Peer-to-Peer Spatial Queries in Sensor Networks

Murat Demirbas Hakan Ferhatosmanoglu

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210 USA
{demirbas,hakan}@cis.ohio-state.edu

Abstract

Sensor networks, that consist of potentially several thousands of nodes each with sensing (heat, sound, light, magnetism, etc.) and wireless communication capabilities, provide great opportunities for monitoring spatial information about a region of interest. Although spatial query execution has been studied extensively in the context of database systems (e.g., indexing technologies), these solutions are not directly applicable in the context of sensor networks due to the decentralized nature of the sensor networks and the limited computational power and energy scarcity of individual sensor nodes.

In this paper, we present a peer-to-peer indexing structure, namely peer-tree, in order to address the problem of energy- and time-efficient execution of spatial queries (such as nearest-neighbor queries) in sensor networks. Loosely speaking, our peer-tree structure can be interpreted as a peer-to-peer version of the centralized R-tree index structure. Using the peer-tree as a building block, we present a peer-to-peer query processing model where a query can be posed in any node of the network without the need of a central server. For achieving minimal energy consumption and minimal response time, our query processing model ensures that only the relevant nodes for the correct execution of a query are involved in the query execution.

Keywords : sensor networks, peer-to-peer spatial queries, distributed index structures

1 Introduction

With the recent advancements in micro-electro-mechanical-systems (MEMS) related technology, it has now become feasible to manufacture low power sensors that integrate detection of infrared radiation, heat, sound, vibration, and magnetism together with on-chip intelligence and wireless communication [17]. Progress in this area is significant for its applications in deploying large ad-hoc wireless sensor networks (potentially consisting of several thousands of nodes) to retrieve spatial information about an area of interest. Sensor networks find applications in ecology (e.g., environmental and habitat monitoring [24]), in precision agriculture (e.g., monitoring of temperature and humidity), in civil engineering (e.g., monitoring stress levels of buildings under earthquake simulations), in military and surveillance (e.g., tracking of an intruder [9]), in aerospace industry (e.g., fairing of cargo in a rocket), etc.

In this paper we focus on efficient execution of spatial queries in sensor networks. In particular, we are concerned with efficient execution of nearest neighbor (NN) queries in sensor networks: "What is the location of the nearest data object to coordinates (x,y)?". The query can be initiated locally, without the need of a central server. The coordinates given in the query can be either the current location of the user or a distant location of interest.

In the context of database systems, there has already been an extensive work on indexing structures that enable efficient execution of nearest neighbor queries. We identify R-trees [1, 15] as amenable for the type of queries we are interested in this paper. However, the work on spatial query execution in the database systems are not readily applicable in the context of sensor networks, since sensor networks introduce the following challenges over database systems:

- Nodes in sensor networks have very limited computational resources (e.g., 8K RAM) and are energy constrained (e.g., 2 AA batteries per node); thus, centralized programs are not suitable for sensor networks due to their larger computational and communication requirements. In particular, since a message send operation may spend at least 1000 times more battery than a local operation (e.g., sense operation), sensor network programs should avoid unnecessary communication as much as possible. For example, a centralized program that demands all sensor nodes to periodically feed their data into a central repository (e.g., a base station) is not feasible because it will drain the battery power of the sensor nodes quickly.

  Decentralized versions of spatial query execution programs are needed for sensor networks: by using a decentralized program instead of a centralized approach, it would be possible to contact only the relevant nodes for the execution of a spatial query and hence achieve minimal energy consumption.

- In sensor networks any node should be able to introduce a query to the system. For example, in the context of a fire evacuation scenario a firefighter should be able to query a nearby sensor node for the closest exit where a safe path exist. Therefore, a peer-to-peer query processing model is required. To the best of our knowledge, we are not aware of any peer-to-peer programs for spatial query execution.

