A P2P Technique for Continuous k-Nearest-Neighbor Query in Road Networks

Fuyu Liu, Kien A. Hua, and Tai T. Do

School of EECS, University of Central Florida, Orlando, FL, USA
{fliu,kienhua,tdo}@cs.ucf.edu

Abstract. Due to the high frequency in location updates and the expensive cost of continuous query processing, server computation capacity and wireless communication bandwidth are the two limiting factors for large-scale deployment of moving object database systems. Many techniques have been proposed to address the server bottleneck including one using distributed servers. To address both of the scalability factors, P2P computing has been considered. These schemes enable moving objects to participate as a peer in query processing to substantially reduce the demand on server computation, and wireless communications associated with location updates. Most of these techniques, however, assume an open-space environment. In this paper, we investigate a P2P computing technique for continuous kNN queries in a network environment. Since network distance is different from Euclidean distance, techniques designed specifically for an open space cannot be easily adapted for our environment. We present the details of the proposed technique, and discuss our simulation study. The performance results indicate that this technique can significantly reduce server workload and wireless communication costs.

1 Introduction

With the advances in wireless communication technology and advanced positioning systems, a variety of location based services become available to the public. Among them, one important service is to continuously provide *k-nearest-neighbor* (*k*NN) search for a moving object. Early research effort has focused on moving query over static points of interest. Recently, interest has been shifted to monitoring moving queries over moving objects, e.g., "Give me the five nearest BMW cars while I am driving on Colonial Drive." This new type of query, demanding constant updates from moving objects to keep the query results accurate, raises a great challenge.

A simple mobile query processing system consists of a centralized server and a large number of moving objects. There are two scalability issues for such systems: (1) query processing cost, and (2) location update cost. Addressing the first issue has been the focus of the majority of the existing work [2, 4, 5, 6, 7, 12, 13, 16, 17]. These researches focus on query processing techniques and do not worry about the communication cost associated with location updates. To address the second issue, namely update cost, using distributed servers has been proposed [14] to leverage the aggregate bandwidth of the servers. Another interesting idea for reducing location updates is to use *safe regions* [8, 15] or *thresholds* [19], where an object moving within a *safe region* or a *threshold* does not need to update its location. To address both issues, i.e., expensive query processing cost and intensive location updates, *Peer-to-peer* (P2P) techniques were investigated in [1, 3, 9, 20]. In these schemes, each moving object participates in query processing as a peer by monitoring nearby queries and updating their result if the object's new location affects the query results. The benefits of this strategy are twofold. First, server computation is no longer a bottleneck (i.e., first scalability issue); and second, moving objects need to update their location much less frequently (i.e., second scalability issue).

Most P2P solutions [1, 3, 20] assume an open space environment, where the distance between two objects is the straight line distance between them. In real-life scenarios, many moving objects (e.g., cars) are restricted to move on a network (e.g., road network). Since the distance between two objects in a network is defined as the shortest network distance between them, techniques developed specifically for an open space environment cannot be easily extended to a road network. A more recent P2P technique has been proposed in [9] for dynamic range queries over a network.

In this paper, we focus on *k*NN queries over road networks. Unlike range queries, there are no fixed ranges for *k*NN queries and as objects move around, the ranges constantly change. Therefore, new definitions and new techniques must be developed to address the challenge. We solve the problem by proposing an efficient P2P solution. In our approach, each moving object monitors queries in the neighboring road segments, and will update a query result maintained on a server if the object becomes one of the *k*NN or is no longer one of the *k*NN of the affected query. Besides saving server computation costs, this scheme reduces communications as much less messages are communicated between objects and server compared with a centralized solution.

The contributions of this paper are summarized as follows:

- We introduce a novel way to define the *range* for *k*NN queries in road networks.
- We propose a P2P solution to process *k*NN queries over road networks which has less server computation cost and communication cost.
- We provide simulation study to show the benefits of using the proposed P2P solution.

The remainder of this paper is organized as follows. Related work is discussed in Section 2. Section 3 covers formal definitions and background information. The proposed solution is introduced in Section 4. In Section 5, we present the simulation study. Finally, Section 6 concludes the paper.

