Eleventh International Conference on Mobile Data Management

A Generic Platform for Efficient Processing of Spatial Monitoring Queries in Mobile Peer-to-Peer Networks

Patricio Galdames, Kihwan Kim and Ying Cai
Department of Computer Science
Iowa State University
Ames, IA 50011
Email:{patricio, khkim, yingcai}@cs.iastate.edu

***Abstract*—A *spatial monitoring query* (SMQ) retrieves the set of mobile nodes that satisfy some spatial constraints, and provides real-time updates whenever this set of nodes changes. Efficient processing of such queries is essential to moving objects database management. Existing techniques rely on one or more central servers for query management and assume each mobile node can communicate with some server directly. These limitations prevent them from being used in application scenarios where no such server exists. This paper assumes a mobile peer-to-peer system where mobile nodes are the only computing devices, and investigates the challenges of allowing mobile nodes to collaborate in query processing. We present a cost-effective technique to process a primitive type of SMQs and then show that other types of queries can be converted into the primitive type of queries. As such, different types of queries can now be supported within a common platform and without relying on any stationary server. We also evaluate, through both mathematical analysis and simulation, the performance of the proposed platform in terms of mobile communication costs incurred in query processing.**

***Keywords*-Spatial Monitoring Queries; Mobile Peer-to-Peer Networks; Moving Objects Databases;**

I. Introduction

Given a set of nodes moving on a 2-dimensional space, a *spatial monitoring query* (SMQ) allows users to monitor those that satisfy some spatial constraints. The four most important types of SMQs are:

- *Stationary Range Monitoring Query* (S-RMQ): An S-RMQ retrieves the set of mobile nodes that are inside a user-defined geographic region, and provides real-time updates whenever a node crosses over the border of the region. As nodes move, the query results may keep changing.
- *Mobile Range Monitoring Query* (M-RMQ): An M-RMQ also allows one to monitor the mobile population inside a user-defined region, but the query region is associated with a mobile node, referred to as a *focal node*. As the node moves, the query region changes.
- *Stationary KNN Monitoring Query* (S-KNNMQ): An S-KNNMQ retrieves the set of K mobile nodes that are nearest to a user-defined query point, and provides real-time updates whenever this set of nodes changes.
- *Mobile KNN Monitoring Query* (M-KNNMQ): An M-KNNMQ is the same as an S-KNNMQ, except that the query point is associated with some mobile node. As the focal node moves, the query point changes.

Unlike regular spatial queries, an SMQ is a continuous query and may stay active for a certain time of period. As basic operators in moving objects database management systems, efficient processing of the above four types of SMQs has been investigated intensively, and separately, in the context of centralized environment, wherein mobile nodes are assumed to be able to communicate with some central server directly. In general, the proposed techniques rely on the server for query management, and can be classified into two categories. The techniques in the first category (e.g., S/M-RMQ [1], [2], S/M-KNNMQ [3], [4], [5]) let mobile nodes report the server their velocity information. At the server side, the trajectories of mobile nodes are indexed using some spatial data structure to minimize disk I/O and CPU costs in query evaluation. In these schemes, the server handles all query evaluation and mobile nodes just need to report their location information. In contrast, the techniques in the second category (e.g., S/M-RMQ [6], [7], [8], [9], S/M-KNNMQ [10], [11], [12]) let mobile nodes participate in query evaluation. Each mobile node needs to monitor their movement with respect to some boundaries, and when crossing over a boundary, reports the server. Upon receiving a location update, the server checks if any query result needs to be updated.

Our research considers the problems of handling SMQs in a fully distributed mobile networking environment. Specifically, given a set of mobile nodes that forms a mobile ad hoc network (MANET), we want to perform SMQs over these nodes. Our work is motivated by the potential uses of SMQs in the application scenarios (e.g., battlefields, emergency management) where fixed communication infrastructures may not exist. In the absence of any central/stationary server, efficient query processing is challenging. A simple solution is to let mobile nodes broadcast their every movement to the entire network. Whenever a query creator receives a location update, it computes the query result by itself. This strategy allows one to provide real-time and accurate query results, but is not scalable. Excessive location updates will quickly exhaust mobile nodes' battery, but also suspend the whole network.

978-0-7695-4048-1/10 $26.00 © 2010 IEEE
DOI 10.1109/MDM.2010.11

In this paper, we present a novel technique, which we will refer to as *Peer-to-Peer Monitoring Query Management* (P2P-MQM), for real-time and cost-effective processing of SMQs. Our technique supports S-RMQs and the rest three types of SMQs by converting them into S-RMQs. Thus, all queries can be managed with one common platform. To minimize communication costs incurred in query management, our approach stores queries among mobile nodes and does so dynamically to make each node aware of its nearby queries. More specifically, we partition the network domain into a number of disjointed grid cells and let each node to carry the queries that overlap with the cell where it resides. When a node moves into a new cell, it can retrieve the cell's overlapping queries from any other nodes in the cell. In the case that the cell does not have any other nodes, our technique ensures its overlapping queries can be retrieved from a node in some neighboring grid. We evaluate the proposed technique through both theoretical modeling and simulation. Our extensive performance study shows that, when the network is fully connected, our technique can provide real-time and accurate query results with a reasonable amount of communication cost.

The remainder of this paper is organized as follows. We present our technique for efficient management of S-RMQs in Section II. In Section III, we discuss how the other types of SMQs can be converted into S-RMQs and managed together within one platform. In Section IV, we model the performance of the proposed technique with a detailed mathematical analysis and verify the accuracy of our model using simulation. More performance evaluation based on simulation is given in Section V. We discuss more related work in Section VI and conclude this paper in Section VII.

II. Handling Stationary Range Monitoring Queries

We consider a mobile ad hoc network formed by a set of location-aware mobile nodes. Since many communication protocols (e.g., LAR [13], DREAM [14], GPSR [15]) have been developed for such networks, we simply assume these nodes can communicate with each other through packet relaying and will not concern how this is actually implemented. In this section, we focus on efficient processing of S-RMQs. The query region is assumed to be either rectangular or circular. In the absence of any central server, a major problem is how to store queries among mobile nodes and retrieve them for execution whenever necessary to ensure real-time and accurate query results. As mentioned early, replicating each query among all mobile nodes is the simplest solution, but may incur a large amount of communication overhead. Our research has developed a cost-effective solution that allows mobile nodes to dynamically cache and evaluate their nearby queries.