- Sensor networks are deployed in adverse conditions: message losses and corruptions (due to collusion, and hidden node effect), and node failures (due to crash and energy exhaustion) are the norm rather than the exception. Furthermore, due to the vast area that the nodes are spread over, it is not feasible for a human operator to set up and maintain the sensor network. Therefore, the proposed programs should be self-stabilizing [10].

Even though there has been extensive work on indexing structures for exact match queries for data sharing in the context of peer-to-peer systems [14, 27, 29], to the best of our knowledge the use of index structures for peer-to-peer execution of spatial queries has not been explored before. Furthermore, the work on indexing in peer-to-peer systems do not take into account the constraint on energy usage which is crucial in design of sensor network programs.

Contributions of this paper. In this paper, we address the problem of efficient execution of spatial queries in sensor networks. Our main contributions are as follows:

1. We establish an efficient access structure on sensor networks in order to contact only the relevant nodes for the execution of a query and hence achieve minimal energy consumption, minimal response time, and an accurate response. To this end, we introduce a peer-to-peer indexing structure: *peer-tree* (a peer-to-peer version of the R-tree data structure).

2. We present a peer-to-peer query processing model over peer-trees that enables execution of NN and other spatial queries. In our peer-to-peer query execution model, a query can be posed in any node of the network, as opposed to using the base station as the sole entry point for all queries. We discuss trade-offs of various implementation alternatives for spatial queries.

3. We present a self-stabilizing peer-tree construction program and thus render the above peer-to-peer query processing model to be self-stabilizing.

Organization of the rest of this paper. In Section 2 we present the system and fault model used in this paper. In Section 3 we present our self-stabilizing peer-tree construction program. In Section 4 we show how to efficiently perform nearest neighbor queries in sensor networks and also discuss extensions for execution of other spatial queries. We compare and contrast our work with related work in Section 5. Finally, in Section 6, we conclude the paper.

2 System and Fault Model

We consider a geometric network model: sensor nodes lie in a 2-D coordinate space. We assume a uniform distribution of nodes and a connected topology, and that the cost of communication (potentially through multihops) between nodes that are distance d apart takes $O(d)$ amount of energy.

Each sensor node has a unique identifier, id. Each node has a field of communication, within which it is capable of receiving/transmitting messages. All nodes within this unit field are its immediate neighbors (duplex links). For a node v, we denote v's immediate neighbors as 1-neighborhood of v, $Nbr(v,1)$, and denote v's r-neighborhood, $Nbr(v,r)$, as the set of nodes within a radius r of v . We assume that the nodes have been deployed, and arranged themselves into a connected topology, protocols for routing of the messages, location discovery of the nodes, and maintenance of the network exist, and are transparent to the application layer. Such protocols are proposed in [5, 7, 32].

Fault model. We consider transient faults to affect the sensor network. Nodes can fail for sometime and wake up, and their state can be arbitrarily and transiently corrupted. We do not assume any Byzantine faults in our model.

A program is *stabilizing fault-tolerant* iff starting from an arbitrary state the program eventually recovers to a state from where its specification is satisfied.

Problem statement. The problem is to design a peer-to-peer query processing model where

- only the relevant nodes for the correct execution of a spatial query are involved in the query execution (for achieving minimal energy consumption and minimal response time), and

- the underlying peer-to-peer indexing structure is self-stabilizing (for achieving eventually correct answers to the queries in the presence of faults).

3 Peer-trees

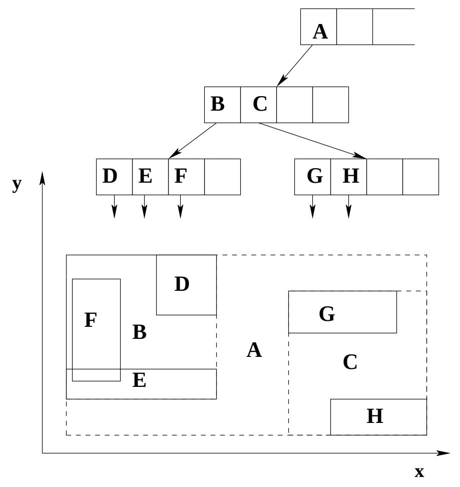
In this section we present a construction program for a peer-to-peer R-tree implementation (peer-tree). To this end, we first recall the R-tree [15] data-structure briefly in the following subsection.

