KPT: A Dynamic KNN Query Processing Algorithm for Location-aware Sensor Networks

Julian Winter Wang-Chien Lee

Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
Email: {jwinter, wlee}@cse.psu.edu

Abstract

An important type of spatial queries for sensor networks are K Nearest Neighbor (KNN) queries. Currently, research proposals for KNN query processing is based on index structures, which are typically expensive in terms of energy consumption. In addition, they are vulnerable to node failure and are difficult to maintain in dynamic sensor networks. In this paper, we propose KPT, an algorithm for dynamically processing KNN queries in location-aware sensor networks. KPT shows great potential for energy savings and improved query latency. Since the tree infrastructure is constructed only temporarily, KPT is less vulnerable to sensor node failure.

1 Introduction

Recent research on accessing data available in sensor networks has been focused on index structures, data storage, routing algorithms, data dissemination and aggregation techniques [2, 4, 6, 8, 7, 9, 15]. A major goal of these proposals is to support various types of queries posed to a sensor network from any location. A query is transmitted from the query source to the sensor nodes or network locations that contain the data needed to satisfy the query. The results (i.e., data collected at the sensor nodes) are then aggregated (if allowed) and returned back to the query source. The main requirement for query processing is to incur as little energy expenditure as possible without dropping the queries or sacrificing execution latency.

Spatial queries such as window/range queries and k nearest neighbors (KNN) search are particularly relevant to sensor network applications because the data needed for these applications is often geographically distributed in the network. Several approaches have been proposed that support window/range queries in sensor networks [5, 14], while a preliminary study of the KNN queries in sensor networks, called *Peer-Tree*, has just started [1]. Peer-tree, a distributed index structure based on the design principle of R-trees, ignores the fact that sensor nodes are susceptible to radio interference, signal attenuation, and fading. As a result of these radio problems index structures are difficult to implement in sensor networks and expensive to maintain in terms of energy consumption. This paper introduces the KNN Perimeter Tree (KPT) Algorithm for supporting KNN queries. KPT exploits the fact that KNN queries are geographically-based to achieve energy savings and increased fault tolerance. A preliminary performance evaluation is given to demonstrate the capabilities of KPT. For this paper we are assuming a stationary, location-aware sensor network. KPT assumes that sensors are aware of their geographical neighbors needed to support geographical routing. Sensor data is stored using local storage which can be organized as cache lines based on sensing event types. A given sensor can aggregate data over a period of time; for example a line in the cache may represent the sensing data of a minute.

This paper is organized as follows. In Section 2 we introduce KNN queries in sensor networks and review relevant research efforts. Then we introduce the KPT algorithm in Section 3 and its analysis in Section 4. Finally, Section 5 concludes this study and discusses the future work.

2 Related Work

In this section we describe the background of sensor networks, KNN queries and related research contributions.

2.1 KNN Query in Sensor Networks

k-Nearest Neighbor (KNN) queries of spatial data have been an interesting research topic for some time [10, 11]. A KNN query is initiated by a query source node and involves finding the k spatially nearest objects to a given query point

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

within the sensor network. Centralized or distributed index structures such as the R-tree have provided support for KNN queries [3]. However, in the context of sensor networks, technical issues such as node failures (caused by depleted energy resources or communication problems) make such index structures unwieldy and inefficient for executing KNN queries.

KNN queries can be classified into two types for sensor networks. For Type 1 queries, we assume that all sensor nodes locally store sensor data and are able to answer a specific query constrained by a geographical query condition. For example, assume that a query desires the k nearest temperature readings to some query point and all sensor nodes have a sensing component to measure temperature. In this case, the query needs to be transmitted to the k geographically nearest sensor nodes to the desired query point. The KNN nodes sample the temperature data and return it back to the query source node.

For Type 2 KNN queries, we assume that some additional query condition precludes the ability of all sensors to satisfy a query despite being located inside the desired geographic region. Type 2 queries request sensor data about the k nearest events to some given query point. These event locations are unpredictable and therefore determining which k sensors to transmit the query to for execution is more complicated than Type 1. In this paper, we consider only Type 1 KNN queries and leave support of Type 2 KNN queries as future work.

2.2 Geographical Routing

We assume for this research that sensor networks are stationary and location aware and that sensor nodes are knowledgeable about neighbor nodes within their radio range. Given these assumptions, several algorithms exist that can route messages towards geographic locations.