2 Related Work

Mouratidis et al. [12] studied the *k*NN monitoring query problem in road networks, where query and data objects all move around. However, their techniques only focused on reducing server workload without worrying about the communication cost and the update cost. As we pointed out in the introduction section, these costs will undermine the scalability of the system. Recently, Wu et al. [20] proposed a distributed solution to answer moving *k*NN queries; nevertheless, the proposed solution is only applicable to open space environments.

To the best of our knowledge, the work most related to ours is the research presented by Jensen et al. in [11], in which an algorithm was given for continuous *k*NN queries. This algorithm takes a client-server approach with the server keeps the location information of all the clients. For a given new query, the server performs a *k*NN search to identify a *Nearest Neighbor Candidate* set (NNC set) and a *distance limit*. This information is sent to the query object, which subsequently needs to repeatedly estimate distances between the clients in the NNC set and the query object to maintain the query result. When the number of clients in the NNC set with a distance to the query object greater than the *distance limit* exceeds a predefined certain threshold, the query object needs to contact the server to refresh the NNC set. A drawback of this approach is the potentially low accuracy in the *k*NN approximation because the criterion employed to refresh the NNC set does not consider the clients outside the NNC set, which could become the query's *k*NNs even when the criterion is still satisfied.

In summary, although there have been a tremendous amount of work in *k*NN query processing, there is no existing P2P solution for such queries in a road network environment, which allows all objects to participate in query processing in order to reduce both server computation and communication costs.

3 Preliminaries

In this section, we first define the underlying spatial network, and then give definitions for moving objects, *k*NN queries and monitoring regions.

Definition 1. (Network) A network is modeled as an undirected graph $G = (N, E)$, where N is a set of nodes, and E is a set of edges. An edge is expressed as $<n_i, n_j>$, where n_i and n_j represent the *start* node and the *end* node. To avoid ambiguity, we use a numbering system such that i is always less than j. The distance between two nodes n_i and n_j is denoted by $d(n_i, n_j)$, which is the shortest network distance from n_i to n_j.

Please note that for simplicity, a road network is modeled as an undirected graph where edges are considered to be bidirectional, but our techniques can be easily extended to networks with unidirectional edges. Also, in this paper, road segment and edge are used interchangeably whenever there is no confusion.

Definition 2. (Edge Distance) Based on the types of nodes connecting two edges together, we classify the distance between two edges into the following four types: *SS*, *SE*, *ES*, and *EE*. We call the resultant distance associated with a specific type as *Edge Distance*. For example, the distance type is *SS* if both nodes are *start* (*S*) nodes; the distance type is *SE* if one node is *start* (*S*) node while another is *end* (*E*) node. Formally, given $e_i = <n_{is}, n_{ie}>$ and $e_j = <n_{js}, n_{je}>$, $d_{xy}(e_i, e_j) = d(n_{ix}, n_{jy})$, where $x, y \in \{S, E\}$. To make the definition complete, we add an extra distance type called *SAME* (*SM*) to cover the case when the two edges are identical. Formally, if $i = j$, $d_{SM}(e_i, e_j) = 0$, otherwise, $d_{SM}(e_i, e_j) = \infty$. As a result, the shortest distance between any two edges e_i and e_j can be expressed as: $d(e_i, e_j) = min_{type \in \{SM, SS, SE, ES, EE\}}\{ d_{type}(e_i, e_j)\}$.

Definition 3. (Moving Object) A moving object is represented by a moving point in the road network. At any one time, an object *o* can be described as <*e, pos, direction, speed, reportTime* >, where *e* is the edge that *o* is moving on and *pos* is the distance from *o* to the *start* node of *e*. The value of *direction* is set to 1 if *o* is moving from the *start* node of *e* to the *end* node of *e*; otherwise, it is set to -1. *reportTime* records the time when the *pos* is reported.

Distance between any two objects o_i and o_j, denoted as $d(o_i, o_j)$, is the shortest network distance from o_i to o_j. It can be calculated as below.