A. Basic Idea

We partition the network domain into a set of disjointed grid cells (or subdomains), as showed in Figure 1. The size of each cell is set to be $\frac{r}{\sqrt{2}}$ x $\frac{r}{\sqrt{2}}$, where r is node transmission

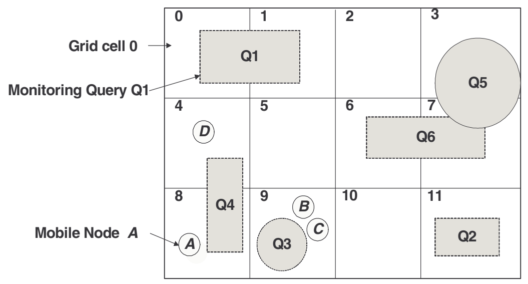


Fig. 1. Safe-Boundary

radius. Thus, when a node broadcasts a message, it covers the entire cell where the node resides. The network partitioning is made known to all mobile nodes, and each mobile node caches the queries that are relevant to its home cell. We say a cell is a node's home cell if the node is currently inside the cell, and a query is relevant to a cell if the query overlaps with the cell. A query may be relevant to more than one cell since its range may span over several cells. For instances, query Q_3 is relevant only to cell 9 while Q_5 is relevant to both cells 3 and 7. With this network partitioning in place, creating/removing a query can be done by first determining its relevant cells and then geocasting [16], [17] the operation to all nodes in these cells. When a node moves inside its home cell, it monitors its movement against the cell's relevant queries. When it detects that it has crossed over any query boundary, it notifies the query's creator accordingly. Since a node knows exactly when such a report is needed, this approach ensures real-time query results. In this scheme, a cell's relevant queries are replicated among all nodes in the cell. Thus, when a node moves into a new cell, it can retrieve the new cell's relevant queries from any other node in the cell.

A technical problem here is how to handle the situation when a cell does not have any node. When this happens, the cell's relevant queries may be lost, and a node moving into this cell may not find these queries. To address this problem, we use the following approach. When a node moves out of its home cell, it checks if it is the last node in the cell. If there are some other nodes in the cell, the node can simply delete the queries that are relevant to this cell. Otherwise, it keeps these queries, and this node becomes the cell's *retaining node*. For example, if node A in Figure 1 is moving out of cell 8, it will become the retaining node to this cell. Note that a cell can have at most one retaining node; and this happens only when there is no node inside the cell. When a node becomes a cell's retaining node, it keeps the cell's relevant queries until they are retrieved when some node moves into the cell.

The concept of retaining node allows a node to retrieve a cell's relevant queries from its retaining node, if the cell contains no other node. However, the cost of locating a cell's retaining node can be high if the node has moved far away from the cell. To reduce this cost, we use the following approach to keep a cell's retaining node as close as possible to the cell. Suppose node H is the retaining node to cell i, and

H is moving from its current cell j into a new cell k. If the distance from k to i is farther than that from j to i, then H can send the queries that are relevant to cell i to another node, if any, in cell j. This node then becomes the new retaining node to cell i. As an example, suppose node A in Figure 1 moves from cell 8 to cell 9, and then to cell 10. If A is the retaining node to cell 8, it may unload the cell's relevant queries to either B or C.

When the network is fully connected, the above technique ensures that a node moving into a new cell can always retrieve the cell's relevant queries, either from another node currently in the cell or from the cell's retaining node (if the cell contains no other node), which is likely to be nearby. This scheme also guarantees real-time query processing. Since each node is made aware of the queries relevant to its home cell, it can monitor its movement against them and notify a query's creator whenever necessary. In reality, the network may be disconnected due to various factors such as node failures. When this happens, installing/removing a query may fail. In particular, a node moving into a new cell may not be able to retrieve the cell's relevant queries. When the network is disconnected, no technique can provide accurate query results. However, we can minimze the impact caused by the disconnection with some minor revisions on our original design. When a node fails to install/remove a query, it can repeat the operation until it is successful. Similarly, when a node becomes a cell's retaining node, it can periodically try to geocast the relevant queries back to the cell. In this way, the queries can be restored as soon as the network path to the cell is reconnected.

We refer to the above technique as *Peer-to-Peer Monitoring Query Management* (P2P-MQM). From a user's perspective, each cell is conceptually a storage device where one can store/remove information (i.e., queries). The problem of saving data on the ground in the context of MANETs has been investigated in [18], [19], [20]. The techniques in [18], [19] are intended for location services and allow a node to save its position information to a predefined area called virtual home region (VHR) for other nodes to retrieve. These approaches, however, do not guarantee storage persistency: a data item stored in a VHR will be lost when there is no node in the VHR. In contrast, our technique allows a cell to keep its data even if it contains no node. The technique investigated in [20] was designed for information sharing among mobile users in cellular networks. This scheme allows users to save information in a number of designated regions called bazaars, but similar to the VHR concept, it requires each bazaar to contain at least one node in order to ensure data availability.

B. Detailed Design

We now give a more formal description of our technique. Our design of a mobile node mainly consists of two components: *MessageListener* and *RegionMonitor*. To facilitate our discussion, we define the following notations for the data structures maintained by a mobile node:

- *myID*: The node's unique identifier;
- *myPos*: The node's current position;
- *myCell*: The node's current home cell;
- *myQueries*: The list of queries that are relevant to *myCell*;
- *myRetains*: The list of cells to which the node is currently a retaining node and their relevant queries;

MessageListener: The mobile node listens to these messages and processes them as follows:

- *SearchNode(cell)*: If *myPos* is inside *cell*, or *cell* is listed in *myRetains*, reply with a *candidate* message including $myID$ and $myPos$.
- *RetrieveQueries(cell)*:
  - Send all queries relevant to *cell* to the requester;
  - If *cell* is listed in *myRetains*, remove the cell and its relevant queries from *myRetains*.
- $SetRetainingNode(cell, queries)$:
  - Add $cell$ and $queries$ to $myRetains$.
- *InstallQuery(Q)*:
  - If query Q overlaps with *myCell*, add Q to *myQueries*;
  - For each cell g listed in $myRetains$, if g overlaps with Q, add Q to the list of g's relevant queries.
- *RemoveQuery(Q)*:
  - If query Q overlaps with *myCell*, remove Q from *myQueries*;
  - For each cell g listed in *myRetains*, remove Q from the list of g's relevant queries.

RegionMonitor: When the mobile node moves, it monitors its movement and does the followings:

1) If it crosses over the boundary of any query listed in *myQueries*, notify the query's creator;
2) If it moves into a new cell, say $newCell$, do the followings:
   - Set $CandidateList = NULL$;
   - Broadcast message $SearchNode(myCell)$ within $myCell$;
   - Collect all *candidate* messages and add them to $CandidateList$;
   - If $CandidateList = NULL$, add $myCell$ and $myQueries$ to $myRetains$ (becoming the cell's retaining node);
   - Otherwise, do the followings:
     - For each cell c (if any) in $myRetains$, check its distance to $myCell$ and to $newCell$;
     - If c is closer to $myCell$ than to $newCell$, then find the node in the $CandidateList$ that is nearest to c, and send a message $SetRetainingNode(g, queries)$ to the node, where $queries$ is the list of queries relevant to c.
   - Set $myCell = newCell$, $CandidateList = NULL$, and $TTL = 1$;
   - While $CandidateList$ is $NULL$, do the followings:
     - Broadcast message $Searchnode(myCell)$ within a scope of TTL hops;

  - Collect all *candidate* messages and add them to $CandidateList$;
  - Increase TTL by 1.
- Among all nodes in $CandidateList$, find the one which is the closest to $myPos$, and send it a message $RetrieveQueries(myCell)$ to retrieve $myCell$'s relevant queries.

