On Efficient and Scalable Support of Continuous Queries in Mobile Peer-to-Peer Environments

Chi-Yin Chow, *Graduate Student Member, IEEE,* Mohamed F. Mokbel, *Member, IEEE,* and Hong Va Leong, *Member, IEEE Computer Society*

Abstract—In this paper, we propose an efficient and scalable query processing framework for continuous spatial queries (range and k-nearest-neighbor queries) in mobile peer-to-peer (P2P) environments, where no fixed communication infrastructure or centralized/distributed servers are available. Due to the limitations in mobile P2P environments, for example, user mobility, limited battery power, limited communication range, and scarce communication bandwidth, it is costly to maintain the exact answer of continuous spatial queries. To this end, our framework enables the user to find an approximate answer with quality guarantees. In particular, we design two key features to adapt continuous spatial query processing to mobile P2P environments. (1) Each mobile user can specify the desired quality of services (QoS) for query answers in a personalized QoS profile. The QoS profile consists of two parameters, namely, *coverage* and *accuracy*. The coverage parameter indicates the desired level of completeness of the available information for computing an approximate answer, and the accuracy parameter indicates the desired level of accuracy of the approximate answer. (2) We design a *continuous answer maintenance scheme* to enable the user to collaborate with other peers to continuously maintain her query answer. With these two features in our framework, the user can obtain a query answer from her local cache if the answer satisfies her QoS requirements. Otherwise, the user enlists neighbors for help to share their cached information to refine the answer. If the refined answer still cannot satisfy the QoS requirements, the user broadcasts the query to the peers residing within the required search area of the query to find the most accurate answer. Experimental results show that our framework is efficient and scalable and provides an effective tradeoff between the communication overhead and the quality of query answers.

Index Terms—Mobile computing, peer-to-peer computing, continuous query processing, spatio-temporal databases, and GIS

1 Introduction

With the advance in new peer-to-peer (or P2P for short) wireless communication technologies, for example, IEEE 802.11 and Bluetooth, and the computational and storage capacity of portable devices, a new information access paradigm, known as *mobile P2P information access*, has rapidly taken shape. This paradigm is important for environments, where no fixed communication infrastructure or centralized/distributed servers are available, such as battlefield and rescue operations. It is also useful for other business and civilian applications, such as traffic monitoring and resource locator. Existing spatial query processing frameworks in mobile environments either rely on fixed communication infrastructure and/or centralized servers (e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]), or only support snapshot queries [12], [13], [14]. The limitations in these existing works motivate us to design a new continuous spatial query processing framework that does not require any fixed communication infrastructure or centralized/distributed servers. In particular, we focus on the two most common spatial queries, *range queries* and *k-nearest-neighbor (k-NN) queries*. Examples of these spatial queries include that the rescuers issue range queries to continuously keep track of the ambulances within a certain range in a disaster site, and the soldiers issue k-NN queries to continuously monitor their k-nearest tanks in a battlefield.

- *Chi-Yin Chow and Mohamed F. Mokbel are with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA, email: {cchow, mokbel}@cs.umn.edu.*
- *Hong Va Leong is with the Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, email: cshleong@comp.polyu.edu.hk.*

Due to the limitations in mobile P2P environments, for example, user mobility (the query issuer and the data object are continuously roaming), limited communication range, limited battery power, and scarce communication resources, it is costly to maintain the exact answer of continuous spatial queries. To this end, we propose a continuous spatial query processing framework to provide approximate answers for mobile users with quality guarantees. In particular, we design two key features to adapt continuous spatial query processing to mobile P2P environments. (1) The user can specify the desired *quality of services* (QoS) for query answers in a personalized QoS profile. The QoS profile consists of two parameters, namely, *coverage* and *accuracy*. The coverage parameter indicates the desired level of completeness of the available information for computing an approximate answer, and the accuracy parameter indicates the desired level of accuracy of the approximate answer. (2) We design a *continuous answer maintenance scheme* that allows the user to collaborate with peers to continuously maintain her answer, instead of always processing the query from scratch, in order to reduce communication overhead.

The main idea of our framework is that a user can obtain a query answer from her local cache if the answer satisfies her QoS requirements, that is, the information stored in the local cache satisfies the coverage requirement and the answer derived from the local cache satisfies the accuracy requirement. In case that the answer does not satisfy the QoS requirements, the user asks neighbors to share their cached information, in order to refine the answer. If the refined query answer still does not satisfy the QoS requirements, the user enlists the peers residing within the required search area of the query for help

to find the most accurate answer, and then updates her local cache. Since the query issuer and data objects are moving, the query answer may become stale. Thus we propose the continuous answer maintenance scheme that enables the user to collaborate with other peers to continuously maintain the query answer. When the answer derived from the local cache no longer satisfies the user's QoS requirements, the user needs to enlist peers for help to refine the answer again.

We evaluate our framework through simulated experiments. The results show that our framework is efficient and scalable in terms of communication overhead and power consumption and provides a tradeoff between the communication overhead and the quality of query answers. In this paper, we focus on continuous range and k-NN queries. However, our framework can be extended to support other types of continuous spatial queries, if their query answers can be abstracted into monitoring regions, for example, continuous reverse-nearest-neighbor queries [15] and continuous density queries [16].

The rest of this paper is organized as follows. Section 2 surveys previous works related to query processing in mobile P2P environments. Our system model is presented in Section 3. Section 4 describes our continuous spatial query processing framework for both range and k-NN queries. We delineate our simulation model in Section 5. Experimental results are presented in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

Existing spatial query processing frameworks in mobile environments can be categorized into four architecture approaches: *centralized* [6], [9], [11], [17], *semi-distributed* [1], [4], *semi-distributed with client cooperation* [2], [3], [5], [7], [8], [10], [18], [19], and *wireless sensor networks* [20], [21], [22]. (1) The centralized approach requires fixed communication infrastructure and centralized database servers for query processing. (2) The semi-distributed approach is similar to the centralized approach, as it also relies on centralized database servers for query processing and data storage; however the servers delegate some query monitoring tasks to the mobile user in order to reduce communication overhead. (3) The semi-distributed with client cooperation approach considers peer collaboration in mobile P2P environments. In this approach, mobile users can only share their cached information with other peers, in which the information is previously retrieved from centralized database servers; thus, this approach does not consider query processing among the mobile users. (4) The algorithms designed for wireless sensor networks rely on stationary sensors. Since these sensors work like distributed servers to process spatial queries, these algorithms cannot be applied to mobile P2P environments. Due to the dependency on fixed communication infrastructure and/or centralized/distributed servers, none of these techniques can be applicable to spatial query processing in mobile P2P environments.

The closest works to ours are the systems that support resource discovery in mobile P2P environments [12], [13], [14]. These resource discovery algorithms enable a mobile or stationary resource provider, e.g., a taxi and ATM machine, to periodically broadcast its information to mobile users, in order to look for potential customers or users. On the other hand, a mobile user broadcasts a query to peers to express an interest in certain types of resources, e.g., "Where is my nearest available taxi". Once the peer finds a match between the query and its requested resource, the peer sends the information about the available requested resource to the requesting user. Our work distinguishes itself from these works, as it (a) supports spatial constraints for queries, for example, the user can ask for her k-nearest resources and certain types of resources within a certain distance range, (b) enables the user to specify the desired QoS, namely, coverage and accuracy, for query answers, in order to achieve a tradeoff between the communication overhead and the quality of query answers, and (c) enables the user to collaborate with peers to continuously maintain query answers, while the existing resource discovery works only consider snapshot queries.

3 System Model

We present the system model of our continuous spatial query processing framework in mobile P2P environments.

Mobile users. Each mobile user belongs to an object type, such as taxis and police cars. A querying user issues a continuous query asking for objects of a specific type, and the peers who belong to the requested object type are referred to as the *objects of interest* of the query. Every mobile device is equipped with a positioning device, such as GPS, to determine its location, which is represented by a coordinate (x, y). The mobile user can communicate with all neighbors through broadcast communication, a neighbor through point-to-point communication, and a multi-hop peer through a multi-hop routing protocol which contains a sequence of point-to-point communications. Furthermore, each user employs a neighbor discovery protocol (NDP) [23], [24] to maintain a list of neighbors. The basic idea of NDP is that each user periodically broadcasts a beacon message with her identity to neighbors. If the user has not received the beacon message of a neighbor for a beacon period or certain beacon periods, the user considers that the peer no longer resides in her transmission range; therefore the peer is removed from the neighbor list. However, if the user receives the beacon message of a peer who is not included in the neighbor list, the user considers the peer as a newly discovered neighbor. The user sends a message to request for the peer's information (*ID*, *Loc*, *TS*, *MaxSpeed*, and *Type*), where *ID* is the peer's identity, *Loc* is the peer's current location at timestamp *TS*, *MaxSpeed* is the peer's maximum mobility speed, and *Type* is the peer's object type, through point-to-point communication, and inserts the peer into the neighbor list. In practice, *MaxSpeed* can be set to the maximum legal speed in the system area or the highest speed that has been recorded by the user for a certain time period, for example, one week.

User QoS profile. Each mobile user maintains a quality of services (QoS) profile that consists of two parameters, namely, coverage (C_{min}) and accuracy (A_{min}), where $0 \leq C_{min}, A_{min} \leq 1$. C_{min} specifies the desired level of completeness of the available information for computing an approximate answer, and A_{min} specifies the desired level of

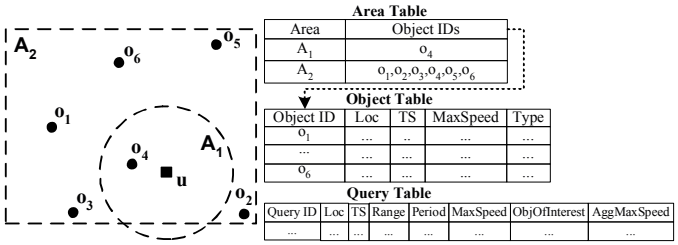


Fig. 1: The structure of a local cache.

accuracy of the approximate answer. The user can specify larger C_{min} and A_{min} values to obtain an answer with better QoS, but they incur higher communication overhead. Therefore the user can tune these QoS requirements to achieve a performance trade-off between the communication overhead and the quality of query answers. The user can change her QoS requirements at any time.