Property 1. Assume the positions of objects o_i and o_j are denoted as $<e_i, pos_i>$ and $<e_j, pos_j>$, where $e_i = <n_{is}, n_{ie}>$, $e_j = <n_{js}, n_{je}>$, and the lengths of e_i and e_j are $e_i.length$ and $e_j.length$, respectively. The distance between o_i and o_j can be calculated as the minimum of the following five items:

$$d(o_i, o_j) = min\{d_{SM}(e_i, e_j) + |pos_i - pos_j|,\ d_{SS}(e_i, e_j) + pos_i + pos_j,\ d_{SE}(e_i, e_j) + pos_i + e_j.length - pos_j,\ d_{ES}(e_i, e_j) + e_i.length - pos_i + pos_j,\ d_{EE}(e_i, e_j) + e_i.length - pos_i + e_j.length - pos_j\}$$

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

Definition 4. (k-Nearest-Neighbor Query) A *k*NN query *q* can be denoted as <*o, k*>, where *o* is the object issuing the query (or the focus of the issued query), and *k* is the number of nearest neighbors interested in. Denote the set of all other moving objects (i.e. excluding *o*) as *O*, a *k*NN query *q* returns a subset $O' \subseteq O$ of *k* objects, such that for any object o_i in *O'* and any object o_j in $(O - O')$, $d(o_i, o) \leq d(o_j, o)$.

For a given *k*NN query <*o, k*>, we call the object *o* as the *query object*, all objects in the set (*O – O'*) as the *data objects*. Among all objects in the query results *O'*, we name the object that has the largest distance to *o* as the *kNN object*, and all other objects in the set *O'* as the *(k-i)NN object*s, with *i* = 1, …, *k*-1.

Definition 5. (Range of *k*NN Query) Given a *k*NN query *q* = <*o, k*>, with object *o* moving on edge e_o . Assume the *kNN object* for this query is object o_{NN} , which is moving on edge e_{NN}, then the *range* of the *k*NN query is defined as $e_o.length$ if e_o and e_{NN} are identical, otherwise, the *range* is defined as $e_o.length + e_{NN}.length + d(e_o, e_{NN})$. Please note that the *range* is the allowed maximum distance between the *query object* and the *kNN object* given that both objects move on their own edges.

As shown in the following Definition 6, this *range* concept is utilized to prune out objects that certainly can not become query result.

Definition 6. (Monitoring Region) A monitoring region of a *k*NN query is a set of edges that can be reached by the query's *range* while the *query object* and the query's *kNN object* both move within their own current edges. Formally, for a query *q* = <*o, k*> where *o* moves on edge *e*, if the query's range is *q.range*, then its monitoring region $r = \{e_i \mid e_i \in E, d(e, e_i) < q.range\}$. If an edge is included in a query's monitoring region, we say that this edge *intersects* with the query's monitoring region.

The monitoring region can be computed with a depth-first search by expanding from the *start* and the *end* node of edge *e*. The detailed algorithm is omitted. The interested reader is referred to [9]. The output of the algorithm, denoted by *mrOutput*, has the following format: $mrOutput = \{<e_i, type, distance> \mid e_i \in E, type \in \{SM, SS, SE, ES, EE\}, distance = d_{type}(e_i, e) < q.range\}$. For an object moving on edge $e_{i'}$ in that monitoring region, it stores locally a subset of the above *mrOutput* as $\{<e_i, type, distance> \mid <e_i, type, distance> \in mrOutput, e_i = e_{i'}\}$, to facilitate computing its distance to the *query object*. As a result, moving objects do not need to store the whole road network and perform the computation-intensive shortest-path algorithm. This is considered as one nice feature of our proposed technique.

To illustrate the above definitions, we give an example below. A partial road network is drawn in Figure 1, where nodes are denoted as n_1, n_2, etc. Each edge's length is indicated by the number close to that edge. Notations like n_1n_2, n_1n_3, are used to represent edges. Assume that there is one object *A* (represented by a star) moving on edge n_1n_4, and we are interested in its 2-NNs, which have been determined to be *B* and *C* (represented by triangles). Based on Definition 4, *A* is the *query object*, *B* is the *(k-i)NN object*, *C* is the *kNN object*, and all other objects (represented by circles) are *data objects*. Since *C* is moving on edge n_2n_3, and the shortest distance between edge n_1n_4 and n_2n_3 is 1 (through edge n_3n_4), based on Definition 5, the range of this query is computed as the sum of the lengths of edge n_1n_4 and edge n_2n_3, then added by 1, which gives (3 + 2 + 1) = 6. The monitoring region is then computed by expanding from both nodes (n_1 and n_4) of edge n_1n_4. The results are shown in the figure by the thick edges. All objects moving in the monitoring region will monitor this query.