3.1 R-tree

An R-tree is an approximately balanced search tree that is widely used for handling spatial data in traditional database systems. In particular, R-tree is a generalization of $B+$-tree developed for efficient processing of intersection queries on spatial databases. R-trees keep a set of (hyper-) rectangles and allow the handling of arbitrarily shaped objects by representing each one with its minimum bounding rectangle (*MBR*). The algorithms extend to higher dimensions in a straight forward manner, but to keep the exposition clear we only consider the 2-dimensional case.

An R-tree node is allowed to hold between n and N, $N \geq 2 * n$, number of (mbr, c) pairs where mbr denotes the minimum bounding rectangle (MBR) of all rectangles stored in the subtree pointed to by the children pointer c. Note that: (a) rectangles at any level may be overlapping, and (b) every descending path in the tree is a sequence of nested rectangles with the last one containing actual positions of nodes in the sensor network.

A section of an R-tree and the corresponding MBRs are depicted in the following figure.



3.2 Peer-tree construction

An effective adaptation of R-trees to a de-centralized model offers many advantages including efficiency in searching over peer-to-peer systems. Here we present a distributed program for constructing a peer-to-peer version of R-tree indexing structure.

Loosely speaking, our program constructs a hierarchical partitioning of a sensor network into rectangle-shaped clusters based on the number of nodes contained in the clusters. In our hierarchical partitioning program, every node cooperates at level 0 of the peer-tree construction, and only clusterheads of level i cooperate for construction of level $i+1$ of the peer-tree. Every node v maintains a variable $l.v$ (read level of v) denoting the highest level of the peer-tree construction that v has participated in. For every level i, $0 \leq i \leq l.v$, v maintains a variable $p.v(i)$ (read parent of v at level i) to denote v's immediate clusterhead for level i. Dually, as discussed in Section 3.1, v also maintains $c.v(i)$ (read children of v for level i), $0 < i \leq l.v$, in order to point to all nodes w such that $p.w(i-1) = v$. That means, v is the clusterhead of the MBR that contains $c.v(i)$. As we describe in Section 4, using this doubly-linked structure, any node can query the peer-tree in a localized manner.

In our program we have two actions: (1) "*join/form*" a cluster and (2) "*split*" a cluster.

We first describe the join/form cluster action. A node v with $l.v = i$ executes the join/form cluster action if its random timeout is expired and yet it is not included in the cluster of a level $i+1$ node (i.e., $p.v(i) = nil$). In this case, v first tries to contact a neighboring node u with $l.u = i+1$ by searching increasingly larger radii, r, and join u's cluster (by setting $p.v(i) = u$). If no such u exists and v encounters n nodes in level i within its r-neighborhood, then v waits for a random time before setting $l.v = i+1$ and becoming the clusterhead of these n nodes in one atomic step. (Several self-stabilizing programs [8] exist for performing this operation.) If within the random wait period, v is contacted by another node u with $l.u = i+1$, v simply joins u's cluster. This way, the random wait period takes care of the formation of premature clusters that are concurrently started by multiple nodes within the same region. Thus, the join/form cluster action for node v at a level i is as follows:

```
(join/form a cluster)
if ($l.v = i$ and $p.v(i) = nil$ and $wait(T_{random})$)
then { $r := 1$;
        //search for increasingly larger radii
        while ($|Nbr(v,r)| < m$ or $p.v(i) \neq nil$) {
            if ($\exists u : u \in Nbr(v,r) : l.u = i+1$) [1]
            then { $p.v(i) := u$; $c.u(i) := c.u(i) \cup \{v\}$; }
            $r := r + 1$;
        } //end-while
        wait($T_{random}$);
        if ($p.v(i) = nil$ and
              $|\{u \mid u \in Nbr(v,r)$ and $p.u(i) = \perp\}| \geq n$)
        then { // form a cluster using Nbr($v,r$)
            ($\forall u : u \in Nbr(v,r) : p.u(i) := v$);
            Calculate $mbr.v(i)$ using $Nbr(v,r)$;
            $p.v(i) := v$; $l.v := i + 1$;
            $c.v(i+1) := Nbr(v,r)$;
        } //end-if
}
```

