Authentication of Moving Range Queries

Duncan Yung Eric Lo Man Lung Yiu*
Department of Computing
Hong Kong Polytechnic University
{cskwyung, ericlo, csmlyiu}@comp.polyu.edu.hk

ABSTRACT

A moving range query continuously reports the query result (e.g., restaurants) that are within radius r from a moving query point (e.g., moving tourist). To minimize the communication cost with the mobile clients, a service provider that evaluates moving range queries also returns a safe region that bounds the validity of query results. However, an untrustworthy service provider may report incorrect safe regions to mobile clients. In this paper, we present efficient techniques for authenticating the safe regions of moving range queries. We theoretically proved that our methods for authenticating moving range queries can minimize the data sent between the service provider and the mobile clients. Extensive experiments are carried out using both real and synthetic datasets and results show that our methods incur small communication costs and overhead.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms, Performance

Keywords

authentication, moving queries

1. INTRODUCTION

Moving spatial queries [33, 14] continuously report updated query result to a mobile client and they have numerous mobile applications. For example, a moving range query continuously reports all tourist attractions that are within 3 km from a moving car.

Location-based service providers (LBS) that offer moving querying services return mobile users the query result and a corresponding *safe region* [33, 14]. Given a moving query point q, its safe region is a region where the result of q remains unchanged as long as q moves within it. In other words, it allows a mobile user to issue a new query to the LBS (to get the latest query result) only when the user leaves the safe region, Thus, safe region is a powerful optimization for significantly reducing the communication frequency between the user and the service provider.

The query results and safe regions returned by LBS, however, may not always be accurate. For instance, a hacker may infiltrate the LBS's servers [25] so that results of queries all include a particular location (e.g., the White House). Furthermore, the LBS may be self-compromised and thus ranks sponsored facilities higher than actual query result. The LBS may also return an overly large safe region to the clients for the sake of saving computing resources and communication bandwidth [22, 16, 30]. On the other hand, the LBS may opt to return overly small safe regions so that the clients have to request new safe regions more frequently, if the LBS charges fee for each request, or if the LBS wishes to boost its request counts — a figure that could influence its advertisement revenue.

Recently, techniques for authenticating query result have received significant attentions, e.g., the authentication of relational queries [15, 10, 27], data stream queries [11, 20], text similarity queries [16], static spatial queries [28, 19], and shortest path queries [30]. Most authentication techniques [10, 27, 11, 20, 28, 19, 16, 30] are based on *Merkle tree* [12], which is an *authenticated data structure* (ADS) that is built on the dataset. Recently, Yang et al. [28] developed an authenticated data structure called Merkle R-tree (MR-tree) for authenticating spatial queries. The issue of authenticating *moving spatial queries*, however, has not been addressed yet. Existing techniques for authenticating static spatial queries such as [28, 19, 6] cannot help in authenticating moving spatial queries because their authentication target is the *query result*, which *is a subset of the dataset*, whereas the authentication targets of moving queries include both the query result and the *safe region* — the latter is a geometric entity that is dynamically computed by the LBS at run-time and is **not** part of the dataset. Since a safe region is defined based on both points in the query result as well as points not in the query result, the missing of a non-result point during the authentication process may also fail the authentication of the safe region.

This paper is devoted to the authentication of moving range queries. The specific technical contributions include:

1. The design of *verification objects* $\mathcal{VO}$ specific for authenticating the safe regions (and also the result) of moving range queries. Note that the existing notion of verification objects [10, 28, 16, 30, 31] cannot be used to authenticate safe regions of moving range queries. Algorithms for constructing and verifying those verification objects during moving range query processing are included in this paper as well.

2. Communication optimization techniques that allow a mobile client to refresh its safe region without communicating

*Supported by grant PolyU 5333/10E from Hong Kong RGC.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

with the LBS even when the client leaves its current safe region. These techniques are useful to reduce the communication overhead during authentication-enabled moving spatial query processing.

3. We introduce a new notion called $\mathcal{VO}$*-optimal*, to describe authentication techniques that put the minimum data points and MR-tree entries into the $\mathcal{VO}$. We show that our solutions achieve such optimality.

4. An extensive experimental study on real data and synthetic data to study the efficiency of our proposed methods.

The rest of this paper is organized as follows. We discuss related work in Section 2. We present the framework for authenticating moving queries in Section 3. In Section 4, we present our $\mathcal{VO}$-optimal methods for authenticating moving range queries. The experimental study is presented in Section 5. Finally, we conclude this paper in Section 6.

2. RELATED WORK

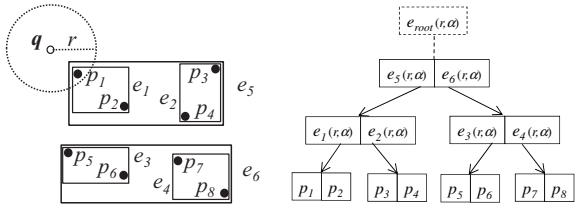
Query authentication In the literature, most authentication techniques [10, 27, 11, 20, 28, 19, 16, 30] are based on *Merkle tree* [12] with the public key infrastructure [21]. Merkle tree is an *authenticated data structure* (ADS) that is built on the dataset. Digests of nodes in the tree are first recursively computed from the leaf level to the root level.[1] Then, the signature of the dataset is obtained by signing the root digest using the data owner's private key. *Signature aggregation* [15, 17] is an alternative of Merkle tree. It works by having a signature for each tuple in the dataset. Recently, Yi et al. [29] propose a probabilistic approach for authenticating aggregation queries; and Kundu et al. study the authentication of trees [8] and graphs [9] without revealing any data objects beyond the user's access rights.

To authenticate spatial queries, Yang et al. [28] develops an authenticated data structure (ADS) called Merkle R-tree (MR-tree) based on R*-tree [1] and Merkle tree [12]. Figure 1b shows an MR-tree for the dataset shown in Figure 1a. A leaf entry p_i stores a data point. A non-leaf entry e_i stores a rectangle $e_i.b$ and a digest $e_i.\alpha$, where $e_i.b$ is the minimum bounding rectangle of its child node, and $e_i.\alpha$ is the digest of the concatenation (denoted by $|$) of binary representation of all entries in its child node. For instance, $e_1.\alpha = h(p_1|p_2)$, $e_5.\alpha = h(e_1|e_2)$, and $e_{root}.\alpha = h(e_5|e_6)$. The *root signature* is generated by signing the digest of the root node $e_{root}.\alpha$ using the data owner's private key.

Consider the range query q with radius r in Figure 1. Its result is p_1. In order to let the client verify the correctness of the range results, the LBS utilizes the MR-tree (provided by the data owner) to generate a verification object $\mathcal{VO}$. First, it defines a circular verification region $\odot(q, r)$ with center q and radius r. Then, it computes the $\mathcal{VO}$ by a depth-first traversal of the MR-tree, with the following conditions: (i) if a non-leaf entry e does not intersect $\odot(q, r)$, then e is added to the $\mathcal{VO}$ (e.g., e_2, e_6) and its subtree will not be visited; (ii) data points in any visited leaf node are added into the $\mathcal{VO}$ (e.g., p_1, p_2). In this example, we have $\mathcal{VO} = \{\{\{p_1, p_2\}, e_2\}, e_6\}$, where $\{$ and $\}$ are tokens for marking the start and end of a node.

Upon receiving the $\mathcal{VO}$, the client first checks the correctness of the $\mathcal{VO}$ by reconstructing the digest of root of the MR-tree from the $\mathcal{VO}$ and then verifying it against the root signature using the data owner's public key. If the verification is successful, the client next finds the range result directly from the data points extracted from the $\mathcal{VO}$ (ignoring the non-leaf entries). Then, the client defines the verification region $\odot(q, r)$ and checks whether every non-leaf entry in the $\mathcal{VO}$ satisfies $e.b \cap \odot(q, r) = \emptyset$. If so, the client can assure that the computed range result is correct. The reason is that, if a non-leaf entry e in the $\mathcal{VO}$ does not intersect $\odot(q, r)$, that means all the points in e cannot become the range result of q.