III. Handling Other Types of SMQs

We now consider how to support other types of queries, including M-RMQ, S-KNNMQ, and M-KNNMQ. We show that given such a query, we can find a set of *safe boundaries*; unless there is a crossing event, the query result does not change. Thus, the query needs to be re-evaluated only when some node crosses over a boundary. Since each boundary can be considered as a stationary range monitoring query, the technique presented in the previous section can then be used to manage all these types of queries.

A. Mobile Range Monitoring Query

An M-RMQ is associated with a mobile node, referred to as a focal node. As the node moves, the query region changes accordingly. An example use of such queries is for a commander to monitor the soldiers that are within 1 mile of his current position.

Figure 2 shows a focal node N at position (x_n, y_n) and its current query region $R[(x_q, y_q), (x'_q, y'_q)]$. Given a rectangle R, we will denote it as $R[(x, y), (x', y')]$, where points (x, y) and (x', y') are its low-left and upper-right points, respectively. Let $R_i[(x_i, y_i), (x'_i, y'_i)]$ be the minimum bounding rectangle that contains all nodes that are currently inside the query region, and let $R_o[(x_o, y_o), (x'_o, y'_o)]$ be the maximum bounding rectangle that contains the query region and does not have any node that are currently outside the query region. Then we call the following three rectangular boundaries, B_i $[(\frac{x_q+x_i}{2}, \frac{y_q+y_i}{2}), (\frac{x'_q+x'_i}{2}, \frac{y'_q+y'_i}{2})]$, B_o $[(\frac{x_q+x_o}{2}, \frac{y_q+y_o}{2}), (\frac{x'_q+x'_o}{2}, \frac{y'_q+y'_o}{2})]$, and B_f $[(x_n - \min(\frac{x_q-x_o}{2}, \frac{x'_i-x'_q}{2}), y_n - \min(\frac{y_q-y_o}{2}, \frac{y'_q-y'_i}{2})), (x_n + \min(\frac{x'_o-x'_q}{2}, \frac{x_i-x_q}{2}), y_n + \min(\frac{y'_o-y'_q}{2}, \frac{y_i-y_q}{2}))]$, as the M-RMQ's *outer safe boundary*, *inner safe boundary*, and *focal safe boundary*, respectively. Our key observation is that, the query result will remain the same unless the focal node moves out of B_f, some node currently inside R crosses over B_i, or some node currently outside of R crosses over B_o. Thus, the query can be processed by making the focal node to monitor its movement against B_f, and other nodes B_i and B_o. If anyone detects it crosses over a boundary, it notifies the focal node to re-evaluate the query and provide latest query result. Note that the above discussion assumes each query region is a rectangular area. In the case it is a circular area, we can compute the circular safe boundaries in a similar way.

To create a new M-RMQ, a node notifies the selected focal node, which then executes the following M-RMQ(N_p, R_q) procedure, where N_p is the current position of the focal node and R_q is the current query region.

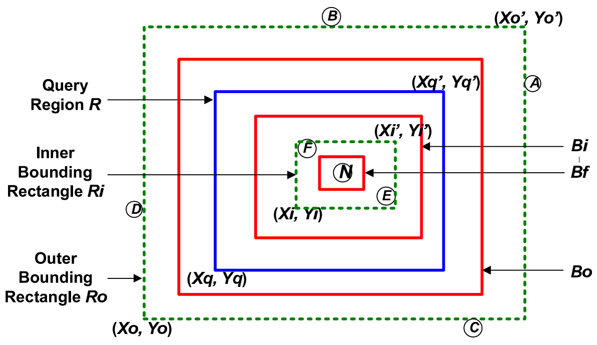


Fig. 2. Safe Boundaries for an M-RMQ

M-RMQ(N_p, R_q)

1) Find all cells that overlap with R_q;
2) Retrieve the position of all nodes inside these overlapping cells;
3) Provide the initial query result (i.e., find all nodes inside R_q);
4) Compute R_o and R_i;
5) Compute B_o, B_i, and B_f, according to N_p, R_o, R_i, and the position information of the nodes inside the overlapping cells, and do the followings:
   - Submit B_o and B_i as two S-RMQs with the additional information that tells a node to perform the followings whenever crossing over B_o or B_i,
     - Remove the two S-RMQs created with B_o and B_i;
     - Notify the focal node to re-evaluate the query by calling **M-RMQ**(N_p, R_q).
   - Monitor movement against B_f and if moving out of it, re-evaluate the query by calling **M-RMQ**(N_p, R_q);

B. Stationary KNN Monitoring Query

When a user issues an S-KNNMQ on point p, the system needs to return the k nodes that are nearest to the query point p, and provide continuous update whenever the query result changes. Let N_k be the kth nearest node to p with a distance of d_k, and N_{k+1} be the $(k+1)$th nearest node to p with a distance of d_{k+1}. Let B be the circular boundary that centers on p with a radius of $\frac{d_k+d_{k+1}}{2}$. These notations are illustrated in Figure 3. We observe that the set of k nodes nearest to p does not change as long as no node moves across B. Thus, B can serve as the query's safe boundary. When a node crosses over the boundary, it computes a new safe boundary by identifying the $k + 1$ nodes nearest to p and informs the query creator of the new set of k nearest nodes.

The above approach converts an S-KNNMQ into a circular S-RMQ. A more formal description for this algorithm is as follows, where p is the query point and k is the number of nearest neighbors to monitor.

S-KNNMQ(p, k)

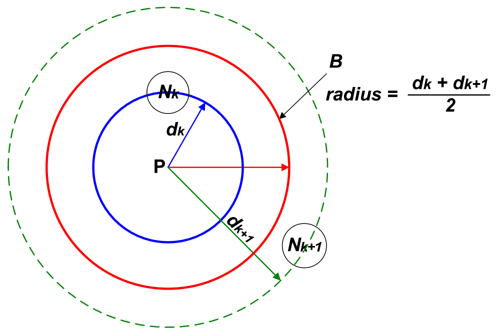


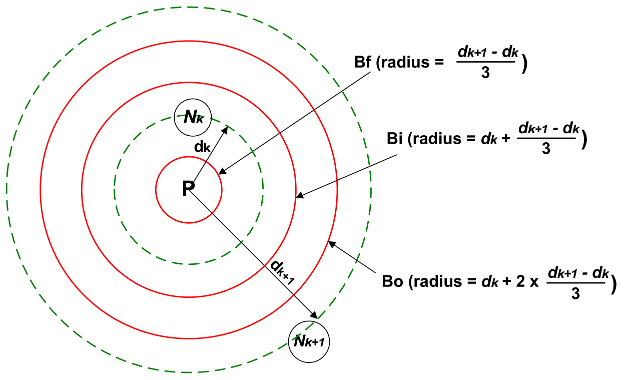
Fig. 3. Safe Boundary for an S-KNNMQ

1) Find the $k+1$ nodes that are nearest to p by search the cell that contains p (and expand the search scope if necessary);
2) Notify the query creator the k nearest nodes;
3) Compute the safe boundary B according to p, the positions of the k and $k+1$ nearest nodes;
4) Submit B as an S-RMQ with the additional information that tells a node, when crossing over B, to re-evaluate the query by calling **S-KNNMQ**.

