcomponent contributed by objects requesting new queries from the server. This part of communication cost is linearly associated with the number of moving objects. As mentioned before, a typical moving object system usually has a large number of moving clients, which implies that a major portion of communication cost is from the first component. In the proposed hybrid solution, due to the addition of a broadcast channel, that part is eliminated. Thus, the required total number of messages in one time unit for the proposed solution is

$$\begin{aligned}
total_com_cost_proposed = & \\
& \frac{2\ avg_v\ n_{mq}}{\alpha} + 2p n_{mq} + \\
& (\min(\frac{avg_v}{4\ avg_r}, \frac{1}{4}) + \min(\frac{avg_v}{2\alpha}, \frac{1}{4})) \times \\
& \frac{(2n + 1)^2}{n_{cell}} n_o n_{mq} \tag{3.12}
\end{aligned}$$

The above two equations show that the total communication cost increases as the average moving velocity increases, and decreases as the cell size increases. The increases of the number of moving objects and the number of moving queries also lead to a higher message cost. To cross-validate the correctness of the above analysis, we can compare the results obtained with analysis to those from simulation studies. To ensure a fair comparison, we need to choose the parameters close to the ones specified in Section 3.6.1. We first set $\alpha = 2$

miles and $p = 10\%$, same as those used in the simulation. Secondly, since the range radius is chosen from $[1, 3]$ miles, we set $avg_r = 2$ miles, then consequently $n = 1$. Finally, avg_spd is set to 0.3 mile/min, close to the median of the numbers generated in simulation (Recall that the speeds are in the range of $[0.1, v_{max}]$ miles/min following Zipf distribution with a deviation factor of 0.7, and the default v_{max} value is 1 mile per minute). These two equations become

$$total_com_cost = 0.3 n_o + 0.5 n_{mq} + \frac{1.05}{1024} n_o n_{mq} \quad (3.13)$$

$$total_com_cost_proposed = 0.5 n_{mq} + \frac{1.05}{1024} n_o n_{mq} \quad (3.14)$$

Using the above two equations, we first vary the number of objects from 2000 to 10000 while keeping the number of moving queries at 200, and compute the communication cost for the two techniques respectively. The results are shown in Fig. 3.16, which is very close to the results obtained by simulation study. (Also plotted in the figure for easy comparison.). Similarly, the number of moving queries is varied from 200 to 1000 while keeping the number of moving object at 10000 to get Fig. 3.17. The plot shows the results obtained from analytical study are similar to those of simulation study in terms of trend, but when the number of queries is high, the estimated communication cost from analytical study is much higher. This is because Equation 3.6 only gives an upper bound estimation, when the number of queries increases, the effect is magnified. (See the last term in Equation 3.11 and Equation 3.12, respectively.)

3.6.4 Comparison of Proposed Indexing Schemes

The Grid Index (GI) and the Direction Index (DI) are compared using access time and tuning time. We also include a scheme without using any index, called the No Index (NI).

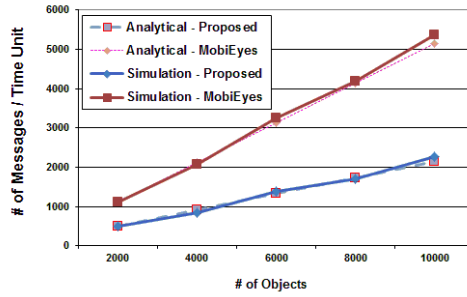


Figure 3.16: Effect of number of objects on communication cost based on analytical study

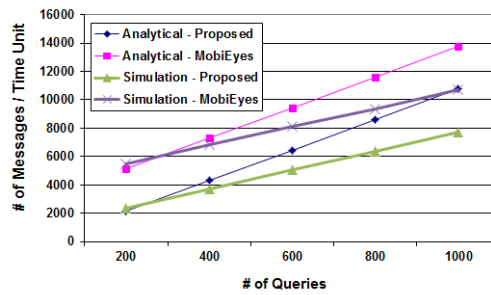


Figure 3.17: Effect of number of queries on communication cost based on analytical study

The No Index scheme only broadcasts data packets; clients have to download all the data packets to determine which one to keep or discard. Since NI does not use any index, it has the minimal access time, and thus serves as a good baseline for comparison. On the other hand, the tuning time for NI is prohibitively long, so we do not include it when measuring tuning time.

In Fig. 3.18, we vary the number of objects. The figure shows that the access time increases linearly when the number of objects increases. This is expected because the access time is measured as an average per time unit. When there are more moving objects, the access time for all objects adds up linearly. The figure also shows that both GI and DI need longer access time compared to the No Index scheme, but the difference is not very big. The access times with GI and with DI are about 1.5 times and 2 times that of NI, respectively.

As discussed in Section 3.4, the lower-level index in DI does not include type 9 queries, which leads to a smaller lower-level index when compared to GI. However, because of the

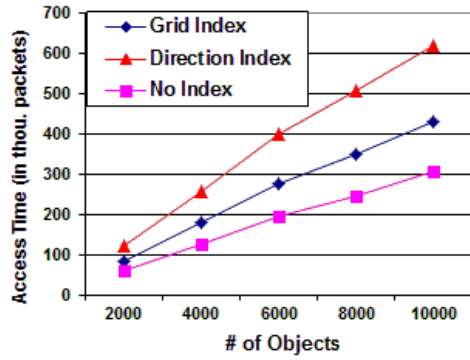


Figure 3.18: Effect of number of objects on access time

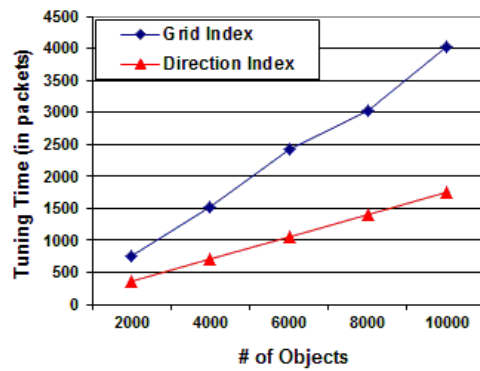


Figure 3.19: Effect of number of objects on tuning time

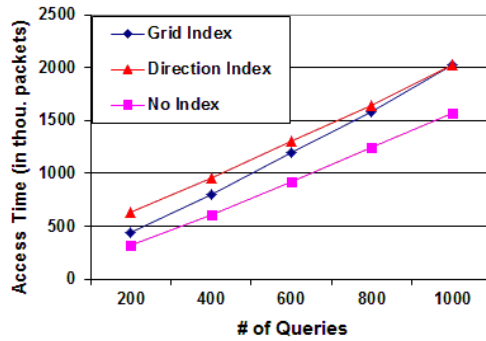


Figure 3.20: Effect of number of queries on access time

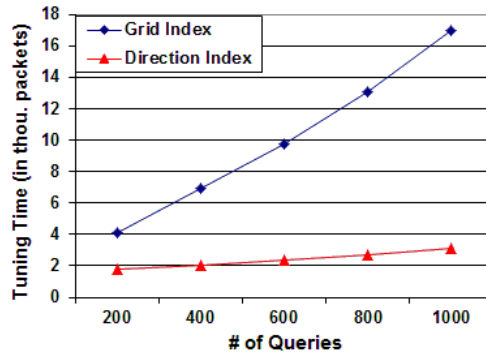


Figure 3.21: Effect of number of queries on tuning time

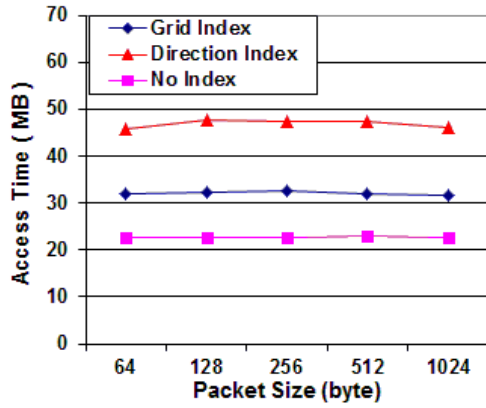


Figure 3.22: Effect of packet size on access time

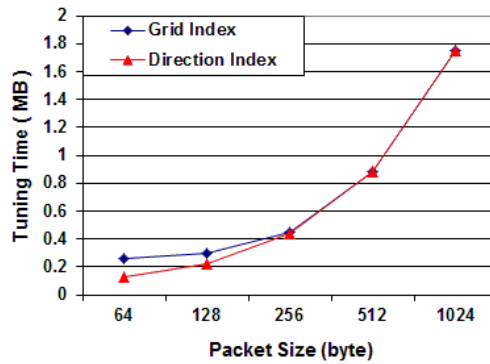


Figure 3.23: Effect of packet size on tuning time

middle-level index, the total size of DI is still larger than the total size of GI, which explains why DI needs longer access time.

We also study the tuning time of the two indexes when varying the number of objects. As shown in Fig. 3.19, the tuning times of both indexing schemes increase linearly with the number of objects, while the GI experiences a steeper slope. This is due to the fact that an object needs to download more queries in GI.

In Fig. 3.20 and 3.21, we study the effect of varying the number of queries. Fig. 3.20 shows the result on access time. When the number of queries is still small, GI is better than DI in terms of access time. However, when the number of queries increases, GI gradually loses its advantage over DI. This can be explained as follows. A larger number of queries implies that more queries are classified into type 9 queries, which leads to an even smaller lower-level index when compared to the GI. This effect is more evident as the number of queries increases. Eventually, the size of DI is smaller than the size of GI. That is exactly why DI demands less access time when the system has a large number of queries.

In Fig. 3.21, we can see the tuning time for GI is very sensitive to the number of queries, while the tuning time for DI only increases slowly with the increase of the number of queries. This is because when there are more queries, there are on average more queries to be monitored for each cell. This translates into more queries to be downloaded in GI when an object switches cell. But for DI, this problem is not very severe since only a subset of the monitoring queries for a cell needs to be downloaded, explaining why the curve for DI only increases slowly.

For a system using broadcast channels, packet size plays an important role in performance. We study the effect of packet size by varying it from 64 bytes to 1024 bytes. The results are presented in Fig. 3.22 and 3.23. Fig. 3.22 shows that the access times for three indexes only change slightly when packet size varies. This is because that access time is dominated by the total length of one broadcast cycle, and the effect of packet size on the total cycle length is almost negligible.

Fig. 3.23 demonstrates that the tuning times for both indexes increase quickly with the increase in packet size. When the packet size is the smallest, the system has the shortest tuning time. This is expected due to the overhead associated with using larger packets over smaller packets. One interesting finding is that DI requires shorter tuning time than GI when the packet size is small; but the tuning times for the two indexes converge when the packet size becomes larger. That is because more queries can be fit in a larger packet; and when an object downloads one packet, it downloads more queries than it actually needs. This cancels the advantage of using DI.

In conclusion, when there are a large number of queries and/or objects, DI is a better index than GI, because DI requires much shorter tuning time, and only demands slightly longer or even shorter access time.

3.7 Discussions

We first analyze the communication cost by considering the cost of the broadcast channel, then discuss the query retrieval latency when mobile objects retrieve data from the broadcast channel.

3.7.1 Communication Cost Revisited

In Section 3.6.3, we compared our solution against MobiEyes using the total number of messages communicated. However, the comparison does not take into account the extra broadcast channel used in our proposed technique. Here, we start with analyzing the cost of having an extra channel, and then demonstrate that our solution is still superior to MobiEyes.

Assume that in the system, the total available wireless bandwidth is B , and it can be divided into c channels, with each channel of bandwidth $\frac{B}{c}$. In our solution, one channel is dedicated to the broadcast channel, and the remaining $(c - 1)$ channels are used for on-demand communications. Using Equations 3.13 and 3.14, we can calculate Normalized

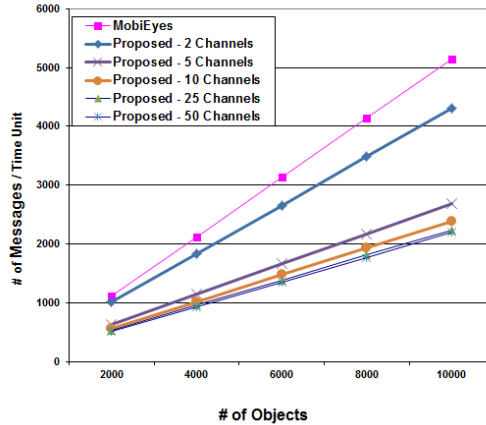


Figure 3.24: Effect of total number of channels on Normalized Communication Cost

Communication Cost (NCC), defined as the total number of messages communicated as if there were c on-demand channels, while keeping the throughput of each channel the same. For MobiEyes, NCC is just the total number of messages obtained in Section 3.6.3.1, since all the c channels are used for on-demand communications. However, for our proposed solution, NCC is equal to the total number of messages multiplied by $\frac{c}{c-1}$. Using NCC ensures that the comparison is based on the same number of channels. In Fig. 3.24, we re-plot Fig. 3.16 using NCC. The plot shows that when there are only two channels, the proposed solution is still better than MobiEyes, even though the saving is only about 10%. When the number of channels increases, especially after over 25 channels, the saving starts to stabilize, and the plot is very similar to that in Figure 3.16. Since in reality, a cellular cell typically has up to 50 channels available [15], using the proposed solution can significantly reduce communication cost.

3.7.2 Query Retrieval Latency

As mentioned in Section 3.3, when an object moves to a new cell, it needs to tune into the broadcast channel to fetch a new set of queries to monitor. However, the object may not

be able to retrieve new queries right away. In this section, we first estimate the length of a broadcast cycle, then discuss its effect on query retrieval latency.

We choose to analyze the Grid Index due to its simplicity. In the Grid Index, the index consists of two levels: the upper level and the lower level. Assume the size of a cell id as $length_c$ bytes, the size of a pointer to a query as $length_q$ bytes, and the query size in a data block as $length_{data}$, we have

$$\begin{aligned}
 Cycle_length = & \\
 & n_{cell} \times length_c + n_{mq} (2n + 1)^2 \times length_q + \\
 & n_{mq} \times length_{data} \tag{3.15}
 \end{aligned}$$

In this equation, the first component indicates the total length of the upper level index. The second component shows the total length of the lower level index, because on average, each cell intersects with the monitoring regions of $\frac{n_{mq}(2n+1)^2}{n_{cell}}$ queries, as discussed in Section 3.6.3.1. The last component corresponds to data buckets, which store detailed information for queries.

Let $length_c = 2$ bytes, $length_q = 2$ bytes, $length_{data} = 50$ bytes, $n_{cell} = 1024$ bytes, and $n = 1$, by varying the number of queries from 200 to 1000 we get the plot in Fig. 3.25. The cycle length increases as the number of queries increases. For example, when there are 1000 queries, the size of a full cycle is about 100 KB. The approximated access time (measured in KB, roughly equal to half of the broadcast cycle) is 50 KB. If the broadcast channel's bandwidth is 100 Kbps, then the average access latency is 4 seconds. This should be acceptable in reality. Moreover, 100 Kbps is a conservative estimation. Typical broadcast bandwidth in practice could be higher than this value [30].