The Greedy Perimeter Stateless Routing (GPSR) algorithm is a geographical routing algorithm which operates in two modes in location-aware sensor networks: *greedy* mode and *perimeter* mode [4]. In greedy mode, the forwarding node forwards the message to the neighbor nearest the destination. If no such neighbor exists, the algorithm switches to perimeter mode, which, given a planarized graph of the network topology, routes messages around voids in the network. GPSR can be employed for routing Nearest Neighbor (NN) queries in sensor networks. Given a desired location, GPSR can continue to route the query message until the NN to the query point is reached. The nearest neighbor sensor node can be confirmed by routing in perimeter mode around the query point. Due to this nice property, GPSR was selected as the routing protocol for implementing KPT.

2.3 Peer-Tree

To the best of our knowledge, Peer-Tree (PT) is the only other proposal in the literature that is able to support KNN queries. Peer-tree applies the decentralized R-tree index structure to ad-hoc sensor networks in order to support location-based queries [1].

Like with the R-tree, the sensor network is partitioned into Minimum Bounding Rectangles (MBRs). Each MBR covers a geographical region and includes as a member any sensor node inside that area. The clusters are then organized in a hierarchical fashion until one overall cluster geographically spans the entire network. For each cluster, a specific node is designated as a clusterhead, which knows the location and ID of all sensors that belong to the MBR cluster. Furthermore, it knows the location and ID of the clusterheads of any child clusters and its parent clusterhead. Although the authors do not discuss the physical layer of the network topology directly, it is logical that that the authors assume the clusterhead can communicate with all nodes within its MBR as well as its parent.

In Peer-Tree, queries do not originate at the root of the tree, but come up from the level 0 child nodes since it is desirable to allow queries to be spawned from random locations in the network. NN queries can be locally scoped to include only the largest MBR necessary for satisfying the query. For handling NN queries, the source node routes the query message to its clusterhead. The clusterhead determines whether the query point is within its MBR. If so, the clusterhead then begins the algorithm for finding the NN. If it is not, the clusterhead forwards the query to its parent for processing. Eventually a clusterhead is reached that covers the area that contains both the query source and query point. This clusterhead becomes the Peer-Tree root node for processing the query.

The traditional branch-and-bound algorithm [10] is executed by the root node. Beginning with the child MBRs of the root, the partition list is sorted by MINDIST and the Peer-Tree is recursively traversed while a NN leaf node candidate is maintained and used for pruning MBRs. Supporting KNN queries with Peer-Tree is more complicated and not discussed by the Peer-Tree authors. For Peer-Tree to execute the query, it must be sent to the parent of the highest clusterhead required for finding the NN in order to guarantee that all candidate nodes will be evaluated (unless the query is already at the root clusterhead). At this point, the same branch-and-bound technique is employed except that a sorted buffer of at most k nearest neighbors is maintained and pruning is done according to the distance of the furthest nearest neighbor in this buffer.

There are several problems with the Peer-Tree approach. First, query messages must typically be routed through several layers of clusterheads. Transmission between clusterheads is executed largely independently of the physical geographic direction and distance. Depending on the network topology and the locations of clusterheads, it is possible that many unnecessary hops are included when routing messages towards query points. Furthermore, the clusterheads become communication bottlenecks where network congestion is likely (depending on the rate of submitted queries) especially if the distances between clusterheads is large and additional transmitting power is required. Ad-

ditionally, adding hierarchical infrastructure to sensor networks is inherently problematic since sensor networks are highly unstable. To handle the issue of fault tolerance, the authors propose using a lease period for all clusterhead nodes so that the hierarchical infrastructure is re-evaluated periodically.

3 KNN Perimeter Tree

Our hypothesis is that geographical routing algorithms such as GPSR can be used to approach shortest-path routing such that overall improved performance and fault tolerance is possible for KNN queries. Minimizing the individual responsibilities of sensor nodes makes the network less vulnerable to failure since there are no critical nodes in the network. Furthermore, less communication is necessary to maintain index or topology information in the network.

The KNN Perimeter Tree (KPT) builds upon GPSR [4] for processing KNN queries. KPT is deployed at all sensor nodes during network deployment. GPSR can successfully deliver messages to the nearest neighbor of any query point in the network. Since data is only available at the sensor nodes that generate them, a query need only be routed to the sensor nodes that own the data. All nodes in the network may participate in processing/forwarding queries.