[1] A secure-hash function is often used to compute a fix-length digest.



(a) data points on a plane (b) Merkle R-tree (T_D)

Figure 1: Authentication of range queries

In this paper, we use MR-tree because of its popularity and low construction cost, and our concept of $\mathcal{VO}$-optimality is also based on MR-tree. MR*-tree is an extension of MR-tree [28], where each node is embedded with a conceptual Merkle KD-tree defined on entries in the node. If a query region intersects with only one side of a split axis (in a KD-tree), then it skips inserting the entries on the other side into the $\mathcal{VO}$. This technique can reduce the number of entries in the $\mathcal{VO}$ at the cost of higher construction and computation time. Reference [13] discusses another spatial ADS called PMKd-tree, which separates the query processing part and the $\mathcal{VO}$ construction part in order to obtain better query efficiency and smaller verification objects. To the best of our knowledge, we are the first to study the authentication of moving range queries. Existing spatial authentication techniques [28, 19, 6] cannot help in authenticating moving queries because they focus on *static* queries, so that the authentication targets (i.e., the query results) are part of the dataset. In contrast, authentication of moving queries require authenticating both the query results (part of the dataset) and the safe regions (*not* part of the dataset).

Our recent work [31] examined how to authenticate the safe regions of moving kNN queries. However, the safe regions for kNN queries (e.g., convex polygons) are different from those for range queries (e.g., set unions/differences of circles). This renders the specific techniques in [31] inapplicable to our problem here. Also, we will prove that our method is $\mathcal{VO}$-optimal, in which [31] did not introduce this notion yet.

Moving query processing In moving query processing, [24] computes the nearest neighbors for each possible query point on a given line segment, whereas [7] studies how to maintain the user's future kNN upon a change of the user's velocity. Both [24] and [7] model the user's movement as a linear function. When the user's future movement is unknown, the buffering approach [23] and the safe region approach [33, 14] are more appropriate for efficient moving query processing.

In the buffering approach [23], the LBS returns users the original result points and some additional result points to constitute a *buffer region*. While moving within the buffer region, the client's latest result can be recomputed locally from the result of the previous query. However, it is not easy to tune the size of the buffer region in practice, and that value actually influences the communication frequency and the number of objects sent per communication.

In the safe region approach, the LBS reports a safe region [33, 14, 3] for the query result, such that the result remains unchanged while the user moves within the safe region. Unlike the buffering approach, this approach does not require the client to compute

result locally. This approach reduces both the communication frequency and the computational overhead for the user.

Finally, note that the solutions in [23, 24, 7, 14] focus on moving kNN queries rather than moving range queries. The safe regions of moving range queries were studied in [33, 3]: rectangular range queries [33] and circular range queries [3].

3. BASELINE AND FRAMEWORK

Following [33, 14, 3], in this paper, we consider a generic problem setting in which q is a moving object whose future locations cannot be predicted in advance. In a moving query environment, the question is when and where the client should (re-)issue query in order to get the most updated query range result as q moves.

Baselines for moving query authentication A brute-force method is that the client periodically issues a query to the server for every T time units. The correctness of the results can then be authenticated using an MR-tree. However, there is no way to guarantee that result is always up-to-date even when a very small T is used, rendering this method impractical.

To authenticate moving spatial queries, a baseline method uses the buffering approach to compute the query result and then authenticate the results by using an MR-tree. Specifically, a client issues a range query with a range Δr larger than the original query range r. Again, let q_{last} be the last location sent to server. As long as the current location of q moves within the buffer region $\odot(q_{last}, \Delta r)$, the latest query result can be derived from the result of the last query because any ranges that have center within $\odot(q_{last}, \Delta r)$ and radius r are totally inside $\odot(q_{last}, r + \Delta r)$.

Although simple, the buffering approach requires finding the optimal value of Δr. A small Δr leads to more frequent communication, thereby increasing the communication cost. A large Δr, unfortunately, also increases the communication cost because the size of the $\mathcal{VO}$ increases.

Our safe region approach The buffering approach is not a true safe region approach because even when staying within the buffer region the client is still required to recompute result locally. Our approach is to return a safe region for the query result, such that the result remains unchanged as long as the user's location q stays within the safe region. This approach helps reducing the communication frequency between the LBS and the client. However, for the various reasons we mentioned in the introduction, the LBS may return incorrect safe regions, rendering the user's future result incorrect. Our goal is thus to devise methods for the client to verify the correctness of the safe region returned by LBS.

Framework for moving query authentication Figure 2 illustrates the framework for answering moving spatial queries that supports query correctness verification. A map provider (e.g., the government's land department, NAVTEQ[2] and TeleAtlas[3]) collects points-of-interests into a spatial dataset. It builds the MR-tree [28] of the dataset and signs the digest of the root node, before distributing/selling it to a service provider (i.e., LBS). Initially, a mobile user downloads the root signature from the LBS and the map provider's public key from a certificate authority (e.g., VeriSign). Afterwards, the user sends its location to the LBS, and obtains the query result, the safe region, and the $\mathcal{VO}$. The correctness of the query result and the safe region can be verified at the client by using the received $\mathcal{VO}$, the root signature and the map provider's public key. The client needs to issue a query to the LBS again only when it leaves its safe region.

[2]NAVTEQ Maps and Traffic. http://www.navteq.com
[3]TeleAtlas Digital Mapping. http://www.teleatlas.com

The spatial dataset is expected to have infrequent updates (e.g., monthly map updates). In case the user requires *fresh* results (i.e., obtained from the latest datasets), the map provider could follow [10] to include a timestamp in the root signature of the tree.

Our adversary model is the same as in [28]. Except the data owner's private key, adversaries are assumed to know all other information, e.g., the data owner's public key, the secure-hash function, the Merkle R-tree, its root signature, and our authentication algorithms. The security is guaranteed by the Merkle R-tree [28].

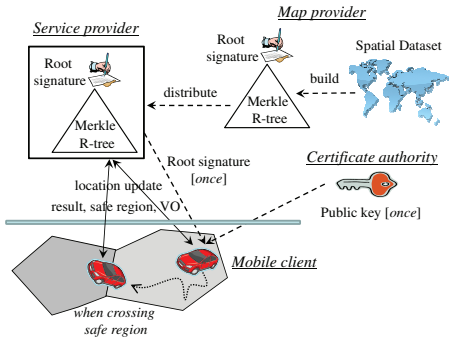


Figure 2: Moving query authentication

4. PROPOSED SOLUTION

There are two main types of range query, *rectangular range query* [5, 6] and *circular range query* [4, 6]. The former specifies the query range as a rectangle whereas the latter specifies the query range as a circle. In this paper, we focus on the authentication of moving circular range queries. Hereinafter, we use the term range query to denote circular range query. A summary of notation used in this paper is given in Table 1.

In the following, we first introduce some basic definitions for our problem. Then, we present our Arc-Based (AB) method for constructing $\mathcal{VO}$ for moving range query authentication (Section 4.2). As we will elaborate later, the safe region for a moving range query is constituted by a set of arcs. Hence, our AB method exploits this property to construct a compact $\mathcal{VO}$. Furthermore, we can prove that our AB method is $\mathcal{VO}$-optimal, i.e., it puts the minimum data points and MR-tree entries into the $\mathcal{VO}$ (Section 4.3). Afterwards, we analyze the size of the $\mathcal{VO}$ constructed by this method (Section 4.4). At the end, we present a method to further reduce the communication frequency (Section 4.5) by reusing the $\mathcal{VO}$ when the client crosses a safe region.