As we learn from performance studies, our solution is more scalable than the MobiEyes approach, especially when the number of queries is high or when there are a large number of objects constantly moving around. Nevertheless, if the mobility of objects is limited, or if the

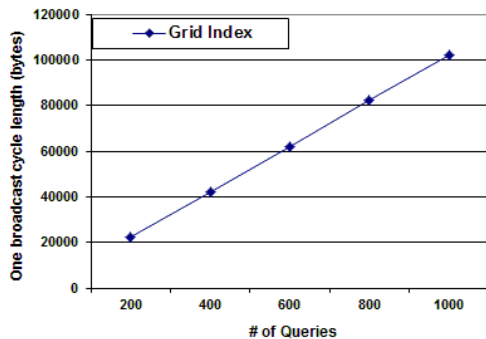


Figure 3.25: Effect of number of queries on one broadcast cycle length based on analytical study

number of objects is small, then there are only a few cell-switching activities. In this case, solutions using only on-demand communications, such as MobiEyes, are more effective.

3.8 Conclusion

In a moving query monitoring system, one typical solution is to use distributed processing to reduce the server-side computation bottleneck. However, a distributed approach usually relies heavily on the on-demand communications between server and client, leading to considerably high communication cost. In this chapter, we propose a hybrid solution, which combines on-demand communications and periodic broadcasting to reduce the overall system communication cost. Two indexing schemes: the Grid Index and the Direction Index are designed specifically to facilitate the query downloading process from the broadcast channel. The simulation results indicate that the proposed hybrid solution can achieve 30% to 60% savings over MobiEyes, a typical distributed approach. The finding is also verified by analytical analysis. Furthermore, the two proposed indexing schemes are compared using simulation studies. The results show that both schemes have pros and cons, while the Direction Index is better when there are a large number of queries and/or objects.

In conclusion, our approach is more scalable than the MobiEyes approach, and thus very desirable for medium-to-large-scale deployments. On the other hand, if the number

of queries to be monitored is small, or there are only a small number of moving objects, MobiEyes or other similar distributed solutions using only on-demand communications can also be applied directly. The moving query monitoring system discussed in this chapter is quite general and has a wide range of applications in mobile commerce. We believe that our result will help future applications to decide which solution to choose in practice.

CHAPTER 4: PROTECTING USER PRIVACY BETTER WITH QUERY L-DIVERSITY

4.1 Introduction

Location-based queries typically consist of a location, usually the location of the query issuer, and a question. There are a lot of challenges to answer this type of query for mobile users. One of the challenges is how to protect user privacy. As we can see, to get a query answered, one has to provide a location as well as a query question to the service provider, which raises a privacy concern if the service provider is not trustworthy.

Many researches have been done to address this challenge [23, 18, 48, 12, 5, 70, 71]. Most existing solutions assume a three-tier architecture, in which mobile users first send the location and query information to a *trusted* anonymizer server, then the anonymizer server performs some cloaking procedure to enlarge the query's location into a region, finally forwards that region to a service provider. Typically, the goal of this cloaking procedure is to enforce location k -anonymity. That is, the cloaked region must contain at least $(k - 1)$ other users, such that an adversary can only link the cloaked query to the actual query issuer with $\frac{1}{k}$ probability. To protect against a *query sampling attack* [12], techniques have been proposed to ensure that all users included in the same cloaked region must report this region as their cloaked region.

Enforcing k -anonymity alone is not sufficient to ensure privacy. Let us consider a scenario, in which all users from a cloaked region are interested in the same type of service such as the location of a special club. In this case, even an adversary cannot link an individual query back to a specific user, it is still known to the adversary that all the users in the cloaked region have inquired about that special club. While this example depicts an extreme case, in reality, it is not uncommon that users from the same cloaked region request only a limited number of services. Consequently, an adversary can still infer that some user has issued

a query on a certain service with a high probability. This kind of attack is referred to as *query homogeneity attack* [66], and renders the existing k -anonymity model vulnerable. To counter this kind of attack, we modify the l -diversity concept [46], originally proposed for the relational database domain, and apply it in LBS domain to protect query contents. The key idea is to ensure that for all queries sharing the same cloaked region, their query contents must be different enough, such that the probability of linking a query to its original issuer is less than some pre-defined threshold.

In this chapter, we first formally define the problem, and then propose two cloaking techniques that can counter against query homogeneity attack. Both of these techniques first divide the whole terrain into grid cells. Their space partitioning schemes, however, are different. The first technique starts from the center cell, and gradually expands over the space in all directions in search for a good way to partition the space. In contrast, the second technique first maps the two-dimensional grid space into a one-dimensional line of grid cells using a space filling curve, and then sequentially scans these cells to find the best partitioning strategy. We will describe these techniques in details later, and give simulation results to show that they are significantly better than the improved Interval Cloak technique [23].

The contributions we make in this chapter can be summarized as follows:

- To the best of our knowledge, we are the first to use the l -diversity concept to address the query homogeneity attack.
- We consider a new anonymization criteria: $\langle k, l \rangle$ -sharing region, and propose two cloaking techniques to partition the space using this new criteria.
- We conduct extensive simulation studies to evaluate the proposed techniques.

The remainder of this chapter is organized as follows. We first discuss related work in Section 4.2. The preliminary and some definitions are then presented in Section 4.3 to

facilitate further discussion. The two proposed cloaking techniques are introduced in Section 4.4, followed by the simulation study in Section 4.5. Finally, we conclude this chapter in Section 4.6.

4.2 Related Work

In this section, we discuss the two concepts: k -anonymity and l -diversity in relational databases and their applications in location-based services.

4.2.1 k -anonymity and l -diversity in relational databases

In a relational database, to publish data (such as censor or medical data) to support third-party data mining applications, it is important to prevent an adversary from linking the published data back to an individual. One obvious solution is to remove the *identifiers* such as a person's name and social security number for each published record. However, this is not enough since there still exist so-called *quansi-identifiers* such as age, height, zip code, etc., which can be used to infer a person's identify. To address this problem, the k -anonymity concept is proposed in [60], [56]. The key idea is to make a record indistinguishable from other $(k - 1)$ records with the same set of quansi-identifier. All the records sharing the same set of quansi-identifier form an *anonymization set*, and the size of this set should be larger than or equal to k . There are mainly two techniques in achieving k -anonymity: through *suppression* or *generalization*. Suppression means to remove the quansi-identifiers, while generalization is to replace the quansi-identifiers with more general terms, for example, replacing a person's age with a range.

Recently, it is pointed out in [46] that maintaining k -anonymity alone is not sufficient. In each anonymization set, the number of distinct sensitive values is more important than the size of the set, namely, k . To address this concern, a new notion called l -diversity is proposed in [46], which requires each distinct sensitive value in an anonymization set to be

well represented; and two different metrics are introduced to measure the representativeness. This scheme uses an algorithm based on the technique used in [43] to generate anonymization sets meeting both k -anonymity and l -diversity properties. More recently, a linear algorithm is introduced in [65] to generate anonymization sets meeting the l -diversity requirement. In [21], the multidimensional quansi-identifiers are mapped into one dimension to solve the k -anonymity and l -diversity problems. In [22], the authors solve the privacy problem for datasets with high dimensions.

4.2.2 k -anonymity and l -diversity in location-based services

Location k -anonymity has been studied quite extensively in the location-based services community. The idea is similar to that in relational database. Given a location-based query, there is a trusted anonymizer server to cloak the location of the query issuer into a region, with the requirement that there are at least $(k - 1)$ other users in that cloaked region.

Most of existing researches focus on designing an efficient cloaking algorithm to achieve location k -anonymity. The Interval Cloak technique, based on the quad tree structure [16], is proposed in [23]. Given a query, the algorithm recursively divides the area into quadrants, and checks the quadrant where the query is located to see if the quadrant contains more than k users. If it does not contain at least k users, the quadrant's parent is used as the cloaked region. In [18], the Clique Cloak algorithm is presented, in which a set of users are combined to form a graph clique, and these users form an anonymization set. In [48], a system called Casper is introduced, which uses a pyramid data structure to quickly find a cloaking box. In [12], the distinction between location k -anonymity and query k -anonymity is made, where the latter is to assure that the cloaked region for a query should also be shared by other $(k - 1)$ queries as their cloaked region. In [5], grid based approaches are

investigated to achieve location k -anonymity and location l -diversity, where the location l -diversity is proposed to ensure that the query issuer cannot be identified from l different physical locations (such as buildings and postal addresses).

The aforementioned techniques focus on how to cloak a query location, but ignore the protection of query content. The latter was considered recently in [66], in which queries are divided into two types: *sensitive* and *insensitive*. For an anonymization set, which includes both sensitive and insensitive queries, the percentage of sensitive queries shall not exceed a certain threshold to protect privacy. A Partition-Enumeration tree (PE-tree) structure is proposed in [66] to facilitate the cloaking process. This technique has a few drawbacks. First, since the PE-tree has to store each mobile user’s location, given that location updates are frequent in typical location-based applications, the maintenance cost for the tree could be very high. Second, queries are divided into sensitive and insensitive types, but how to define a query as a sensitive one is quite subjective. A sensitive query to one user might be deemed as insensitive to another user.

4.3 Preliminaries

In this section, we first introduce the system architecture, then go over some properties that are important for privacy-preserving applications, including the novel $\langle k, l \rangle$ -sharing region property, finally, we present the goal for anonymization techniques.

4.3.1 System Architecture

We consider a system consisting of a large number of mobile users, a trusted anonymizer server, and one or more service providers, as illustrated in Fig. 4.1. Mobile users send location-based queries to the anonymizer server using an authenticated and encrypted wireless connection. An LBS query typically includes a user ID, the location of the user, a time stamp, and service-specific information. After receiving a query from a mobile user, the

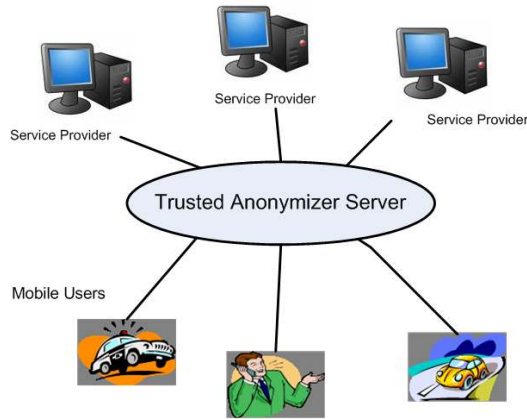


Figure 4.1: System Architecture

anonymizer server first decrypts the message, then performs a cloaking operation, which typically consists of the following two steps: (1) replace the user ID with a pseudo-identifier, (2) cloak the query location into a region. Finally, the anonymizer server forwards the cloaked (anonymized) query without its exact location (i.e. with the enlarged region only) to the selected service provider. Upon receiving an anonymized query, the service provider answers the query and sends the result back to the anonymizer server. The anonymizer server then refines the result and forwards the refined result to the mobile user.

4.3.2 Privacy-preserving Properties

However, one obvious problem with the above approach is that if the query issuer is the only user residing in the cloaked region, an adversary can easily link the query to the query issuer, and makes the effort to protect query's actual location useless. To fix this problem, the following property is necessary for a cloaked region.

Property 1. (k -anonymity region) This property requires that for a cloaked region, it must contain at least k different mobile users. Therefore, the adversary can only link the cloaked query to the query issuer with probability $\frac{1}{k}$.

Nevertheless, as pointed out in [12], when the locations of mobile users are revealed to adversaries, even with the k -anonymity region property, the adversary can still use *query sampling attack* to link a cloaked query back to the query issuer. To counter this type of attack, [12] identified the following property.

Property 2. (k -sharing region) This property states that the cloaked region contains at least k users, and this region is also reported by at least k of these users as their cloaked region.

One special case with this property is that for a cloaked region with more than k users, all users report this region as their cloaked region. As we can see, the k -sharing region is a stronger requirement than the k -anonymity region. Even when all users' locations are known to an adversary, since there are at least k different queries originated from the same cloaked region, the adversary still only has $\frac{1}{k}$ probability to link a query back to the query issuer. Based on the analysis presented in paper [12], among existing techniques, only the techniques proposed in [32] and [18] have the k -sharing region property.

However, if the number of distinct services requested by users residing in the same k -sharing region are small, in this scenario, even though a query can not be linked back to a specific user, the adversary can deduce with high probability that some users are interested in some service. This type of attack is called *query homogeneity attack*. To counter this type of attack, we need to assure that among the queries submitted by users from the same k -sharing region, the requested services are different enough.

In other words, for a group of queries sharing the same cloaked region, the group should possess enough query diversity to stand query homogeneity attack. However, how to define and measure query diversity becomes a challenge. Inspired by the fact that yellow page companies divide different businesses into different categories, we also divide different services into categories.

Definition 4.10 (Category) *The services requested by LBS queries are classified into different categories according to the point of interest, such as restaurant, hospital, bar, and gas station.*

We assume that the number of categories available in the system is pre-known. In the rest of this chapter, when we refer to query category, we mean the category of the service requested by a query. With the definition for category, we have the following definition for query entropy.

Definition 4.11 (Query Entropy) *Given a set of queries $\{q\}$, with each query q associated with a query category c , then the percentage that a query category c_i is requested can be computed as $p_i = \frac{|\{q|q.c=c_i\}|}{|\{q\}|}$. Then query entropy is calculated as:*

$$\text{query entropy} = - \sum p_i \log p_i. \quad (4.1)$$

Property 3. (Query l -diversity) This property means that for a set of queries $\{q\}$, given an integer l , the query entropy is equal to or greater than $\log l$.

Finally, we define a new property, called $\langle k, l \rangle$ -sharing region, which takes k -anonymity and query l -diversity into consideration.

Property 4. ($\langle k, l \rangle$ -sharing region) This property is more restrictive than the k -sharing region property. Besides the restrictions imposed by the k -sharing region, this property also requires that among all queries submitted by users from this region, they must possess the query l -diversity property.

4.3.3 Anonymization Goal

In the system, we assume that each mobile user has a privacy profile, which specifies the requirement on privacy. The profile is a two-tuple $\langle k, l \rangle$, as explained in the following definition. When a user sends a query to the trusted anonymizer server, he/she also includes his/her privacy profile in the request.

Definition 4.12 (*Personalized Privacy Query*) *A personalized query sent from a user to the trusted anonymization server has the following format: $\langle id, \langle k, l \rangle, \langle t, x, y \rangle, C \rangle$, in which the id is the unique id assigned to the user, $\langle k, l \rangle$ is the user's privacy profile, with k as the number of users required in the k -sharing region, while l indicating the requirement for query l -diversity, $\langle t, x, y \rangle$ specifies the time and location where the query is issued, and C is the content of the query (with the type of service specified).*

After the anonymizer server receives a personalized query, it performs the cloaking operation to obtain a cloaked query, and sends the cloaked query to the service provider.