The KPT algorithm can be broken down into phases as follows:

1. find the nearest neighbor and a maximum KNN boundary;
2. find $k-1$ nearest neighbors;
3. disseminate and execute query;
4. return result.

3.1 Find NN and a Maximum KNN Boundary

The query message is geographically routed from the query source towards the query point specified in the query. Based on GPSR, the message will eventually reach the geographically nearest neighbor to the query point. This node is designated as the *home node* of the KNN query. The home node is assigned temporary responsibilities for organizing the dissemination of the query and processing the results. This responsibility does make the home node vulnerable to node failure however only for the short duration of the time needed to process the query.

To avoid flooding a query to the whole network, a maximum KNN boundary is estimated to restrict the search space for finding the remaining $k-1$ nearest neighbors. We consider several approaches for determining this boundary while the query message is being routed to the home node. These approaches seek to determine a circular boundary in terms of a radius distance centered at the query point which is guaranteed to contain the KNN sensor nodes and the approaches have different tradeoffs.

An intuitive approach (called SUMDIST) for determining the boundary is to add the position of each sensor node on the forwarding path from the query source to the home node to a list in the query message. When the home node is reached, the distance between the home node position and the k-th position in the list serves as the maximum boundary. This approach has a higher communication cost since up to k locations are transmitted along with the query at every hop. For large values of k, this cost can be large.

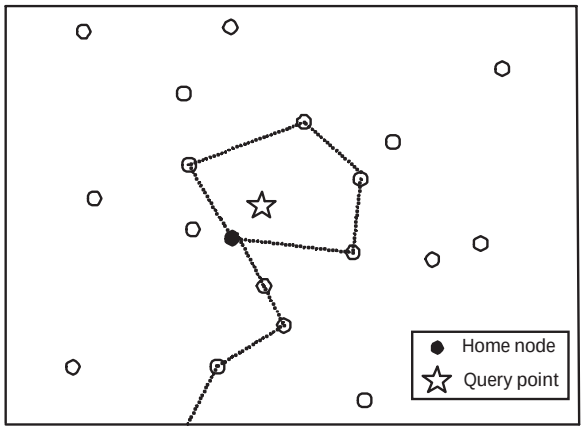

Figure 1: KPT home node and perimeter

A second approach (called MHD-1) includes only a counter variable, and a maximum hop distance (MHD) value which represents the largest distance value for any one hop on the route between the query source node and the home node. The counter variable is incremented at each forwarding hop until it reaches k. MHD always maintains the largest hop distance visited. After the query message reaches the home node, the maximum KNN boundary value can be determined by multiplying the MHD value by k. The advantage of this approach is that the cost of determining the maximum KNN boundary is less than the naive approach since only a few values are transmitted with the query message (independent of k). However, the search boundary is likely to be larger (and thus less efficient) than the boundary obtained from the naive approach.

An improvement on the second approach (called MHD-2) is to minimize the MHD value by plotting the hop distance along the direct path between the query source and query destination using geometry instead of taking the direct hop distance between neighbor nodes. However, the location of the query source node has to be added to the query message at an additional energy cost.

An assumption that is made for all three methods is that at least k hops occur on the route between the query source and the home node. Therefore, it is necessary to consider the case when fewer than k hops occur. To solve this problem we estimate the boundary by taking the MHD value and multiplying it by k (even for the naive approach). We believe that this estimation should be fairly good for many cases; however in implementing the KPT algorithm, we must consider the case when the estimation fails.

Figure 1 demonstrates the state of the KPT algorithm after the query has been routed to the nearest neighbor home node and the perimeter has been established. The query point is illustrated with a star and the home node which connects the incoming geographical route with the perime-

ter route is solid.

3.2 Find $k-1$ Nearest Neighbors

Given that the query is at the home node which knows the maximal KNN boundary, the next step is to determine the IDs and locations of the $k-1$ nearest neighbor nodes. A naive approach is to simply flood the query to all nodes within the circular KNN boundary centered at the query point. However, flooding expends excess energy, particularly if nodes are densely packed with much overlapping of radio and sensing ranges.

We propose the Perimeter Tree which is designed to reduce the number of total messages required to determine the $(k-1)$-NN nodes and for disseminating the query to them. The philosophy of this approach is to divide the boundary circle into regions for each of which a minimum spanning tree can be constructed that is rooted at a perimeter node. The subtrees expand in the direction away from the destination. The individual trees are bounded by the circular boundary and the two subtree boundaries on both sides.