4.1 Preliminary and Definitions

A (circular) range query is a circle $\odot(q, r)$, where q is a query point and r is a radius (in Euclidean distance). A data point p is in the result set S if its distance from q is less than or equal to r, i.e., $dist(q, p) \leq r$. In contrast, a data point p is not in the result S if its distance from q is greater than r, i.e., $dist(q, p) > r$. In Figure 3(a), the bold circle $\odot(q, r)$ is a circular range query with center q and radius r. Data points g_1, g_2, g_3, and p_1 are inside $\odot(q, r)$, so they are result points and in S. Since g_4 is outside $\odot(q, r)$, g_4 is a not a result point and not in S.

When a client moves, its range query result may change. In order to get the updated result, basically it has to submit a new query to the LBS using its updated location q'. To minimize the communication frequency and the communication cost, we adopt the safe region approach. When the server receives a range query, it returns the client a safe region SR in addition to the query result S.

Table 1: Summary of Frequently Used Symbols

Symbol	Meaning
D	the dataset
r	radius
$\overset{\frown}{\text{AB}}$	arc AB
$\odot(c, r)$	circular region with center c and radius r
S	range query result
$SR(S, D)$	safe region
g_i	a guard object
g_i^{arc}	a border of safe region contributed by g_i
A_{end}^1, A_{end}^2	two ending points of g_i^{arc}
G	a set of guard object
$\lhd(c, r)$	a sector with center c and radius r
$rRVR$	query result verification region
$GOVR_{g_i}$	guard object verification region
$GOEPC_{g_i}$	guard object end point circle
$GOSD_{g_i}$	guard object sector difference
$\circlearrowleft$	circumference of $\bigcup_{g_i \in G} GOVR_{g_i}$
$rSRVR$	range safe region verification region
Π	verification region

Therefore, the client does not need to issue a new query to the LBS as long as it moves within SR. The safe region for moving range queries [4] is stated below.

Definition 1. **Safe Region of Range Query (Ref. [4])** The safe region $SR(S, D)$ of a circular range query is defined as:

$$SR(S, D) = \bigcap_{p_i \in S} \odot(p_i, r) - \bigcup_{p_j \in D - S} \odot(p_j, r)$$

where S is the set of result points of range query $\odot(q, r)$ and D is the dataset.

In Figure 3(a), the light grey region is the safe region $SR(S, D)$ for the range query result $S = \{g_1, g_2, g_3, p_1\}$. It is formed by $\odot(g_1, r) \cap \odot(g_2, r) \cap \odot(g_3, r) \cap \odot(p_1, r)$, excluding $\odot(g_4, r)$.

Note that the safe region $SR(S, D)$ is enclosed by four arcs: $\overset{\frown}{\text{ab}}$, $\overset{\frown}{\text{bd}}$, $\overset{\frown}{\text{dc}}$, and $\overset{\frown}{\text{ca}}$, which are originated from circles $\odot(g_1, r)$, $\odot(g_4, r)$, $\odot(g_2, r)$, and $\odot(g_3, r)$, respectively. In other words, $\odot(p_1, r)$ does not contribute to the final safe region $SR(S, D)$ (in fact $SR(S, D)$ is completely inside $\odot(p_1, r)$). So, [4] distinguishes the set of data points G that are necessary for representing the safe region, which are called *guard objects*, from the others.

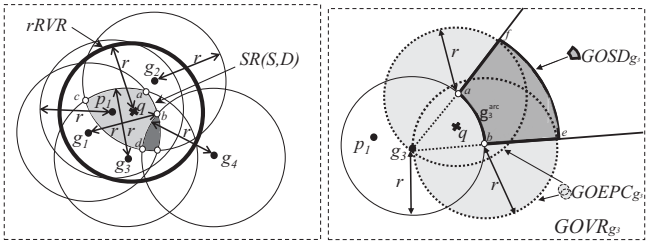
Definition 2. **Guard Object (Ref. [4])** Given a point g_i, g_i is a guard object if part of $\odot(g_i, r)$, which is an arc g_i^{arc}, can contribute to the final safe region $SR(S, D)$.

In Figure 3(a), points g_1, g_4, g_2, and g_3 are guard objects because their corresponding arcs $\overset{\frown}{\text{ab}}$, $\overset{\frown}{\text{bd}}$, $\overset{\frown}{\text{dc}}$, and $\overset{\frown}{\text{ca}}$ form the safe region. In contrast, point p_1 is not a guard object.

The challenges of verifying the correctness of the safe region SR can be explained using Figure 3(a). Assume that the LBS needs to evaluate a range query $\odot(q, r)$ and its safe region (i.e., the light grey region). By Definition 2, the LBS can represent the safe region using four guard objects g_1, g_2, g_3, and g_4. Having these guard objects, the client is able to construct the safe region.

Suppose that the LBS intentionally omits guard object g_4 and only sends guard objects g_1, g_2, and g_3 to the client. In this case, the client will get an incorrect (larger) safe region, which is the union of the light grey region and the dark grey region. The challenge here is that the client cannot notice that one guard object (g_4) is missing. Note that all guard objects g_1, g_2, and g_3 are originated from the dataset D (thus they pass the data correctness checking). However, the client cannot determine whether the set of guard objects provided by the LBS is complete. Similarly, the LBS may also report a smaller safe region by returning fake objects as guard objects. Again such case is easy to be detected by checking the root signatures.

By Definition 1, $SR(S, D)$ is constructed by *all data points* in D. So, a brute-force solution is to return the whole dataset D to the client so that the client is guaranteed to compute $SR(S, D)$ correctly. Our goal is to design efficient methods to construct verification objects $\mathcal{VO}$ for verifying the correctness of the safe region, yet the size of $\mathcal{VO}$ should be as small as possible.



(a) safe region and rRVR (b) $GOVR$, $GOEPC$, and $GOSD$

Figure 3: Illustration of regions

4.2 Arc-Based Method

Our Arc-Based method constructs a compact $\mathcal{VO}$ for authenticating a moving range query. Its idea is to exploit the arcs of a safe region to construct a $\mathcal{VO}$ with minimal size. The method consists of a server algorithm and a client algorithm. At the server side, everything that is necessary for verifying the range query result and the safe region is put into $\mathcal{VO}$ and sent to the client. At the client side, the correctness of the query result and the safe region is verified based on the data stored in the $\mathcal{VO}$.

Algorithm 1 Arc-Based Method (Server)

Receive from client: (Query point q, Integer r)
Using MR-tree T_D (on dataset D)
1: S := compute the result points of q from the tree T_D
2: compute $SR(S, D)$ from the tree T_D (using the method of [4])
3: $rRVR := \odot(q, r)$
4: G := set of guard objects g_i
5: $GOVR_{g_i} := (\lhd(g_i, 2r) - \lhd(g_i, r)) \cup \odot(p_e^1, r) \cup \odot(p_e^2, r)$
6: $rSRVR := \bigcup_{g_i \in G} GOVR_{g_i} - \circlearrowleft$
7: $\Pi := rRVR \cup rSRVR$
8: $\mathcal{VO}$:= DepthFirstRangeSearch(T_D.root, Π)
9: **send** $\mathcal{VO}$ to the client

4.2.1 Server Algorithm

Algorithm 1 is the pseudo-code of the server algorithm. Upon receiving the user location q and the radius r, the server computes the result points from an MR-tree T_D (Line 1). Then, it computes the safe region by using safe region computation method in [4] efficiently (Line 2).

Next, it defines a verification region Π so as to identify data points that are useful for verifying the range query result and the safe region and put them into VO. More specifically, the verification region Π is defined as the union of (i) the *range query result verification region* ($rRVR$) and (ii) the *range query safe region verification region* ($rSRVR$).

Definition 3. **Range query result verification region ($rRVR$).** The *range query result verification region* ($rRVR$) is defined as the circular region $\odot(q, r)$ at the current query location q with radius r.

In Figure 3(a), the bold circle $\odot(q, r)$ is $rRVR$.

The definition of the range query safe region verification region $rSRVR$ is quite complicated. Therefore, we first state some preliminary definitions.