Definition 4.13 (*Cloaked Query*) *A cloaked query has the following format: $\langle id', t, MBR, C \rangle$, where id' is the pseudo-identifier for the query, t is the time stamp, MBR is minimal bounding rectangle for the cloaked region, and C is the content of the query.*

The goal of an anonymization algorithm is to find regions with the $\langle k, l \rangle$ -sharing region property, and the identified region satisfies the $\langle k, l \rangle$ requirements for all users in the region. Formally, denote a personalized query as q , and its cloaked version as q' , and the whole set of cloaked queries at any given time t as Q , and the goal of an anonymization algorithm is to find a Q' for each q , such that:

$$(Q' \subset Q) \wedge (\forall \{q'_i, q'_j\} \in Q', q'_i.MBR = q'_j.MBR) \wedge (\forall q' \in Q', (|Q'| \geq q.k \wedge Q'.entropy \geq \log q.l))$$

4.4 Cloaking Algorithms

We start this section by discussing data structure used on the anonymizer server, then present the two proposed cloaking algorithms.

4.4.1 Data Structure

The whole terrain is divided into grid cells, where each cell is a square with size α . For each query received at every time unit, the anonymizer server records the query information, including the query issuer's id, location, $\langle k, l \rangle$ profile, and content. Also, for each cell, we keep the following aggregated variables for queries belonging to the cell: (1) A counter n to keep the total number of queries at current time unit. (2). A variable k_{spec} to store the maximum of all k 's, among all query's privacy profile. (3). Similarly, a variable l_{spec} to store the maximum of all l 's. (4). A signature, as described below.

Definition 4.14 (Signature) *Each cell keeps a bit vector, called signature, to indicate category information for all the queries in the cell. Recall that for each submitted query, from the content C , we can map the query into a certain category. Assume that the maximum number of category available is cat_{max} , then the vector is a cat_{max} bit vector. For each cell, if it covers a query with category i , the vector's i th bit is set to 1, otherwise, it is set to 0. Given a signature sig , we refer to the number of one's in sig as its cardinality, denoted as $sig.card$.*

Definition 4.15 (Signature Union) *A union of two signatures is defined as the bit union for the two bit vectors of the two signatures.*

In other words, if the i th bit of either signature is 1, the resultant bit vector also has 1 at the i th bit; if both signatures have 0 at the i th bit, then the resultant bit vector has 0 at that bit. Note that a union of two signatures is still a signature. This union operation is

useful when multiple cells are merged into one area, in order to obtain the signature for the combined area, we can simply compute the union of the signatures.

Note that for a given area, if the cardinality of the area’s signature is equal to l , the maximum query entropy of this area is obtained only when the distribution of query categories are uniform, and the maximum is $\log l$. Therefore, to test if an area meets the query l -diversity requirement, we can compare the cardinality with l first. Only if the cardinality is greater than or equal to l , there is a need to calculate the area’s entropy.

4.4.2 Expand Cloak

The first proposed cloaking algorithm, named as Expand Cloak, is described in Algorithm 4.1. The algorithm starts with a *for* loop to examine every cell in the area. The examine order can be linear, starting from the bottom-left cell and moving to right and top, or be spiral, from the center of the area. We prefer the latter, noticing that usually more mobile users are around the center area than the boundary area.

First, we check if the cell is already cloaked (lines 2-4). If yes, the algorithm moves to the next cell. Then we initialize the *area* variable using the cell c (line 5). We call the *CheckAreaValidity* algorithm as shown in Algorithm 4.2, with the details to be discussed later. If the area meets its privacy requirement, we then create a new cloaked region for this area/cell, and label this area/cell as cloaked (lines 6-10), then move to the next cell. Otherwise, we start to check the area’s neighbor cells. If all neighbor cells are already cloaked or none of the neighbor cell contains a query, we give up in cloaking this area (lines 13-15). Then we examine each of the area’s neighbor cell by calling the *CheckAreaValidity* algorithm (lines 17-22). If after a neighbor cell is unioned with the area, the updated area meets the privacy requirement, that neighbor cell is saved into the *result*. If the *result* is not empty (line 23), we pick the cell with the fewest queries to form a new cloaked region with the *area*

Algorithm 4.1 Expand Cloak

```
1: for each cell  $c$  do
2:   if  $c$  is already cloaked then
3:     Continue;
4:   end if
5:    $area \leftarrow \{c\}$ ; {Initialization}
6:   if CheckAreaValidity( $area$ ) then
7:     Create a cloak region for  $area$ ;
8:     Label  $area$  as cloaked;
9:     Continue;
10:  end if
11:  while  $area$  is not cloaked do
12:     $NC \leftarrow area$ 's neighbor cells that are not cloaked;
13:    if  $NC == \emptyset \parallel \forall c_i \in NC, c_i.n == 0$  then
14:      Continue; {Do not cloak this area}
15:    end if
16:     $result \leftarrow \emptyset$ ; {To store candidate neighbor cells}
17:    for each cell  $c_i$  in  $NC$  do
18:       $tmp\_area \leftarrow area \cup c_i$ ;
19:      if CheckAreaValidity( $tmp\_area$ ) then
20:         $result \leftarrow result \cup c_i$ ;
21:      end if
22:    end for
23:    if  $result \neq \emptyset$  then
24:       $c_p \leftarrow$  the cell in  $result$  with the fewest queries;
25:      Create a cloak region for  $area \cup c_p$ ;
26:      Label  $area$  and  $c_p$  as cloaked;
27:    else
28:       $c_p \leftarrow$  the cell in  $NC$  with the most queries;
29:       $area \leftarrow area \cup c_p$ 
30:    end if
31:  end while
32: end for
```

Algorithm 4.2 Check Area Validity

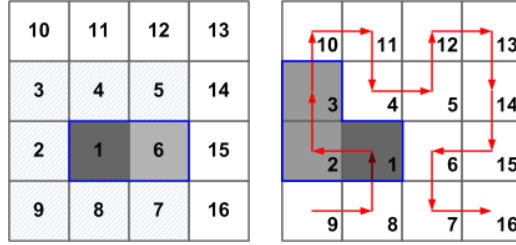
CheckAreaValidity(*area*)**Require:** an area consists of one or more cells**Ensure:** true or false

```
1:  $k_{act} \leftarrow 0$ ; {actual number of queries for the area}
2:  $k_{spec} \leftarrow 0$ ; {specified maximum  $k$  for the area}
3:  $sig \leftarrow \emptyset$ ; {signature}
4:  $l_{spec} \leftarrow 0$ ; {specified maximum  $l$  for the area}
5: for each cell  $c$  in  $area$  do
6:    $k_{act} \leftarrow k_{act} + c.n$ ;
7:    $k_{spec} \leftarrow \max(c.k_{spec}, k_{spec})$ ;
8:    $sig \leftarrow sig \cup c.signature$ ;
9:    $l_{spec} \leftarrow \max(c.l_{spec}, l_{spec})$ ;
10: end for
11: if  $k_{act} \geq k_{spec}$  then
12:   if  $sig.card \geq l_{spec}$  then
13:      $entropy \leftarrow$  calculate the area's entropy;
14:     if  $entropy \geq \log l_{spec}$  then
15:       return true;
16:     end if
17:   end if
18: end if
19: return false;
```

(lines 24-26), otherwise, the cell with the most queries is picked to form an updated *area* (lines 28-29), and the while loop is executed again.

The input to Algorithm 4.2 is an area consisting of multiple cells. The goal is to examine if the input area can meet the privacy requirement. First, we initialize some variables (lines 1-4). Then, we iterate through each cell in the *area* to update these variables (lines 5-10). After that, we check if the area meets the k -anonymity requirement, if yes, then we use the cardinality of the signature union to check if the area could potentially meet the l -diversity requirement. Finally, we compute the actual entropy for the area and compare it with the specified entropy to determine if the area meets both privacy requirements (lines 11-18).

We demonstrate the Expand Cloak algorithm using an example. Fig. 4.2.a shows the space divided into 16 cells, with each cell assigned a cell ID. The data stored in cells from



4.2.a: Expand Cloak 4.2.b: Hilbert Cloak

Figure 4.2: Illustrations for Expand Cloak and Hilbert Cloak

Table 4.1: Data in Cells

Cell ID	n	k_{spec}	l_{spec}	signature	query count
1	3	6	3	0011	{0, 0, 2, 1}
2	3	3	2	0011	{0, 0, 1, 2}
3	2	3	2	1100	{1, 1, 0, 0}
4	4	3	2	1100	{3, 1, 0, 0}
5	0	0	0	0000	{0, 0, 0, 0}
6	3	4	2	1100	{2, 1, 0, 0}
7	1	4	3	0001	{0, 0, 0, 1}
8	0	0	0	0000	{0, 0, 0, 0}
9	0	0	0	0000	{0, 0, 0, 0}

1 to 9 are detailed in Table 4.1, where n , k_{spec} , l_{spec} , and *signature* have the values as specified in Section 4.4.1. For the ease of explanation, we assume there are only four types of query categories, therefore, each *signature* is a four-bit vector. The *query count* indicates the number of queries for each query category. For example, Cell 1 has 3 queries, with the specified k_{spec} and l_{spec} as 6 and 3, respectively. There are two queries belonging to the third category and one query belonging to the fourth category, as a result, the *signature* is 0011.

Suppose we want to cloak Cell 1 (the dark shaded cell in the middle). First, we check if the cell by itself can meet the privacy requirement. Since there are only 3 queries in Cell 1, and less than the k_{spec} , expansion is needed. The algorithm checks all the surrounding eight cells and filters out cells 3, 5, 7, 8, 9 out immediately due to the small number of queries. In the next step, the algorithm checks the cardinality of the signature unions, which then eliminates Cell 2 because the signature union shows that there are two types of query categories and can not meet the l -diversity privacy requirement. Till now, the cells left

are Cell 4 and Cell 6. After we calculate the entropies for the two combined area {Cell 1, Cell 4}, and {Cell 1, Cell 6}, the results show that both combined areas satisfy the privacy requirement. Then, the algorithm decides to merge Cell 6 with Cell 1 to form a new cloaked region, since Cell 6 has fewer queries. Fig. 4.2.a shows Cell 1 and Cell 6 form a new cloaked region, all the other examined cells are decorated with stripes.

For the Expand Cloak technique, there are two scenarios when queries from certain cell can not be successfully cloaked. The first scenario is that all neighbor cells are empty, so the algorithm considers that the cell (or area) is an isolated cell (or area) and gives up on it. The second scenario is that the cell is very close to the boundary of the whole space, which also makes it difficult to find suitable neighbor cells to merge with. In Section 4.5, we use a metrics called *Relative Success Rate* to measure the percentage of these two scenarios occurring.

4.4.3 Hilbert Cloak

The proposed Expand Cloak technique starts from one cell, and expands to both horizontal and vertical directions. Because all neighbor cells must be checked and evaluated before the algorithm can make a decision, the computation cost could be high. To remedy this problem, we propose to use space filling curve to design a cloaking algorithm. Hilbert curve is a type of space filling curve. In Fig. 4.2.b, we show a 4×4 hilbert curve, which can map a two dimensional space into one dimension, thus save us from expanding toward two directions.

Here we present the second proposed cloaking algorithm, Hilbert Cloak, as shown in Algorithm 4.3. Hilbert Cloak first sorts the cells based on its Hilbert curve value (line 1). Then, for each cell, the algorithm checks if it is already cloaked or empty (lines 3-5). If not, it examines the cell to see if the cell by itself meets the privacy requirement. If yes, a cloaked region is created and the cell is labeled as cloaked (lines 8-12). If not, the next cell

(based on Hilbert Curve order) is included into the *area* (lines 13-14), then the updated *area* is checked again using the while loop until a satisfying region is found.

We use the same example from the previous section to illustrate Hilbert Cloak algorithm. The algorithm starts from Cell 9. Based on Table 4.1, both Cells 9 and 8 are empty and thus skipped. The next cell is Cell 1, which does not meet the privacy requirement by itself. Then the next cell, Cell 2, is merged with Cell 1 to form an *area*. However, the signature union check indicates that the merged area is still not good enough. Then, the next cell, Cell 3, is added into the *area*, which passes the validity test. Therefore, Cells 1, 2, and 3 form a cloaked region, shown in Fig. 4.2.b as shaded cells.

Algorithm 4.3 Hilbert Cloak

```

1: Sort the cells using Hilbert Curve values;
2: for each cell  $c$  do
3:   if  $c$  is already cloaked  $\|c.n == 0$  then
4:     Continue;
5:   end if
6:    $area \leftarrow \{c\}$ ; {Initialization}
7:   while  $area$  is not cloaked do
8:     if CheckAreaValidity( $area$ ) then
9:       Create a cloaked region for  $area$ ;
10:      Label  $area$  as cloaked;
11:      Break;
12:     end if
13:      $c_n \leftarrow$  the next cell;
14:      $area \leftarrow area \cup c_n$ ;
15:   end while
16: end for

```

4.4.4 Discussions

The degree of query l -diversity is measured using the entropy concept, which has been pointed out as being too restrictive [46]. Note that since the proposed techniques are orthogonal to the method used to define diversity, our techniques can work with other methods with straightforward modifications.

For k -anonymity in relational databases, it has been shown in [47] that the optimal k -anonymization problem of relations is NP -hard. The problem studied in this chapter, involving one more parameter l , therefore, is also very difficult. The two proposed techniques can not guarantee that the optimal solutions can be found, nevertheless, as demonstrated by the simulation study, the performances are acceptable in practice.

For queries that can not be cloaked in the Expand Cloak technique, typically due to the lack of neighboring queries, we can insert some dummy queries to help cloak the queries [34]. As what to be shown in the simulation study, the percentage of queries not being cloaked is quite low, which means the number of dummy queries needs to be generated is also small.

4.5 Performance Study

Simulation studies are used to study the effectiveness and scalability of the proposed cloaking techniques. First, we introduce the performance metrics, then cover the experimental setup, finally, we present the detailed simulation results.

4.5.1 Performance Metrics

The first measure is *Query Anonymization Time* (QAT), which measures the run-time efficiency of a cloaking technique. However, the QAT only shows how fast a technique runs, to evaluate the quality of the generated cloaked regions, other performance metrics are called for. The following metrics are employed to gauge the anonymization quality.

The second measure is called *Query Success Rate* (QSR), which indicates the percentage of queries whose locations are successfully cloaked into regions. A good cloaking technique should have a QSR very close to one. Should this number be equal to one, it means that the technique cloaks all queries successfully.

Another measure is *Relative Anonymization Area* (RAA). RAA is defined as the ratio of the sum of the sizes of all anonymization areas to the size of the entire system area, where an

anonymization area is defined as the minimal bounding rectangle of a cloaked region, which consists of one or multiple cells. This metrics is quite important. Recall that in the three tier architecture (Fig. 4.1), after query locations are cloaked into regions, the anonymizer server needs to send the region information to service providers. The smaller the size of a region, the lighter workload imposed on service providers.