C. Mobile KNN Monitoring Query

An M-KNNMQ, like an M-RMQ, is associated with a focal node; as the node moves, the query point p changes accordingly. Figure 4 shows an M-KNNMQ having its current query point at p. Again, let N_k be the kth nearest node to p with a distance of d_k, and N_{k+1} be the $(k+1)$th nearest node to p with a distance of d_{k+1}. We define the following three p-centered circumcircles, B_f, B_i, and B_o, the radius of which are $R_f = \frac{d_{k+1}-d_k}{3}$, $R_i = d_k + \frac{d_{k+1}-d_k}{3}$, and $R_o = d_k + 2 \times \frac{d_{k+1}-d_k}{3}$, respectively. These notations are illustrated in Figure 4. Clearly, the set of k nodes nearest to p does not change as long as the focal node does not moves out of B_f, and no node crosses over B_i or B_o. Thus, we just need to make the focal node to monitor its movement against B_f, and other nodes to monitor their movement against B_i and B_o. If a crossing event occurs, the focal node is notified to re-evaluate the query. A more formal description of this algorithm (executed by the focal node) is as follows, where p is the focal node's current position and k the number of nearest nodes to monitor.

```
M-KNNMQ$(p,k)$
1) Find the $k+1$ nodes that are nearest to $p$ by searching
   the cell that contains $p$ (and expand the search scope if
   necessary);
2) Notify the query creator the $k$ nearest nodes;
3) Compute $B_f$, $B_i$, and $B_o$ according to $p$, the positions
   of the $k$ and $k+1$ nearest nodes;
4) Submit $B_i$ and $B_o$ as two S-RMQs with the additional
   information that tells a node to do the followings when
   crossing over $B_o$ or $B_i$,
     • Remove the two S-RMQs created with $B_i$ and $B_o$;
     • Notify the focal node to re-evaluate the query by
       calling M-KNNMQ$(p,k)$.
5) Monitor movement against $B_f$ and if moving out of it,
   re-evaluate the query by calling M-KNNMQ$(p,k)$;
```



Bf (radius = $\frac{d_{k+1} - d_k}{3}$)
Bi (radius = $d_k + \frac{d_{k+1} - d_k}{3}$)
Bo (radius = $d_k + 2 \times \frac{d_{k+1} - d_k}{3}$)
N_k
d_k
P
d_{k+1}
N_{k+1}

Fig. 4. Safe Boundaries for an M-KNNMQ

IV. Analytical Model

In this section, we analyze the communication costs incurred in managing S-RMQs. Our focus is on S-RMQs because other types of monitoring queries are essentially converted into S-RMQs. The cost of managing S-RMQs mainly comes from two events:

- *A node moves into a new cell*: When this happens, the node needs to retrieve the new cell's relevant queries.
- *A query is created or removed*: In this case, the query needs to be broadcasted to the nodes in the cells that overlap with the query boundary.

We assume the network is dense enough that all nodes are connected. Many techniques can be used for regional broadcast (e.g., broadcasting within a fixed region or a number of hops) and they can result in very different communication cost. The cost itself can have different measures (e.g., the number of packets sent and/or received, or the size of total network traffic). To avoid assuming some particular technique and/or specific metric, we simply assume the cost of 1-hop broadcast is c_o and the cost of broadcasting a fixed region is proportional to the area of the region. Thus, the cost of broadcasting within a range of i hops is $C_i = i^2 \cdot c_o$.

A. Cost of Retrieving Relevant Queries

When a node h moves into a cell c, it needs to retrieve the cell's relevant queries. We call this node a requesting node. If there are other nodes inside the cell, any of them can supply the queries. Since the cell is covered entirely by a node's transmission range, the search cost in this case is just one-hop broadcast, i.e., $Cost_{inside} = c_o$. If there is no other node inside the cell, the requesting node needs to find the cell's retaining node, which is done by expanding the search scope sequentially, first within 2 hops, then 3 hops, ..., until it is found [1]. Suppose the retaining node is k-hop away from the requesting node. Then it will take k rounds of search to locate. The total cost can be calculated as $cost(k) = \sum_{i=2}^{k} i^2 \cdot c_o = [\frac{k(k+1)(2k+1)}{6} - 1] \cdot c_o$.

We now analyze where the retaining node is actually located. Suppose the network domain $h \times w$ is partitioned into M cells, each being $\frac{r}{\sqrt{2}}$ x $\frac{r}{\sqrt{2}}$, where r is node transmission radius. Let p_0 be the position of the requesting node, which must be on the boundary of cell c. According to [21], the time duration that a randomly moving unit may stay in a spatial region can be approximated as an exponential distribution and the mean staying time is

$$\bar{t} = \frac{\pi A}{E[v]L}, \tag{1}$$

where A is the area of the region, L is the perimeter of the region, and $E[v]$ is the average speed of the mobile unit. Thus, the average time duration that a node stays inside a square of $q \times q$ is $\bar{t}_q = \frac{\pi q}{4E[v]}$, and the probability density function of the corresponding exponential distribution is $f_{t_s}(t, q) = \frac{1}{\bar{t}_q} e^{-\frac{t}{\bar{t}_q}}$. Let N be the total number of mobile nodes. Assuming these nodes are uniformly distributed and move randomly in the network domain, given a specific close region, the average frequency of the event that some node moves into this region is equal to that of the event that some node moves out of this region. Since the average number of nodes in a cell is $\frac{N}{M}$, the average time interval of two consecutive move-in events on a specific cell is $\bar{t}_e = \frac{\pi r \cdot M}{4\sqrt{2}E[v]N}$, and the probability density function of the corresponding exponential distribution is $f_{t_e}(t_e) = \frac{1}{\bar{t}_e} e^{-\frac{t_e}{\bar{t}_e}}$.

Let t_u be the interval from the time when cell c's retaining node moves out of c to the time when the requesting node moves into c, and let p_r be the position of the retaining node when the requesting node asks for c's relevant queries. Then the number of hops from the requesting node to the retaining node is $H(p_r, p_0) = \lceil \frac{dist(p_0, p_r)}{r} \rceil$, where r is node transmission radius. Let S_i denote the square that centers on cell c and consists of $(2i-1) \times (2i-1)$ cells, and let D_i denote the geographic region that is inside of S_{i+1} but outside of S_i. Then the probability that p_r locates inside D_i is

$$\begin{aligned} p_i(t_u) &= \int_0^{t_u} f_{t_s}(t, (2i-1)d) - f_{t_s}(t, (2i+1)d)\, dt \\ &= e^{-\frac{t_u}{(2i+1)t_g}} - e^{-\frac{t_u}{(2i-1)t_g}} \end{aligned} \tag{2}$$

The average cost of locating the retaining node in D_i is

$$\overline{C}_{D_i} = \frac{1}{4d}\frac{1}{8i^2d^2}\int_B \iint_{D_i} cost(H(p_r, p_0))dBdD_i, \tag{3}$$

where B is the boundary of cell c. Similar to $\overline{C}_n$, $Cost_{outside}$ can be computed using numerical approaches, say, *Monte Carlo* method [22].