Definition 4. **Guard object end points circles ($GOEPC_{g_i}$)** Given a guard object g_i, the *guard object end points circles* of g_i ($GOEPC_{g_i}$) is the union of $\odot(A^1_{end}, r)$ and $\odot(A^2_{end}, r)$, where A^1_{end} and A^2_{end} are two end points of g_i^{arc}, an arc that contributes to the final safe region $SR(S, D)$.

Definition 5. **Guard object sector difference ($GOSD_{g_i}$)** Given a guard object g_i, the *guard object sector difference* of g_i ($GOSD_{g_i}$) is the difference between a sector $\lhd(g_i, 2r)$ with center g_i and radius $2r$, and a sector $\lhd(g_i, r)$ with the same center g_i but with radius r. The angle subtended by the sectors are determined by the end points A^1_{end} and A^2_{end} described in Definition 4.

In Figure 3(b), given a guard object g_3, and assume that g_3^{arc} is the arc that contributes to the final safe region $SR(S, D)$. The two end points of g_3^{arc} are a and b. So, the guard object end points circles of g_3 , i.e., $GOEPC_{g_3}$, is the union of circles $\odot(a, r)$ and $\odot(b, r)$. Note that $GOSD_{g_3}$ is the dark grey region $abef$.

With the definitions of guard object end points circles and guard object sector difference, we now define the verification region of a guard object as follows.

Definition 6. **Guard object verification region** ($GOVR_{g_i}$)

$$GOVR_{g_i} = GOEPC_{g_i} \cup GOSD_{g_i}$$

Figure 3(b) shows the verification region of guard object g_3, which is the union of the light grey region and the dark grey region.

Now, we define range query safe region verification region ($rSRVR$) as follows.

Definition 7. **Range query safe region verification region ($rSRVR$)**
The *range query safe region verification region* ($rSRVR$) is defined as the union of the $GOVR_{g_i}$ of each guard object g_i in G, except its circumference $\circlearrowleft$:

$$rSRVR = \bigcup_{g_i \in G} GOVR_{g_i} - \circlearrowleft$$

where G is the set of guard objects and $\circlearrowleft$ is the circumference of $\bigcup_{g_i \in G} GOVR_{g_i}$, i.e., $\circlearrowleft = \{z \in \mathbb{R}^2 | \exists$ location l on g_i^{arc}, $dist(l, z) = r, \forall$ locations l' on g_i^{arc}, $dist(l', z) \geq r\}$

Finally, the verification region Π for authenticating (moving) range query is defined as the union of the range query result verification region ($rRVR$) and the range query safe region verification region ($rSRVR$).

Definition 8. **Verification region** Π

$$\Pi = rRVR \cup rSRVR$$

With the verification region Π for a moving range query clearly defined, the $\mathcal{VO}$ can be constructed by a depth-first traversal of the MR-tree (Line 8 in Algorithm 1). Points inside Π are put in the $\mathcal{VO}$ and the rest are put in the $\mathcal{VO}$ as some non-leaf entries.

We now prove that any point p^* outside the verification region Π *cannot* alter the range query result S and its corresponding safe region $SR(S, D)$ and thus it is safe to exclude them in the $\mathcal{VO}$.

THEOREM 1. **[Points p^* outside Π *cannot* alter the query result and the safe region $SR(S, D)$]**

PROOF: To prove the theorem, we prove that (i) any point p^* outside $rRVR$ *cannot* be range query result (Lemma 1), and (ii) any point p^* outside $rSRVR$ *cannot* alter the safe region (Lemma 2). Since the verification region Π is the union of $rRVR$ and $rSRVR$, the theorem is proved if Lemma 1 and Lemma 2 hold.

LEMMA 1. **[Points outside $rRVR$ are not range query result]** *We have $\forall$ p^* outside $rRVR$, $p^* \notin S$, where S is the range query result set.*

Proof: If p^* is outside $rRVR$, then p^* is not inside the query range. Thus, $p^* \notin S$. □

LEMMA 2. **[Points outside $rSRVR$ cannot alter safe region $SR(S, D)$]** *If p^* is outside $rSRVR$, then $SR(S, D) = SR(S, D \backslash \{p^*\})$.*

Proof: Recall that the safe region $SR(S, D)$ is enclosed by a set of arcs g_i^{arc}. Therefore, if p^* is outside $GOVR_{g_i}$ for all $g_i \in G$, Lemma 3 from [4] implies that $\odot(p^*, r)$ does not intersect any arc g_i^{arc} and thus $\odot(p^*, r)$ does not overlap the safe region $SR(S, D)$. With that, Lemma 2 is proven.

LEMMA 3. *If p^* is outside $GOVR_{g_i}$, then $\odot(p^*, r)$ does not intersect arc g_i^{arc}. (Ref. [4])*

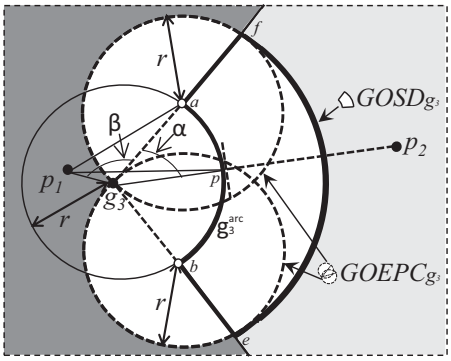


Figure 4: Example for Lemma 3

Now, with Lemma 1 and Lemma 2, Theorem 1 is proven.

The shape of verification region Π here is irregular (not a simple shape like circle). For efficient implementation, the server actually does not render the complex shape of the verification region Π. Instead, we can check whether the following condition holds during the tree traversal (Line 8 in Algorithm 1), and if it does not, we add e into the $\mathcal{VO}$:

$$(mindist(e, q) \leq r) \vee \bigvee_{A^1_{end} \in g_i^{arc}} (mindist(e, A^1_{end}) < r) \vee \bigvee_{A^2_{end} \in g_i^{arc}} (mindist(e, A^2_{end}) < r) \vee \bigvee_{g_i \in G} overlap(GOSD_{g_i}, e) \quad (1)$$

The first term of Equation 1 checks whether the minimum distance from non-leaf entry e to q is smaller than or equal to r. If the distance is smaller than or equal to r, e overlaps or touches $rRVR$. The second and third terms check whether the minimum distance from e to every end points A^1_{end} and A^2_{end} of arc g_i^{arc} is smaller than r. If the distance is smaller than r, e overlaps with $GOEPC_{g_i}$. The last term checks whether e overlaps any $GOSD_{g_i}$.

4.2.2 Client Algorithm

Algorithm 2 is the pseudo-code of the client algorithm. Upon receiving the $\mathcal{VO}$, the client first reconstructs the root digest from the $\mathcal{VO}$ (Line 1) and verifies it against the MR-tree root signature signed by the data owner (Line 2). If the verification is successful, the $\mathcal{VO}$ is guaranteed to contain only entries from the original MR-tree (Line 3). After that, it proceeds to verify the correctness of result points and the safe region provided by the $\mathcal{VO}$. It extracts from the $\mathcal{VO}$ (i) a set D' of data points, and (ii) a set R' of non-leaf entries, (Line 4-5) and then computes the result point set S' from D' (Line 6).

S' is correct if every non-leaf entry of R' does not intersect $rRVR$ (Lines 7–8). If the result points are correct, the client can proceed to check the safe region. The client next computes the safe region $SR(S', D')$ from D' (Line 9). The client can ensure that $SR(S', D')$ is correct if every non-leaf entry does not intersect Π (Lines 10–14). If so, the client can treat the result points and the safe region as correct (Line 15).