As discussed in Section 4.3, when an user issues a query, a personal privacy profile is specified by a tuple $\langle k, l \rangle$. Given an anonymization area, the ideal scenario is that the area should meet the privacy requirements for all users included in the area, but without exceeding the requirements by too much. To measure this effect, we can use the *Relative Anonymity Level (RAL)*, which is defined as follows:

$$RAL = \frac{k_{act}}{k_{spec}} * \frac{e_{act}}{e_{spec}} \quad (4.2)$$

where k_{act} is the actual number of queries in a cloaked region, and k_{spec} is the maximum of the specified k among all queries in the cloaked region, similarly, e_{act} is the actual query entropy for queries in the region, and e_{spec} is the calculated entropy using the maximum of the specified l among all queries in the cloaked region.

In a word, to evaluate a cloaking technique, all the above four metrics should be taken into consideration. A good cloaking technique must behave well in all the metrics.

4.5.2 Experimental Setup

The Brinkhoff data generator [6] is used to generate moving objects on the map of oldenburg, Germany. The outputs of the data generator are saved into files, which are then read by our simulator. In the initialization phase of the simulation, every moving object specifies a personal privacy profile $\langle k, l \rangle$. If we denote k_{max} and l_{max} as the supported maximum k and l of the system, respectively, then in each object's privacy profile, k is a randomly selected value in the range of $[1, k_{max}]$, and l is randomly selected from $[1, l_{max}]$. Each object issues

a location-based query inquiring about a randomly chosen query category. The anonymizer server then cloaks the queries for all objects using the proposed cloaking techniques.

The simulation is implemented in Java, and the running environment is a desktop computer with Intel Pentium 3.06GHz CPU and 2G memory, running a Linux operation system. In the experiments, we vary different parameters, as listed in Table 3.1, to study the efficiency and scalability of the proposed cloaking techniques. If not otherwise specified, the experiment takes the default values. For each parameter setting, the simulation is run with different input trace files for 100 times, and the averaged results are reported.

Table 4.2: Simulation Parameters

Parameter	Description	Value Set	Default
n_{cell}	Number of cells	{64, 256, 1024, 4096}	1024
k_{max}	Maximum of allowed k	{10, 20, 30, 40, 50}	20
l_{max}	Maximum of allowed l	{2, 6, 10, 14, 18}	6
n_q	Number of queries	{1000, 2000, 3000, 4000, 5000}	2000
n_{cat}	Number of categories for queries	{10, 20, 30, 40, 50}	20

4.5.3 Experimental Results

In this section, we first study the effects of varying the number of cells n_{cell} , then measure the performance of the techniques by varying k_{max} and l_{max} , finally, we study the scalability of the proposed techniques by varying the number of moving queries n_q and the number of query categories n_{cat} .

4.5.3.1 Varying number of cells

Since the whole terrain is divided into grid cells, we want to study the effect of grid cell size on the system performance. Note that a small grid cell size means a large number of cells. The total number of cells is varied from 64 to 4096, with the results shown in Fig. 4.3.

In Fig. 4.3.a, the effect on QAT is studied. Please note that the vertical axis is in logarithmic scale. The plot shows that more cells mean longer anonymization time for both

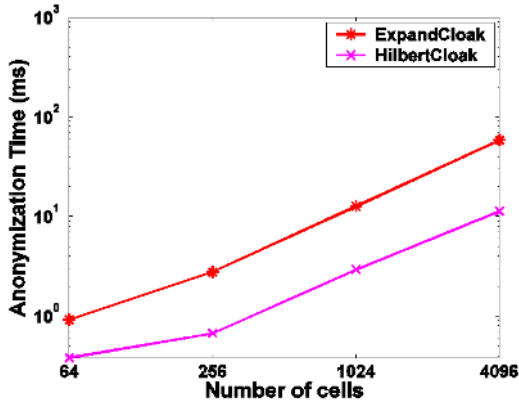
techniques. This is as expected, as more computations are necessary when cell size becomes smaller. The plot also shows that Expand Cloak technique runs slower than Hilbert Cloak, due to the expensive cell-expanding operations.

From Fig. 4.3.b, we learn that the QSR of Hilbert Cloak is always one, which means all queries can be successfully cloaked. However, the Expand Cloak technique experiences variations. Initially, the QSR increases as the number of cells increases, but if there are too many cells, the rate starts to drop. This can be explained as follows. When the number of cells is small (i.e. the size of a grid cell is large), the cloaked region is also large, which leads to over-anonymization, as shown in Fig. 4.3.d. As a result, it adds difficulty for some cells (especially boundary ones) to find neighbor cells to form a cloaked region. As n_{cell} increases, more boundary cells can be successfully cloaked, which leads to an improved QSR . However, if there are too many cells, cell size becomes very small, then some cells can not find non-empty neighbor cells to be combined with, which also leads to the drop of the success rate.

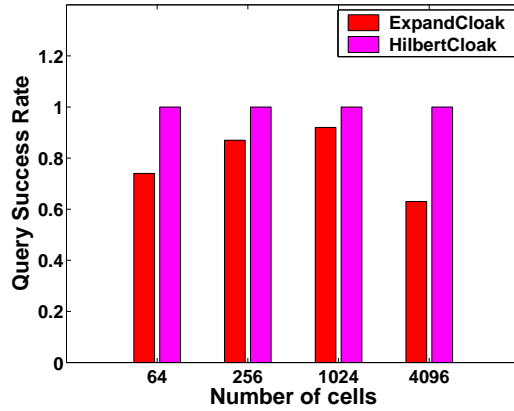
In Fig. 4.3.c, we studied the effect of n_{cell} on RAA . The results show that as n_{cell} increases, the RAA for Expand Cloak decreases while the RAA for Hilbert Cloak increases. This is because that for the Expand Cloak technique, smaller cell size can lead to smaller cloaked region; however, for the Hilbert Cloak technique, due to the one-dimensional nature of the Hilbert Curve, there are many overlappings among different cloaked regions. As a result, the more cloaked regions, the more overlapped space there are, which explains the RAA increases as the cell size gets smaller. Fig. 4.3.d shows that both techniques give a relatively low anonymity level. When n_{cell} increases, the RAL drops, which is more desirable.

4.5.3.2 Varying k and l

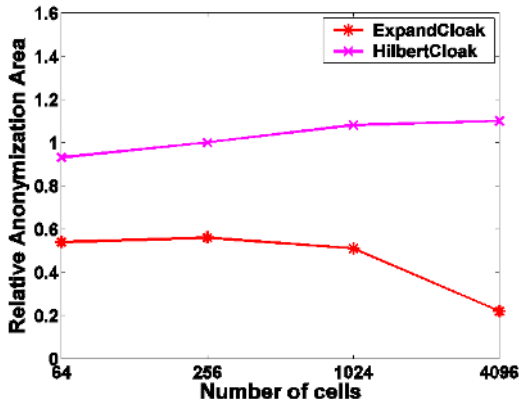
The other two important factors: the user defined k and l , are evaluated here. Furthermore, the Interval Cloak technique [23] described in Section 4.2, is modified to generate cloaked



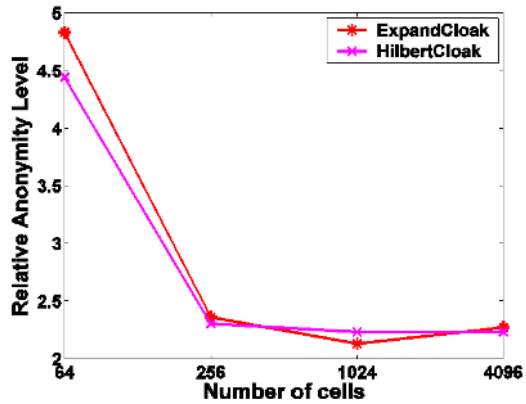
4.3.a: Query Anonymization Time



4.3.b: Query Success Rate



4.3.c: Relative Anonymization Area



4.3.d: Relative Anonymity Level

Figure 4.3: Varying number of cells

regions meeting the $\langle k, l \rangle$ -sharing region requirement. The proposed techniques are then compared against the improved Interval Cloak technique.

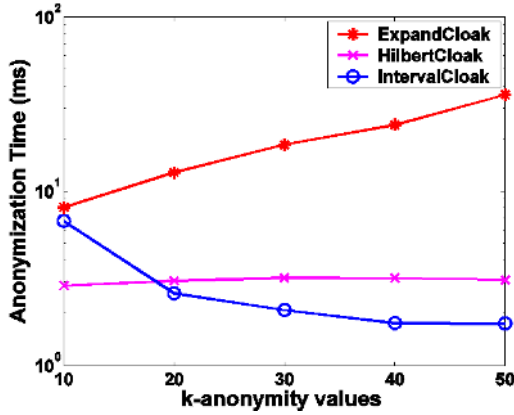
In Fig. 4.4, we study the effects of varying k_{max} from 10 to 50. Recall that the user specified k in k -anonymity is randomly selected from the range $[1, k_{max}]$. In Fig. 4.4.a, the plot shows that as k_{max} increases, the QAT increases. This is quite reasonable since a larger k_{max} means that the user defined k would be larger, which then requires more cell-expanding operations. Interestingly, the Hilbert Cloak technique seems to be immune to the k_{max} changes. This is because we only need to scan the grid cells to determine cloaked regions. A larger k_{max} leads to a larger cloaked region, however, the overall execution time

is not affected. Also quite interestingly, the Interval Cloak technique demands less execution time as k_{max} increases, which is due to the fact that the Interval Cloak technique is a top-down approach, a larger k_{max} means larger cloaked regions and fewer number of iterations.

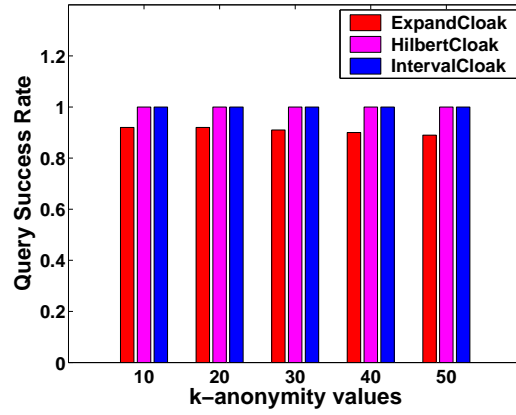
Fig. 4.4.b and Fig. 4.4.c show that all three techniques have high QSR , and the RAA increases with the increase of k_{max} . Among the three techniques, the Expand Cloak technique incurs the smallest anonymization area size, while the Hilbert Cloak technique requires the largest anonymization area size. This is because the Expand Cloak technique calculates a cloaked region using cell expanding technique, which can reduce the including of unnecessary dead space. On the other hand, since Hilbert Cloak relies on the one-dimensional Hilbert Curve, the cloaked regions have a lot of overlapping, which leads to a large anonymization area.

Fig. 4.4.d measures the RAL for the three techniques. The plot shows that both Expand Cloak and Hilbert Cloak have smaller RAL compared to Interval Cloak, which indicates that the cloaked regions meet the user defined requirement better. On the contrary, Interval Cloak has a much larger RAL , the result of creating unnecessarily large regions.

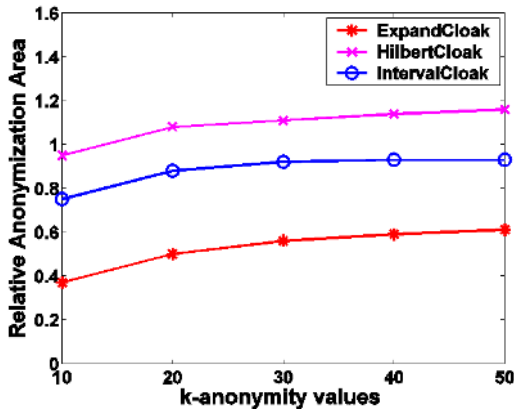
Fig. 4.5 studies the effect when l_{max} is varied from 2 to 18. First of all, Fig. 4.5.a shows that as l_{max} increases, both Expand Cloak and Hilbert Cloak incur longer computation time, however the anonymization time for Interval Cloak decreases. When l_{max} increases, for the Expand Cloak technique, to find a cloaked region for a given cell, more neighboring cells must be examined and included to meet the l -diversity requirement, which explains why the anonymization time is longer. However, for the Hilbert Cloak technique, the increase in anonymization time is mainly due to the cost in calculating query entropy. When the l is getting larger, the cost for calculating query entropy is also higher. Interestingly, the time needed for Interval Cloak decreases, which is because Interval Cloak stops after a few iterations. This leads to larger cloaked regions when compared to the other two techniques, as we will see from the RAL metrics.



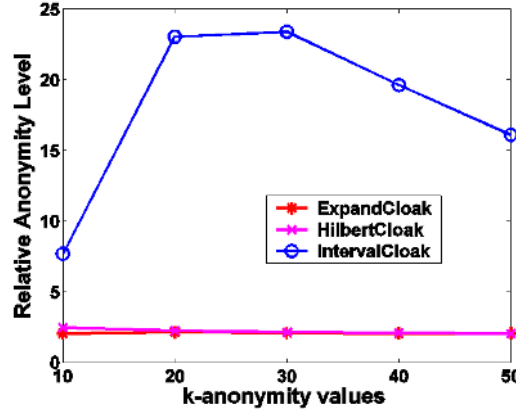
4.4.a: Query Anonymization Time



4.4.b: Query Success Rate



4.4.c: Relative Anonymization Area

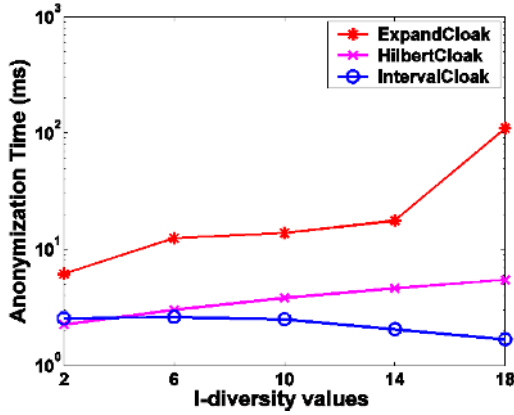


4.4.d: Relative Anonymity Level

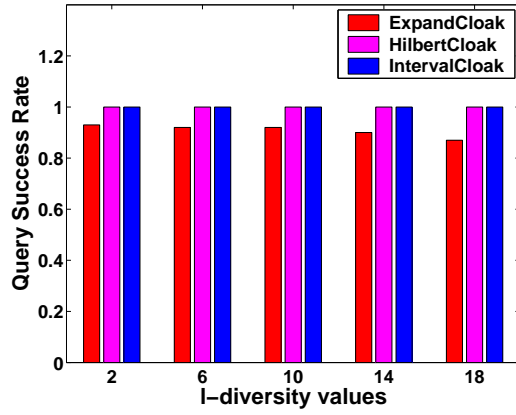
Figure 4.4: Varying k in k -anonymity

Fig. 4.5.b, Fig 4.5.c, and Fig. 4.5.d compare the quality of the three cloaking techniques. Fig. 4.5.b tells us that all three techniques have high QSR , and Fig. 4.5.c again shows that the Expand Cloak technique leads to cloaked regions with the smallest size, similar to the results obtained when varying k_{max} . In Fig. 4.5.d, we find out that although Interval Cloak do well in terms of the two performance metrics: QSR and RAA , it actually over-anonymizes many queries, as demonstrated by the high RAL values.