Spatial queries. A range query is in a form (*ID*, *Range*, *Period*, *Loc*, *TS*, *MaxSpeed*, *ObjectOfInterest*), where *ID* is the query issuer's identity, *Range* is the distance range of the query, *Period* is the valid time frame of the query, *Loc* is the query issuer's current location, *TS* is the current timestamp, *MaxSpeed* is the query issuer's maximum mobility speed, and *ObjectOfInterest* is the object type requested by the query. Given a range query, its answer includes the objects of interest residing in a rectangular query region whose left-bottom and right-top vertices are *(Loc.X−Range, Loc.Y−Range)* and *(Loc.X+Range, Loc.Y+Range)*, respectively. A k-NN query is in a form (*ID*, *k*, *Period*, *Loc*, *TS*, *MaxSpeed*, *ObjectOfInterest)*, where k is the required number of objects of interest of the query and other attributes are the same as in the range query. Given a k-NN query, its answer includes the k-nearest objects of interest to *Loc*. Although we only focus on range and k-NN queries in this work, our framework is applicable to other continuous spatial query types if their query answers can be abstracted into monitoring regions. For example, our framework can be extended to support reverse-nearest-neighbor queries [15] and density queries [16] because recent research efforts have shown that the answer of these query types can be maintained by monitoring a region.

Local cache. A local cache is a user's storage space dedicated to our framework. The local cache stores three tables, *area table*, *object table*, and *query table*. The area table and object table maintain the received object information, while the query table maintains the received continuous queries for the continuous query answer maintenance scheme. The details of the query table will be described in Section 4.4. Each user maintains two types of object information, *neighbors* and *query answers*. For the information of neighbors, it is modeled by a circular area as the user's transmission range, which is stored in the area table, and the object table stores the information of each neighbor, where a peer's information is in a form (*ID*, *Loc*, *TS*, *MaxSpeed*, *Type*), where *Loc* is the peer's latest location information sent by the peer at timestamp *TS*. For the information of a range (or k-NN) query answer, it is modeled by a rectangular (or circular) area as the query's required search area, which is stored in the area table, and the object table stores the information of

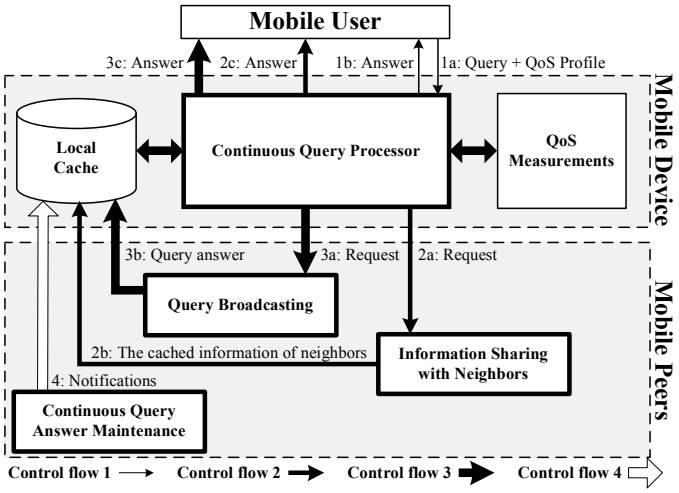


Fig. 2: The four control flows in our framework.

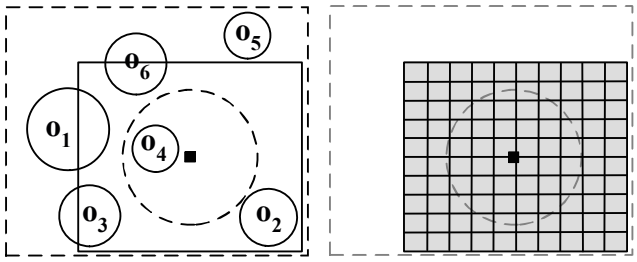
the objects of interest that could reside in the required search area. If an object is included in multiple areas in the area table, the object table only keeps one entry for the object with the latest location information received by the user through the neighbor discovery protocol, the information sharing with neighbors control flow (Section 4.2), the query broadcasting control flow (Section 4.3), or the continuous query answer maintenance control flow (Section 4.4). Whenever the user receives the more recent location information of an object stored in the object table, the user updates the *Loc* and *TS* of the object's entry accordingly. If an object is no longer referred by any entry in the area table, the object is removed from the object table. Figure 1 depicts the local cache of a mobile user u, where u's location is represented by a square and the peers are represented by circles with unique labels. The area table contains two areas, where the user's transmission range A_1 is represented by a dotted circle with one object o_4, and the required search area of a continuous range query A_2 is represented by a dotted rectangle with six objects o_1 to o_6. Thus the object table contains one entry for each of the objects of A_1 and A_2, o_1 to o_6.

4 Query Processing Framework

Figure 2 depicts the four main control flows in our framework: (1) QoS measurements for a local cache, (2) information sharing with neighbors, (3) query broadcasting, and (4) continuous query answer maintenance. Algorithm 1 outlines the first three control flows and Algorithm 2 outlines the fourth control flow. We will describe each control flow in detail.

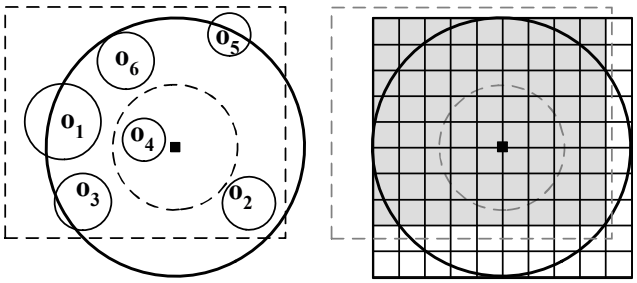
4.1 QoS Measurements for a Local Cache

This is the first control flow in our framework, which is indicated by thin lines in Figure 2 (Lines 2 to 7 in Algorithm 1). This control flow is completely executed on the mobile user side without enlisting peers for help. Due to user mobility, the cached peer location information could become stale. To capture location uncertainty, we employ a conservative approach that models an uncertain location as an *adjusted location region*. The adjusted location region of a peer's location is a circular region centered at the peer's



(a) Range query (b) Coverage measurement

Fig. 3: Coverage measurement for a range query.



(a) 3-NN query (b) Coverage measurement

Fig. 4: Coverage measurement for a 3-NN query.

location with a radius of $(t_{current} - TS) \times MaxSpeed$, where $t_{current}$ is the current time and *TS* is the timestamp of the latest location update. In other words, the peer can move in any direction at the maximum mobility speed *MaxSpeed*. We will present how to determine the coverage and accuracy for a query with respect to the information stored in a local cache for both range queries and k-nearest-neighbor (k-NN) queries.

4.1.1 Coverage Measurement

The coverage measurement indicates the level of completeness of the available information in a local cache for computing a query answer. Given the required search area of a query and a local cache, the coverage of the information in the local cache is measured by the ratio of the intersection of the required search area and the union of the areas stored in the area table to the required search area. Formally, given a required search area S and a set of n areas $A_1, A_2, \ldots, A_n$ in the area table in a local cache $\mathcal{L}$ intersecting S, the coverage of $\mathcal{L}$ with respect to S is calculated as:

$$Cov(S, \mathcal{L}) = Area(S \cap (A_1 \cup \ldots \cup A_n))/Area(S), \quad (1)$$

where $Area(R)$ is the area of a region R.

Since calculating the exact coverage of a large number of overlapping areas is costly, we use a histogram approach to get approximate coverage measurement. The basic idea is that we model the required search area by a uniform grid structure and maintain a bitmap, in which each bit corresponds to a unique gird cell. Initially, all bits in the bitmap are set to zero. For each grid cell, if it is totally covered by some area stored in the area table, the corresponding bit is set to one. The coverage is the number of one in the bitmap divided by the total number of bits in the bitmap. Since the bit of a grid cell is set to one if its area is totally covered by some monitored area, a larger grid cell area results in a larger underestimation of the coverage measurement. Although a smaller grid cell area gives higher computation precision, it incurs higher computational overhead. In Section 6.3, we study this performance tradeoff between the computational overhead and the coverage measurement precision. We will discuss the coverage measurement for range and k-NN queries.

Range queries. Figure 3a shows a user's local cache, which stores the information about the user's neighbors (the area of the user's transmission range is represented by a dotted circle) and the adjusted required search area of the user's continuous range query is represented by a dotted rectangle. The detail of how to adjust the required search area of a continuous range query will be discussed in Section 4.4. In this example, the user wants to measure the coverage of the information stored in the local cache with respect to the same continuous range query, where the user's location is represented by a filled square, the query region[1] of the range query is represented by a rectangle, and the adjusted location region of each object stored in the object table is represented by a circle with a unique label. Since the required search area of a range query is the same as its query region, the user can calculate the coverage without enlisting any peers for help. In Figure 3b, we assume that the required search area is modeled by a uniform grid structure of 10×10 cells. Since all grid cells, which are represented by shaded cells, are totally covered by the areas stored in the area table, A_1 and/or A_2, all bits in the bitmap are set to one; hence the coverage is 100/100 = 1.

k-NN queries. Figure 4a shows the same local cache as in Figure 3, where the local cache stores the information about the user's neighbors (the area of the user's transmission range is represented by a dotted circle) and the adjusted required search area of the user's outstanding continuous range query is represented by a dotted rectangle. In this example, a user wants to measure the coverage of the information stored in the local cache with respect to a new k-NN query. The required search area of a k-NN query is a minimal circle that totally covers the exact location or the adjusted location region of k objects of interest. These k objects are not necessary to be the k-nearest objects to the user. However, if the user cannot find at least k objects of interest in the local cache, the user is unable to determine the required search area of the query, $S = null$. Thus the user cannot perform the coverage measurement, and the user proceeds to the next control flow: information sharing with neighbors. In this example, we assume that all objects o_1 to o_6 are the objects of interest of the query; therefore the required search area of the query is a circle (represented by a bold circle) that totally covers the adjusted location regions of 3-nearest objects, o_2, o_4, and o_6. Then we construct a minimum bounding rectangle of the required search area and model the rectangle by a uniform grid structure of 10×10 grid cells, as depicted in Figure 4b. Since 72 grid cells, which are represented by shaded cells, are totally covered by the areas, there are 72 bits in the bitmap are set to one; hence the coverage is 72/100 = 0.72.

1. The query region of a range query is a rectangular area whose left-bottom and right-top vertices are *(Loc.X−Range, Loc.Y−Range)* and *(Loc.X+Range, Loc.Y+Range)*, respectively, as defined in Section 3.