Algorithm 2 Arc-Based Method (Client)

Receive from server: (Verification Object $\mathcal{VO}$)
1: h'_{root} := reconstruct the root digest from $\mathcal{VO}$
2: verify h'_{root} against the MR-tree root signature
3: **if** h'_{root} is correct **then**
4: D' := the set of data points extracted from $\mathcal{VO}$
5: R' := the set of non-leaf entries extracted from $\mathcal{VO}$
6: S' := compute the result points of q from D'
7: $rRVR := \odot(q, r)$
8: **if** $\forall\ e \in R',\ e \cap rRVR = \emptyset$ **then** ▷ authenticate query result
9: $\mathcal{V}$:=compute $SR(S', D')$ ▷ authenticate safe region
10: G := set of guard objects g_i
11: $GOVR_{g_i} := (\lhd(g_i, 2r) - \lhd(g_i, r)) \cup \odot(A^1_{end}, r) \cup \odot(A^2_{end}, r)$
12: $rSRVR := \bigcup_{g_i \in G} GOVR_{g_i} - \circlearrowleft$
13: $\Pi := rRVR \cup rSRVR$
14: **if** $\forall\ e \in R', e \cap \Pi = \emptyset$ **then**
15: **return** range query result S' and safe region SR
16: **return** authentication failed

Up to now, we have introduced the server algorithm and the client algorithm of the Arc-Based (AB) method. Now, we prove that the client can verify the range query result and the safe region by using the $\mathcal{VO}$ constructed by this method.

LEMMA 4. **[Client can verify the correctness of range result]**

Proof: Direct results from [28]. □

LEMMA 5. **[Client can verify the correctness of safe region]** *Following the Arc-Based (AB) method, a client can verify that the constructed safe region $SR(S', D')$ equals to the correct safe region $SR(S, D)$, i.e., $SR(S', D') = SR(S, D)$.*

Proof: First, from Lemma 4, we know $S' = S$. $---(\blacklozenge)$

Next, line 1 and line 2 of Algorithm 2 ensure that all data points $p \in D'$ and all non-leaf entries $e \in R'$ in the $\mathcal{VO}$ are originated from the data owner [28]. Then, the client algorithm checks whether any non-leaf entry $e \in R'$ intersect $rSRVR$, and if no, it regards the safe region $SR(S', D')$ as correct, i.e., $SR(S', D') = SR(S, D)$. We prove its correctness by contradiction.

Assume the client regards $SR(S', D') = SR(S, D)$ even when a non-leaf entry $e \in R'$ in $\mathcal{VO}$ intersects $rSRVR$. When e intersects $rSRVR$, it is possible that $\exists p' \in e$ inside $rSRVR$.

Let P' be the set of $p' \in e$ that is inside $rSRVR$ and P^* be the set of $p^* \in e$ that is outside $rSRVR$. By Lemma 2, $p' \in P'$ may (or may not) alter the safe region. Since e covers points in $P' \cup P^*$, a point p in e may (or may not) alter the safe region, i.e., it is possible that $SR(S, D \backslash \{P' \cup P^*\}) \neq SR(S, D)$. $---(\bigstar)$

In our AB-method, the client computes the safe region from D' (the set of data points in $\mathcal{VO}$), as points in $P' \cup P^*$ are represented by a non-leaf entry e, they are not in D', therefore, we have $D' = D \backslash \{P' \cup P^*\}$. And hence, we have $SR(S', D') = SR(S', D \backslash \{P' \cup P^*\})$.

By $(\blacklozenge)$, we get $SR(S', D') = SR(S', D \backslash \{P' \cup P^*\}) = SR(S, D \backslash (P' \cup P^*)$. Combining this with $(\bigstar)$, we obtain that $SR(S', D')$ may not equal to $SR(S, D)$, i.e., it possible that $SR(S', D') \neq SR(S, D)$, which contradicts with the assumption. □

4.3 $\mathcal{VO}$-optimality

In this section, we prove that the Arc-Based (AB) method is $\mathcal{VO}$-optimal, i.e., it puts the minimum data points and MR-tree entries into the $\mathcal{VO}$.

THEOREM 2. **[The AB method is $\mathcal{VO}$-optimal]**

PROOF: To prove the theorem, we need to prove that (i) Points inside Π are sufficient for the client to verify the correctness of the query result and the safe region (Lemma 6). (ii) Points inside Π are necessary for the client to verify the correctness of the query result and the safe region (Lemma 7). Since the establishment of (i) shows that all points in Π are sufficient and the establishment of (ii) shows that all points in Π are necessary, the $\mathcal{VO}$ contains the minimum data points. Furthermore, since the MR-tree is being visited from the root node, and traversed until we find a non-leaf entry e that does not intersect Π, so the number of non-leaf entries that put into $\mathcal{VO}$ is also minimum.

LEMMA 6. **[Points p inside Π are sufficient for client to verify the correctness of range query result and its safe region]**

Proof: Lemma 4 implies that points p inside $rRVR$ are sufficient for the client to verify the correctness of the query result. Lemma 5 implies that points p inside $rSRVR$ are sufficient for the client to verify the correctness of the safe region. Since Π is the union of $rRVR$ and $rSRVR$, points p inside Π are sufficient for client to verify the correctness of the range query result and the safe region. □

LEMMA 7. **[Points p inside Π are necessary for client to verify the correctness of range query result and its safe region]**

Proof: First, we prove that all points inside $rRVR$ are necessary for the client to verify the correctness of query result. $---(\blacklozenge)$ Since all points inside $rRVR$ are the query result, they must be returned to the client. Hence all points inside $rRVR$ are necessary.

Next, we prove that all points p inside $rSRVR - rRVR$ are necessary for the client to verify the correctness of the safe region.$---(\bigstar)$

We prove this by contradiction. Suppose there exists non-result point p' inside $rSRVR - rRVR$ such that $SR(S', D') = SR(S', D' \backslash \{p'\})$. By definition of $rSRVR$ (Definition 7), we know that there exists a location l inside $SR(S', D' \backslash \{p'\})$ such that $dist(p', l) \leq r$. Therefore, we have

$$SR(S', D' \backslash \{p'\}) \cap \odot(p', r) \neq \emptyset$$

By this, we can conclude that:

$$SR(S', D'\backslash\{p'\}) - \odot(p', r) \neq SR(S', D'\backslash\{p'\}) \quad (2)$$

By the definition of safe region (Definition 1), we know

$$SR(S', D'\backslash\{p'\}) - \odot(p', r) = SR(S', D') \quad (3)$$

Combining (2) and (3), we can conclude that $SR(S', D') = SR(S', D'\backslash\{p'\}) - \odot(p', r) \neq SR(S', D'\backslash\{p'\})$. This leads to a contradiction.

Finally, since $rRVR \cup (rSRVR - rRVR) = rRVR \cup rSRVR$, and Π is the union of $rRVR$ and $rSRVR$, by ($\blacklozenge$) and ($\bigstar$), all points p inside Π are necessary for client to verify the correctness of the query result and the safe region. □

4.4 $\mathcal{VO}$ Size Analysis

Next, we provide a theoretical analysis on the size of the $\mathcal{VO}$ constructed by our AB-method. Given a region, the size of the $\mathcal{VO}$ can be estimated by the cost formula in [28]. So, here we focus on estimating the size of the verification region Π.

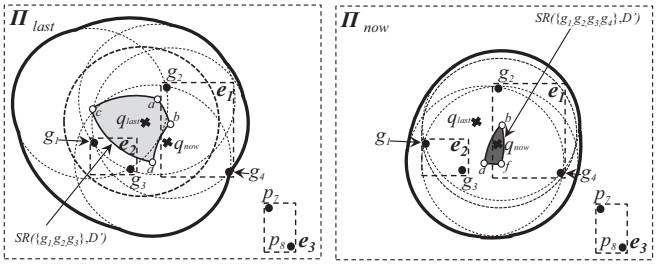
For simplicity, we assume that data points are uniformly distributed. Let d_{max} be the maximum distance between query point q and any point in the safe region. Hence, the circular region $\odot(q, d_{max})$ can completely cover the safe region. If a circular region with radius r and center outside $\odot(q, r + d_{max})$, it cannot overlap with $\odot(q, d_{max})$ and thus it cannot overlap with the safe region. Therefore, $\odot(q, r + d_{max})$ is a valid verification region. Since Π is $\mathcal{VO}$-optimal and $\odot(q, r + d_{max})$ is a valid verification region, Π is completely covered by $\odot(q, r + d_{max})$. In [4], the upper bound of the expected distance m_{up} that a point can move straightly within a safe region is estimated as $\frac{0.33}{rn}$, where n is the number of data points in $\mathcal{D}$. Here, we use m_{up} to approximate d_{max}. Therefore, the size of the $\mathcal{VO}$ is estimated by substituting the area $\pi(r + \frac{0.33}{rn})^2$ into the MR-tree $\mathcal{VO}$ size analysis in [28].