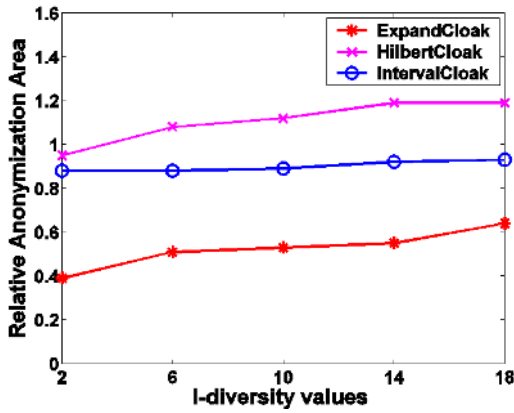
We also observe that when l_{max} is increased from 14 to 18, the QAT for Expand Cloak experiences a sharp increase. This is because the default number for the total available number of query is 20, when l_{max} is getting larger and close to 20, it becomes very difficult



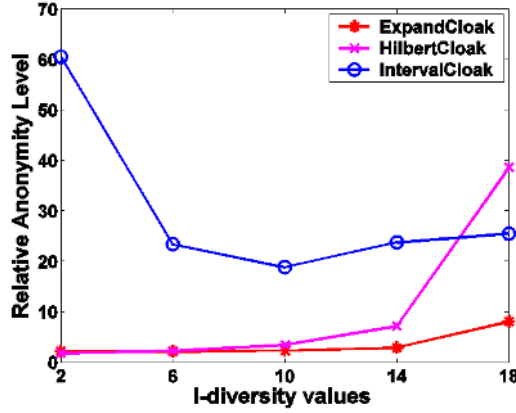
4.5.a: Query Anonymization Time



4.5.b: Query Success Rate



4.5.c: Relative Anonymization Area



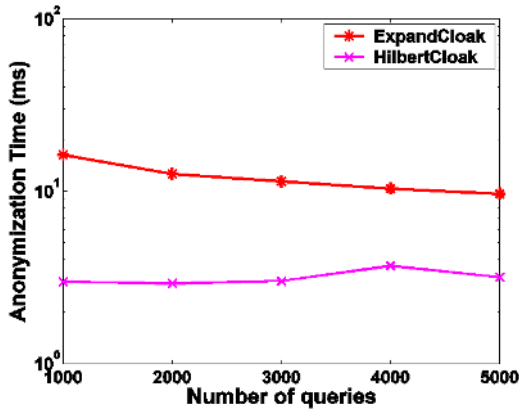
4.5.d: Relative Anonymity Level

Figure 4.5: Varying l in l -diversity

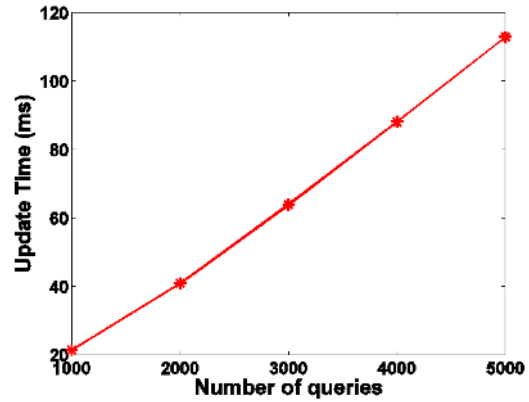
to find a region that can meet the privacy requirement. As a result, it takes longer to cloak queries, and the resultant cloaked regions are larger. This impact is also observed from the sharp raise in RAL and the drop in QSR when l_{max} is increased from 14 to 18.

4.5.3.3 Scalability Study

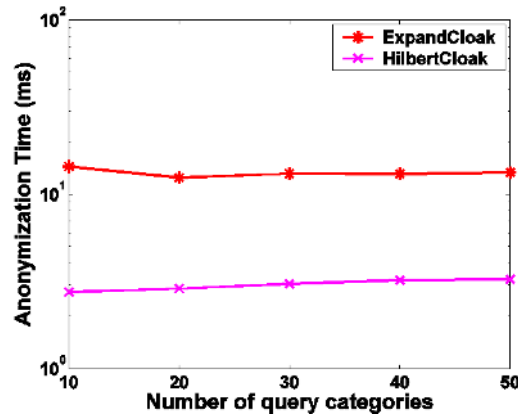
The scalability study of the proposed techniques is shown in Fig. 4.6. First, in Fig. 4.6.a, we vary the number of queries from 1000 to 5000 and measure its effect on anonymization time. The plot shows that the effect on the Hilbert Cloak technique is very small. However, the Expand Cloak technique incurs less anonymization time with the increase of queries,



4.6.a: Query Anonymization Time



4.6.b: Update Cost



4.6.c: Query Anonymization Time

Figure 4.6: Scalability Study by varying number of queries and query categories

this is because the more queries in each grid cell, the less cell-expanding is necessary, which reduces the overall anonymization time.

Second, Fig. 4.6.b measures the time to update the grid-based data structure when the number of queries increases. When new queries are sent to the anonymizer server, the grid data structure needs to be updated accordingly. In our simulation, the grid data structure is kept as an in-memory data structure. As we can see, the update cost is linear to the number of queries. Since the grid-based index is a flat structure, it has advantage over a tree-based index, which requires more expensive operations to maintain its tree structure.

Third, we increase the total number of query categories from 10 to 50 and study its impact on anonymization time. The obtained results are shown in Fig. 4.6.c, which shows

that both techniques are not sensitive to the number of query categories. For the Expand Cloak technique, when the number of categories is increased from 10 to 20, the anonymization time drops slightly, indicating that it is easier to find cloaked regions due to more available query categories. However, after that point, while the number of categories keeps increasing, the effect is not noticeable.

4.5.3.4 Discussions

Please note that although the Expand Cloak technique needs longer time when compared to the other two, the time actually needed is only a few milliseconds. Given that it has the lowest *RAA* and a close-to-one *QSR*, it has advantage over the other two techniques in practice.

4.6 Conclusion

In privacy-aware mobile information system, it is important for a cloaking technique to hide both user locations and the content of the issued queries. Existing techniques typically only focus on anonymizing user locations. In this chapter, we first propose to use both location k -anonymity and query l -diversity to better protect user privacy. A new property called $\langle k, l \rangle$ -sharing region is then identified as the guide to design new cloaking algorithms. This property is used to design the Expand Cloak and Hilbert Cloak techniques to achieve both location k -anonymity and query l -diversity. To assess their performance, we also design an improved version of the original Interval Cloak technique [23] to handle query l -diversity. The simulation results indicate that Expand Cloak generates cloaked regions with smaller size and lower *Relative Anonymity Level*, while Hilbert Cloak is faster but leads to larger cloaked regions. Both techniques are significantly better than the improved Interval Cloak technique in providing user privacy protection.

CHAPTER 5: USING BROADCAST TO PROTECT USER PRIVACY

5.1 Introduction

To protect user privacy in answering location-based queries, there are mainly two directions. One direction is to assume a three-tier architecture, consisting of mobile users, a trusted anonymizer server, and service providers, as discussed in Section 4.3.1. Another direction employs a two-tier approach: mobile users and service providers. Mobile users send queries directly to service providers, but to ensure privacy, techniques such as Private Information Retrieval [20], Space Transformation [33], Peer-to-Peer [13], and Incremental Query Processing [73] are used.

From Chapter 3, we also learn that in a wireless environment, a server can communicate with clients in the following two ways: on-demand communication and broadcast communication. In the on-demand communication, a client sends a request to the server and the server sends the result back to the client. In the broadcast communication, the server puts what clients need in a broadcast channel; a client can just tune into the channel and download the desired information. In Chapter 3, we propose to use a hybrid communication technique to reduce communication cost.

In this chapter, we propose a hybrid three-tier architecture, where the trusted anonymizer server also serves as a broadcast server. It first groups mobile users into clusters, then for each cluster, it fetches query results from service providers before mobile users issuing queries. The query results are broadcast in an air channel. As a result, mobile users can tune into the air channel first to determine if a query can be answered. If not, mobile users then issue a traditional location-based query to the trusted anonymizer server. The advantages of doing this are two-folds: (1). For clients that can obtain the needed information from the

broadcast channel, there is no privacy exposure at all. (2). For clients that can not get the information, they can fall back to use the traditional technique to have query answered.

The remainder of this chapter is organized as follows. The proposed solution is introduced in Section 5.2, followed by the simulation study in Section 5.3. Finally, we conclude this chapter in Section 5.4.

5.2 Proposed Solution

We first present the proposed new system architecture, then discuss the anonymization goal, followed by the clustering algorithms. Finally, we cover how to broadcast query results in an air channel.

5.2.1 System Architecture

The system consists of a large number of mobile users, a trusted third-party anonymizer server, and one or more service providers, as shown in Fig. 5.1. Mobile objects send location updates periodically to the trusted anonymizer server. The trusted anonymizer server then groups objects into clusters. For each cluster, the trusted anonymizer server sends a location-based query to the service provider, using the minimal bounding rectangle of the cluster as the query's region. After receiving query results back, the trusted anonymizer server broadcasts the query results on an air channel. Mobile objects can then tune into the air channel and get the query results.

5.2.2 Anonymization Goals

For each cluster, we want to make sure that there are at least k mobile objects, to ensure that an adversary can not link a query request to a specific mobile object. In other words, the cluster must meet the k -anonymity region requirement (Section 4.3.2).

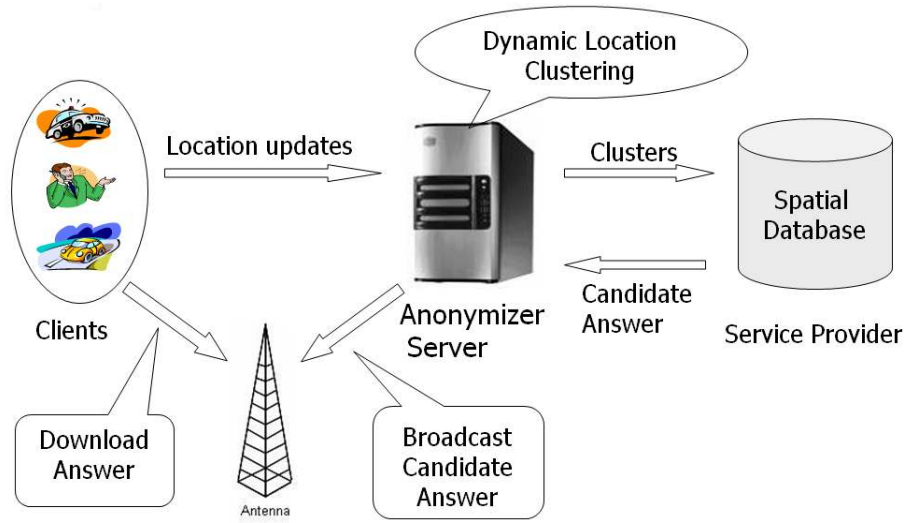


Figure 5.1: System Architecture

Because the query results returned from service providers are for the whole cloaked region, typically, mobile objects have to refine the query results to find the desired ones. To ensure the Quality of Service, it is better to limit the size of the allowed cloaked region, such that mobile objects can find the desired query result quickly.

Property 5. (Maximum Region Size) For a cloaked region, the size of the region must not be larger than a pre-defined size S .

5.2.3 Clustering Algorithms

As mentioned in Section 5.2.1, the trusted anonymizer server needs to group mobile objects into clusters. We assume that the terrain is divided into grid cells of the same size. A cluster area consists of one or multiple adjacent cells. Consequently, all mobile objects residing in a cluster area form a cluster. To facilitate the clustering, on the anonymizer server, we maintain a counter for each cell to record the number of mobile objects currently moving in that cell. The counter value is updated periodically, after location updates are received from mobile objects.

Two clustering algorithms are proposed. The first one is a simple scanning algorithm, called *ScanCluster*, as shown in Algorithm 5.1. The algorithm simply examines the number of mobile objects in that cell to determine if it is greater than the pre-specified k value. If yes, the cell forms a cluster. Otherwise, the cell is ignored.

Algorithm 5.1 Scan Clustering Algorithm

```

1: for each cell  $c$  do
2:   if  $c.n \geq k$  then
3:     Create a cluster for Cell  $c$ ;
4:   end if
5: end for

```

The second clustering algorithm is called *ExpandCluster*, as shown in Algorithm 5.2. The intuition is that if a cell does not meet the privacy requirement by itself, some neighboring cells of that cell are included to form a cluster.

The clustering algorithm starts with a *for* loop to examine every cell in the area. First, we check if the cell is already included by some cluster (lines 2-4). If yes, the algorithm moves to the next cell. Otherwise, we check if the cell by itself can meet the requirement to form a cluster (lines 6-9). If a cell does not contain enough mobile objects, we will try to combine the cell with one of its neighboring cells to form a cluster. If a cell does not have any neighboring cells that are not included by some clusters, or all the neighboring cells do not contain any mobile objects, we simply ignore this cell (lines 12-14). Otherwise, all the neighboring cells that, after combined with the current area, can meet the k -anonymity requirement while satisfying maximum region size requirement, are kept in the set r (lines 16-21). If there are more than one candidate neighboring cell, the one with the fewest objects is selected (lines 23-24); otherwise, the one with the most objects is selected to form a temporary area (lines 26-27). The while loop continues until a cluster is formed or no cluster can be formed.

Algorithm 5.2 Expand Clustering Algorithm

```
1: for each cell  $c$  do
2:   if  $c$  is already included by a cluster then
3:     Continue;
4:   end if
5:    $area \leftarrow \{c\}$ ; {Initialization}
6:   if  $area.n \geq k$  then
7:     Create a cluster for  $area$ ;
8:     Continue;
9:   end if
10:  while  $area$  is not included by a cluster do
11:     $nc \leftarrow area$ 's neighbor cells that are not included by a cluster;
12:    if  $nc == \emptyset$  ||  $\forall c_i \in nc, c_i.n == 0$  then
13:      Break; {Ignore this area for clustering}
14:    end if
15:     $r \leftarrow \emptyset$ ; {To store candidate neighbor cells}
16:    for each cell  $c_i$  in  $nc$  do
17:       $tmp\_area \leftarrow area \cup c_i$ ;
18:      if  $tmp\_area.n \geq k$  &&  $tmp\_area.size \leq S$  then
19:         $r \leftarrow r \cup c_i$ ;
20:      end if
21:    end for
22:    if  $r \neq \emptyset$  then
23:       $c_p \leftarrow$  the cell in  $r$  with the fewest objects;
24:      Create a cluster for  $area \cup c_p$ ;
25:    else
26:       $c_p \leftarrow$  the cell in  $nc$  with the most objects;
27:       $area \leftarrow area \cup c_p$ 
28:    end if
29:  end while
30: end for
```

5.2.4 Broadcast Channel

The trusted anonymizer server also serves as a broadcast server. Upon receiving query results, the anonymizer server broadcasts the query results on a broadcast channel. The interleaving mechanism, as shown in Fig. 3.2, is used in the broadcast. For a broadcast channel, the most important thing is to design a broadcast index such that mobile objects can obtain the data from the broadcast channel quickly. Fig. 5.2 shows the broadcast index. The first layer is the index layer, which lists all the cells. For each cell, there is a pointer that

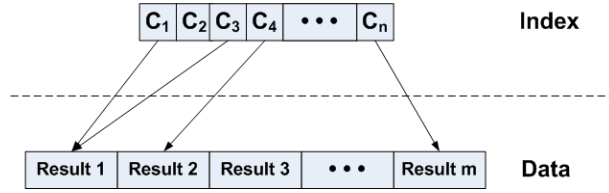


Figure 5.2: Broadcast Index

pointing to the query result for that cell (data layer). Since one cluster could cover multiple cells (i.e. If Algorithm 5.2 is used), multiple cells could point to the same data block in the data layer. For example, Cell 1 and Cell 3 are pointing to the same Result 1. Also, some cells are not included by any cluster, hence there is no pointer originating from those cells, such as Cell 2 in the figure.