4.1.2 Accuracy Measurement

The accuracy measurement indicates the level of accuracy of an approximate answer, which is derived from a local cache. It is important to note that we only determine an approximate query answer if the coverage requirement is satisfied. Since it is very costly to determine an exact probabilistic answer for a k-NN query, even an approximate approach is computationally expensive [25], these approaches are not suitable for mobile devices with limited computational resources. To this end, we use a heuristic approach to measure the accuracy of a range or k-NN query answer. The accuracy measurement is defined as follows: Given a query with a required search area S, a set of m objects of interest of the query whose exact locations or adjusted location regions intersect S, $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, the probability p_i of each object $o_i \in \mathcal{O}$ being part of the query answer, where $p_i \geq 0$, and an approximate answer $\mathcal{A} = \{o_1, o_2, \ldots, o_n\}$, where $\mathcal{A} \subseteq \mathcal{O}$, the accuracy, $Acc(\mathcal{O}, \mathcal{A})$, is computed as the average of (a) the probability p_i of correctly making a decision of including an object o_i in $\mathcal{A}$, and (b) the probability $1 - p_i$ of correctly making a decision of excluding an object o_i from $\mathcal{A}$; hence the accuracy of $\mathcal{A}$ is computed as:

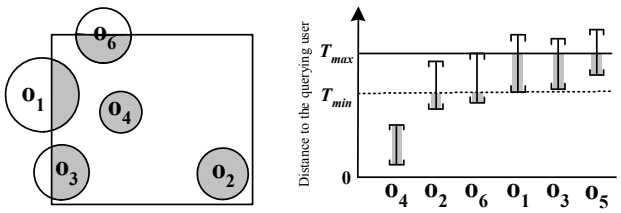
$$Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{m}\left[\sum_{o_i \in \mathcal{A}} p_i + \sum_{o_i \in \mathcal{O}-\mathcal{A}} (1 - p_i)\right]. \quad (2)$$

We assume a uniform distribution for the actual location of an object o_i in its adjusted location region when computing p_i for both range and k-NN queries; however our algorithm is completely independent of how we compute p_i. The following theorem shows the correctness of Equation 2.

Theorem 1: Given a query with a required search area S, a set of m objects of interest of the query whose exact locations or adjusted location regions intersect S, $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, the probability p_i of each object $o_i \in \mathcal{O}$ being part of the query answer, where $p_i \geq 0$, and an approximate answer $\mathcal{A} = \{o_1, o_2, \ldots, o_n\}$, where $\mathcal{A} \subseteq \mathcal{O}$, the accuracy of $\mathcal{A}$ is $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{m}[\sum_{o_i \in \mathcal{A}} p_i + \sum_{o_i \in \mathcal{O}-\mathcal{A}} (1 - p_i)]$.

Proof: If all objects in $\mathcal{O}$ are included in $\mathcal{A}$, $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{m}\sum_{o_i \in \mathcal{O}} p_i = \frac{1}{n}\sum_{o_i \in \mathcal{A}} p_i$. However, if k objects from $\mathcal{O}$ are excluded from $\mathcal{A}$, $\overline{\mathcal{A}} = \{o_{j_1}, o_{j_2}, \ldots, o_{j_k}\}$, the average accuracy of making correct decisions of including objects in $\mathcal{A}$ is $\frac{1}{|\mathcal{A}|}|\sum_{o_i \in \mathcal{A}} p_i = \frac{1}{n}\sum_{o_i \in \mathcal{A}} p_i$, while the average accuracy of making correct decisions of excluding objects from $\mathcal{A}$ is $\frac{1}{|\mathcal{A}|}[(1 - p_{j_1}) + (1 - p_{j_2}) + \ldots + (1 - p_{j_k})] = \frac{1}{k}\sum_{o_j \in \mathcal{O}-\mathcal{A}}(1 - p_j)$. Thus the overall average accuracy $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{m}[\sum_{o_i \in \mathcal{A}} p_i + \sum_{o_i \in \mathcal{O}-\mathcal{A}} (1 - p_i)]$. □

Range queries. Figure 5a shows a range query, where its query region is represented by a rectangle and the adjusted location region of each object of interest of the query that intersects the query region is represented by a circle. Since the required search area of a range query is the same as its query region, $\mathcal{O}$ contains the objects of interest of the query whose exact locations or adjusted location regions intersect the query region; thus $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_6\}$ in this example. For each object $o_i \in \mathcal{O}$ with an exact location, p_i is set to one because p_i must reside in the query region. On the other hand, for each object $o_i \in \mathcal{O}$ with an adjusted location region, since we consider a uniform distribution, p_i is computed as the ratio of the area of the intersection of the adjusted location region and the query region to the area of the adjusted location region. We can maximize the accuracy of an approximate range query answer by maximizing p_i in each term in $Acc(\mathcal{O}, \mathcal{A})$. Thus if $p_i \geq 0.5$, o_i should be included in $\mathcal{A}$; however if $p_i < 0.5$, o_i should be excluded from $\mathcal{A}$. The correctness of this selection threshold is proved in Theorem 2. In this example, since o_2 and o_4 are inside the query region, $p_2 = p_4 = 1$. For the other objects, o_1, o_3, and o_6, $p_1 = 0.4$, $p_3 = 0.7$, and $p_6 = 0.5$. For the objects with $p_i \geq 0.5$, they are selected to $\mathcal{A}$; hence $\mathcal{A} = \{o_2, o_3, o_4, o_6\}$ and $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{5}[(1 - p_1) + p_2 + p_3 + p_4 + p_6] = \frac{1}{5}[(1 - 0.4) + 1 + 0.7 + 1 + 0.5] = 0.76$.



(a) Range query (b) 3-NN query

Fig. 5: Accuracy measurement.

Theorem 2: For a range query, we can maximize the accuracy of its approximate answer $\mathcal{A}$, $Acc(\mathcal{O}, \mathcal{A})$, by including an object o_i with a probability of being part of its answer $p_i \geq 0.5$ in $\mathcal{A}$ and excluding o_i with $p_i < 0.5$ from $\mathcal{A}$.

Proof: To maximum $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{|\mathcal{O}|}[\sum_{o_i \in \mathcal{A}} p_i + \sum_{o_i \in \mathcal{O}-\mathcal{A}}(1 - p_i)]$, we need to select $\max(p_i, 1 - p_i)$ in each term in the summation. Thus the maximal value of $Acc(\mathcal{O}, \mathcal{A})$ is $\frac{1}{|\mathcal{O}|}\sum_{o_i \in \mathcal{O}} \max(p_i, 1 - p_i)$. If $p_i \geq 0.5$, we include o_i in $\mathcal{A}$ to yield a term p_i (since $p_i \geq 1 - p_i$). However, if $p_i < 0.5$, we exclude o_i from $\mathcal{A}$ to yield a term $1 - p_i$ (since $1 - p_i > p_i$). □

k-NN queries. The accuracy measurement for k-NN queries has two steps.

Step 1: Distance threshold step. We already find the required search area of a k-NN query during the coverage measurement. We consider the objects of interest of the query $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$ whose exact locations or adjusted location regions intersect the required search area. For each object $o_i \in \mathcal{O}$ with an adjusted location region, we determine a distance range $[d_{min_i}, d_{max_i}]$, where d_{min_i} and d_{max_i} are the smallest and largest possible distances between o_i and the querying user u, respectively. On the other hand, for each object $o_i \in \mathcal{O}$ with an exact location, both d_{min_i} and d_{max_i} are set to the distance between o_i and u. Then the objects in $\mathcal{O}$ are sorted by their smallest possible distances in increasing order. We find the smallest possible distance of the $(k+1)$-st object in the sorted $\mathcal{O}$ as a *minimum threshold distance*, T_{min}, and the largest possible distance of the k-th object in the sorted $\mathcal{O}$ as a *maximum threshold distance*, T_{max}. Figure 5b shows the sorted $\mathcal{O} = \{o_4, o_2, o_6, o_1, o_3, o_5\}$, where the smallest and largest possible distances of the adjusted location region of each object to u are represented by $\sqcup$ and $\sqcap$, respectively. Since o_1 is the $(k+1)$-st object, $T_{min} = d_{min_1}$ that is represented by a dotted line. Since o_6 is the k-th object, $T_{max} = d_{max_6}$ that is represented by a line.

Step 2: Answer selection step. We select the k objects with the smallest minimum possible distance to an answer set $\mathcal{A}$.

Algorithm 1 Continuous spatial query processing

1: **function** PROCESSING(*Query* Q, *LocalCache* $\mathcal{L}$, *Float* C_{min}, A_{min})
   // *Control Flow 1: QoS measurements for a local cache*
2: $S \leftarrow$ the required search area of Q
3: **if** $S \neq$ *null* **and** $cov(S, \mathcal{L}) \geq C_{min}$ **then**
4: $\quad \mathcal{O} \leftarrow$ the objects of interest of Q in $\mathcal{L}$ intersecting S
5: $\quad \mathcal{A} \leftarrow$ an answer derived from $\mathcal{O}$
6: $\quad$ **if** $Acc(\mathcal{O}, \mathcal{A}) \geq A_{min}$ **then return** $\mathcal{A}$
7: **end if**
   // *Control Flow 2: Information sharing with neighbors*
8: $\mathcal{L}_p \leftarrow$ the neighbor's transmission range and the monitored area of the query whose requested object type is the same as Q, and their objects
9: $S \leftarrow$ the required search area of Q
10: **if** $S \neq$ *null* **and** $cov(S, \mathcal{L} \cup \mathcal{L}_p) \geq C_{min}$ **then**
11: $\quad \mathcal{O} \leftarrow$ the objects of interest of Q in $\mathcal{L} \cup \mathcal{L}_p$ intersecting S
12: $\quad \mathcal{A} \leftarrow$ an answer derived from $\mathcal{O}$
13: $\quad$ **if** $Acc(\mathcal{O}, \mathcal{A}) \geq A_{min}$ **then return** $\mathcal{A}$
14: **end if**
   // *Control Flow 3: Query broadcasting*
15: $S \leftarrow$ the required search area of Q
16: Find the objects of interest residing in S
17: Update the area table and object table accordingly
18: Compute a query answer $\mathcal{A}$
19: **return** $\mathcal{A}$

These selected objects are the first k objects in the sorted $\mathcal{O}$ and their minimum possible distances are less than T_{max}. For each object $o_i \in \mathcal{A}$, $p_i = \min(T_{min} - d_{min_i}, d_{max_i} - d_{min_i})/(d_{max_i} - d_{min_i})$ because if the actual distance between o_i and u is equal to or less than T_{min}, o_i must be one of the k-NN to u. Thus the probability of correctly making a decision of including o_i in $\mathcal{A}$ is p_i. For each object $o_j \notin \mathcal{A}$, $p_j = (T_{max} - d_{min_j})/(d_{max_j} - d_{min_j})$ because if the actual distance between o_j and u is equal to or less than T_{max}, o_j could be one of the k-NN to u. Thus the probability of correctly making a decision of excluding o_j from $\mathcal{A}$ is $1 - p_j$.