[1]The search scope may be expanded in a more aggressive way to balance the search cost and latency. For instance, we can double the hop count in each iteration, i.e., first within 2 hops, then 2^2 hops, and so on so forth. This approach, which is used in some MANET routing protocols like AODV and DSR for route discovery, can reduce the search latency from $O(K^2)$ to $O(K)$, where K is the number of hops from the requesting node to the retaining node. This paper does not model the search latency.

Note that the retaining node can be anywhere outside of cell c, t_u can be from 0 to infinity, and the size of q can range from $\frac{r}{\sqrt{2}}$ to $\max(h, w)$. Let $Q = \lfloor \frac{\max(w,l)}{\frac{r}{\sqrt{2}}} \rfloor$. Then the expected cost of locating this node is

$$\begin{aligned} Cost_{outside} &= \sum_{i=1}^{Q} \int_0^{\infty} f_{t_e}(t_u) \cdot \overline{C}_{D_i} \cdot p_i(t_u)\, dt_u \\ &= \sum_{i=1}^{Q} (\frac{(2i+1)\bar{t}_g}{\bar{t}_e + (2i+1)\bar{t}_g} - \frac{(2i-1)\bar{t}_g}{\bar{t}_e + (2i-1)\bar{t}_g}) \cdot \overline{C}_{D_i} \end{aligned} \tag{4}$$

Suppose the network area is partitioned into M cells. Then given a total of N mobile nodes, the probability of having no node (excluding the requesting node) in a particular cell c is $P_0 = (1 - \frac{1}{M})^{N-1}$. Thus, the total cost for a node to retrieve the queries relevant to its new home cell is

$$\begin{aligned} Cost_{retrieval} &= (1 - P_0) \cdot Cost_{inside} + P_0 \cdot Cost_{outside} \\ &= (1 - P_0) \cdot c_o + P_0 \cdot Cost_{outside} \end{aligned} \tag{5}$$

B. Cost of Installing or Removing Queries

We now analyze the communication cost incurred in creating/removing queries. For simplicity, we assume each query region is a square of $l \times l$. The cell that contains the center position of this query can be seen as a concatenation of a number of rectangles. Let $a = \lceil \frac{l}{\frac{r}{\sqrt{2}}} \rceil$ and $b = \lceil \frac{l}{\frac{r}{\sqrt{2}}} \rceil - \frac{l}{\frac{r}{\sqrt{2}}}$, where $\frac{r}{\sqrt{2}}$ is the cell size. Figure 5 shows these rectangles and their sizes. These rectangles can be classified into three categories: R_{1X} (including R_1 along), R_{2X} (including R_{21}, R_{22}, R_{23}, and R_{24}), and R_{3X} (including R_{31}, R_{32}, R_{33}, and R_{34}). The number of cells that overlaps with the range of the query depends on where the center position of the query is located:

- Case 1: If the position is inside R_1, the query overlaps with $(a+1)^2$ cells;
- Case 2: If the position is inside some R_{2X} rectangle, the query overlaps with a^2 cells;
- Case 3: If the position locates in some R_{3X} rectangle, the number of cells that overlap with the query is $a \cdot (a+1)$.

Suppose the queries are uniformly distributed in the network domain. Then the probability of each case is proportional to the size of its corresponding area. Specifically, the probability of case 1 is b^2; the probability of case 2 is $(1-b)^2$; and the probability of case 3 is $2b(1-b)$. Thus, the average number of cells that overlap with this query is

$$\begin{aligned} n_o &= a^2 \cdot b^2 + (a+1)^2 \cdot (1-b)^2 + a(a+1) \cdot 2b(1-b) \\ &= (a - b + 1)^2 = (\frac{r + \sqrt{2}l}{r})^2 \end{aligned} \tag{6}$$

When a mobile node needs to install or remove a query, it will need to broadcast all mobile nodes located in the overlapping cells. Thus, the cost of installing or removing a query can be calculated as: $Cost_{update} = n_o \cdot c_o$

C. Overall Cost Per Time Unit

Equation 5 allows us to determine the cost of one time retrieval of relevant queries. We now analyze the frequency of such retrievals. According to equation 1, the average time duration that a node stays inside a cell is equal to $\frac{\pi r}{4\sqrt{2}E[v]}$.

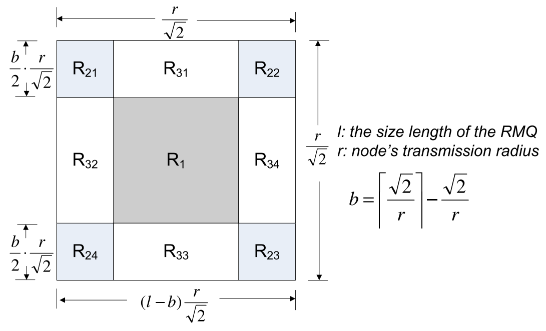

$\frac{r}{\sqrt{2}}$
$\frac{b}{2}\cdot\frac{r}{\sqrt{2}}$
R21
R31
R22
R32
R1
R34
R24
R33
R23
$\frac{b}{2}\cdot\frac{r}{\sqrt{2}}$
$\frac{r}{\sqrt{2}}$
l: the size length of the RMQ
r: node's transmission radius
$b = \left\lceil \frac{\sqrt{2}}{r} \right\rceil - \frac{\sqrt{2}}{r}$
$(l-b)\frac{r}{\sqrt{2}}$

Fig. 5. Possible regions for a query's center position

Thus, given that there are N mobile nodes, the frequency of the events that some node moves out its current cell is $Freq_{retrieval} = \frac{4\sqrt{2}E[v]N}{\pi r}$. Each time a node moves into a new cell, it needs to ask for the cell's relevant queries. Thus, the average cost per time unit for a query retrieval at the network wide can be calculated as

$$Cost_{retrieval\_ptu} = Cost_{retrieval} \times Freq_{retrieval} \quad (7)$$

On the other hand, if $Freq_{update}$ denotes the frequency of query installation or removal, then the cost of installation incurred per time unit is

$$Cost_{update\_ptu} = Cost_{update} \times Freq_{update} \quad (8)$$

Thus, the overall cost per time unit occurred in the entire system can be calculated as

$$Cost_{ptu} = Cost_{retrieval\_ptu} + Cost_{update\_ptu} \quad (9)$$

D. Analysis Validation

To assess the accuracy of our analytical model, we validate Equation 9 using simulation. Our simulator is built on top of an existing MANET testbed JIST/SWANS [23]. In our study, we fixed the network domain to be $5,000 \times 5,000$ $meter^2$ and placed on it a number of mobile nodes. The number of node is set to be 500 to 950 $nodes$. These nodes are uniformly distributed in the network domain and they move at random speeds, ranging from 0 to 10 $meter/second$. The node transmission radius is fixed at 500 $meter$. To broadcast a packet to all mobile nodes in some geographic region, we used an existing technique called *Priority Forwarding* [24]. The performance of this technique has been shown to be little sensitive to the node density – the number of nodes participating in forwarding a broadcast message is close to be proportional to the intended broadcast area. In our study, we count the cost of broadcasting a message as the number of nodes that rebroadcast this message. Thus, the cost of 1-hop broadcast is $c = 1$.