The perimeter nodes that encircle the query point each make up a root of a minimum spanning tree that expands away from the destination and is bounded by the circular KNN boundary. The perimeter nodes are determined when the query message is transmitted by the home node in GPSR perimeter mode to validate the home node as the NN to the query point similar to the Perimeter Refresh Protocol in GHT [9]. At each hop around the perimeter, the midpoint on the line between Perimeter nodes is computed and by plotting a line from the query point through the midpoint to the circular boundary the subtree boundaries are determined, similar to a Voronoi cell [12].

The next step is to establish the spanning trees in each of the bounded areas that are rooted at the perimeter nodes. The goal is to build a tree with as few messages transmitted as possible and with also the shortest possible latency. By having multiple trees rooted at the perimeter nodes instead of one tree rooted at the home node the maximum height of the trees is reduced which reduces the overall query latency, although in highly irregular networks balancing the tree may not be possible which would affect the query latency but not the correctness. The construction of the tree begins with the perimeter root node which knows the query point, the two subtree boundaries (the midpoints between it and its two perimeter neighbors) and the circular KNN boundary. At a minimum, this information is transmitted to its potential children along with other information specified in Phase 3. In a tree, nodes only have one parent and belong to a certain level of the tree. Finally, a child node responds to its parent after hearing from its children and transmitting all node level information including node IDs and locations. This information is forwarded to the perimeter root which then transmits it to the home node. The home node then has all the locations of all nodes within the circular KNN boundary which it can then sort by their distance from the query point and thus determine the KNN node set.

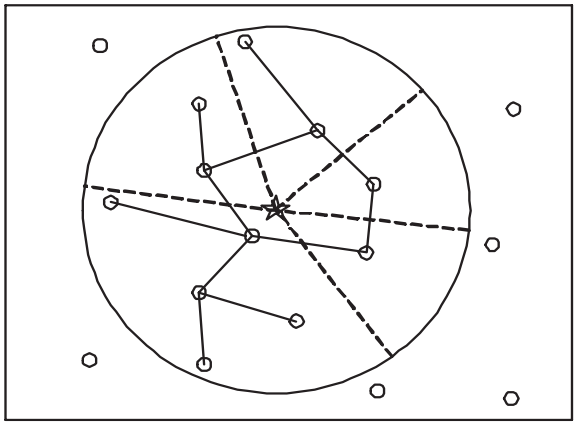

Figure 2: KNN Perimeter Tree

The perimeter boundaries are employed in order to keep the tree as balanced as possible and thus reduce the overall query latency. However, strictly enforcing this boundary for construction of the tree may exclude nodes that are within the circular boundary but are out of communication range of all potential parent nodes within its median boundary. Therefore we allow nodes to select a parent outside its tree boundary, but only if it does not hear a request from another potential parent from within its tree boundary. Although it may be possible for a sensor node to exist within the circular boundary and be completely disconnected from all other nodes within the circular boundary, it is unlikely. Furthermore, this would tend to happen towards the edge of the circular boundary reducing the probability that the disconnected node belongs to the KNN set.

Figure 2 demonstrates the state of the KPT after the Perimeter Tree has been established. The perimeter nodes are used to construct the tree boundaries to minimize the total height of the tree.

3.3 Disseminate and Execute Query

After Phase 2, the home node is aware of the IDs and locations of the KNN nodes. The next step is for the query to be disseminated for execution. A naive approach is for the home node to unicast or multicast using the Perimeter Tree the query to the KNN nodes. In order to reduce the overall latency, we propose combining the query dissemination with the Perimeter Tree establishment from Phase 2. As the Perimeter Tree is constructed, the actual query is transmitted to all tree members for automatic execution. This approach should have drastically improved latency, but less efficient energy performance since more than the KNN nodes actually execute the query. Imposing a quota system on the number of nodes to execute a query per subtree can reduce the execution cost without increasing the latency. The quota estimation method assigns the top q nodes of every subtree to execute the query automatically where q is a quota estimation defined in Equation (1) and p is the number of perimeter nodes and c is an adjustable parameter which trades off the quota size and the number of retransmissions needed when quota estimations fail.

$$q = \frac{k}{p} + c \quad (1)$$

The q value is set by the perimeter root node and decremented as it is assigned to nodes farther down the tree. The nodes assigned to execute the query do so and return the results back to the home node as the tree is constructed. The remaining nodes in the tree that are not assigned by the quota to execute the query automatically simply return location information.