4.5 Communication Optimization

Our Arc-Based method computes safe regions in order to minimize the communication frequency between the mobile client and the server. In this section, we discuss how the client can reuse the previously received verification object $\mathcal{VO}$ in order to further reduce the communication frequency between the server and the client even when the client leaves the safe region.

Let us consider the example in Figure 5. In Figure 5a, a client is located at q_{now} and its previous location was at q_{last}. At q_{last}, the range query result set was $\{g_1, g_2, g_3\}$ and its safe region $SR(\{g_1, g_2, g_3\}, D')$ was the region $abdc$. The verification region Π_{last} was the light grey region. Since Π_{last} intersected the non-leaf entries e_1, e_2, their leaf nodes were visited and their data points were inserted into the verification object $\mathcal{VO}_{last}$. Π_{last} did not intersect e_3 so e_3 was inserted into $\mathcal{VO}_{last}$, and the subtree of e_3 was not visited. Thus, $\mathcal{VO}_{last} = \{g_1, g_2, g_3, g_4, e_3\}$.

Later on, the client moves to a new location q_{now}, which is outside the safe region. Before sending a new query to the server, the client can run Algorithm 2 again using the previous $\mathcal{VO}_{last}$ and its current client location q_{now} as inputs. First, the client finds the query result set of q_{now} from $\mathcal{VO}_{last}$ and the results are $\{g_1, g_2, g_3, g_4\}$. Then, the corresponding safe region $SR(\{g_1, g_2, g_3, g_4\}, D')$ is computed as the region *bdf* and the corresponding verification region Π_{now} is computed as the dark grey region shown in Figure 5b. Observe that Π_{now} does not intersect any non-leaf entries in $\mathcal{VO}_{last}$ (e.g., e_3), the client actually can compute the results and the safe region without any data points from e_3. Therefore, it does not issue a new query to the server. This example illustrates how the client is able to refresh the safe region without issuing a new query to the server. By using this technique, the client needs to send a new query to the server only when the new verification region Π_{now} intersects some non-leaf entry in $\mathcal{VO}_{last}$.



(a) verification region Π_{last} for q_{last} (b) verification region Π_{now} for q_{now}

Figure 5: Reusing $\mathcal{VO}$ for moving range query authentication

5. EXPERIMENTAL STUDY

We evaluate our methods using both synthetic and real datasets. We implemented all methods in C++ with the cryptographic functions in the Crypto++ library[4]. All experiments were run on a 2.5 GHz Intel PC running Ubuntu with 8 GB of RAM. The page size of MR-trees is set to 4 Kbytes.

Datasets and query trajectories The real datasets are NA (North America, 175K points), ARG (Argentina, 85K points), CHINA (China, 32K points) [5]. These datasets have also been used in [18, 30] to model points-of-interests in different countries. We also generated UNI (synthetic uniform) and GAU (synthetic Gaussian) datasets for the scalability experiments. We followed [2, 32, 26] to generate each GAU dataset such that it contains 100 Gaussian bells of equal size and every Gaussian bell has a standard deviation as 2.5% of the domain space length. The query workload contains trajectories of 50 moving objects generated by trajectory generator in [14]. Each trajectory simulates an object running in Euclidean space (directional movement) and has a location record at every timestamp; there are 10,000 timestamps in total and the time between adjacent timestamps is 1 second. Therefore, each object's journey is about 10,000 seconds (i.e., about 2.7 hours). We try to simulate the scenario of a car (default speed 50km/hr) moving at the country level (i.e., on CHINA, ARG, and NA datasets).

Table 2 shows the building time of MR-trees on the real datasets and synthetic datasets (50K data points to 1000K data points). The time of building MR-trees scales well with the data size.[6]

Table 2: MR-Tree building time

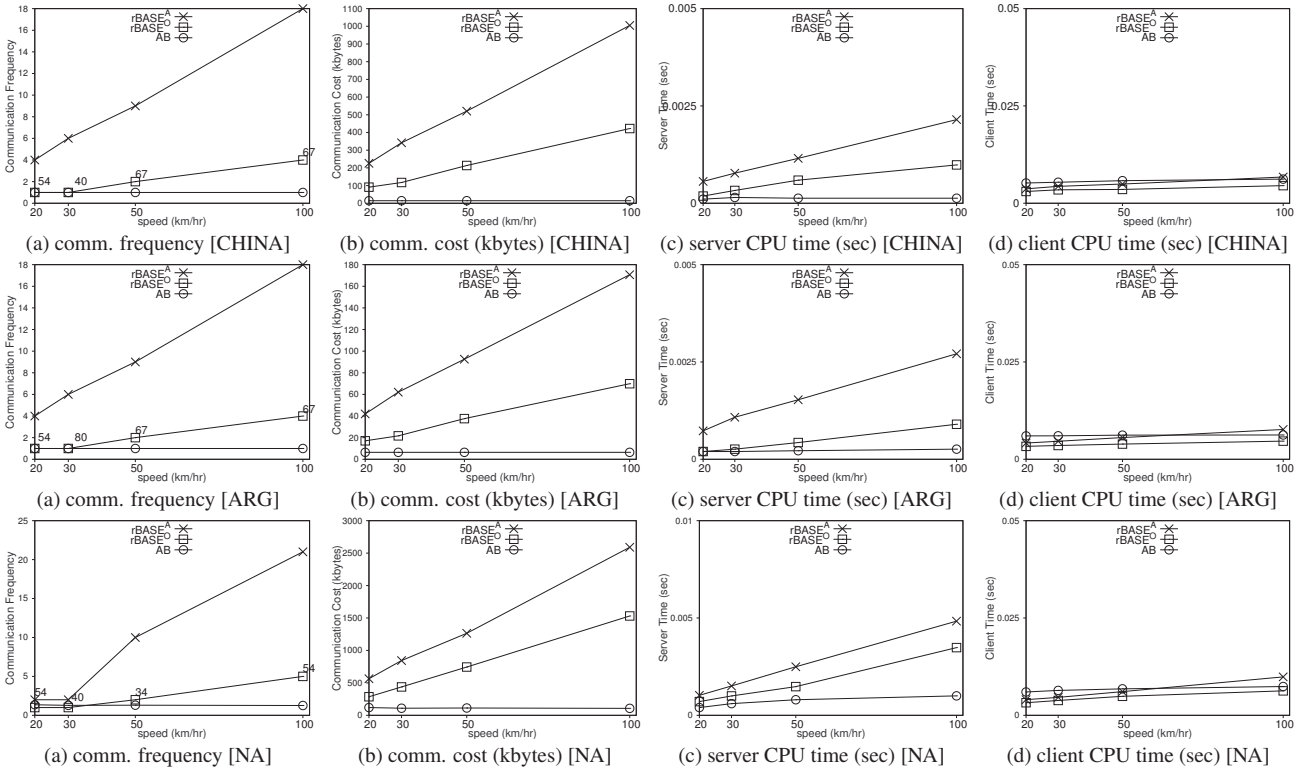
Dataset	Building Time (seconds)
	MR-tree
CHINA	21.75
ARG	62.71
NA	124.05
50,000—UNI/GAU	30.11 / 35.4
100,000—UNI/GAU	63.11 / 74.78
200,000—UNI/GAU	128.80 / 151.03
500,000—UNI/GAU	334.30 / 396.82
1,000,000—UNI/GAU	732.92 / 885.07

Performance measure Following [14], we measure the cumulative total cost for each object's trajectory. In our experiments, we

[4]Crypto++ library: http://www.cryptopp.com/

[5]Downloaded from http://www.maproom.psu.edu/dcw/ and http://www.dis.uniroma1.it/∼challenge9/download.shtml

[6]We have repeated the whole experimental study using MR*-tree. The experimental results are similar and not described here.