[1] A formula ($op\ i : R.i : X.i$) denotes the value obtained by performing the (commutative and associative) op on the $X.i$ values for all i that satisfy $R.i$. As special cases, where op is conjunction, we write $(\forall i : R.i : X.i)$, and where op is disjunction, we write $(\exists i : R.i : X.i)$. Thus, $(\forall i : R.i : X.i)$ may be read as "if $R.i$ is true then so is $X.i$", and $(\exists i : R.i : X.i)$ may be read as "there exists an i such that both $R.i$ and $X.i$ are true". Where $R.i$ is true, we omit $R.i$. If X is a statement then $(\forall i : R.i : X.i)$ denotes that X is executed for all i that satisfy $R.i$.

Note that during concurrent executions of join action by multiple nodes, a clusterhead may end up having more than N children. In the split cluster action, a clusterhead v with more than N children at level i (i.e., $|c.v| > N$) splits its cluster into clusters with number of children greater than or equal to n but less than N. In the literature several programs exist for split operation [15], and since clusterhead v has information on mbr's of its children any of these programs is applicable at v. After the split operation the resulting new level i clusterheads inform their parent cluster about the modification as another split operation may need to be scheduled at level $i+1$ as a result of this split operation.

```
(Split a cluster)
if  ($c.v(i) > N$)      then    split($mbr.v(i)$);
```

3.3 Self-stabilization of peer-tree

In order to achieve stabilization of the peer-tree, we employ a technique called "soft-state" stabilization [30, 33]. That is, we require a *lease* on critical variables of the peer-tree structure so as to check and correct these variables periodically (i.e., when the lease expires).

More specifically, we assign a lease (timeout period) on children variable of each clusterhead v at every level i. When the lease expires, v checks its $c.v(i)$ for correctness. If all nodes w in $c.v(i)$ are $i-1$ level clusterheads with $p.w(i-1) = v$ and are within $mbr.v(i)$, and moreover $n \leq c.v(i) \leq N$ holds, no correction is needed. If $c.v(i) \geq N$ split action takes care of the correction. If $c.v(i) < n$ we destroy v's i'th level cluster, $mbr.v(i)$, so that the members can join neighboring i'th level clusters. Similarly, if there exists a member w of $c.v(i)$ such that $w \notin mbr.v(i)$ or $p.w \neq v$, we also reset v's cluster.

```
(Collapse a cluster)
if  ( timeout ( lease($c.v(i)$) ) )
then {
    if ( $c.v(i) < n$ or
        ($\exists w : w \in c.v(i) : w \notin mbr.v(i) \;\vee\; p.w \neq v$))
    then {  collapse ($mbr.v(i)$); }
}
```

In order to conserve energy we set the lease period on $c.v(i)$ to be proportional to the level i of the cluster. The higher the cluster level, the higher is the energy required for the correctness check, and hence, we require such a check less frequently.

Another tolerance property that might be desirable for sensor networks is to mask the failure of a clusterhead. That is, when a clusterhead responsible for some region fails, we want another node within that region to assume responsibility for the region. One such protocol for rotating leader nodes is discussed in [32].