Figure 5b shows the sorted $\mathcal{O} = \{o_4, o_2, o_6, o_1, o_3, o_5\}$ of a 3-NN query. The first three objects, o_4, o_2, and o_6, are selected to an answer set $\mathcal{A}$. Each object $o_i \in \mathcal{A}$ must be one of the k-NN to u if the actual distance between o_i and the querying user u is within a distance range $\min(T_{min} - d_{min_i}, d_{max_i} - d_{min_i})$, which is represented by a gray bold line. We assume $p_4 = 1$, $p_2 = 0.3$, and $p_6 = 0.2$. For each object $o_j \notin \mathcal{A}$, o_1, o_3, and o_5, o_j could be one of the k-NN to u, if the actual distance between o_j and u is within a distance range $T_{max} - d_{min_j}$, which is also represented by a gray bold line. We assume $p_1 = 0.7$, $p_3 = 0.65$, and $p_5 = 0.5$. Therefore $Acc(\mathcal{O}, \mathcal{A}) = \frac{1}{6}[p_4 + p_2 + p_6 + (1 - p_1) + (1 - p_3) + (1 - p_5)] = \frac{1}{6}[1 + 0.3 + 0.2 + (1 - 0.7) + (1 - 0.65) + (1 - 0.5)] = 0.44$.

After the user submits a query and QoS profile to the *continuous query processor*, which is indicated by the flow labeled by 1a in Figure 2, if the continuous query processor finds an approximate answer that satisfies the user's coverage and accuracy requirements, it returns the answer to the user (this flow is labeled by 1b). Otherwise, the continuous query processor proceeds to the next control flow, that is, information sharing with neighbors (this flow is labeled by 2a).

4.2 Information Sharing with Neighbors

This is the second control flow in our framework, which is indicated by thick lines in Figure 2 (Lines 8 to 14 in Algorithm 1). When a querying user u fails to get an answer from the local cache, the continuous query processor initiates this control flow. Since this control flow is very simple, we only present its main idea for both range and k-NN queries, which can be summarized into two steps.

Step 1: Information sharing step. In this step, u sends the query to the neighbors through broadcast communication. Each neighbor p replies to u with (1) the area of p's transmission range along with its objects that belong to the requested object type of u's query; and/or (2) the monitored required search area of each p's continuous query along with its monitored objects if its requested object type is the same as u's query, through point-to-point communication.

Step 2: Answer refinement step. After u receives the information from her neighbors, u updates the location information of the objects stored in the object table accordingly (this flow is labeled by 2b in Figure 2). If the coverage of the information stored in u's local cache and the information returned by the neighbors with respect to the query satisfies u's coverage requirement, u derives a new answer from the local cache. If the answer also satisfies u's accuracy requirement, the continuous query processor returns the answer to u (this flow is labeled by 2c). If the answer still cannot satisfy u's QoS requirements, the query processor proceeds to the query broadcasting control flow (this flow is labeled by 3a).

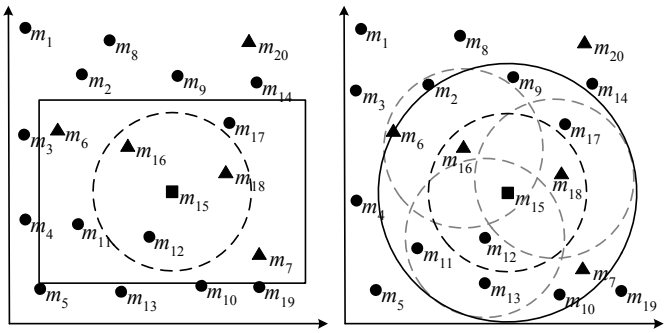
4.3 Query Broadcasting

This is the third control flow in our framework, which is indicated by very thick lines in Figure 2 (Lines 15 to 19 in Algorithm 1). The key functions of this control flow are to (1) search the objects of interest of a query residing in its required search area in order to find the most accurate answer; and (2) have an opportunity for a peer to update an object's location stored in the local cache when the peer is involved in routing messages for query processing, which contains the more updated location of the object. The latter function is useful to reduce the location uncertainty of the objects stored in the local cache. Figures 6a and 6b illustrate this control flow for range and k-NN queries, respectively. In both examples, there are 20 mobile users, m_1 to m_{20}, where m_{15} is a querying user (represented by a square), the objects of interest of the query are m_6, m_7, m_{16}, m_{18}, and m_{20}, which are represented by triangles, and other users are represented by circles. A user's transmission range is represented by a dotted circle. Since this control flow will provide the most accurate answer for a query, the relevant information of this query is removed from the query table and the object table before initiating this control flow. In general, the query broadcasting control flow has two main steps.

4.3.1 Required search area step

We will discuss how to find the required search area of a range or k-NN query.

Range queries. Since the required search area of a range query is the same as its query region, the querying user u can determine the required search area without enlisting peers for help. In Figure 6a, the required search area of the range query is represented by a rectangle.



(a) Range query (b) 3-NN query

Fig. 6: Query broadcasting for range and NN queries.

k-NN queries. For a k-NN query, the querying user u needs to broadcast the query to peers to find at least k objects. Then the required search area of the query is a circular area centered at u with a radius of the required search range of the query, which is the distance from u to the k-th nearest object. To find at least k objects, u broadcasts the query with a hop distance $h = 1$ to the neighbors. If u cannot find at least k objects within one hop, u increases h by one and rebroadcasts the query with the updated h to the neighbors. When a peer p receives the query, p sends its information to u through point-to-point communication. If the received h is larger than one, p decreases h by one and forwards the query with the updated h to the neighbors through broadcast communication. p simply drops duplicate messages without processing or forwarding them. It is important to note that when a peer participates in routing a reply to u, the peer has an opportunity to update the location information of the objects stored in the local cache. u keeps performing this broadcast process until u finds at least k objects of interest. It is expected that u receives replies from more peers as h increases. Receiving the replies from the same set of peers with two consecutive hop distances, h and $h+1$, implies that the total number of objects of interest of u's query in u's network partition or the system is less than k. When this case takes place, u can postpone this step for a while or proceed to the next step by reducing k to the number of objects of interest that are already found by this step.

In Figure 6b, the querying user $u = m_{15}$ issues a 3-NN query. After u broadcasts the query with a hop distance $h = 1$ to the neighbors, u receives replies from three peers, m_{12}, m_{16}, and m_{18}, which are located in u's transmission range (represented by a black dotted circle). Among these three peers, u finds two objects of interest, m_{16} and m_{18}. Since u requires three objects to determine a required search area, u rebroadcasts the query with an increased hop distance $h = 2$. Then u receives replies from five more peers, m_2, m_6, m_{11}, m_{13}, and m_{17}, which are located in the transmission ranges of m_{12}, m_{16}, and m_{18} (represented by gray dotted circles). Since u finds three objects of interest, m_6, m_{16}, and m_{18}, u terminates the broadcast process and determines the required search area. The required search area is a circle centered at u with a radius of the required search range, which is the distance from u to the 3-th nearest object, m_6, which is represented by a black circle. Although u already finds three objects of interest, they do not constitute a correct query answer. This is because the correct answer is m_{16}, m_{18} and, m_7. This missing of m_7, which is outside the searched area with $h = 2$, will be resolved in the next step.

Algorithm 2 Continuous query answer maintenance

```
1: function MAINTENANCE(Query Q, LocalCache L, Float C_min, A_min)
   // Control Flow 4: Continuous query answer maintenance
2: if Q is a k-NN query then
3:     Send Q and AggregateMaxSpeed to the peers residing in S
4: end if
5: Update the object table when receiving a notification message
6: Periodically evaluate the query answer A
7: if A becomes uncertain and Acc(A) < A_min then
8:     Go to Line 2 in Algorithm 1 to start the Control Flow 1
9: else return A
```

4.3.2 Query dissemination step

Once we finish the required search area step, both range and k-NN queries are boiled down to range queries with their required search areas as the range query region. The main idea of this step to retrieve the objects of interest within the range query region. u broadcasts the query along with the range query region to the neighbors. When a peer p receives the query, if p belongs to the requested object type, p sends its information to u through point-to-point communication. In addition, if p's transmission range intersects the range query region, p rebroadcasts the query to the neighbors. Similar to the required search area step, the peer participating in routing messages has an opportunity to update the peer location information stored in the local cache. After u receives the replies from the objects of interest within the range query region, u computes the answer.

Range queries. The answer of a range query simply includes the objects of interest located in the required search area. Figure 6a shows that u receives replies from four objects of interest, m_6, m_7, m_{16}, and m_{18}, and these objects constitute the query answer.

k-NN queries. Among the objects of interest located in the required search area of the query, u selects the k-nearest objects as the answer. Figure 6b shows that u receives replies from four objects of interest, m_6, m_7, m_{16}, and m_{18}, u realizes that m_7 is closer to herself than m_6. Thus u can find a correct answer, which includes m_7, m_{16}, and m_{18}.

After the continuous query processor finds the query answer, it updates the local cache accordingly (this flow is labeled by 3b). The required search space of the query along with the ID list of its objects of interest residing in the required search area is inserted into the area table. Then the information of these objects is inserted into the object table. Finally the answer is returned to the user (this flow is labeled by 3c). We will describe how to monitor the answer in the continuous query answer maintenance control flow.

4.4 Continuous Query Answer Maintenance

This is the fourth control flow in our framework, which is indicated by a very thick white line in Figure 2 and outlined by Algorithm 2. The main tasks of this control flow are (1) collaborative query maintenance: the querying user

collaborates with peers to forward the query to other peers who could become part of the answer (Section 4.4.1); and (2) query evaluation: the querying user computes the answer locally as long as the answer derived from the local cache satisfies her QoS requirements (Section 4.4.2).