The range of each S-RMQ generated is a square and the size is uniformly distributed from 200 x 200 to 400 x 400. At the network wide, 10 S-RMQs are installed. The grid size is set to be 350×350 $meter^2$. During each simulation run, we periodically compute the average communication cost incurred during a fixed time interval, and collect the performance data when the average cost per time unit becomes stabilized. Figure 6 shows both the simulation results and the results computed using Equation 9. We observe that the formula is quite accurate in predicting the average cost per time unit in each simulation setting. The slight error mainly comes from the assumption that the broadcast cost is proportional to the area of a broadcast region. In our analytical model, the distance between the requesting node and the potential supplying node is round to an integer. In particular, the cost of broadcasting a message in a rectangular region (e.g., within a grid or a combination of a number of grids that overlap with a query) is also approximated by its area. In simulation, however, the area covered by each broadcast retransmission is actually a circle.

V. Performance Evaluation

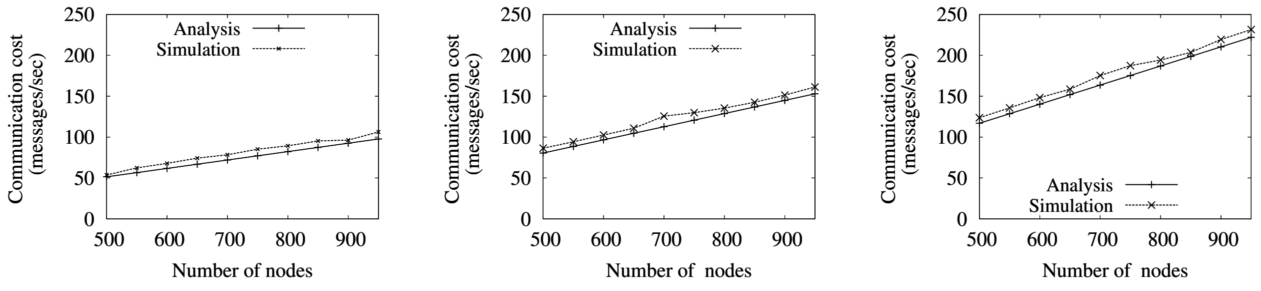
Our analysis allows us to conveniently evaluate the performance of our technique under different settings. However, it assumes the network is fully connected, and does not consider the delay in updating query results. In this section, we use our simulator for a more comprehensive performance evaluation of S-RMQs, wherein the network may be partially disconnected. We simulate a small battlefield where the movement of mobile nodes follows the random walk model [25]. The parameters used in our simulation is summarized in Table I. For performance comparison, we have also implemented a baseline approach. In this scheme, a node monitors the mobile population in a region of interest by periodically querying the nodes inside the cells that overlap with the region. Our study focuses on the following performance metrics:

- *Communication cost per time unit:* The total number of packets transmitted by each mobile node during the simulation divided by the total simulation time.
- *Miss rate:* The percentage of times that a mobile node that crossed into the region delimited by a safe boundary, it left the same region without being detected by a technique.
- *Delay:* The period from the time a node actually crosses over a query boundary to the time when the node knows that it has crosses over the boundary. This metric measures the delay in executing a query.

TABLE I
Parameters

parameter	default	unit
number of nodes	300	$nodes$
average node speed	5	$meter/sec$
network area	5,000 x 5,000	$meter^2$
cell size	350 x 350	$meter^2$
transmission radius	500	$meter$
number of S-RMQs	100	

In the next subsections, we report how the performance of the two techniques (i.e., proposed and baseline) is affected by the number of nodes and node mobility.



(a) Average node speed = 1 (b) Average node speed = 5 (c) Average node speed = 10

Fig. 6. Analysis Validation

A. *Effect of the number of nodes*

We generated a number of mobile nodes, from 100 to 500 and placed them on the network domain. For each simulation, the average of node speed is set to be 5 $meters/second$. In the baseline approach, the result of each query is updated at three different periods (T): 1, 5, and 10 seconds. Figure 7(a) shows the communication costs incurred by the proposed approach and the baseline approaches. The baseline approaches are insensitive to the number of nodes since the cost of query execution is determined by the frequency of updating query results. However, as the number of nodes increases, the communication cost of the proposed approach is initially reduced and then starts to increase. When the number of nodes is small (e.g., 100 nodes), the network is partially disconnected, and the chance is high that a node moving into a new cell cannot find any node in the cell for relevant queries and therefore has to search for the cell's retaining node. When the number of nodes is large, most likely a node will get its new cell's relevant queries within one hop. However, the number of the events of having some node crossing over a cell boundary increases, which causes the communication cost to increases.

Figure 7(b) shows the miss rate of the proposed approach and the baseline approaches. It shows that the miss rate of the baseline approaches is affected by the frequency of updating query results. The more frequent, the more accurate query results will be achieved. However, the baseline approach can never achieve 100% accuracy. In contrast, the proposed technique can guarantee accurate query results when the network is fully connected. When the network is slightly disconnected, it can still achieve nearly 0% miss rate.

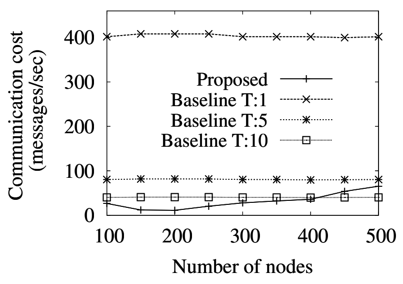
Figure 7(c) shows the delays incurred by the two techniques. It shows that the delay caused by the baseline approach initially drops and then becomes stabilized. This is due to the fact that when the number of nodes is small, many queries are never actually executed, i.e., the miss rate is very high, as showed in Figure 7(b). However, regardless of the number of nodes, the delay under the three baseline approaches is always considerably high. In contrast, the proposed technique can achieve almost instant execution of all queries.