Mobile objects can first tune to the air channel, based on which cell they are in, they can follow the pointer and download the query results. Since the query results are meant for the cell, mobile objects will need to refine the results to meet their needs.

5.2.5 Hybrid Solution

Because of the constraints of the k-anonymity region and the Maximum Region Size, the successful formation of clusters depends heavily on the density of mobile objects. Not all mobile objects can be included by some cluster. Consequently, the objects not included in any cluster can not find the query results in the broadcast channel.

To solve this problem, we propose a hybrid solution. After tuning to the broadcast channel, mobile objects that can not download query results from the broadcast channel should send a query request (note: not location update) to the trusted anonymizer server. The anonymizer server then cloaks the query request using some existing techniques [23] [48] [12] [5] [70] [71], and sends the cloaked query to the service provider.

5.3 Performance Study

In this section, we study the proposed techniques using simulation. First, we introduce performance metrics employed in our study, then cover the experimental setup, finally, we present the detailed simulation results.

5.3.1 Performance Metrics

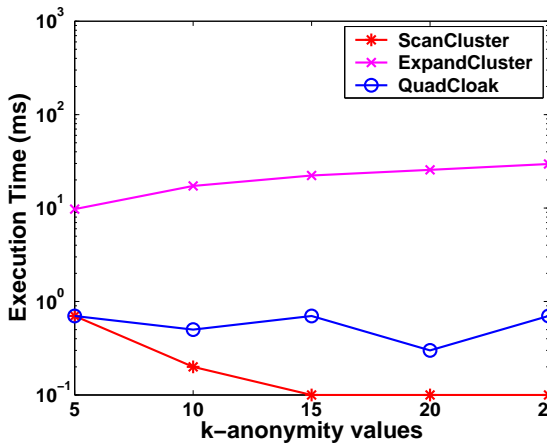
The first measure is *Query Execution Time (QET)*, which measures the run-time efficiency of a cloaking technique. The second measure is *Query Success Rate (QSR)*, which indicates the percentage of queries whose locations are successfully cloaked into regions. A good cloaking technique should have a *QSR* close to one. The third measure is *Number of Communication Messages (NCM)*, to sum the total messages communicated among mobile objects, anonymizer server, and service providers.

5.3.2 Experimental Setup

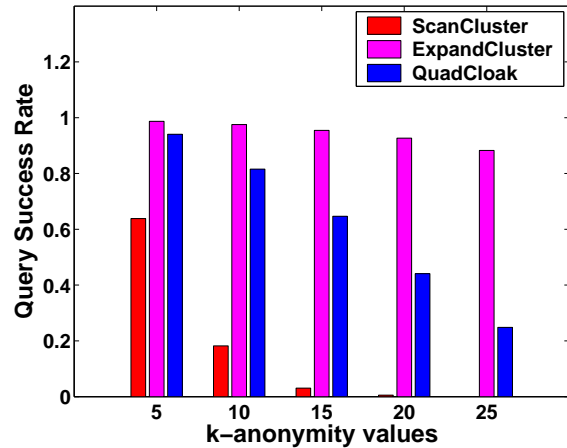
The Brinkhoff data generator [6] is used to generate moving objects on the map of Oldenburg, Germany. The outputs of the data generator are saved into files, which are then read by the simulator. Each object issues a location-based query. The anonymizer server then cloaks the queries for all objects using the proposed techniques. The terrain is divided into 1024 cells. We vary k in k -anonymization and num_obj (the number of objects), as listed in Table 5.1, to study the proposed cloaking techniques. If not otherwise specified, the experiment takes the default values. For each parameter setting, the simulation is run with different input trace files for 100 times, and the averaged results are reported.

Table 5.1: Simulation Parameters

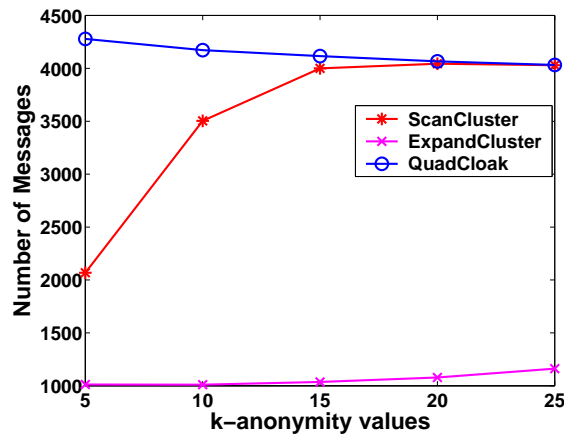
Parameter	Description	Value Set	Default Value
k	k in k-anonymity	{5, 10, 15, 20, 25}	10
num_obj	Number of objects	{1000, 2000, 3000, 4000, 5000}	2000



5.3.a: Query Execution Time



5.3.b: Query Success Rate

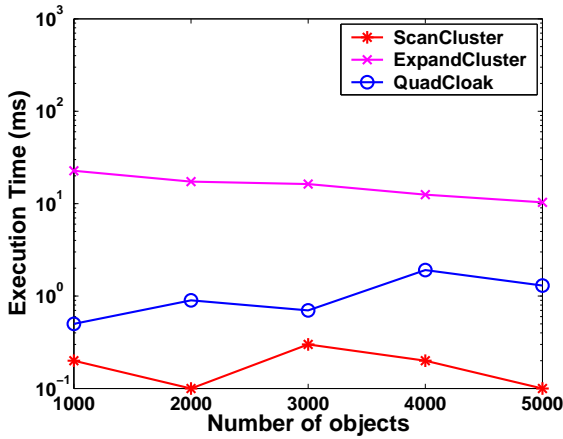


5.3.c: Number of Messages

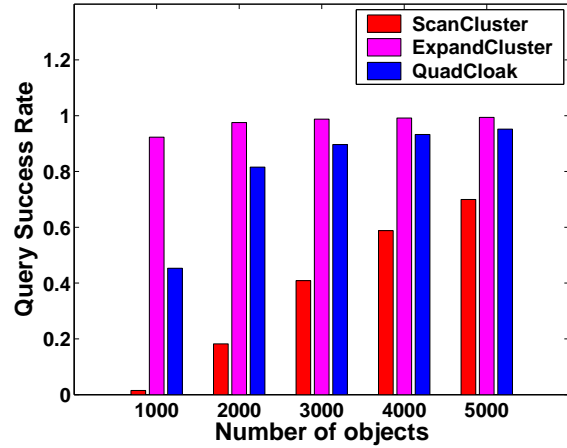
Figure 5.3: Varying k in k -anonymity

5.3.3 Experimental Results

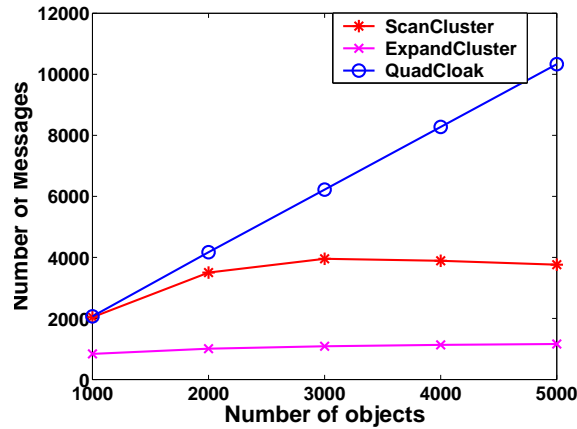
As a comparison, the Interval Cloak technique [23] described in Section 4.2.2, is modified to generate cloaked regions meeting the k -anonymity region and Maximum Spatial Resolution properties discussed in Section 5.2.2. We name the improved Interval Cloak technique as QuadCloak. The proposed techniques are then compared against the QuadCloak technique.



5.4.a: Query Execution Time



5.4.b: Query Success Rate



5.4.c: Number of Messages

Figure 5.4: Varying number of objects

5.3.3.1 Varying k in k -anonymity

In Fig. 5.3, we study the effects of varying k from 5 to 25. Fig. 5.3.a shows that as k increases, the QET increases for the ExpandCluster technique, and decreases for the ScanCluster technique. However, no noticeable trend is observed for the QuadCloak technique. The main reason is that a larger k demands more expanding effort for the ExpandCluster technique, leading to longer time. However, for ScanCluster and QuadCloak, the cloaking area can not be found, thus the algorithms terminate quickly. Due to the same reason, Fig. 5.3.b shows the success rate drops quickly for ScanCluster and QuadCloak when k increases. However,

ExpandCluster still maintains high success rate. That's because with the expand clustering algorithm, more clusters are formed to meet the k -anonymity requirement than the other two methods, thus more query results can be broadcast in the broadcast channel. Fig. 5.3.c plots the number of communication messages versus k . For ExpandCluster, because the query success rate only drops slightly, the need for communicating to server is low, which explains the number of messages only increases slowly. However, for ScanCluster, because the query success rate drops to zero when k is increased, it has to get query results via point to point communication, leading to the abrupt increase of the number of messages. For QuadCloak, the number of messages drops slightly because the success rate drops, hence the trusted anonymizer server sends fewer messages to service providers.

5.3.3.2 Varying number of objects

Fig. 5.4.a shows that as num_obj increases from 1000 to 5000, ExpandCluster incurs shorter computation time, because more objects mean it is easier to find a neighboring cell to form a cluster area, hence reduces computation time. However, the effect is not noticeable with the ScanCluster technique, because the algorithm is a linear algorithm and the execution time is linear to the number of cells. For QuadCloak, the execution time presents a rising trend overall, because more objects demand smaller cloaking areas, which requires more iterations of divisions, hence more time. Fig. 5.4.b is as expected. The QSRs for all three algorithms increase when there are more objects. ExpandCluster has better success rate because more clusters are formed, hence more queries are answered.

In Fig. 5.4.c, it is clear that increasing the number of objects lead to the linear increase of the communication messages for the QuadCloak technique. That is simply because more objects need more messages. However, the number of messages increases initially for the ScanCluster technique, then becomes flat and starts to drop when the number of objects increases. This is because as the success rate increases, more objects can get the query

answered by tuning into the broadcast channel, which can reduce the number of messages. Similarly, we can see that ExpandCluster is only slightly affected by the number of objects, because most objects can get the query results through the broadcast channel, and incurs only low communication cost. The plot demonstrates the superior scalability of the ExpandCluster technique over the other two techniques.

5.4 Conclusion

In this chapter, we propose a hybrid three-tier architecture. The trusted anonymizer server first divides mobile users into clusters, then retrieves query results for each cluster and broadcasts the query results through a broadcast channel. Therefore, mobile users can tune into the broadcast channel and possibly download query results directly without issuing a query. Two clustering techniques: ScanCluster and ExpandCluster, are proposed and compared against a modified Interval Cloak technique. The simulation results show that ExpandCluster is the best technique overall.

CHAPTER 6: PRIVACY PROTECTED QUERY PROCESSING WITH ROAD NETWORK EMBEDDING

6.1 Introduction

To protect user privacy, location-based queries are typically answered by a three-tier system: mobile users, trusted anonymizer, and server providers, as shown in Section 4.3.1. Most existing researches in this area focus on the trusted anonymizer by designing better cloaking algorithms [23, 18, 48, 12, 5, 70, 45]. In a cloaked query sent to a service provider, the location is typically represented by a *Minimal Bounding Rectangle* (MBR), not a point. If this query is issued in an open space, where Euclidean distance is used, query processing can be easily extended from existing techniques. However, if the query is issued in a road network environment, it is more challenging because the distance computation is significantly more complex.

There have been some query processing techniques proposed for a road network environment, as reviewed in Section 2.2. However, very little research has been done to support privacy-enabled queries in such an environment. To the best of our knowledge, [39] is the first to attack this problem. The authors consider both privacy-protected spatial network nearest neighbor query and privacy-protected spatial network range query. However, the proposed algorithms are based on the original two-dimensional space and adopt the network expansion techniques for query processing, which is quite inefficient.

One efficient solution to compute network distance is the *Road Network Embedding* (RNE) technique [57]. RNE transforms points in a two-dimensional space to a k -vector space and uses L_p metric to approximate the network distance in the high-dimensional space. In this chapter, we propose to use RNE to answer privacy-enabled queries. In particular, we discuss how to answer privacy-enabled k -nearest neighbor queries, and extend the solution to answer

range queries and queries over private objects. The performance of the proposed techniques is studied extensively using simulation studies.

The remainder of this chapter is organized as follows. The Road Network Embedding technique is presented in Section 6.2, followed by the proposed solution in Section 6.3. In Section 6.4, we discuss the performance study and give the simulation results. Finally, we conclude this chapter in Section 6.5.

6.2 Road Network Embedding

A road network is modeled as an undirected graph $G = (N, E)$, as discussed in Section 2.3.2. *Road Network Embedding* (RNE) is a way to compute network distance in a road network, proposed by Shahabi et al. in [57]. RNE uses *Linial, London, and Robinovich* (LLR) embedding technique, a special form of Lipschitz embedding on road networks [44].

RNE transforms points in the original space into points in a high dimensional space. The transformation works as follows. Let S denote the set of all the points in the original space, and n is the size of this set. We define a set R as a set of subsets of S : $R = \{S_{1,1}, \dots, S_{1,\beta}, \dots, S_{\alpha,1}, \dots, S_{\alpha,\beta}\}$, where $\alpha = O(\log(n))$, $\beta = O(\log(n))$ and $S_{i,j}$ contains randomly chosen 2^i elements from S . So the size of R is $O(\log^2 n)$.

Given $S_{i,j}$ as a subset of S , we define the distance between a node u and $S_{i,j}$ as $D(u, S_{i,j}) = \min_{v \in S_{i,j}} \text{dist}(u, v)$, which means the shortest distance to a node in $S_{i,j}$. The *dist* function can be one of the L_p metric:

$$L_p(u, v) = \left[\sum_{i=1}^k |u_i - v_i|^p \right]^{\frac{1}{p}}$$

where u and v are two points in the space, and u_i and v_i are their i th coordinates, respectively. p refers to the order of *Minkowski* distance. When p equals to 1, the distance is called *Manhattan* distance; and when p equals to 2, the distance is known as *Euclidean* distance.