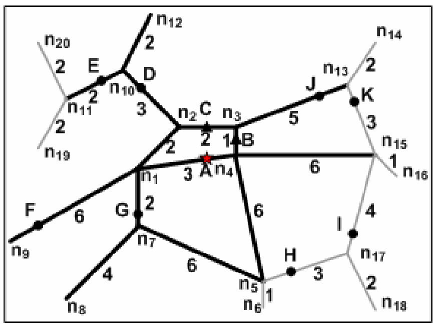


Fig. 1. Example of Monitoring Region

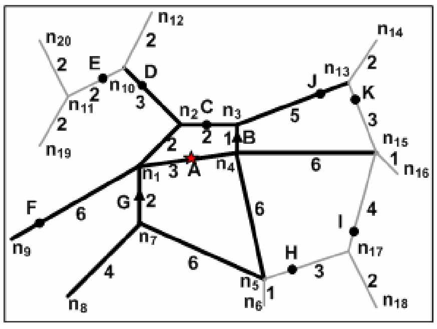


Fig. 2. Example of Message Processing

To deal with long road segments, such as highways, we set a maximum for the allowed segment length. Any segment that is longer than this maximum will be divided into multiple shorter pieces. We add a virtual node at each position where the original segment is divided, and the resultant shorter segments become virtual edges. In our system, we do not differentiate virtual node (edge) from real node (edge).

4 Proposed Solution

4.1 Assumptions and System Overview

We have the following two assumptions: (1) every moving object is equipped with some positioning devices. (2) every moving object has some computing power for data processing.

The proposed system adopts a server-client architecture. On the server side, all information about *query objects, kNN objects, (k-i)NN objects,* and queries are stored. The server determines the monitoring region for each query and sends the query to objects moving in that monitoring region. The moving objects save the received information in their local storage space. Periodically, based on the saved information, a moving object needs to calculate its distance to the *query object*, and compares that distance with the distance from the *kNN object* to the *query object*. For a *data object*, if it moves closer to the *query object* than the *kNN object*, it sends a message to the server to trigger an update. Similarly, for a *(k-i)NN object*, if it moves further away from the *query object* than the *kNN object*, it also notifies the server.

4.2 Server Data Structure

A number of excellent disk-based storage structures have been proposed for road networks [10, 18]. Any of these techniques can be easily adapted for our network database to achieve good access locality and therefore low I/O cost.

There are mainly three tables used: (1) a *query-object-table* to store *query objects* in the form of <*oid, eid, pos, direction, speed, reportTime*>, (2) a *query-table* to store monitoring queries in the form of <*qid, oid, k, kNN object, (k-i)NN objects, mrOutput*>, where the *mrOutput* is the output from the algorithm for monitoring region calculation, as mentioned in Section 3, and (3) a *segment-query-table*, where for each edge, the *qid*s of all queries whose monitoring regions *intersect* with that edge are stored. An entry in this table has the form of <*eid,* {*qid*}>. To facilitate the initialization step (to be discussed in Section 4.4), we also keep track of how many objects currently moving on each edge.

4.3 Moving Object Data Structure

A moving object stores all queries whose monitoring regions intersect with the edge where it is moving. We use a table for that need. For a moving object moving on edge *e*, Each entry in the table has the following format: <*qid*, *oid*, *eLength*, {<*e*, *type*, *dist*>}, *nn_oid*, *nn_eLength*, {<*e'*, *nn_type*, *nn_dist*>}>, where *qid* is the query id, *oid* is the corresponding *query object*'s id, *eLength* is the length of the edge that the *query object* is on, and {<*e, type, dist*>} stores a subset of *mrOutput* (the attribute inside the *query-table* on the server), where each tuple specifies the edge distance type and the actual edge distance from the moving object's segment to the *query object*'s segment. With *eLength* and the set {<*e, type, dist*>}, the moving object can calculate its distance to the *query object*. Similarly, *nn_oid* denotes the *kNN object*'s object id, *nn_eLength* is the length of the edge where the *kNN object* is on, and {<*e'*, *nn_type*, *nn_dist*>} stores a set of tuples which help to calculate the distance from the *kNN* *object* to the *query object*. Please note that *e'* is the edge where the *kNN object* is moving on. In order to estimate the locations of the *query object* and the *kNN object* at different time units other than at the saved *reportTime*, we also store the information about the *query object* and the corresponding *kNN object* on moving objects.