B. *Effect of Node Movement*

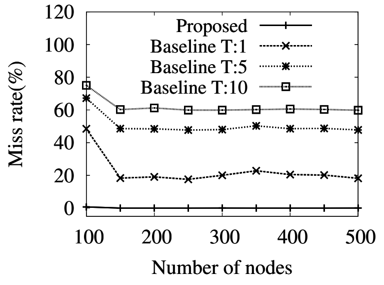
In this study, we generated 300 mobile nodes and randomly placed them on the network domain. We vary the average speed of node movement from 1 to 10 $meters/second$. Figure 8(a) shows the communication cost incurred by the proposed approach and the baseline approaches. The proposed technique incurs more communication costs as the node mobility increases. This is not surprising because the frequency of crossing cells increases when the average speed of mobile nodes increases. Each time a node moves into a new cell, it needs to retrieve the cell's relevant queries. As for the baseline approach, the communication cost is not affected by the node mobility. Figure 8(b) shows the miss rate of the proposed approach and the baseline approaches. The proposed approach shows no miss rate regardless of node speeds, because the network is fully connected given 300 nodes. As such, each query can be executed as soon as a node crosses over its boundary. As for the baseline approach, the miss rates under the three settings of query frequency increases as the average speed of nodes increases. As nodes move faster, the chance of crossing a query boundary without knowing the query increases. Figure 8(c) shows that the delay in query execution under each technique. Again, the proposed approach is able to guarantee real-time execution while the baseline approaches incur significant amount of latency.

VI. RELATED WORK

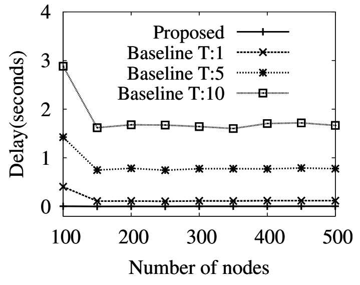
Existing research on spatial monitoring queries assumes a centralized environment wherein one or more servers are used for query management. The first technique for real-time processing of S-RMQs was Q-index [26],[27]. In this scheme, the server computes a safe region for each node, and a node needs to report its location only when it moves out of its safe region. Q-index reduces mobile communication cost, but it requires intensive server computation. More efficient techniques were developed later, including MQM [7] and SINA [9]. While these techniques assume a single server, the work in [28] considers using a number of regional servers was used for fault-tolerant query management. These servers are interconnected with high-speed networks and each is responsible for handling the queries inside some specific region.



(a) Communication cost

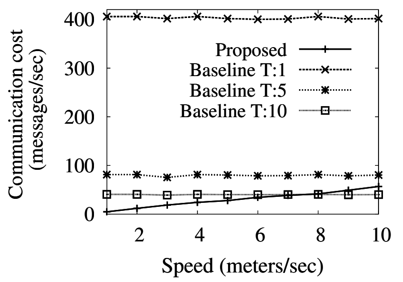


(b) Miss rate

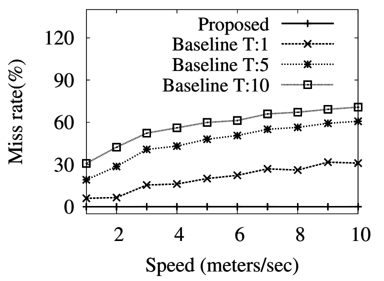


(c) Delay

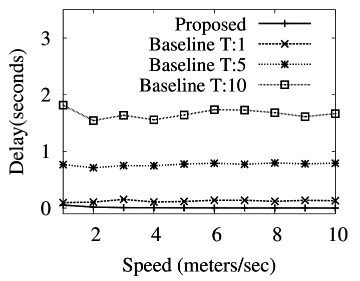
Fig. 7. Effect of the Number of Nodes



(a) Communication cost



(b) Miss rate



(c) Delay

Fig. 8. Effect of Node Movement

When a server is down, the corresponding queries are migrated to another server.

Early works on S-KNNMQs such as [29], [30], [31], [3] made assumptions on the motion patterns of mobile nodes and focus on indexing their trajectories to minize server disk I/O. These techniques usually can provide only approximate query results. More recent works on S-KNNMQs can be found in [5], [4], [32]. The proposed techniques do not consider how the clients update their location, but simply assume the server knows their location and focus on minimizing disk I/O incurred in query processing by indexing both moving objects and queries. Providing approximate results of S-KNNMQs was also considered in [33]. The proposed technique minimizes location update cost while ensuring the query results are reasonably accurate. In [10], the proposed technique lets mobile nodes participate in processing of S-KNNMQs.

Some techniques have also been developed to support more than one type of queries within one unified framework. Hu et al [12] proposed an approach to process S-RMQs and S-KNNMQs based on the concept of *quarantine area*. This area guarantees that as long as all result nodes stay inside it and all non-result nodes stay outside it, the query result will not change. Separately, Gedik et al proposed a generic technique, called *Motion Adaptive Indexing (MAI)* [34], which can support also M-RMQs and M-KNNMQs. Assuming the knowledge of the speed and direction of mobile nodes, it uses the concept of motion sensitive bounding boxes to model both mobile nodes and mobile queries in order to minimize the number location updates needed from the mobile nodes. The nodes and queries are indexed based on these bounding boxes, which are modified automatically based on the mobility of the nodes. Despise their differences in query processing, all of them rely on some central server.

VII. Conclusion

We have presented a novel technique for efficient processing of stationary range monitoring queries (S-RMQs), and showed that other types of queries, including mobile range monitoring query and stationary/mobile KNN monitoring query can be converted into a number of S-RMQs. Thus, they can be all supported within one unified platform. Unlike existing research, our technique does not rely on any central server. Instead, mobile nodes collaborate in query processing. Our technique is efficient because each mobile node needs to cache only its nearby queries. As a node moves, it can evaluate the queries it knows. As a result, accurate and real-time query results can be provided when the network is fully connected. We have developed a detailed mathematical model to evaluate the mobile communication costs incurred in query management. For comprehensive performance evaluation, we have also implemented a query management simulator based on a mobile ad hoc network.

In our current solution, the network partitioning is predetermined with a fixed cell size. While this simplifies our technique design, it may not achieve optimal system performance. A larger cell size reduces the chance of having a node moving out of it, and so decreases the communication costs incurred in retrieving relevant queries. However, it increases the cost of installing/removing a query, which needs to broadcast to all nodes inside a cell. Clearly, various factors such as node mobility and query activities need to be considered in order to determine a good cell size. As these parameters may change from time to time, dynamic network partitioning is necessary to ensure consistent good performance. We leave this as our future work.

REFERENCES

[1] B. Gedik, K.-L. Wu, P. Yu, and L. Liu, "Motion Adaptive Indexing for Moving Continual Queries over Moving Objects," in *Proc. of the 13th Conf. on Information and. Knowledge Management, (ACM CIKM'04)*, Washington, D.C., USA, 2004, pp. 427–436.

[2] D. Stojanovic, S. djordjevic, and B. Predic, "Incremental Evaluation of Continuous Range Queries over Objects Moving on Known Network Paths," *Lecture Notes in Computer Science*, vol. 3833, no. 1, pp. 168–182, November 2005.

[3] Y. Li, J. Yang, and J. Han, "Continuous k-nearest neighbor search for moving objects," in *Proc. of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, Washington, DC, USA, June 21 - 23 2004, p. 123.

[4] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proc. of the 21st Int'l Conf. on Data Engineering (ICDE'05)*, Tokyo, Japan, 2005.

[5] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proc. of the 21st Int'l Conf. on Data Engineering (ICDE'05)*, Tokyo, Japan, 2005.