(a) comm. frequency [CHINA] (b) comm. cost (kbytes) [CHINA] (c) server CPU time (sec) [CHINA] (d) client CPU time (sec) [CHINA]

(a) comm. frequency [ARG] (b) comm. cost (kbytes) [ARG] (c) server CPU time (sec) [ARG] (d) client CPU time (sec) [ARG]

(a) comm. frequency [NA] (b) comm. cost (kbytes) [NA] (c) server CPU time (sec) [NA] (d) client CPU time (sec) [NA]

Figure 6: Effect of query speed (on various real datasets)

report the *communication cost* (in kilobytes) as the total size of $\mathcal{VO}$ per object. This is the most important measure in this paper and we hope to minimize it. We also report the *communication frequency*, which is the number of clients' queries received by the server, per object. In addition, we report the *server* and *client CPU time* (in seconds), per object journey.

Competitors We now report the experimental results of moving range queries. The default query range r is 5 km for a moving client (i.e., on CHINA, NA, and ARG datasets). We compare our AB-method with the baseline buffering method (rBASE) mentioned in Section 3. Again, rBASE's performance depends on the Δr value, which in turns depends on various factors such as the query range, query speed, data size, and data distribution. The optimal Δr for a particular setting cannot be found unless we exhaustively try every possible Δr value. However, for comparison purpose, we also carried out such a manual tuning process for rBASE in all our experiments. We use rBASE^O to denote the best performance of rBASE that uses the optimal Δr value. We use rBASE^A to represent the average performance of rBASE over every Δr value.

5.1 Effect of Query Speed

We study the effect of moving query speed on the performance of our AB-method, rBASE^A, and rBASE^O. Figure 6 shows (a) the total communication frequency, (b) the total communication cost, (c) the total server CPU time, and (d) the total client CPU time, of each moving object, with respect to different query speeds, in its 2.7-hour journey, on the three datasets.

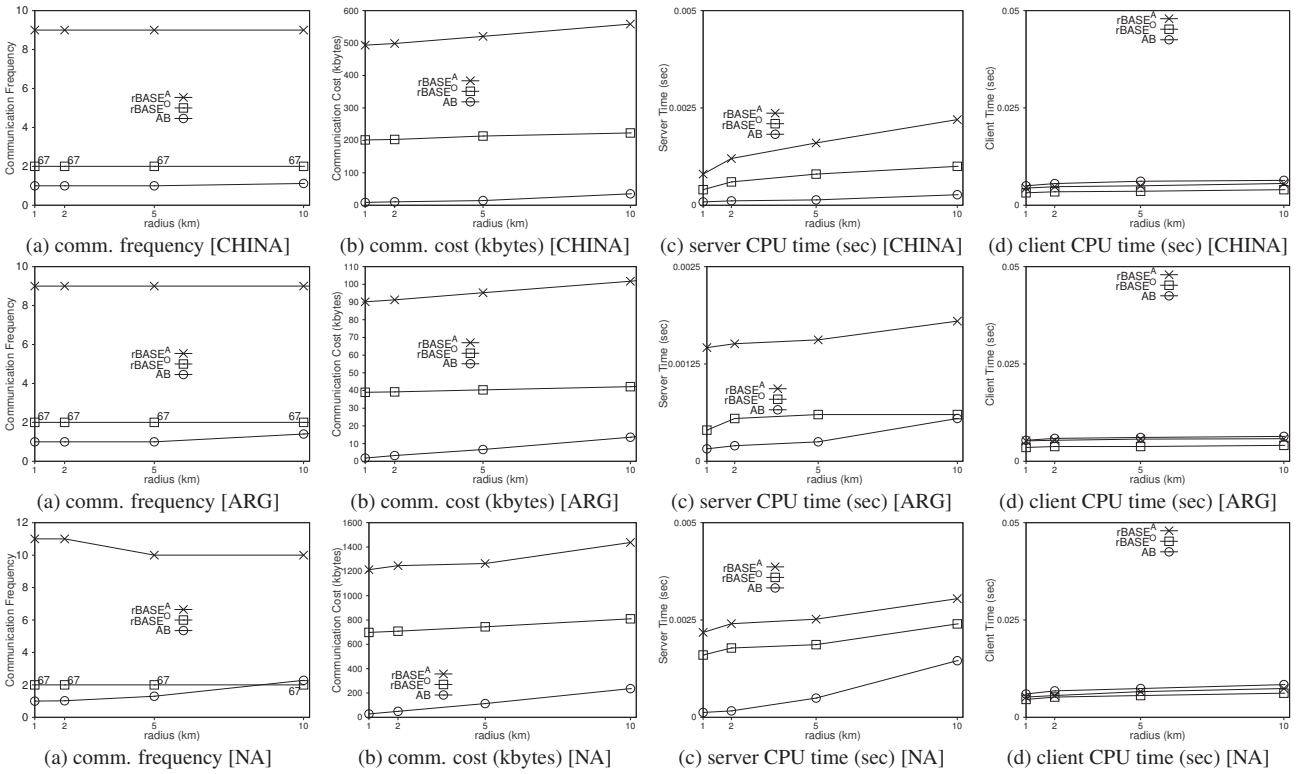
In Figure 6a, the communication frequency of rBASE^O is annotated with the corresponding optimal Δr value (in km) found by our manual tuning process. When the query speed increases, a client using the baseline methods leaves its buffer region easier, so its communication frequency increases. The communication frequency of our AB-method rises slowly because our method reuses the $\mathcal{VO}$ whenever applicable. Together with the fact that our $\mathcal{VO}$-optimal method constructs very compact $\mathcal{VO}$, this explains why the communication cost of our AB-method is much smaller than that of the baseline methods in Figure 6b.

The server computation overhead of our AB-method is very small (Figure 6c). That is because the communication frequency of AB-method is small and also computing safe regions for moving range queries is not expensive.

The client CPU times of all methods are shown in Figure 6d. We can see that all methods incur almost negligible computational overhead on the client side, which is less than 0.01 CPU seconds for a 2.7-hour journey.

5.2 Effect of the Query Range r

Figure 7 shows the effect of the query range r. In Figure 7a, we can see that the communication frequencies of the baseline methods are quite stable when r increases. That is because the optimal Δr values for the baseline methods do not vary when r increases, so all buffering regions have the same size. Since the speed of the client is constant in this experiment, with constant size buffering regions, the communication frequencies of the baseline methods remain unchanged. The communication frequency of our AB-method increases slowly when r increases. That is because the number of result points increases when r increases. As the safe region of moving range query is defined as the intersection of result points excluding the union of non-result points, the increase of result points results in a smaller safe region. This explains the increase of communication frequency with r. However, the communication cost of our AB-method is much lower than the baseline methods (Figure 7b) because our method is $\mathcal{VO}$-optimal and thus is able to construct very compact $\mathcal{VO}$. When r increases, the query

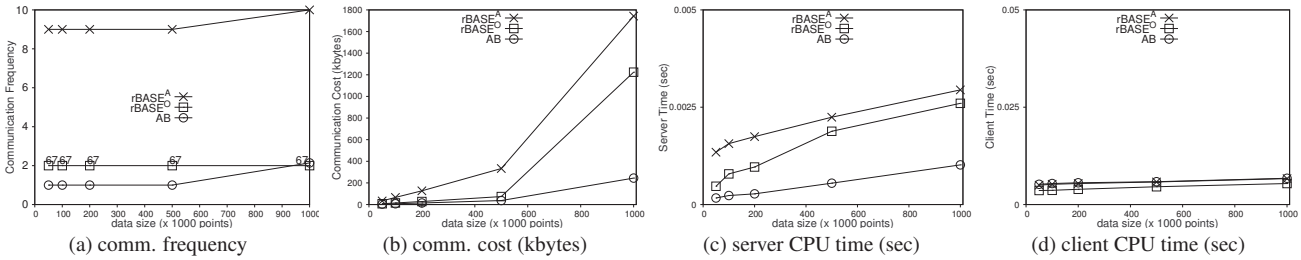


(a) comm. frequency [CHINA] (b) comm. cost (kbytes) [CHINA] (c) server CPU time (sec) [CHINA] (d) client CPU time (sec) [CHINA]