4 Peer-to-Peer Nearest Neighbor Queries in Sensor Networks

We first present an overview of nearest neighbor queries in Section 4.1 and then, in Section 4.2, we present a method for efficiently performing peer-to-peer nearest neighbor queries in wireless sensor networks. In Section 4.3 we discuss the tradeoffs involved in the method and in Section 4.4 we discuss extensions to the method for handling other types of spatial queries.

4.1 Nearest neighbor queries

A nearest neighbor (NN) query is defined as follows: Given a set of data objects, find the data object from the data set which is closer to a given query object than any other object in the data set. The widely used NN algorithm is a branch-and-bound technique implemented over a tree-based index structure [28]. The tree, which consists of minimum bounding rectangles (MBR), is traversed and several of the MBRs are pruned if they are guaranteed not to have the closest data point(s). At each level, the pruning comparison involves distances to the children nodes as well as the distance to the previously found nearest neighbor candidate. Only nodes that can lead to a possible nearest neighbor are further considered, while the remaining nodes and corresponding subtrees are pruned out of the search space. There are a number of cases where MBRs can safely be pruned. For example, if MINDIST of an MBR, i.e., the shortest possible distance between the query point and any point in the MBR, is larger than the current computed NN distance, then there is no need to check the data points within that MBR.

An efficient implementation of NN algorithm over R-trees is given below [2].

```
initialize PartitionList with the subpartitions of the root
sort PartitionList by MINDIST;
while (PartitionList is not empty) {
    if (top of PartitionList is a leaf)
    then {find nearest point NNC in the leaf;
            if (NNC closer than NN)
            then { prune PartitionList with NNC;
                    NN:=NNC; }   }
    else { replace top of PartitionList with its children; }
resort PartitionList by MINDIST;
} //end-while
output NN
```

4.2 Peer-to-peer nearest neighbor queries

We now discuss techniques for performing peer-to-peer nearest neighbor queries over a sensor network. Using peer-tree as the indexing mechanism, the task of performing NN queries in a distributed fashion is significantly simplified. The remaining issue is to enable querying of the index structure with initiation from any node, as opposed to initiation only from the root: In traditional database systems, NN queries are executed always starting from the root of the tree, i.e., the top-most level clustering; in contrast, in a peer-to-peer environment any node should have the capability of initiating the query.

In our peer-tree structure, data (e.g., sensor readings, interesting events such as the detection of a trespasser) is stored only at the physical sensor nodes, i.e., nodes that are at level 0 of the hierarchical clustering (this is also the case in the original R-tree index structure). Thus, at some level of the hierarchical clustering peer-to-peer NN queries should start heading downwards to reach the data at level 0. However, in our peer-to-peer NN query execution model, we are able to limit an NN query only to the relevant nodes and can prevent it to propagate till up the root (i.e., topmost) partition: If the query can be answered locally, there is no need to go up till the root partition and query the root. By limiting the query to only the relevant nodes, we conserve energy. In particular, in our program the NN query is propagated upwards (i.e., query is sent to parent) only when the query point (x, y) is not within MBR of the current clusterhead. This way we are guaranteed at some level to reach a clusterhead whose MBR contains (x, y), from whereon the downwards journey of the NN query starts.

```
**Execution of NN($x, y$) at $v$**

if ($\exists i : 0 < i \leq l.v : (x, y) \in mbr.v(i)$
                    and $(x, y) \notin mbr.v(i-1)$)
then {
  initialize PartitionList with $c.v(i)$
  sort PartitionList by MINDIST;
  while (PartitionList is not empty) {
      if (top of PartitionList is at level 0)
      then { return nearest point NNC in level 0 $mbr$;}
      else { send query to child at top of PartitionList;
             if ( returned NNC is closer than NN)
             then { NN:=NNC; }
      } // end-if
      prune PartitionList with NN;
  } // end-while
}
else { send query to $p.v(l.v)$;}
```

Above is the peer-to-peer program at node v for execution of an NN query asking for the closest data to a location (x, y). (In our query execution model any node can insert an NN query to the system with respect to any arbitrary (x, y) coordinate.) Note that peer-to-peer execution of NN is very similar to its centralized counterpart except that when moving to a lower level cluster control is transferred to the node responsible for the respective cluster. In the next subsection we discuss the tradeoffs involved in this portion of the query execution.