4.4.1 Collaborative query maintenance

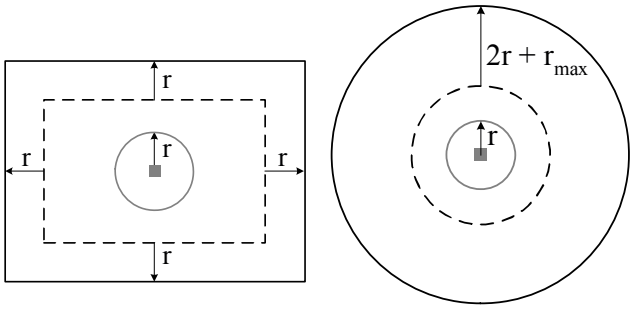
Each user maintains a *query table*. In the query table, a range query is stored in a form (*ID*, *Loc*, *TS*, *Range*, *Period*, *MaxSpeed*, *ObjectOfInterest*), where *ID* is the query issuer's identity, *Loc* is the query location point, *TS* records the time when the query is issued, *Range* is the required search range of the query, *Period* is the valid time period of the query, *MaxSpeed* is the maximum mobility speed of the query issuer, and *ObjectOfInterest* is the requested object type of the query. Likewise, a k-NN query is stored in a similar form (*ID*, *Loc*, *TS*, *Range*, *Period*, *MaxSpeed*, *ObjectOfInterest*, *AggregateMaxSpeed*), where *AggregateMaxSpeed* is the maximum mobility speed of the objects of interest that could reside in the required search area of the query. In general, the collaborative query maintenance task has three main steps.

Step 1: Query dissemination step. This step mainly disseminates a query to the peer located in the required search area. For a range query, since its required search area is the same as its query region, the peer can store the query in the query table during the query dissemination step in the query broadcasting control flow, as discussed in Section 4.3. On the other hand, for a k-NN query, it needs to disseminate an extra parameter for the query, *AggregateMaxSpeed*, to the peer located in its required search area. Thus after the querying user u gets a query answer, u broadcasts the query and its *AggregateMaxSpeed* to the peer residing in the required search area. Then the peer stores the query in the query table.

Step 2: Query table synchronization step. When a peer p discovers a new neighbor p', this step takes place to synchronize the query tables of p and p' through a three-way message exchange via point-to-point communication. Without loss of generality, we assume that the peer with a smaller ID initiates this step. We consider that p initiates this step. (1) p sends a list $p.L$ of the query IDs in the query table to p'. (2) After p' receives $p.L$, p' generates a list $p'.L$ of the query IDs in the query table. Then p' replies to p with the information of the queries in $p'.L$ but not in $p.L$, $p'.L \setminus p.L$, and a list of query IDs in $p.L$ but not in $p'.L$, $p.L \setminus p'.L$. (3) p stores the received queries in the query table and sends the information of the queries included in the received list to p'. p' stores the received queries in the query table.

Step 3: Notification step. For each new query received during the query table synchronization step, the peer, p and/or p', performs this step to decide whether to send its information to the query issuer. Due to user mobility, the peer has to adjust the required search area of a range or k-NN query to capture the effect of location uncertainty. We will discuss how to adjust the required search area of a range or k-NN query.

Range queries. Figure 7a shows a range query, where the original query location point is represented by a gray square and the original required search area is represented by a dotted rectangle. Similar to the location adjustment of peer locations, we use a conservative approach to adjust the required search area of the range query, where the querying user u can move at the maximum mobility speed $\textit{MaxSpeed}_u$ in any direction. Thus the distance from the original query location point and the u's current location is at most $r = (t_{current} - \textit{TS}) \times \textit{MaxSpeed}_u$. In other words, u could be anywhere within a circular location region centered at the original query location point with a radius of r, which is represented by a gray circle. To ensure that an adjusted required search area contains all the possible range query regions, regardless of the u's actual location within u's adjusted location region, we extend each edge of the original required search area by r. The adjusted required search area is represented by a rectangle.



(a) Range Queries (b) k-NN Queries

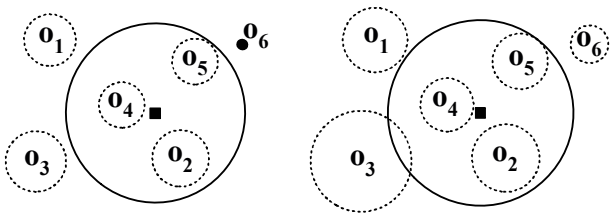
Fig. 7: Required search area adjustment.

k-NN queries. Figure 7b shows a k-NN query, where the original query location point is represented by a gray square and the original required search area is represented by a dotted circle. Similar to range queries, the adjusted location region of a querying user u is centered at the original query location point with a radius of $r = (t_{current} - \textit{TS}) \times \textit{MaxSpeed}_u$, which is represented by a gray circle. Consider a case that all the objects of interest residing in the required search area move at *AggregateMaxSpeed*; therefore the maximum possible distance between u and each of these objects is $r + r_{max} + \textit{Range}$, where $r_{max} = (t_{current} - \textit{TS}) \times \textit{AggregateMaxSpeed}$, and *Range* is the original required search range, which is the original distance between u and the k-th nearest object in the answer. Since u could be anywhere within the adjusted location region, the adjusted required search area is a circular area centered at the original query location point with a radius of $2 \times r + r_{max} + \textit{Range}$. Whenever u finds a larger *AggregateMaxSpeed*, u broadcasts it to the objects residing in the adjusted required search area.

After the peer, p and/or p', determines the required search area of a newly received query, if the peer is located in the adjusted required search area, the peer sends her information as a notification message to the querying user through point-to-point communication. Otherwise, the peer will periodically evaluate the query until the query is expired.

4.4.2 Query evaluation

The main idea of this task is that the querying user, u, checks whether to derive a certain query answer from the local cache. A certain query answer can be returned to u without any QoS measurement. If u cannot get a certain answer from the local cache, u can find an answer from the local cache as



(a) Certain answer (b) Uncertain answer

Fig. 8: The answer of a 3-NN query.

long as the answer satisfies her QoS requirements. In case that u cannot find an answer from that satisfies her QoS requirements, u restarts the query processing by executing the second control flow: information sharing with neighbors. Once u receives a notification message through the collaborative query maintenance task, u updates the object table accordingly. It is important to note that u will adjust the required search area of the queries stored in the area table as in Section 4.4.1, when u is requested to return the information of the local cache to a peer in the information sharing step in the second control flow. We will describe how to check the certainty of a range or k-NN query answer.

Range queries. Since the required search area of a range query is the same as its query region, the querying user u always knows the exact required search area of the query. Thus u simply adjusts the location of the objects of interest stored in the local cache, and then selects the objects with adjusted location regions intersecting the query region. If all these objects are totally included in the required search area, the accuracy of an answer that includes all these objects is one; hence this answer is a certain one.

k-NN queries. In contrast to range queries, u needs to determine the required search area of a k-NN query, and calculate the minimum and maximum possible distances between each object of interest in the local cache to u's current location. Then u finds an object p with the k-th smallest maximum possible distance. The required search area is a circle centered as u's current location with a radius of the maximum distance between p and u. If there are only k objects intersecting the required search area, their adjusted locations or exact locations must be totally included in the required search area; therefore these k objects constitute a certain query answer.

Figure 8 shows the answer of a 3-NN query at two time instances, where the current location of the querying user u is represented by a black square, and the exact location or the adjusted location region of each object of interest of the query is represented by a black circle or a dotted circle, respectively. Figure 8a shows an answer, in which o_5 has the third smallest maximum possible distance to u; therefore the required search area (represented by a circle) is a circular area with a radius of the distance from u to the farthest point on the adjusted location region of o_5 from u. Since there are only three objects, o_2, o_4, and o_5, whose adjusted location regions are totally covered by the adjusted required search area, these objects constitute a certain answer. On the other hand, Figure 8b shows the query answer at a later time, where the adjusted location regions of objects, o_1 to o_5, are expanded, and the location of o_6 is adjusted. o_5 still has the third smallest maximum possible distance to u, so the required search area is a circle at u with a radius of the distance from u to the farthest point on the adjusted location region of o_5. Since the adjusted location regions of four objects, o_2, o_3, o_4, and o_5, intersect the required search area, any three of these four objects could be the actual query answer; hence the answer becomes uncertain, and u needs to check the accuracy of the query answer.

5 Simulation Model

In this section, we present a simulation model that is used to evaluate our continuous spatial query processing framework (denoted as ContQP) in a mobile P2P environment.

5.1 Baseline Algorithms

To our best knowledge, ContQP is the first framework realizing continuous spatial query processing in mobile P2P environments. We design two baseline algorithms that have a subset of the control flows of ContQP to evaluate the performance of ContQP. Since the continuous query answer maintenance scheme (the fourth control flow) is the one of the key features of our framework, the baseline algorithms do not have this control flow. (1) In the first baseline algorithm (denoted as LocalQP), the user only monitors the transmission range. If the user cannot find an answer that satisfies her QoS requirements from the local cache, that is, the first control flow: QoS measurements for the local cache, the user executes the query broadcasting control flow (the third control flow) to find the answer. (2) In the second baseline algorithm (denoted as PeerQP), if the user fails to find an answer from a local cache, the user proceeds to the information sharing with neighbors (the second control flow) to find the answer. If the answer still cannot satisfy the user's QoS requirements, the user executes the query broadcasting control flow to find the answer.

5.2 Power Consumption Model

Each mobile user is equipped a wireless network interface card that supports two communication methods: point-to-point and broadcast communication. The user can communicate with all neighbors through broadcast communication, one of the neighbors through point-to-point communication, and a multi-hop peer through a point-to-point multi-hop routing that contains a sequence of point-to-point communication. In this work, we focus on the application layer and do not have any assumption on the underlying multi-hop routing protocol; therefore any multi-hop routing protocol can be applied to our framework. It has shown that the power consumption of wireless communication can be modeled by linear formulas in terms of message sizes and communication methods [26].

For point-to-point communication, P_{point}, a source user S sends a message to a destination user D. The affected users of this communication method are the users who reside in S's transmission range, R_S, D's transmission range, R_D, and their transmission ranges, R_{SD}. The power consumption of P_{point}

TABLE 1: Parameters for point-to-point communication.

Conditions	μW· s/byte	μW · s
S	$v_{send} = 1.9$	$f_{send} = 454$
D	$v_{recv} = 0.5$	$f_{recv} = 356$
Peers $\in R_{SD}$	$v_{sd\_disc} = 0$	$f_{sd\_disc} = 70$
Peers $\in R_S$	$v_{s\_disc} = 0$	$f_{s\_disc} = 24$
Peers $\in R_D$	$v_{d\_disc} = 0$	$f_{d\_disc} = 56$

TABLE 2: Parameters for broadcast communication.