After the tree is constructed, the home node receives the $p \times q$ results along with all the location and ID results from all nodes within the circular KNN boundary. The home node determines the KNN node set and whether the quota results include all necessary data to satisfy the KNN query. If any members of the KNN node set did not return quota estimation results, then the quota failed and must be resolved. The resolution can be handled simply by unicasting the query to the missing nodes and routing the results back, adding additional overhead and latency and is thus undesirable. The c parameter can be adjusted by experiment to determine the appropriate quota size. Flooding is used to execute the query if the circular boundary is underestimated using one of the MHD methods which adds considerable energy and latency costs. However, we feel that this situation will be rare.

3.4 Return Results

After the home node has collected the query results, it needs to transmit them back to the query source by unicasting the results geographically using GPSR. The Perimeter Tree can be destroyed after the location information has been returned to the home node. We reiterate that the Perimeter Tree only exists for a short period of time and therefore is only vulnerable to node failure very briefly unlike Peer-Tree.

4 Preliminary Performance Analysis

To give an idea of the capabilities of KPT versus Peer-Tree, we performed a mathematical analysis on both approaches in terms of the number of messages required to execute a query. For the analysis, we assume that nodes are uniformly distributed. To determine the cost processing KNN queries with KPT and Peer-Tree, we define some parameters which are listed in Table 1.

For analyzing the performance of KNN query processing, we break the execution into three phases for both KPT and PT:

- Phase 1 consists of the number of messages required to reach the home node for KPT or the Peer-Tree MBR root node.
- Phase 2 represents the cost of executing the query by getting the query to the KNN nodes and returning the results back to the Phase 1 home node.

Variable	Definition
h	Height of Peer-Tree
l	Average distance between nodes
n	Number of nodes in network
x	Number of nodes in KNN PT MBR
f	MBR fanout ($.69 \times M$)
s	Square axis of network ($s \times s$)
d	Average query distance
k	Number of nearest neighbors required
m	Minimum children per MBR
M	Maximum children per MBR
P_i	Probability a PT node is accessed at level i

Table 1: Summary of Parameters for Analysis

- Phase 3 represents the cost of returning the query results back to the query source node.

Estimating the query execution cost for KPT is fairly simple. For phases 1 and 3, we can estimate the number of hops required to route a message to the query source node and the home node and back by using the expression $\frac{d}{l}$. For phase 2, we estimate the number of messages as two messages per node inside the circular boundary. We can compute the average number of nodes inside the circular boundary by dividing the area of the circular boundary by the average area per sensor node (density) and thus we define the number of messages as $2 \times (\pi \times (k \times l)^2)/((s^2/n))$.

Performance analysis of Peer-Tree is more complicated. We refer to the analysis of KNN queries for R*-Trees [13] which is similar to Peer-Tree except that message transmissions are used instead of disk accesses when information from a node is needed. For phases 1 and 3, the number of messages required to transmit the query message to the root parent node and the results back is the number of levels in the tree from level 0 to the level of the root parent. The level of the root parent is one above the smallest MBR that contains the query point and the query source node. For estimating the size of the smallest MBR that contains the query point and the source node we assume an average square-shaped MBR where the query distance d makes up half the bisecting hypotenuse with an area of $2 \times d^2$. The number of sensor nodes contained within the parent of the MBR that spans the source node and query point can be estimated as $x = (h \times 2 \times d^2)/((s^2)/(n))$. We can determine the height of the tree needed to execute the query as $h = 1 + \lceil \log_f(\frac{x}{M}) \rceil$ [13].

For computing the cost of phase 2 for Peer-Tree, we use the same formula for node accesses defined as $\sum_{i=0}^{h-1} (n_i \times P_i)$ where h is the height of the tree, P_i is the probability that a node at level i is accessed and n_i is the total number of nodes at level i [13]. Due to the space constraints of this paper, we leave the details to [13]. Two messages are required for each node access, one to deliver the query and one for a response.