4.4 Initialization

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

(1) Since the server knows how many objects are moving on each edge, by comparing with the requested number *k*, the server can decide the set of edges to send a probe message. The probe message has the format of <*qid*, *oid*, *pos*, *eLength*, {<*e*, *type*, *dist*>}>, where *pos* is the position of the *query object* on its edge, and other parameters have the same meanings as those discussed in Section 4.3.

(2) After the probe message is received by all moving objects moving on the identified edges, based on Property 1, moving objects can calculate their distances to the *query object* and send the distances back to the server.

(3) The server compares all the returned distances and picks the *k* smallest ones. The moving objects with the *k* smallest distances are the initial *k* nearest neighbors. Among the *k* identified objects, the one with the largest distance is the *kNN object*.

(4) With the *kNN object* known, the server calculates the query's range using Definition 5, computes the query's monitoring region with Definition 6, and sends a message, which contains the information of the *query object* and the *kNN object*, to all objects in the monitoring region.

4.5 Message Processing

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

4.5.1 Switch Segment Message

Every moving object needs to monitor its own location on the segment it is moving on. If its position on that segment is less than zero or greater than the segment's length, it knows that it has moved to a new segment. At this time, the moving object reports to the server and requests for the new segment's length and a new set of queries. For each query, the server sends the *query object* and the *kNN object*'s information with the relevant edge distances, and the lengths of the edges where the *query object* and the *kNN object* are moving on, respectively. With the received information, later on, the moving object can estimate the position of the *query object* and the *kNN object*. Using saved edge distances, the moving object can calculate its distance to the *query object* and the distance from the *kNN object* to the *query object*.

However, if the moving object itself is also a *query object* (or a *kNN object*) for some query, the monitoring region for that query needs to be updated. The server performs the following three tasks: (1). update the *query-object-table* (or the *query-table*) with the object's new location. (2). compute a new monitoring region for the query and update the *query-table*. (3). send out messages to notify moving objects in the old monitoring region to stop monitoring the query, and notify moving objects in the new monitoring region to add this query for monitoring. If the moving object is a *(k-i)NN object* for some query, although there is no need for monitoring region re-computation, the server needs to update the *query-table* accordingly.

4.5.2 Enter Query Message

For a *data object*, it periodically checks its distance to the monitored *query object*, and compares with the distance from the *kNN object* to the *query object*. If it is getting closer to the *query object* than the *kNN object* is, a message is sent to the server to indicate that it is currently part of the query result.

The server first estimates the current positions of the saved *kNN object* and *(k-i)NN object*s to decide which object should be replaced by the new-coming object. Then the server updates the *query-object-table*. If the replaced object is the *kNN object*, the server also calculates a new monitoring region based on the new *kNN object*, and notifies all affected objects.

4.5.3 Exit Query Message

For a *(k-i)NN object*, since it could move further away from the *query object* and become the *kNN object*, it needs to periodically monitor its distance to the *query object* and compare with that of the *kNN object*. Once the distance is larger than the distance from the *kNN object* to the *query object*, the *(k-i)NN object* needs to report to the server.

After the server receives this type of message, it replaces the current *kNN object* with the one sending out the message, re-calculates the monitoring region, and notifies all affected objects.

4.5.4 Other Messages

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

4.5.5 Optimization

For the sake of clarity, we have assumed that the server can receive only one message per time unit. In reality, the server bandwidth is more plentiful and many messages should be able to arrive at the server per time unit. To reduce server computation and communication cost, for all the messages received during a given time unit requiring monitoring region re-computation (such as "Exit Query Message" and "Enter Query Message"), immediately after the message is received, the server only updates the query result to keep the result accurate. And the server waits until the end of that time unit to re-compute the monitoring region and sends out message to notify moving objects to update their monitoring queries.