Conditions	μW· s/byte	μW · s
S	$v_{bsend} = 1.9$	$f_{bsend} = 266$
Peers $\in R_S$	$v_{brecv} = 0.5$	$f_{brecv} = 56$

is measured by the following equations:

$$P_{point} = \begin{cases} (v_{send} \times |msg|) + f_{send}, & \text{for } S \\ (v_{recv} \times |msg|) + f_{recv}, & \text{for } D \\ (v_{sd\_disc} \times |msg|) + f_{sd\_disc}, & \text{for peers} \in R_{SD} \\ (v_{s\_disc} \times |msg|) + f_{s\_disc}, & \text{for peers} \in R_S \\ (v_{d\_disc} \times |msg|) + f_{d\_disc}, & \text{for peers} \in R_D \end{cases},$$

where f is a fixed setup cost, and v is a variable cost in terms of the size of a message msg in bytes, $|msg|$.

For broadcast communication, P_{bc}, a source user S broadcasts a message to the peers residing in S's transmission range, S_R. The power consumption of P_{bc} is measured by the following equations:

$$P_{bc} = \begin{cases} (v_{bsend} \times |msg|) + f_{bsend}, & \text{for } S \\ (v_{brecv} \times |msg|) + f_{brecv}, & \text{for peers} \in R_S \end{cases}.$$

The parameter settings for P_{point} and P_{bc} are depicted in Tables 1 and 2, respectively.

5.3 Performance Metrics

We evaluate the performance of our framework in terms of four metrics. (1) Number of messages. This metric measures the average number of messages incurred by our framework per query evaluation. (2) Power consumption. This metric measures the average power consumption per query evaluation based on the power consumption models described in Section 5.2. (3) False negative. This metric measures the average relative number of objects missed in an approximate answer compared to an actual one. Given an actual answer set A and an approximate answer set $\widehat{A}$, the relative false negative is computed as $|A \setminus \widehat{A}|/|A|$. (4) False positive. This metric measures the average relative number of extra objects in an approximate answer compared to an actual one. The false positive is computed as $|\widehat{A} \setminus A|/|A|$.

5.4 Simulation Settings

We implement our framework for both range queries "continuously report the object(s) of a specific type within a certain range from a query issuer", and k-NN queries "continuously report the k-nearest object(s) of a specific type to a query issuer" in C++. Each experiment runs 1,000 seconds. Unless mentioned otherwise, we generate 200 mobile users moving at a speed distributed uniformly between 1 and 20 meter(s) per second based on the "random waypoint" model [27] in a 1,000m $\times$ 1,000m space. Each user belongs to one of 10 object types. 20% of the users issue continuous queries for a time period of 1,000 seconds, and they evaluate their query answers every second. The default coverage, C_{min}, and accuracy, A_{min}, requirements are set to 0.8. The grid cell area for the approximate coverage measurement is 16 m^2. The transmission range of each user is 100 meters, and the beacon interval is one second. We first assume that a querying user receives the beacon her neighbors before evaluating a query, and then we remove this assumption by increasing the beacon interval in Section 6.6. For query parameters, the range distance, $Range$, of range queries is 200 meters, and the required number of objects, k, of k-NN queries is five. Table 3 summarizes the parameter settings.

TABLE 3: Parameter settings for experiments.

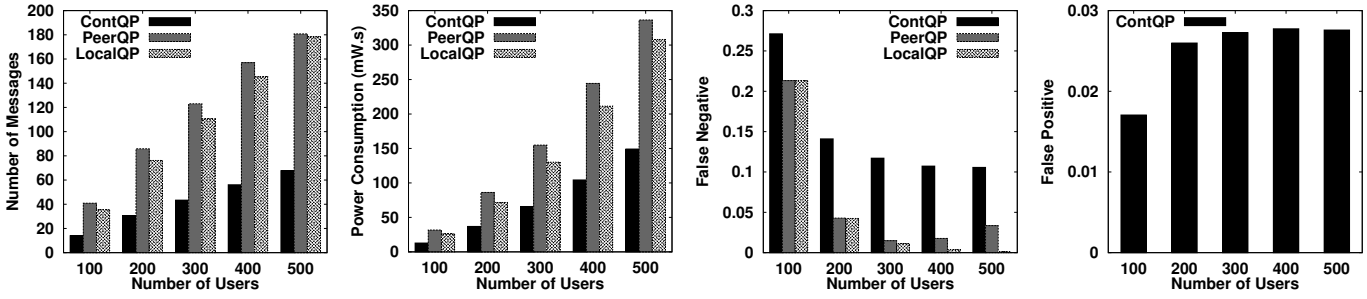
Parameters	Default Values	Ranges
No. of mobile users	200	100 to 500
Coverage (C_{min})	0.8	0.5 to 1.0
Accuracy (A_{min})	0.8	0.5 to 1.0
Mobility speed	$[1, 20]$ m/sec	$[1, 5]$ to $[1, 30]$ m/sec
No. of objects (k)	5	2 to 10
Range distance ($Range$)	200 meters	150 to 350 meters
Grid cell area	4^2 m^2	2^2 to 12^2 m^2
Beacon interval	1 sec	1 to 5 sec
Query period	1,000 sec	2 to 10 sec
No. of queries	20% of users	5% to 40% of users
Transmission range	100 meters	-
No. of object types	10	-

6 Experimental Results

This section presents the experimental results of our framework, ContQP, in comparison with the baseline algorithms, LocalQP and PeerQP, with respect to various numbers of users, numbers of continuous queries, QoS requirements, query parameters (the range distance of range queries and the value of k of k-NN queries), user mobility speeds, beacon intervals, and query period intervals.

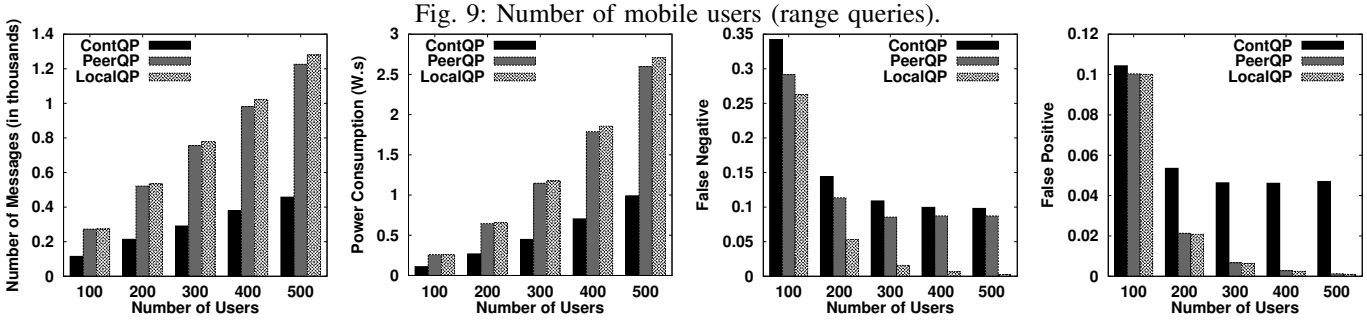
6.1 Effect of the Number of Users

Figures 9a and 9b show that ContQP outperforms the baseline algorithms in terms of number communication overhead. The reason is that LocalQP executes the query broadcasting control flow for each query evaluation and PeerQP only slightly reduces the number of times of executing this control flow. On the other hand, ContQP effectively avoids executing the relatively expensive query broadcasting control flow through the continuous query answer maintenance control flow that efficiently maintains the user's local cache. Increasing the number of users results in more users residing in the required search area of queries, thus the communication overhead increases. Figure 9c shows that the false negative improves, as there are more users. In a sparser environment, the user is more likely to suffer from a network partition problem, in which the user is unable to communicate with all objects residing in the query's required search area. With a higher user density, the user has a lower probability of suffering from the network partition problem, so the false negative reduces. When there are more users, ContQP has a higher probability to select more extra objects to an answer (Figure 9d). Since the false positive of ContQP for range queries in all the experiments is



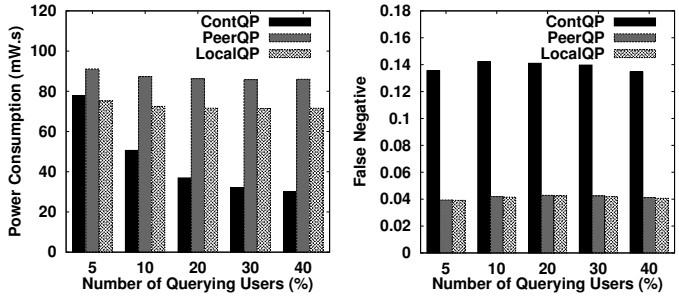
(a) Number of messages (b) Power consumption (c) False negative (d) False positive

Fig. 9: Number of mobile users (range queries).



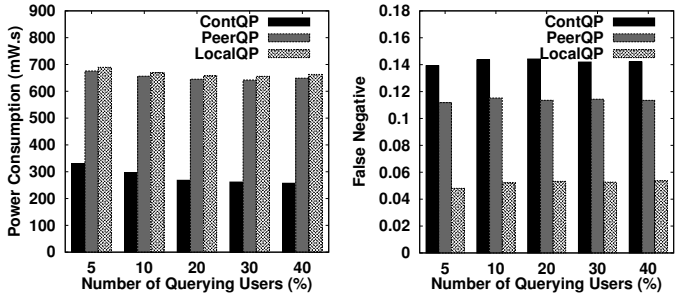
(a) Number of messages (b) Power consumption (c) False negative (d) False positive

Fig. 10: Number of mobile users (k-NN queries).



(a) Power consumption (b) False negative

Fig. 11: Number of querying users (range queries).



(a) Power consumption (b) False negative

Fig. 12: Number of querying users (k-NN queries).

very small, that is, always less than 0.03, we will not present the false positive of range queries in other experiments.

Similar to range queries, ContQP outperforms the baseline algorithms for k-NN queries in terms of communication overhead, as the number of users increases (Figures 10a and 10b). Due to network partitions, all the algorithms may not be able to find the exact k-nearest objects to the user; thus the false negative is always larger than zero (Figure 10c). In contrast to range queries, all the algorithms compute a k-NN query answer with at most k objects, so only presenting their false negatives is good enough to evaluate the answer accuracy. As Figures 9 and 10 show that the number of messages and power consumption have the same trend, only the results of power consumption will be presented in other experiments.