For constructing experiments using the mathematical analysis the following default parameters were used. A network size of 100×100 meters^2 was used with a node

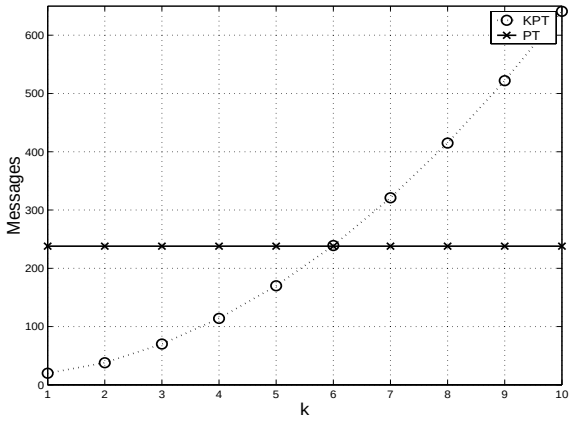

Figure 3: Experiment 1: Effect of k

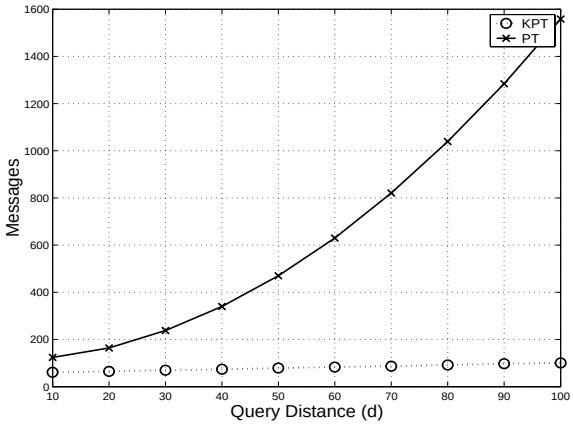

Figure 4: Experiment 2: Effect of query distance

density of 500 uniformly distributed sensors. The average query distance used was 30 meters with a k value of 3. For Peer-Tree, each MBR contained between 3 and 6 children. The metric used for analysis was simply the number of messages required to execute the query for KPT and Peer-Tree.

Figure 3 demonstrates the effect of k on the performance of KPT and Peer-Tree. The results show that while Peer-Tree is not affected by the value of k, KPT performs better for lower k values, specifically with k smaller than 6. This makes sense since the larger the k value, the larger the circular query boundary which includes more nodes in the query.

Figure 4 shows the effect of the query distance on the execution performance of both approaches. The effect of the query distance on KPT is minimal; only a very small linear increase for KPT while Peer-Tree suffers an exponential increase in the number of messages as the query distance increases. This is due to the fact that the size of the spanning parent MBR grows much larger and the height of the tree increases as well. Although not demonstrated here, Peer-Tree is also affected by the size of the child node capacity and the node density of the network.

We acknowledge that this analysis is primitive by simply counting the number of messages of an individual query and does not take into account that the messages for Peer-Tree would likely have to be transmitted at a higher power level and are thus more expensive. The size of the messages, per-bit cost of transmission and query execution costs are also not considered here. Most importantly, this analysis assumes that all required infrastructure for Peer-Tree is in place, i.e., the considerable cost for constructing and maintaining the tree is not demonstrated. Nonetheless, KPT is able to perform often significantly better than Peer-Tree for executing KNN queries. Fault tolerance to node failure is also not demonstrated. Considering fault tolerance and actual energy consumption will be demonstrated through simulation in our future work.

5 Conclusion

We believe that KPT shows potential for improving performance in terms of energy consumption and latency for processing KNN queries in sensor networks. Our preliminary analysis shows that KPT can achieve significant energy savings over Peer-Tree in terms of the number of messages required to execute a KNN query without even comparing the costs required to construct and maintain the Peer-Tree infrastructure when compared to the minimal neighbor information required for geographical routing. Additionally, although not demonstrated through analysis, KPT intuitively is more fault tolerant than Peer-Tree.

For the future work of this project, simulation experiments are under construction that are designed to back up the claims of this paper. Additionally, further improvements of KPT may be possible if assumptions can be made about the node distribution. Furthermore, we intend to also investigate the use of KNN queries in mobile sensor network environments by employing routing protocols for dynamic networks. Finally, we intend to consider supporting Type 2 KNN queries with KPT.

References

[1] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of the 3rd IEEE International Conference on Peer-to-Peer Computing*, Linkping, Sweden, September 2003.

[2] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of the IEEE ICC Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, April 2003.

[3] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[4] B. Karp and H.T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, 2000.