[6] X. Xiong, M. Mokbel, W. Aref, S. Hambrusch, and S. Prabhakar, "Scalable Spatio-temporal Continuous Query Processing for Location-aware Services," in *Proc. of the Int'l Conf. on Scientific and Statistical Database Management (SSDBM'04)*, Santorini, Greece, June 2004.

[7] Y. Cai, K. A. Hua, and G. Cao, "Processing Range-Monitoring Queries on Heterogeneous Mobile Objects," in *IEEE Int'l Conf. on Mobile Data Management (MDM'04)*, Berkeley, CA, U.S.A, January 19-22, 2004, pp. 27–38.

[8] B. Gedik and L. Liu, "MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System," in *9th Int'l Conf. on Extending Database Technology (EDBT'04)*, vol. 2992, Heraklion, Crete, Greece, March 14-18, 2004.

[9] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable Incrementable Processing of Continuous Queries in Spatio-temporal Databases," in *Proc. of the 23th ACM International Conference on Management of Data (SIGMOD '04)*, Paris, France, 2004, pp. 623–634.

[10] W. Wu, W. Guo, and K.-L. Tan, "Distributed processing of moving k nearest-neighbor query on moving objects," in *Proc. of the 23rd International Conference on Data Engineering (ICDE'07)*, Istanbul, Turkey, April 17 - 20 2007, pp. 1116–1125.

[11] F. Liu, K. Hua, , and T. Do, "A p2p technique for continuous knearest-neighbor query in road networks," in *Proc. of the 17th International Conference on Database and Expert Systems Applications (DEXA'07)*, Krakow, Poland, September 04 - 08 2007, pp. 264–276.

[12] H. Hu, J. Xu, and D. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *Proc. of the 2005 ACM international conference on Management of data (SIGMOD'05)*, Baltimore, M, USA, 2005.

[13] Y. Ko and N. H. Vaidya, "Location-Aided Routing (LAR) Mobile Ad Hoc Networks," in *Proc. of ACM Int'l Conf. on Mobile Computing and Networking (MOBICOM98)*, Dallas, TX, U.S.A, October 25-30 1998, pp. 66–75.

[14] E. Royer and C. E. Perkins, "Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol," in *Proc. of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'99)*, Seattle, WA, USA, August 1999, pp. 207–218.

[15] B. Karp and H. T. Kung, "GPSR: Greedy Perimeters Stateless Routing for Wireless Network," in *Proc. of the 6th annual international conference on Mobile computing and networking (MOBICOM'00)*, New York, NY, USA, August 6-11 2000, pp. 243–254.

[16] J. C. Navas and T. Imielinski, "GeoCast – Geographic Addressing and Routing," in *Proc. of the 4th Annual Int'l Conf. on Mobile Computing and Networking (MOBICOM97)*, Budapest, Hungary, 1997, pp. 66–76.

[17] Y. Ko and N. H. Vaidya, "GeoTORA: A Protocol for Geocasting in Mobile Ad Hoc Networks," in *Proc. of the 8th Annual Int'l Conf. on Network Protocols (ICNP'00)*, Osaka, Japon, November 14-17, 2000.

[18] L. Blazevic, L. Buttyan, S. Giordano, J.-P. Hubaux, and J.-Y. L. Boudec, "Self-Organization in Mobile Ad hoc networks: The Approach of Terminodes," in *IEEE Personal Communications*, June 2000, pp. 166–174.

[19] X. Wu, "VPDS: Virtual Home Region based Distributed Position Service in Mobile Ad Hoc Networks," in *Proc. on 25th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, U.S.A, June 06-10, 2005, pp. 113–122.

[20] M. Motani, V. Srinivasan, and P. Nuggehalli, "PeopleNet: Engineering a Wireless Virtual Social Network," in *Proc. of the 11th Annual Int'l Conf. on Mobile Computing and Networking, (MOBICOM'05)*, Cologne, Germany, August 28 - September 2 2005, pp. 243–257.

[21] R. Thomas, H. Gilbert, and G. Mazziotto, "Influence of the Moving of the Mobile stations on the Performance of a Radio Cellular Network," in *Proc. of Third Nordic Seminar*, Copenhagen, Denmark, September 1988.

[22] B. A. Berg, *Markov Chain Monte Carlo Simulations and their Statistical Analysis (with Web-Based Fortran Code)*. World Scientific, 2004, iSBN 981-238-935-0.

[23] R. Barr, Z. Haasand, and R. van Renesse, "JIST/SWANS. Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator," Web page at http://jist.ece.cornell.edu/, May 2005.

[24] K. Kim, Y. Cai, and W. Tavanapong, "A Priority Forwarding Technique for Efficient and Fast Flooding in Wireless Ad Hoc Networks," in *14th Int'l Conf. on Computer Communications and Networks (IC3N'05)*, San Diego, CA, U.S.A, October 17-19, 2005, pp. 223–228.

[25] B.-D. Hughes, *Random Walks and Random Environments*. Oxford University Press, 1996, vol. 2.

[26] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch, "Queries as Data and Expanding Indexes: Techniques for Continuous Queries on Moving Objects," in *TR., Dept. of Computer Science, Purdue University*, 2000.

[27] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambruch, "Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects," *IEEE Transactions on Computers*, vol. 15, no. 10, pp. 1124–1140, October 2002.

[28] H. Wang, R. Zimmermann, and W. Ku, "Distributed Continuous Range Query Processing on Moving Objects," in *17th Int'l Conf. on Database and Expert System Applications (DEXA'06)*, Krakow, Poland, September 4–8, 2006.

[29] R. Benetis, S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest and reverse nearest neighbor queries for moving objects," *The VLDB Journal*, vol. 15, no. 3, pp. 229–249, 2006.

[30] Y. Tao and D. Papadias, "Time-parameterized queries in spatio-temporal databases," in *Proceedings of the 2002 ACM international conference on Management of data (SIGMOD'02)*, New York, NY, USA, 2002, pp. 334–345.

[31] K. Raptopoulou, A. N. Papadopoulos, and Y. Manolopoulos, "Fast nearest-neighbor query processing in moving-object databases," *Geoinformatica*, vol. 7, no. 2, pp. 113–137, 2003.

[32] W. Zhang, J. Li, and H. Pan, "Processing continuous k-nearest neighbor queries in location-dependent application," *Int'l Journal of Computer Science and Network Security*, vol. 6, no. 3, pp. 1–9, 2006.

[33] Y.-L. Hsueh, R. Zimmermann, and M.-H. Yang, "Approximate continuous k nearest neighbor queries for continuous moving objects with pre-defined paths," in *Perspectives in Conceptual Modeling, (ER'05)*, October 24-28 2005, pp. 270–279.

[34] B. Gedik, K.-L. Wu, P. Yu, and L. Liu, "Processing Moving Queries over Moving Objects Using Motion-Adaptive Indexes," *IEEE Transactions On Knowledge and Data Engineering*, vol. 18, no. 5, pp. 651–668, May 2006.