4.6 An Example

In this section, we use the same example as the one used in Section 3 to show how a *data object* keeps monitoring its distance to the *query object* and the distance from the *kNN object* to the *query object*.

For example, at time *t*, as shown in Fig. 1, the *query object A* is at position 2.5 on edge n_1n_4, the *kNN object C* is at position 1 on edge n_2n_3, and a *data object G* is at position 1.5 on edge n_1n_7. Since *G* is inside the query's monitoring region, it has *A* and *C*'s information saved locally. Besides, it stores the edge distance {<n_1n_7, *SS*, 0>} and the length of edge n_1n_4, to determine the distance from itself to *A*. To calculate the distance from *C* to *A*, it also has the edge distances {<n_2n_3, *SS*, 2>, <n_2n_3, *SE*, 3>, <n_2n_3, *ES*, 4>, <n_2n_3, *EE*, 1>} and the length of edge n_2n_3 saved.

At time (*t*+1), as shown in Fig. 2, *data object G* moves to position 1 on edge n_1n_7. It estimates the new position of *A* on edge n_1n_4 using Property 2 and gets 1.5. Similarly, it estimates the new position of *C* on edge n_2n_3 as 1.1. Then with Property 1, it computes its distance to *A* as 2.5 (calculated as: 1 + 1.5 + 0), while the distance from *C* to *A* is 3.4 (through the edge distance <n_2n_3, *EE*, 1>, calculated as: (2 - 1.1) + 1.5 + 1). Since its distance to the *query object* is less than the distance from the *kNN object* to the query object, it sends an enter query message to the server. The server replaces *C* with *G* as the new *kNN object*, determines the new query range, and re-computes the monitoring region. In this example, the new query range is 5 (sum of the lengths of edge n_1n_4 and edge n_1n_7), and the new monitoring region is drawn as thick edges in Figure 2. As we can see, data object *E* on edge $n_{10}n_{11}$ is no longer in the monitoring region.

5 Performance Study

We implemented a simulator to measure the performance of our proposed technique. For a system designed to process monitoring queries, the server could easily become a bottleneck. Whether or not a system can reduce server computation and communication cost is very critical, as a result, we choose the following performance metrics.

- **Server workload.** This cost is measured as the total number of edges accessed in order to answer queries. This is a good measure because server workload consists of I/O time and CPU time, while I/O time is more dominant.
- **Communication cost.** We measure this cost by counting the messages sent out from both the server and the client to reflect the bandwidth consumption.

For server workload, we compare our technique with one popular centralized solution: *query index* [8], which was originally designed for an open space environment. To make the comparison fair, the *query index* scheme is adapted for a road network environment. In the adapted scheme, queries are indexed by a segment-query table (similar to the table used in our technique), where for each segment, all queries whose *query object* can reach that segment within the distance from the *query object* to the *kNN object* are saved. Every time when a moving object sends its updated location to the server, based on the segment where the moving object is on, the server retrieves all queries for that segment from the segment-query table. Then the server computes the distances from the moving object to *query object*s to determine if the moving object belongs to any query result. Also, every time when a *query object*'s location is updated or its *kNN object*'s location is updated, the server updates that segment-query table.

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

5.1 Simulation Setup

Our simulation is based on a terrain of 50 × 50 square miles. We generate a synthetic road network by first placing nodes randomly on the terrain, and then connect nodes together randomly to form edges. There are 2000 nodes and 4000 edges in our setup, with the longest edge as 3 miles. Moving objects are placed randomly on edges with initial speeds and directions. Among all the moving objects, some are specified as *query object*s with a pre-defined number (*k*) of interested nearest neighbors. The speeds are in the range of [0.1, 1] mile/min, following a Zipf distribution with a deviation of 0.7. When an object moves close to a road intersection, it moves to a randomly picked segment. At each time unit, there are a certain percentage of objects changing their speeds. The threshold for changing speed is set as 0.1 mile/min. The time step parameter for the simulation is one minute. We run simulation for 10 times and compute the average as the final output. Each simulation lasts for 200 time units. The simulation was run on a Pentium 4 2.6GHz desktop pc with 2GB memory.