The infinite order:

$$L_\infty = \max_i |u_i - v_i|$$

is the chessboard distance. Based on [57], using chessboard distance is the best for approximating network distances in the embedding space.

The embedding is calculated as follows: Given a point u in the original space,

$$E(u) = (E_{S_{1,1}}(u), \dots, E_{S_{1,\beta}}(u), \dots, E_{S_{\alpha,1}}(u), E_{S_{\alpha,\beta}}(u))$$

where $E_{S_{i,j}}(u) = D(u, S_{i,j})$. Once the points in the original space have been mapped to the embedding space, we can approximate the network distance between any two points by computing their chessboard distance in the embedding space.

To reduce the computational complexity in the embedding space, the authors in [57] proposed *Truncated Embedding Space*, where only the first a few reference sets in the embedding space are used. In Section 6.3, we will use the truncated embedding space to find query results. The effect of the percentage of reference sets used will be studied in Section 6.4.

6.3 Proposed Solution

In general, there are two types of objects stored in the location database, *private* and *public* objects. Public objects are assumed to be stationary in this chapter. Their location information is public. Examples include restaurants, gas stations, and hospitals. Private objects are the mobile or stationary users. Their locations are subject to privacy protection. We are interested in (1) *private query over public objects* where the query issuer is a private object while the queried objects or points of interest are public, and (2) *private query over private objects* where both the query issuer and *points of interest* (POI) are private. More specifically, we first discuss in this section how to answer k -nearest neighbor queries over

public objects, then extend the technique to address range queries over public objects and queries over private objects.

6.3.1 Private kNN Query over Public Objects

Given an MBR as the cloaking area of a private object, the computation of its k nearest neighbors is given in Algorithm 6.1. The first step is to find all the intersection points of the MBR with the road segments (line 1). After that, the algorithm retrieves all the nodes (i.e., road intersections) outside the MBR, that are next to the intersection points with the MBR (lines 2-7). For each of these nodes, its k nearest POI's are determined using the embedding space (line 10). For each of these POI's, if it is within the MBR, its distance to the MBR is zero and is added to the candidate set R (lines 12-13); otherwise, this POI is included in the candidate set only if R currently has less than k candidates or this POI is closer to the MBR than the current k th nearest POI in R is (lines 14-19). Finally, the algorithm sorts the candidates in R according to their distance to the MBR and returns the first k candidates as the result. We note that we define R as a set and need to sort this set many times. This is for the sake of clarity. In practice, one can implement R as an order list to avoid the sorting cost.

We use Figure 6.1 to illustrate the algorithm. The query point q is shown as a star, and the points of interest are shown as triangles (there are five of them). The dark-black square is the MBR. The algorithm first identifies s_1 , s_2 , s_3 , and s_4 as the intersection points with the MBR, then uses them to find n_1 , n_2 , n_3 , and n_4 as the nodes (road intersections) closest to the MBR. For each of these nodes, we find its k nearest POI's. Assume that $k = 1$ in this example, these POI's found are p_1 for n_1 , p_2 for n_2 , p_4 for n_3 , and p_5 for n_4 . Finally, in the refining steps, we conclude that p_5 is the point of interest that is closest to the MBR.

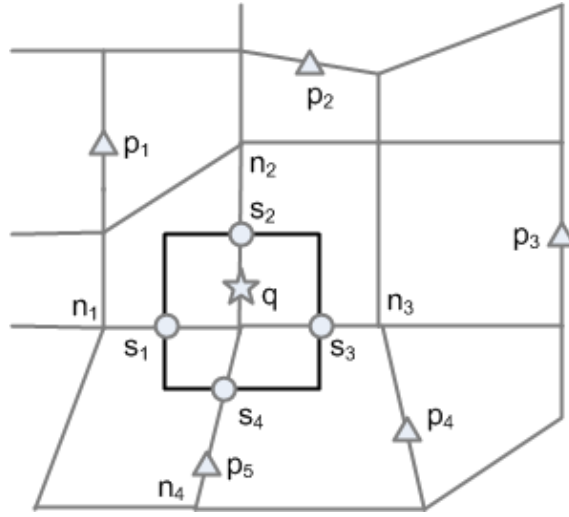


Figure 6.1: Private query over Public Objects

Algorithm 6.1 *k*-Nearest Neighbor Query Algorithm

Require: An MBR defining a cloaking area

Ensure: The *k*-nearest neighbors to the MBR

- 1: Determine the set S of all the intersection points of the MBR with road segments.
 - 2: $N \leftarrow \emptyset$; $\{N$ is used to store the closest nodes to the intersection points in $S\}$
 - 3: **for** each point s in S **do**
 - 4: Find the closest node n that is not inside the MBR
 - 5: $N \leftarrow N \cup n$;
 - 6: Use $n.s$ to refer to the intersection point s
 - 7: **end for**
 - 8: $R \leftarrow \emptyset$; $\{R$ is used to store the final k nearest neighbors $\}$
 - 9: **for** each node n in N **do**
 - 10: Find the set of k nearest POI's, denoted as $n.kNN$, in the embedding space.
 - 11: **for** each POI p in $n.kNN$ **do**
 - 12: **if** p is inside of the MBR **then**
 - 13: $R \leftarrow R \cup p$; $\{\text{The distance from } p \text{ to the MBR is zero}\}$
 - 14: **else**
 - 15: Calculate the distance from p to $n.s$, which is the distance from p to the MBR.
 - 16: **if** $R.size < k$ or the distance is smaller than the distance from the k th nearest POI in R to the MBR **then**
 - 17: $R \leftarrow R \cup p$;
 - 18: **end if**
 - 19: **end if**
 - 20: Sort the results in R based on the distance to the MBR
 - 21: **end for**
 - 22: **end for**
 - 23: Return the first k results in R
-

6.3.2 Extensions

In this section, we discuss how to extend Algorithm 6.1 to cover range queries over public objects and queries over private objects.

6.3.2.1 Private Range Query over Public Objects

A private range query is typically defined by an MBR and a pre-specified range [32]. Upon receiving a private range query, the service provider needs to extend the MBR by the range on all dimensions and return all POIs included in the extended MBR. Algorithm 6.2 shows how to answer private range query over public objects, which is a simple modification of Algorithm 6.1. The difference is that after the nodes closest to the MBR are identified, the algorithm needs to find the points of interests within the specified range, and then followed by the refining steps.

6.3.2.2 Private Query over Private Objects

So far, we have covered the cases when the points of interest are public. When the points of interest are private, they are not represented by points anymore, instead, they are represented by minimum bounding rectangles, as shown by the dashed squares in Figure 6.2. We can modify the above two algorithms by first identifying the intersection points of the POI's MBR with road segments, then using the intersection points to represent the POI. For example, in Figure 6.2, POI p_3 can be represented by two intersection points: t_1 and t_2 .

6.4 Performance Study

We study the proposed techniques using simulation. First, we introduce performance metrics employed in our study, then cover the experimental setup, finally, we present the detailed simulation results.

Algorithm 6.2 Range Query Algorithm

Require: An MBR defining a cloaking area and a specified range

Ensure: The points of interests within the range to the MBR

- 1: Determine the set S of all the intersection points of the MBR with road segments.
 - 2: $N \leftarrow \emptyset$; $\{N$ is used to store the closest nodes to the intersection points in $S\}$
 - 3: **for** each point s in S **do**
 - 4: Find the closest node n that is not inside the MBR
 - 5: $N \leftarrow N \cup n$;
 - 6: Use $n.s$ to refer to the intersection point s
 - 7: **end for**
 - 8: $R \leftarrow \emptyset$; $\{R$ is used to store the final results $\}$
 - 9: **for** each node n in N **do**
 - 10: Find all the points of interests within the specified range in the embedding space.
 Denote as $n.range$.
 - 11: **for** each POI p in $n.range$ **do**
 - 12: **if** p is inside of the MBR **then**
 - 13: $R \leftarrow R \cup p$; $\{\text{POIs inside of the MBR are included automatically}\}$
 - 14: **else**
 - 15: Calculate the distance from p to $n.s$
 - 16: **if** The distance is smaller than the specified range **then**
 - 17: $R \leftarrow R \cup p$;
 - 18: **end if**
 - 19: **end if**
 - 20: **end for**
 - 21: **end for**
 - 22: Return R
-

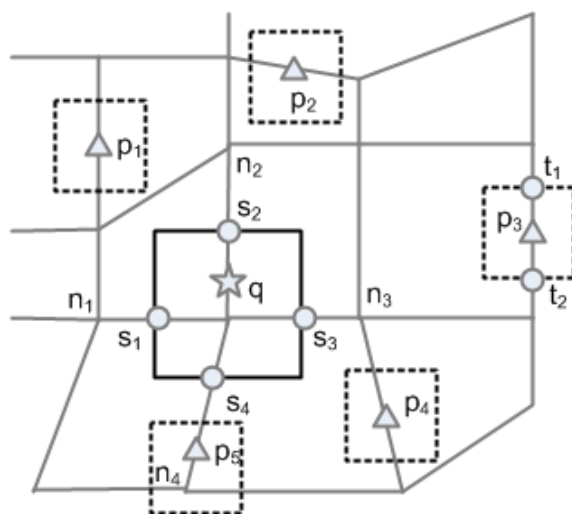


Figure 6.2: Private query over Private Objects

6.4.1 Performance Metrics

First, we want to study the efficiency of the technique, so the query processing time is a good indication of that. Secondly, because transforming points into the embedding space introduces distortion, queries can only be answered approximately. We use query result accuracy to gauge how close the approximation is. Please note that the query results refer to the results presented to the end users. So after obtaining the query results returned by the algorithms for the MBR, we further refine the results for each query in the MBR and compare the results with the ground truth.

6.4.2 Experimental Setup

We use the road network of City Oldenburg, Germany, generated by the Brinkhoff data generator [6]. To simplify the simulation, we assume that each node represents a POI. All the nodes are transformed into the embedding spaces. We place query points randomly on the terrain and construct Minimum Bounding Rectangles (MBR) around the query points. The MBRs used in the simulation are squares with side length of len . When calculating distances between two points in the embedding space, to save computation time, we use only the first few dimensions as the reference sets. Denote the percentage of the dimensions used in calculation as p . We study the effect of p in the simulation as well. Also, please note that we focus on k NN queries in this simulation. The results for range queries and private queries over private objects should be similar.

The simulation is written in Java, and run in a PC with Intel i3 2.13GHz CPU and 4G memory, running a Windows 7 operation system. We vary k , len , and p , as listed in Table 6.1, to study the proposed technique. If not otherwise specified, the experiment takes the default values. For each parameter setting, the simulation is run with a randomly generated MBR for 1000 times, and the averaged results are reported.

Table 6.1: Simulation Parameters

Parameter	Description	Value Set	Default Value
k	k in kNN	{10, 20, 30, 40, 50}	20
len	One side length of MBR	{50, 100, 150, 200, 250}	100
p	Percentage of dimensions used	{5, 10, 20, 30, 40}	10

6.4.3 Experimental Results

Figure 6.3 and 6.4 shows the results with the variation of the value of k . In Figure 6.3, the y axis shows the total time spent in milliseconds to answer 1000 queries. The figure shows that when the value of k increases, the processing time increases slowly. Since we have to find the k NN by examining all nodes in the system, the increase of processing time with the increase of k should not be significant, and the increase is mostly due the maintenance cost for keeping k results. Figure 6.4 indicates that the accuracy level is quite consistent and stable around 80%.

Figure 6.5 and 6.6 shows the changes while the size of MBR varies. Figure 6.5 is as expected, since when the MBR gets larger, more nodes are involved, hence more time are needed to find query results. Figure 6.6 shows that the quality of service improves slightly when the MBR becomes larger. That is because a larger MBR means more margin for error, which is translated into a better query result accuracy.

In Figure 6.7 and 6.8, we study the effect when varying the percentage of dimensions used in the embedding space. Figure 6.7 is as expected, since more dimensions require more time in calculating the distances among nodes. Figure 6.8 demonstrates that when only the first 5% of the dimensions are used, the query accuracy is relatively low, and when there are more than 10% of the dimensions are used, there is no noticeable accuracy improvement, which means that using 10% of the dimensions provides a good query accuracy without incurring a huge computation overhead.

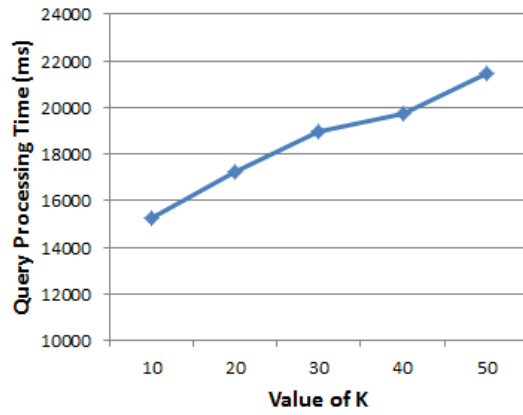


Figure 6.3: Effect on Query Processing Time when varying k in k NN

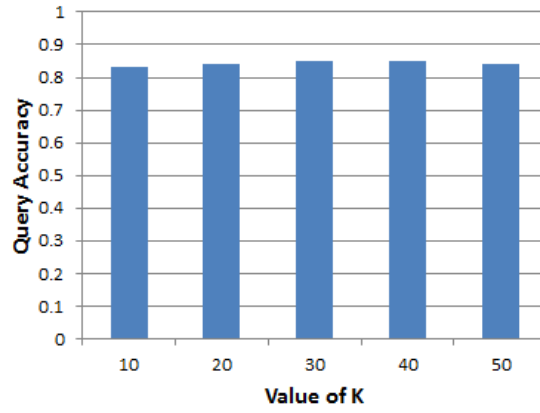


Figure 6.4: Effect on Query Result Accuracy when varying k in k NN

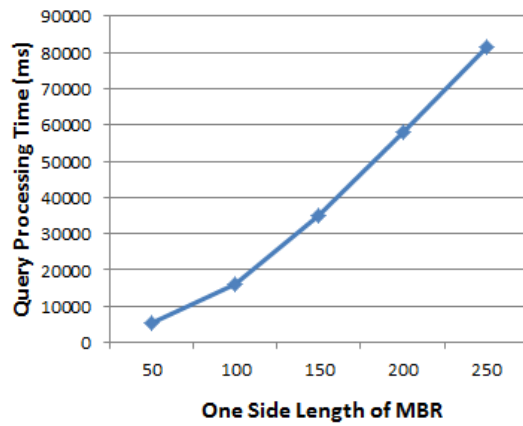


Figure 6.5: Effect on Query Processing Time when varying size of MBR

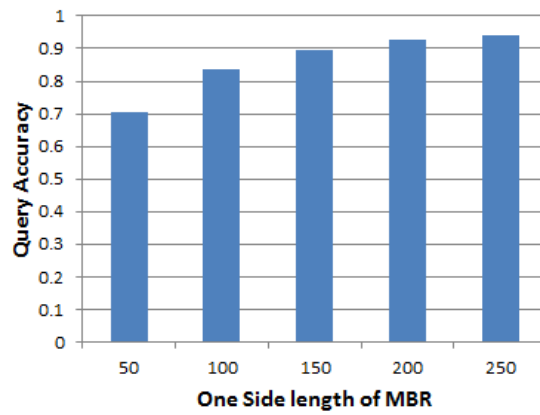


Figure 6.6: Effect on Query Result Accuracy when varying size of MBR

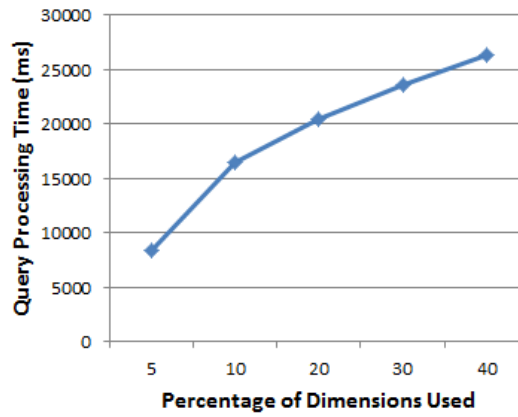


Figure 6.7: Effect on Query Processing Time when varying percentage of dimensions used