6.2 Effect of the Number of Continuous Queries

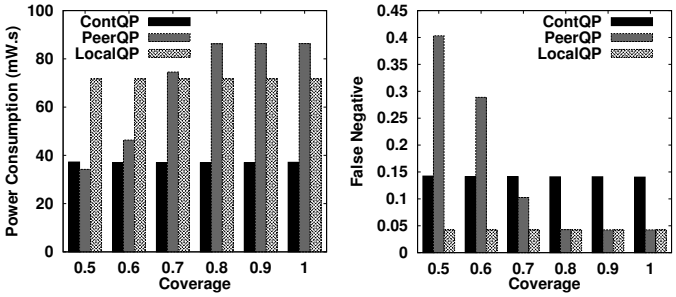
Figures 11 and 12 show the performance of all the algorithms with respect to increasing the number of querying users from 10 (5%) to 80 (40% of the total number of mobile users). Figure 11a shows that ContQP effectively reduces the power consumption, as the number of querying users increases. The main reason is that when there are more querying users, the user of ContQP has a higher probability to get an answer from the information shared by neighbors about the adjusted research search area of their queries and their transmission ranges . Since PeerQP does not continuously maintain the query answer, the user only as a higher chance to get the query answer from the information shared by neighbors about their transmission ranges; thus PeerQP slightly reduces the power consumption. The number of querying users has only slight effect on the false negative (Figure 11b). Figure 12 shows that the results of k-NN queries are similar to that of range queries.

6.3 Effect of QoS Requirements

TABLE 4: Approximate coverage computation.

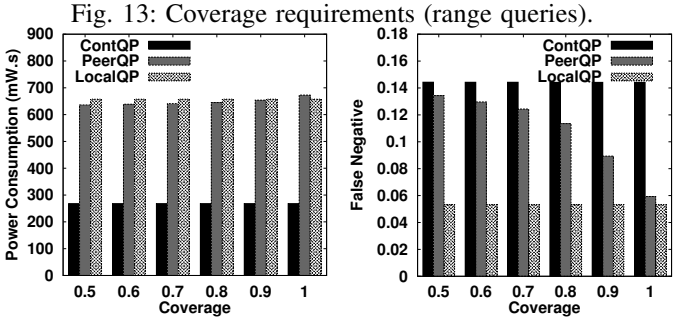
Grid Size Area	2^2	4^2	6^2	8^2	10^2	12^2
Coverage	0.632	0.624	0.612	0.606	0.6	0.586
Computation time (ms)	137.963	35.094	15.77	9.173	5.932	4.228

Table 4 shows the precision of the approximate coverage measurement with respect to increasing the grid cell area from 2^2 to 12^2 m^2. Since the bit of a grid cell is set to one, if it is totally covered by some monitored area that is either stored in a user's local cache or shared by a user's neighbor, increasing



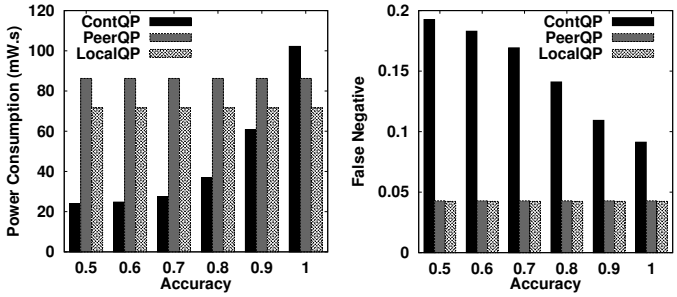
(a) Power consumption (b) False negative

Fig. 13: Coverage requirements (range queries).



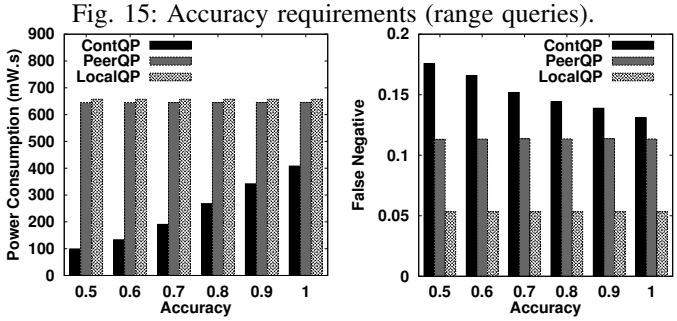
(a) Power consumption (b) False negative

Fig. 14: Coverage requirements (k-NN queries).



(a) Power consumption (b) False negative

Fig. 15: Accuracy requirements (range queries).



(a) Power consumption (b) False negative

Fig. 16: Accuracy requirements (k-NN queries).

the grid cell area leads to a larger underestimation of the approximate coverage measurement. The results indicate that when the grid cell area increases, the precision only degrades slightly, while the computational time significantly decreases. Therefore, we choose 16 m^2 as the default grid cell area for all the experiments because this area size gives the greatest improvement ratio on the computational time.

Figures 13 and 14 show that the coverage requirement, C_{min}, only affects ContQP slightly for range and k-NN queries, respectively. After ContQP finds a query answer, the query's required search area is contained by the adjusted required search area that is monitored by the continuous query answer maintenance control flow, the coverage of the information stored in the local cache with respect to the query is one. Figure 13a shows that PeerQP incurs less communication overhead than ContQP, as C_{min} is small. This is because the user of PeerQP is more likely to find an answer from the local cache or the information shared by neighbors; however, the query answer accuracy is very low (Figure 13b).

Figures 15 and 16 show that when the accuracy requirement, A_{min}, gets larger, the communication overhead of ContQP increases, while its accuracy improves. This is because a larger A_{min} results in a higher probability that the user needs to execute the relatively expensive query broadcasting control flow to find the most accurate possible query answer. However, LocalQP executes this control flow for each query evaluation, while PeerQP has a much higher chance to execute this control flow than ContQP. Therefore, A_{min} achieves a tradeoff between the communication overhead and the query answer accuracy for ContQP.

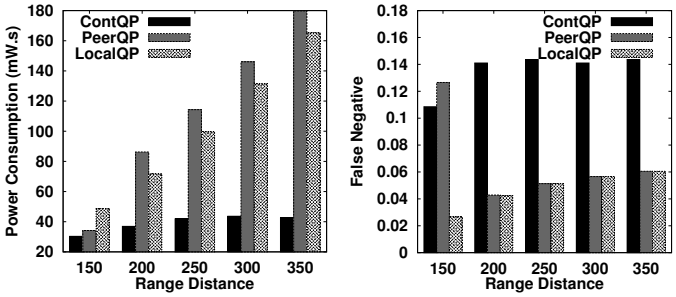
6.4 Effect of Query Parameters

Figure 17 shows the results of all the algorithms with respect to increasing the range distance, *Range*, of range queries from 150 to 350 meters. It is expected that the communication overhead increases, as *Range* gets larger (Figure 17a). ContQP is more scalable than the baseline algorithms, because ContQP has a higher probability to find an answer from the information shared by neighbors, when *Range* increases. The increase of *Range* results in larger required search areas for queries, where the user is more likely to suffer from the network partition problem; thus the false negative increases (Figure 13b).

Figure 18 shows the performance of all the algorithms with respect to increasing the required number of objects of interest, k, of k-NN queries from 2 to 10. Similar to range queries, increasing k incurs higher power consumption and ContQP is more scalable than the baseline algorithms (Figure 18a). When k is larger, there is a higher probability for the user of ContQP and PeerQP to execute the query broadcasting control flow to find the most accurate possible answer; thus their query answer accuracy improves (Figure 18b).

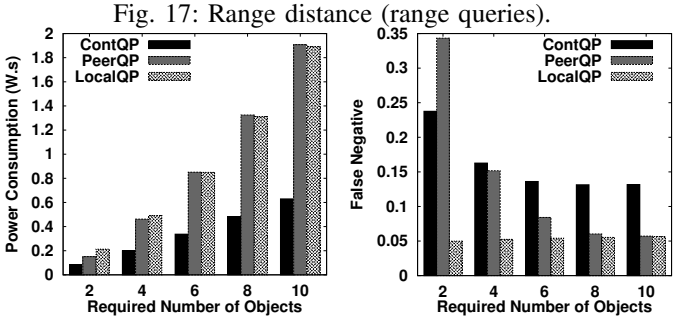
6.5 Effect of Mobility Speeds

Figures 19 and 20 show the performance evaluation with respect to varying the user mobility speed from $[1, 5]$ to $[1, 30]$ meters per second for range and k-NN queries, respectively. Since LocalQP executes the query broadcasting control flow for each query evaluation, it is not affected by the mobility speed. PeerQP is only slightly affected by the mobility speed, as it only avoids a small number of times of executing the query broadcasting control flow. When the mobility speed increases, ContQP incurs higher power consumption (Figures 19a and 20a). The reason is that increasing the mobility speed results in higher uncertainty in location information, which leads to a higher decay rate in the accuracy of an answer derived from a local cache; thus ContQP has a higher probability to execute the relatively expensive query broadcasting control flow to find the answer. Since the user can get



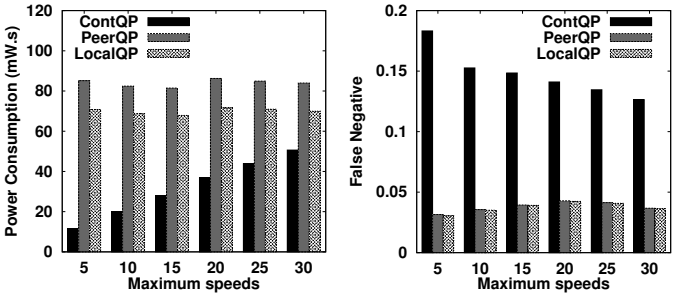
(a) Power consumption (b) False negative

Fig. 17: Range distance (range queries).



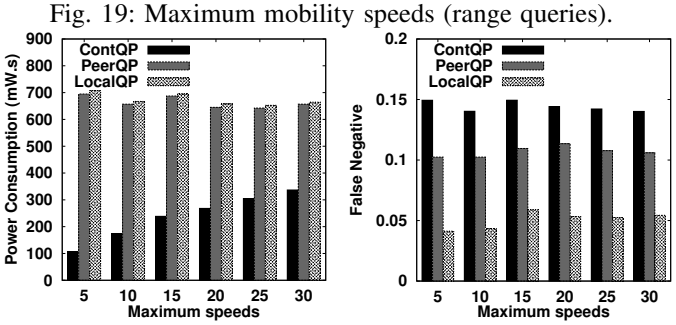
(a) Power consumption (b) False negative

Fig. 18: Required number of objects (k-NN queries).