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

Table 1. Simulation Parameters

Parameter Name	**Value Range**	**Default Value**
Number of Moving Objects	[50000, 100000]	100000
Number of Queries	[10, 1000]	200
Number of Nearest Neighbors (*k*)	[1, 20]	5
Percentage of Objects Changing Speed per Time Unit	[2, 50]	10

5.2 Simulation Results

Figure 3 shows the impact of number of queries on server workload and communication cost. Please note that in Fig 3.a, the vertical axis is in logarithmic scale. The plot shows that both the proposed technique and the *Query Index* method incur more server workload with the increases in the number of concurrent queries. A comparison indicates that the proposed approach is about 50 times better than the *Query Index* method. This huge savings can be attributed to the computations carried out on moving objects, which greatly reduce server workload.

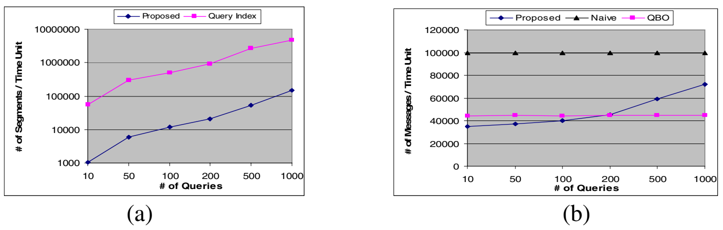
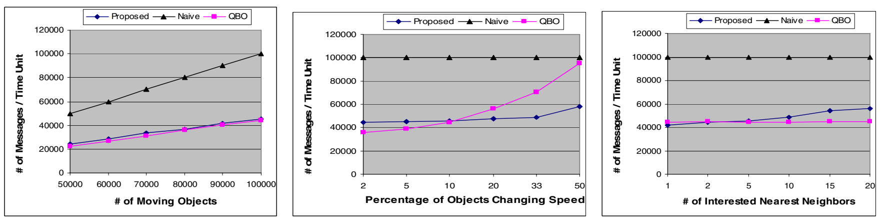


Fig. 3. Effect of number of queries on (a) server workload and (b) communication cost

In Fig 3.b, we compare the communication cost of the proposed technique with those of a Naïve approach and the QBO method. We observe that the curve of the naïve method is flat and it displays the highest communication costs. This is expected since every object updates its location at every time step. The curve of the QBO method is also flat because the communication cost is primarily introduced by objects when they move to new segments or change their speeds; and the occurrences of such activities are independent of the number of concurrent queries. The communication cost of the proposed technique increases as the number of queries increases. This can be explained as follows. Since the P2P strategy needs to update the query results maintained on the remote server, the objects have more query updates to perform with the increases in the number of concurrent queries resulting in a higher communication cost. Nevertheless, the proposed technique performs very well (i.e., comparable to the QBO) for numbers of queries as high as 200. Its performance worsens when the numbers of concurrent queries is greater than 200. Under this circumstance, we note that distributed servers can be used to accommodate the increase in the communication costs. In such an environment, our P2P technique would require a smaller number of distributed servers since it is able to reduce server workload and the demand on server bandwidth.

In Figure 4, we vary the other three parameters to study their effects on communication cost. In Fig 4.a, the number of moving objects is varied from 50000 to 100000. As we can see, for all the three studied methods, the number of messages increases as the number of moving objects increases. Fig 4.b studies the effect of increasing the percentage of objects changing speed per time unit from 2% to 50%. The result shows that for the naïve method, the curve is a flat line as in Fig 3.b. Both our technique and the QBO technique incur more communication cost when there are more objects changing speeds at every time step. Compared to the QBO technique,



(a) (b) (c)

Fig. 4. Effect of (a) number of moving objects, (b) percentage of objects changing speed per time unit, (c) number of interested nearest neighbors, on communication cost

our technique has a much less steeper curve because among all objects that change speeds, only *query object*s, *kNN object*s, and *(k-i)NN object*s, which combined account for a small fraction of the total number of objects, need to contact the server, however, in the QBO technique, all objects changing speeds have to report to the server. We also try to vary the number of interested nearest neighbors (*k*) and the result is shown in Fig 4.c. From the plot, we observe that both the naïve method and the QBO technique are not affected by the number of requested nearest neighbors. For our technique, more messages are needed if there are more nearest neighbors to be found. This is expected since a bigger *k* will make more objects into *(k-i)NN object*s, and quite probably, larger monitoring regions are demanded. Consequently, higher communication cost is necessary.