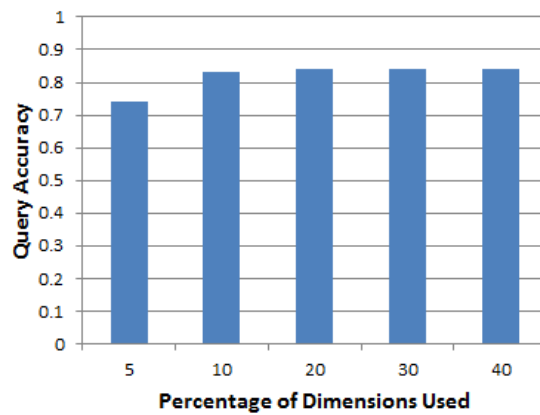


Figure 6.8: Effect on Query Result Accuracy when varying percentage of dimensions used

6.5 Conclusion

In a typical three-tier system used to protect mobile user's privacy, mobile user sends a location-based query to the trusted anonymizer server, which cloaks the query and sends the cloaked query to a service provider. The service provider then needs to answer the query. Most of the existing researches in this area assume that the query processing part on the service provider is straightforward. However, if the underlying environment is not an open space, but a road network, query processing is quite a challenge because of the complexity in network distance calculation. In this chapter, we propose to use the Road Network Embedding technique to answer cloaked queries in a road network. We first discuss how to answer k -nearest neighbor queries, then extend the solution to answer range queries and queries over private objects. Extensive simulation studies are performed to reveal the characteristics of the solution.

CHAPTER 7: CONCLUSION

Location-based services have become more and more popular in recent years. There have been quite some commercial applications emerging out recently. For example, Loopt [2] enables mobile users to find out who is around, what to do, and where to go. Another example is Foursquare [1], where mobile users can update their locations and stay connected with their friends. In this dissertation, we identify five problems in location-based systems and provide the solutions.

The first problem is on how to address moving monitoring queries over moving objects in a spatial network environment. We propose a distributed solution in which mobile users participate in monitoring their surrounding environment, hence reduce the workload on the server. A novel concept called *Edge Distance* is proposed to facilitate the network distance calculation on mobile users.

The second problem is about using a hybrid communication technique to reduce the communication cost in a location-based system. We notice that traditional location-based systems, even with the help of distributed computing, still suffer from high communication cost. Therefore, we propose to use a combination of on-demand communication and broadcast to lower the communication cost.

The third problem is on how to better protect user privacy with query l -diversity. As of now, existing techniques risk exposing user's query content to untrusted parties. We propose a new metric called query l -diversity, and design two cloaking techniques to achieve query l -diversity and location k -anonymity.

The fourth problem is about designing a hybrid three-tier architecture to reduce user privacy exposure. Other than performing anonymization, the trusted anonymizer server also serves as a broadcast server. The most popular query results are obtained based on mobile

user clusters and the results are broadcast through an air channel. Mobile user can tune into the channel and retrieve the query results for most of the time.

The fifth problem is on how to help service providers to process privacy enabled queries in a road network environment. We use the Road Network Embedding (RNE) technique to answer k -nearest neighbor queries, and extend the method to cover range queries and queries over private objects.

Location-based services/applications will get more and more exciting in the coming years. Smart phone sales are about to trump PC sales in 2011. With such a huge amount of mobile devices and heavy investment by wireless carriers to upgrade their network, we will see more opportunities in this field. On the other hand, privacy protection will become even more important once people realize that how easily they can become the victims of privacy breach. It's great that we are in this era that we can witness all the technology improvements and be part of it.

REFERENCES

- [1] Foursquare: Location-based social networking. <http://foursquare.com/>.
- [2] Loopt: Discover the world around you. <http://www.loopt.com/>.
- [3] The u.s. naval observatory gps timing operations. <http://tycho.usno.navy.mil/gps.html>.
- [4] Pankaj K. Agarwal, Lars Arge, and Jeff Erickson. Indexing moving points. In *PODS*, pages 175–186, 2000.
- [5] Bhuvan Bamba, Ling Liu, Péter Pesti, and Ting Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *WWW*, pages 237–246, 2008.
- [6] Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
- [7] Jing Cai, Tsutomu Terada, Takahiro Hara, and Shojiro Nishio. An adaptive control method in the hybrid wireless broadcast environment. In *MDM*, pages 86–93, 2007.
- [8] Ying Cai and Kien A. Hua. An adaptive query management technique for real-time monitoring of spatial regions in mobile database systems. In *IEEE International Performance Computing and Communications Conference*, pages 259–266. IEEE Computer Society, 2002.
- [9] Ying Cai, Kien A. Hua, and Guohong Cao. Processing range-monitoring queries on heterogeneous mobile objects. In *MDM*, pages 27–38, 2004.
- [10] Ying Cai, Kien A. Hua, Guohong Cao, and Toby Xu. Real-time processing of range-monitoring queries in heterogeneous mobile databases. *IEEE Trans. Mob. Comput.*, 5(7):931–942, 2006.

- [11] Hyung-Ju Cho and Chin-Wan Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, pages 865–876, 2005.
- [12] Chi-Yin Chow and Mohamed F. Mokbel. Enabling private continuous queries for revealed user locations. In *SSTD*, pages 258–275, 2007.
- [13] Chi-Yin Chow, Mohamed F. Mokbel, and Xuan Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *GIS*, pages 171–178, 2006.
- [14] Tai T. Do, Fuyu Liu, and Kien A. Hua. When mobile objects’ energy is not so tight: A new perspective on scalability issues of continuous spatial query systems. In *DEXA*, pages 445–458, 2007.
- [15] Tom Farley and Mark V. D. Hoek. Cellular telephone basics. http://www.privateline.com/mt_cellbasics/, 2006.
- [16] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.
- [17] Bugra Gedik and Ling Liu. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *EDBT*, pages 67–87, 2004.
- [18] Bugra Gedik and Ling Liu. Location privacy in mobile systems: A personalized anonymization model. In *ICDCS*, pages 620–629, 2005.
- [19] Bugra Gedik and Ling Liu. Mobieyes: A distributed location monitoring service using moving location queries. *IEEE Trans. Mob. Comput.*, 5(10):1384–1402, 2006.
- [20] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD Conference*, pages 121–132, 2008.

- [21] Gabriel Ghinita, Panagiotis Karras, Panos Kalnis, and Nikos Mamoulis. Fast data anonymization with low information loss. In *VLDB*, pages 758–769, 2007.
- [22] Gabriel Ghinita, Yufei Tao, and Panos Kalnis. On the anonymization of sparse high-dimensional data. In *ICDE*, pages 715–724, 2008.
- [23] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*, 2003.
- [24] Haibo Hu, Dik Lun Lee, and Victor C. S. Lee. Distance indexing on road networks. In *VLDB*, pages 894–905, 2006.
- [25] Haibo Hu, Dik Lun Lee, and Jianliang Xu. Fast nearest neighbor search on road networks. In *EDBT*, pages 186–203, 2006.
- [26] Haibo Hu, Jianliang Xu, and Dik Lun Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD Conference*, pages 479–490, 2005.
- [27] Bo Huang, Zhiyong Huang, Dan Lin, Hua Lu, Yaxiao Song, and Hongga Li. Itqs: An integrated transport query system. In *SIGMOD Conference*, pages 951–952, 2004.
- [28] Xuegang Huang, Christian S. Jensen, and Simonas Saltenis. The islands approach to nearest neighbor querying in spatial networks. In *SSTD*, pages 73–90, 2005.
- [29] Tomasz Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air: Organization and access. *IEEE Trans. Knowl. Data Eng.*, 9(3):353–372, 1997.
- [30] ITU. About mobile technology and imt-2000. <http://www.itu.int/osg/spu/imt-2000/technology.html>, 2005.
- [31] Christian S. Jensen, Jan Kolárvr, Torben Bach Pedersen, and Igor Timko. Nearest neighbor queries in road networks. In *GIS*, pages 1–8, 2003.

- [32] Panos Kalnis, Gabriel Ghinita, Kyriakos Mouratidis, and Dimitris Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Trans. Knowl. Data Eng.*, 19(12):1719–1733, 2007.
- [33] Ali Khoshgozaran and Cyrus Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD*, pages 239–257, 2007.
- [34] Hidetoshi Kido, Yutaka Yanagisawa, and Tetsuji Satoh. Protection of location privacy using dummies for location-based services. In *ICDE Workshops*, page 1248, 2005.
- [35] Mohammad R. Kolahdouzan and Cyrus Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *STDBM*, pages 33–40, 2004.
- [36] Mohammad R. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.
- [37] George Kollios, Dimitrios Gunopulos, and Vassilis J. Tsotras. On indexing mobile objects. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania*, pages 261–272. ACM Press, 1999.
- [38] Hermann Kopetz and Wilhelm Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Trans. Computers*, 36(8):933–940, 1987.
- [39] Wei-Shinn Ku, Roger Zimmermann, Wen-Chih Peng, and Sushama Shroff. Privacy protected query processing on spatial networks. In *ICDE Workshops*, pages 215–220, 2007.
- [40] Wei-Shinn Ku, Roger Zimmermann, and Haixun Wang. Location-based spatial queries with data sharing in wireless broadcast environments. In *ICDE*, pages 1355–1359, 2007.

- [41] Wei-Shinn Ku, Roger Zimmermann, and Haixun Wang. Location-based spatial query processing in wireless broadcast environments. *IEEE Trans. Mob. Comput.*, 7(6):778–791, 2008.
- [42] Wang-Chien Lee and Baihua Zheng. Dsi: A fully distributed spatial index for location-based wireless broadcast services. In *ICDCS*, pages 349–358, 2005.
- [43] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*, pages 49–60, 2005.
- [44] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. In *FOCS*, pages 577–591, 1994.
- [45] Fuyu Liu, Kien A. Hua, and Ying Cai. Query l-diversity in location based services. In *The International Workshop on Privacy-Aware Location-based Mobile Services (PALMS)*, 2009.
- [46] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.
- [47] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *PODS*, pages 223–228, 2004.
- [48] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. The new casper: Query processing for location services without compromising privacy. In *VLDB*, pages 763–774, 2006.
- [49] Mohamed F. Mokbel, Xiaopeng Xiong, and Walid G. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD Conference*, pages 623–634, 2004.

- [50] Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD Conference*, pages 634–645, 2005.
- [51] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras, and Yufei Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *IEEE Trans. Knowl. Data Eng.*, 17(11):1451–1464, 2005.
- [52] Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, and Nikos Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, pages 43–54, 2006.
- [53] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.
- [54] Sunil Prabhakar, Yuni Xia, Dmitri V. Kalashnikov, Walid G. Aref, and Susanne E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers*, 51(10):1124–1140, 2002.
- [55] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, 2000.
- [56] Pierangela Samarati. Protecting respondents’ identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.
- [57] Cyrus Shahabi, Mohammad R. Kolahdouzan, and Mehdi Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. In *ACM-GIS*, pages 94–10, 2002.

- [58] Shashi Shekhar and Duen-Ren Liu. Ccam: A connectivity-clustered access method for networks and network computations. *IEEE Trans. Knowl. Data Eng.*, 9(1):102–119, 1997.
- [59] Bernhard Sterzbach. Gps-based clock synchronization in a mobile, distributed real-time system. *Real-Time Systems*, 12(1):63–75, 1997.
- [60] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
- [61] Yufei Tao and Dimitris Papadias. Time-parameterized queries in spatio-temporal databases. In *SIGMOD Conference*, pages 334–345, 2002.
- [62] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku. Distributed continuous range query processing on moving objects. In *DEXA*, pages 655–665, 2006.
- [63] Wei Wu, Wenyuan Guo, and Kian-Lee Tan. Distributed processing of moving k-nearest-neighbor query on moving objects. In *ICDE*, pages 1116–1125, 2007.
- [64] Yuni Xia, Sunil Prabhakar, Shan Lei, Reynold Cheng, and Rahul Shah. Indexing continuously changing data with mean-variance tree. In *SAC*, pages 1125–1132, 2005.
- [65] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, pages 139–150, 2006.
- [66] Zhen Xiao, Jianliang Xu, and Xiaofeng Meng. p-sensitivity: A semantic privacy-protection model for location-based services. In *The International Workshop on Privacy-Aware Location-Based Mobile Services (PALMS 2008)*, 2008.
- [67] Xiaopeng Xiong, Mohamed F. Mokbel, and Walid G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *ICDE*, pages 643–654, 2005.

- [68] Jianliang Xu, Baihua Zheng, Wang-Chien Lee, and Dik Lun Lee. Energy efficient index for querying location-dependent data in mobile broadcast environments. In *ICDE*, pages 239–252, 2003.
- [69] Jianliang Xu, Baihua Zheng, Wang-Chien Lee, and Dik Lun Lee. The d-tree: An index structure for planar point queries in location-based wireless services. *IEEE Trans. Knowl. Data Eng.*, 16(12):1526–1542, 2004.
- [70] Toby Xu and Ying Cai. Location anonymity in continuous location-based services. In *GIS*, page 39, 2007.
- [71] Toby Xu and Ying Cai. Exploring historical location data for anonymity preservation in location-based services. In *INFOCOM*, 2008.
- [72] Man Lung Yiu, Christian Jensen, Xuegang Huang, and Hua Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE*, 2008.
- [73] Man Lung Yiu, Christian S. Jensen, Xuegang Huang, and Hua Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE*, pages 366–375, 2008.
- [74] Xiaohui Yu, Ken Q. Pu, and Nick Koudas. Monitoring k-nearest neighbor queries over moving objects. In *ICDE*, pages 631–642, 2005.
- [75] Baihua Zheng, Wang-Chien Lee, and Dik Lun Lee. Spatial queries in wireless broadcast systems. *Wireless Networks*, 10(6):723–736, 2004.
- [76] Manli Zhu, Dik Lun Lee, and Jun Zhang. k-closest pair query monitoring over moving objects. In *MDM*, page 14, 2006.