(a) Power consumption (b) False negative

Fig. 19: Maximum mobility speeds (range queries).



(a) Power consumption (b) False negative

Fig. 20: Maximum mobility speeds (k-NN queries).

the most accurate possible answer through this control flow, the query answer accuracy improves, as the mobility speed increases (Figures 19b and 20b).

6.6 Effect of Beacon Intervals

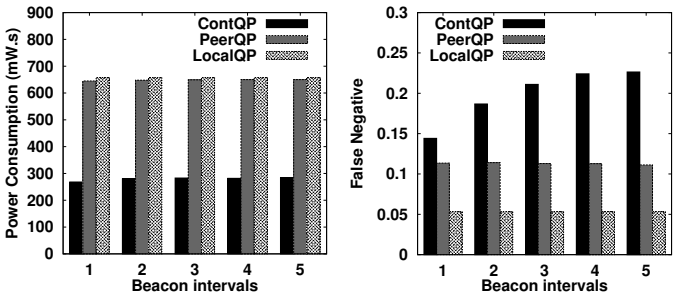
Figure 21 shows the performance of all the algorithms for k-NN queries, as the beacon interval varies from one to five seconds. In ContQP, the continuous query answer maintenance control flow requires the user to disseminate the received queries to newly discovered neighbors. Since increasing the beacon interval leads to a longer delay in this query dissemination process, it takes a longer time for the user to get the notification messages from the peers who could become part of the answer. The delay of these notification messages decreases the accuracy of the answer derived from the user's local cache (Figure 21b). Thus the user has a higher probability to execute the relatively expensive query broadcasting control flow to find the answer; hence ContQP has higher power consumption, as the beacon interval increases (Figure 21a). Although PeerQP does not execute the continuous query answer maintenance control flow, its power consumption also slightly increases with larger beacon intervals. This is because the user of PeerQP takes a longer time to get the location of neighbors that also decreases the accuracy of the answer derived from the user's local cache.

6.7 Effect of Query Period

Figure 22 shows the results of all the algorithms with respect to increasing the time period of continuous range queries from 2 to 10 seconds. Since LocalQP finds a query answer from scratch for each query evaluation, its performance is not affected by the query period. The user of PeerQP has to execute the query broadcasting control flow to an answer in most cases, so its performance is slightly affected by the query period. However, the communication overhead of ContQP reduces and then stabilizes, as the query period increases (Figure 22a). In the experiment, we maintain the same number of querying users, decreasing the query period results in more new querying users that execute the relatively expensive query broadcasting control flow to find initial query answers. Figure 22b shows that the answer accuracy reduces, as the query period increases. This is because the uncertainty of the location information stored in a user's local cache increases for a longer time period.

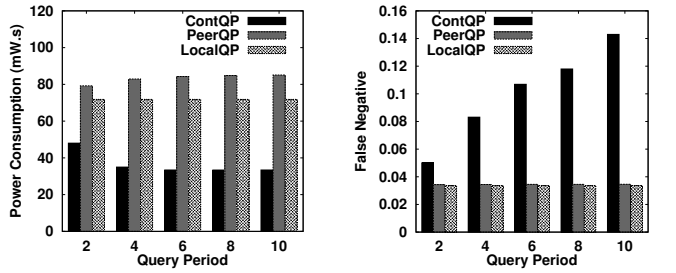
7 Conclusion

We design a continuous query processing framework for range and k-nearest-neighbor queries in mobile peer-to-peer (P2P) environments. Our framework has two key features. (1) Our framework provides an approximate answer for the user with two personalized QoS guarantees, namely, coverage and accuracy. (2) The user is able to collaborate with peers to maintain query answers. Our framework has four main control flows. (i) The user derives an answer from the local cache and measures the quality of the answer. If the answer satisfies the user's QoS requirements, it is returned to the user. (ii) Otherwise, the user enlists neighbors for help to turn in their cached information to refine the answer. (iii) If the refined answer still cannot satisfy the user's QoS requirements, the user searches the required search area of the query to get the answer. (iv) Then the user collaborates with peers to maintain the answer. We evaluate our framework through experiments. The results show that our framework is efficient and scalable in terms of communication overhead, and the QoS requirements achieve a performance tradeoff between the communication overhead and the quality of query answers.



(a) Power consumption (b) False negative

Fig. 21: Beacon intervals (k-NN queries).



(a) Power consumption (b) False negative

Fig. 22: Query period (range queries).

REFERENCES

[1] Y. Cai, K. A. Hua, and G. Cao, "Processing range-monitoring queries on heterogeneous mobile objects," in *Proc. of MDM*, 2004.

[2] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "Distributed group-based cooperative caching in a mobile broadcast environment," in *Proc. of MDM*, 2005.

[3] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "GroCoca: Group-based peer-to-peer cooperative caching in mobile environment," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 179–191, 2007.

[4] B. Gedik and L. Liu, "MobiEyes: Distributed processing of continuously moving queries on moving objects in a mobile system," in *Proc. of EDBT*, 2004.

[5] T. Hara, "Effective replica allocation in ad hoc networks for improving data accessibility," in *Proc. of IEEE INFOCOM*, 2001.

[6] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *Proc. of ACM SIGMOD*, 2005.

[7] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, "Skyline queries against mobile lightweight devices in manets," in *Proc. of IEEE ICDE*, 2006.

[8] W.-S. Ku, R. Zimmermann, and H. Wang, "Location-based spatial queries with data sharing in wireless broadcast environments," *IEEE Transactions on Mobile Computing*, vol. 7, no. 5, 2008.

[9] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable incremental processing of continuous queries in spatio-temporal databases," in *Proc. of ACM SIGMOD*, 2004.

[10] M. Papadopouli and H. Schulzrinne, "Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices," in *Proc. of ACM MobiHoc*, 2001.

[11] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *Proc. of VLDB*, 2002.

[12] H. Cao, O. Wolfson, B. Xu, and H. Yin, "MOBI-DIC: Mobile discovery of local resources in peer-to-peer wireless network," *IEEE Data Engineering Bulletin*, vol. 28, no. 3, pp. 11–18, 2005.

[13] O. Wolfson, B. Xu, and R. M. Tanner, "Mobile peer-to-peer data dissemination with resource constraints," in *Proc. of MDM*, 2007.

[14] O. Wolfson, B. Xu, H. Yin, and H. Cao, "Search-and-discover in mobile P2P network databases," in *Proc. of IEEE ICDCS*, 2005.

[15] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang, "Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors," in *Proc. of IEEE ICDE*, 2007.

[16] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang, "Effective density queries of continuously moving objects," in *Proc. of IEEE ICDE*, 2006.

[17] B. Zheng, J. Xu, W.-C. Lee, and D. L. Lee, "Grid-partition index: A hybrid method for nearest-neighbor queries in wireless location-based services," *VLDB Journal*, vol. 1, no. 15, pp. 21–39, 2006.

[18] N. Chand, R. C. Joshi, and M. Misra, "Cooperative caching strategy in mobile ad hoc networks based on clusters," *Wireless Personal Communications*, vol. 1, no. 43, pp. 41–63, 2007.

[19] W. Wu and K.-L. Tan, "Global cache management in nonuniform mobile broadcast," in *Proc. of MDM*, 2006.

[20] A. Coman, M. A. Nascimento, and J. Sander, "Exploiting redundancy in sensor networks for energy efficient processing of spatiotemporal region queries," in *Proc. of CIKM*, 2005.

[21] N. Dimokas, D. Katsaros, and Y. Manolopoulos, "Cooperative caching in wireless multimedia sensor networks," *Mobile Networks and Applications*, no. 3-4, pp. 337–356, 2008.

[22] Y. Yao, X. Tang, and E.-P. Lim, "Continuous monitoring of kNN queries in wireless sensor networks," in *Proc. of MSN*, 2006.

[23] Z. J. Haas and M. R. Pearlman, "The performance of query control schemes for the zone routing protocol," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 427–438, 2001.

[24] L. Li, J. Y. Halpern, P. Bahl, Y.-M. Wang, and R. Wattenhofer, "Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks," in *Proc. of ACM PODC*, 2001.

[25] R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow, "Probabilistic verifiers: Evaluating constrained nearest neighbor queries over uncertain data," in *Proc. of IEEE ICDE*, 2008.

[26] L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proc. of IEEE INFOCOM*, 2001.

[27] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proc. of ACM MOBICOM*, 1998.

Chi-Yin Chow (M.Sc., 2008, University of Minnesota; M.Phil., 2005, B.A., 2002, The Hong Kong Polytechnic University) is a Ph.D. candidate in the Department of Computer Science and Engineering, University of Minnesota. His main research interests are in spatial and spatio-temporal databases, mobile data management, wireless sensor networks, and data privacy. He was an intern at the IBM Thomas J. Watson Research Center during the summer of 2008.

Mohamed F. Mokbel (Ph.D., Purdue University, 2005, MS, B.Sc., Alexandria University, 1999, 1996) is an assistant professor in the Department of Computer Science and Engineering, University of Minnesota. His main research interests focus on advancing the state of the art in the design and implementation of database engines to cope with the requirements of emerging applications (e.g., location-aware applications and sensor networks). Mohamed was the co-chair of the first and second workshops on privacy-aware location-based mobile services, PALMS, 2007 (Mannheim, Germany) and 2008 (Beijing, China). He is also the PC co-chair for the ACM SIGSPATIAL GIS 2008 and 2009 conferences. Mohamed has spent the summers of 2006 and 2008 as a visiting researcher at Hong Kong Polytechnic University and Microsoft Research, respectively. He is a member of ACM and IEEE.

Hong Va Leong received his Ph.D. from the University of California at Santa Barbara, and is currently an associate professor at the Hong Kong Polytechnic University. He is the program co-chairs of several conferences, including CIC, IMMCN, HS@I Hong Kong Region, and as ACM SAC mobile computing and applications track chair. He has also served on the organizing committees and the program committees of a number of conferences. He is a reviewer for ACM Transactions on Computer Systems, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Computers, Information Systems, Theoretical Computer Science, and other journals. He has published over ninety refereed research papers. His research interests are in mobile computing, internet computing, distributed systems, distributed databases, and digital libraries. He is a member of the ACM and IEEE Computer Society.