6 Conclusions

In this paper, we introduced a P2P technique for continuous *k*NN queries in a network environment. To the best of our knowledge, this is the first P2P solution that fully leverages the computation power of all peers to address the *k*NN problem. This scheme utilizes mobile computing power to reduce server workload and the number of location updates necessary. We presented the detailed design and gave simulation results to show the performance advantages of the proposed technique. When compared to an adapted *Query Index* method, our approach incurs about 50 times less server load. In terms of communication cost, the proposed technique performs comparable to a *Query Blind Optimal* scheme when there are as many as 200 concurrent queries. As the number of concurrent queries increases, the moving objects need to communicate more frequently to update the query results maintained on the server.

References

1. Cai, Y., Hua, K.A., Cao, G.: Processing Range- Monitoring Queries on Heterogeneous Mobile Objects. In: MDM (2004)
2. Mouratidis, K., Hadjieleftheriou, M., Papadias, D.: Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In: SIGMOD (2005)

3. Gedik, B., Liu, L.: MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, Springer, Heidelberg (2004)
4. Hu, H., Lee, D.L., Xu, J.: Fast Nearest Neighbor Search on Road Networks. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Boehm, K., Kemper, A., Grust, T., Boehm, C. (eds.) EDBT 2006. LNCS, vol. 3896, Springer, Heidelberg (2006)
5. Kolahdouzan, M.R., Shahabi, C.: Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In: VLDB, pp. 840–851 (2004)
6. Hu, H., Lee, D.L., Lee, V.C.S.: Distance Indexing on Road Networks. In: VLDB (2006)
7. Xiong, X., Mokbel, M., Aref, W.: SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In: SIGMOD (2004)
8. Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans. on Computers 51(10) (2002)
9. Liu, F., Do, T.T., Hua, K.A.: Dynamic Range Query in Spatial Network Environments. In: Bressan, S., Küng, J., Wagner, R. (eds.) DEXA 2006. LNCS, vol. 4080, Springer, Heidelberg (2006)
10. Shekhar, S., Liu, D.R.: CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. IEEE TKDE 9(1) (1997)
11. Jensen, C.S., Kolar, J., Pedersen, T.B., Timko, I.: Nearest Neighbor Queries in Road Networks. In: Proc. ACMGIS, pp. 1–8 (2003)
12. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous Nearest Neighbor Monitoring in Road Networks. In: VLDB, pp. 43–54 (2006)
13. Cho, H., Chung, C.: An Efficient and Scalable Approach to CNN Queries in a Road Network. In VLDB, pp. 865–876 (2005)
14. Wang, H., Zimmermann, R., Ku, W.S.: Distributed Continuous Range Query Processing on Moving Objects. In: Bressan, S., Küng, J., Wagner, R. (eds.) DEXA 2006. LNCS, vol. 4080, pp. 655–665. Springer, Heidelberg (2006)
15. Hu, H., Xu, J., Lee, D.L.: A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In: SIGMOD (2005)
16. Yu, X., Pu, K.Q., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: IEEE ICDE, IEEE Computer Society Press, Los Alamitos (2005)
17. Xiong, X., Mokbel, M., Aref, W.: SEA-CNN:Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In: IEEE ICDE, IEEE Computer Society Press, Los Alamitos (2005)
18. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query Processing in Spatial Network Databases. In: VLDB (2003)
19. Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y.: A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors. IEEE TKDE 17(11), 1451–1464 (2005)
20. Wu, W., Guo, W., Tan, K.L.: Distributed Processing of Moving K-Nearest-Neighbor Query on Moving Objects. In: IEEE ICDE, IEEE Computer Society Press, Los Alamitos (2007)