(a) comm. frequency [ARG] (b) comm. cost (kbytes) [ARG] (c) server CPU time (sec) [ARG] (d) client CPU time (sec) [ARG]

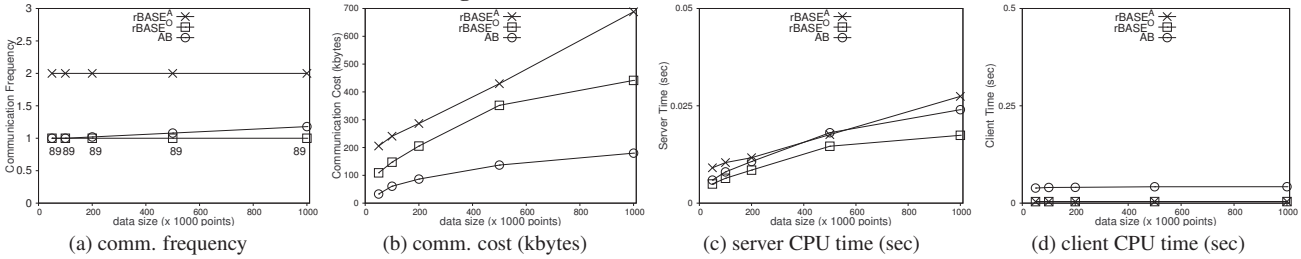
(a) comm. frequency [NA] (b) comm. cost (kbytes) [NA] (c) server CPU time (sec) [NA] (d) client CPU time (sec) [NA]

Figure 7: Effect of r (on various real datasets)



(a) comm. frequency (b) comm. cost (kbytes) (c) server CPU time (sec) (d) client CPU time (sec)

Figure 8: Effect of data size (UNIFORM)



(a) comm. frequency (b) comm. cost (kbytes) (c) server CPU time (sec) (d) client CPU time (sec)

Figure 9: Effect of data size (GAUSSIAN)

results are larger and thus the $\mathcal{VO}$s are larger. Therefore, the computation overhead of our AB-method on both the server side and the client side increase but still stays low (Figure 7c and d).

5.3 Effect of Data Size

We proceed to study the scalability of our methods, by using synthetic data of different sizes. Figure 8 shows the performance of the methods under uniform datasets of different sizes. When the data size increases, the data density rises. Therefore, the query results as well as the $\mathcal{VO}$-size generally would increase, leading to a higher communication cost. Nonetheless, we remark that the communication cost of our AB-method increases mildly because of the compact $\mathcal{VO}$ constructed as well as the reuse of the $\mathcal{VO}$. Similar observations could be found on the Gaussian datasets (see Figure 9).

5.4 Validation of $\mathcal{VO}$ Size Analysis

Finally, we validate the theoretical analysis about the size of $\mathcal{VO}$

in moving range query authentication. Our analysis has made certain simplifying assumptions like uniform data distribution. Thus, the following validation is conducted on uniform data only. Figure 10 shows the actual number of points covered by the verification region Π and the estimated number based on the equations in Section 4.4 during moving range query authentication. The results show that our theoretical analysis is quite robust as it well captures the trend and the error is below 0.18 on uniform datasets.

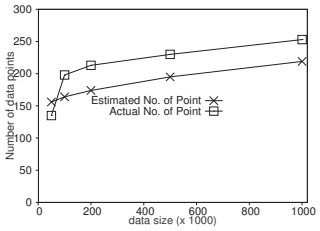


Figure 10: VO analysis (UNIFORM)

6. CONCLUSION

In this paper, we presented a framework with efficient methods to authenticate moving range queries. We proved that our methods are $\mathcal{VO}$-optimal, i.e., the size of the verification object ($\mathcal{VO}$) required to carry out authentication is minimal in size. We also presented optimization techniques that can further reduce the communication frequency between a moving client and the service provider. Experimental results show that our methods efficiently authenticate moving range queries using a small communication cost and computational overhead. As for future work, we plan to extend our methods to use other orthogonal ADS (e.g., PMKd-tree [13]) and establish the corresponding $\mathcal{VO}$-optimality.

7. REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, 1990.

[2] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *SIGMOD*, 2001.

[3] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang. Continuous monitoring of distance based range queries. *TKDE*, 2010.

[4] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang. Multi-Guarded Safe Zone: An Effective Technique to Monitor Moving Circular Range Queries. In *ICDE*, 2010.

[5] D. Gunopulos, G. Kollios, V.J.Tsotras, and C.Domeniconi. Selectivity estimators for multidimensional range queries over real attributes. *VLDB J*, 2003.

[6] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi. Verifying spatial queries using voronoi neighbors. In *GIS*, 2010.

[7] G. S. Iwerks, H. Samet, and K. P. Smith. Maintenance of K-nn and Spatial Join Queries on Continuously Moving Points. *ACM TODS*, 31(2):485–536, 2006.

[8] A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. *PVLDB*, 1(1):138–150, 2008.

[9] A. Kundu and E. Bertino. How to Authenticate Graphs without Leaking. In *EDBT*, 2010.

[10] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *SIGMOD*, 2006.

[11] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-Infused Streams: Enabling Authentication of Sliding Window Queries On Streams. In *VLDB*, 2007.

[12] R. C. Merkle. A Certified Digital Signature. In *CRYPTO*, 1989.

[13] K. Mouratidis, D. Sacharidis, and H. Pang. Partially materialized digest scheme: An efficient verification method for outsourced databases. *VLDB J*, 18(1):363–381, 2009.

[14] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The V*-Diagram: A Query-dependent Approach to Moving KNN Queries. *PVLDB*, 1(1):1095–1106, 2008.

[15] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *SIGMOD*, 2005.

[16] H. Pang and K. Mouratidis. Authenticating the Query Results of Text Search Engines. *PVLDB*, 1(1):126–137, 2008.

[17] H. Pang, J. Zhang, and K. Mouratidis. Scalable Verification for Outsourced Dynamic Databases. *PVLDB*, 2(1):802–813, 2009.

[18] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.

[19] S. Papadopoulos, Y. Yang, S. Bakiras, and D. Papadias. Continuous Spatial Authentication. In *SSTD*, 2009.

[20] S. Papadopoulos, Y. Yang, and D. Papadias. CADS: Continuous Authentication on Data Streams. In *VLDB*, 2007.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[22] R. Sion. Query execution assurance for outsourced databases. In *VLDB*, 2005.

[23] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *SSTD*, 2001.

[24] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, 2002.

[25] L. Wang, S. Noel, and S. Jajodia. Minimum-Cost Network Hardening using Attack Graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[26] X. Xiong, M. F. Mokbel, and W. G. Aref. Lugrid: Update-tolerant grid-based indexing for moving objects. In *MDM*, 2006.

[27] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated Join Processing in Outsourced Databases. In *SIGMOD*, 2009.

[28] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Authenticated Indexing for Outsourced Spatial Databases. *VLDB J.*, 18(3):631–648, 2009.

[29] K. Yi, F. Li, G. Cormode, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Small Synopses for Group-by Query Verification on Outsourced Data Streams. *ACM TODS*, 34(3), 2009.

[30] M. L. Yiu, Y. Lin, and K. Mouratidis. Efficient Verification of Shortest Path Search via Authenticated Hints. In *ICDE*, 2010.

[31] M. L. Yiu, E. Lo, and D. Yung. Authentication of Moving kNN Queries. In *ICDE*, 2011.

[32] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, 2004.

[33] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *SIGMOD*, 2003.