4.3 Tradeoffs in peer-to-peer nearest neighbor queries

In peer-to-peer execution of NN queries our aim is to minimize the energy while minimizing the response time. However, these two objectives can usually be conflicting. The query processing program needs to be aware of the tradeoffs between minimal response time and minimal energy consumption when executing the queries. We explain these next.

Minimal response time. In order to achieve minimal response time, the while loop should be executed in parallel for all partitions in the partition list. That is, at each level each clusterhead sends the query to its children in the partition list in parallel. The clusterhead collects the replies returned by the nodes and selects the reply with the minimum distance as result of the nearest neighbor query.

Minimal energy consumption. In order to achieve minimal energy consumption, the while loop should be executed sequentially so as to prune maximum number of partitions/nodes in the partition list without actually having to query all the sensors covered by the partition list. That is, at each level each clusterhead sends the query to its children in the partition list in sequential order, and at some point in time only one of the children is executing the query. By following a similar proof structure of [2], it can easily be proved that minimum number of nodes will participate in the query processing, i.e., the program requires communication among only the necessary nodes.

Optimizing both the response time and energy consumption. To achieve this, we use a hybrid technique that blends the previous two techniques that separately optimizes the response time and energy consumption. In this method, the partition list is grouped into partition-groups and the while loop is executed in parallel for the partitions within the same partition-group but sequentially for the partition-groups in the partition list.

4.4 Extensions to other spatial queries

The discussion on nearest-neighbor queries can easily be extended to other spatial queries. An obvious extension is the *range query* where the data objects that fall into a query region is asked. Similar to the NN query, the query can be initiated from any node of the peer-tree and the processing needs to be developed in a distributed manner. Below we discuss some other nontrivial types of spatial queries and their potential applications in sensor networks.

Constrained nearest neighbor (CNN). CNN queries are recently formulated in the context of database systems [12]. This type of query is targeted towards users who are particularly interested in the nearest neighbor(s) in a region bounded by certain spatial conditions, rather than in searching for nearest neighbors in the entire data space. This can be seen as performing nearest neighbor and range queries in a single query.

In sensor networks, there are several cases where a user may enforce spatial constraints on a nearest neighbor search. An example of a CNN search over sensor networks is to ask "the nearest object, or objects, to the north-east of a given location (x, y)". The query result is the closest data point to the query point that satisfies the given constraint, i.e. to the north-east of the query point. CNN queries can be efficiently implemented on top of the previously described peer-tree implementation without changing the underlying structure.

Reverse nearest neighbor. Another recently proposed query type in spatial databases is *reverse nearest neighbor* (RNN) queries [21]. Unlike nearest neighbor queries, RNN queries find the set of objects that have the query point as the nearest neighbor. An RNN query can be particularly important for a sensor network environment, both for dynamically establishing the infrastructure of the sensor network and for running real-time spatial queries using RNN. For example, an RNN query can be used to find the best location to set up a new base station in order to minimize the length of the communication routes (and hence energy consumption) in the sensor network. Or similar to the NN query, one can ask all objects that have the query object as the nearest neighbor.

RNN queries can be efficiently implemented on top of the previously described peer-tree implementation with only a minor change on the structure. We need to store one more information, dnn, about each MBR, which is the maximum distance from the points contained in the subtree rooted at this MBR to their nearest neighbors [31]. By using this information, RNN queries can be effectively implemented using the same structure. Since the structure and the algorithm doesn't have major differences from the peer-to-peer NN algorithm, similar efficiency-energy tradeoffs apply also for this case.

Generalization for multi-dimensional data. Besides the location-based queries, the proposed framework can be utilized for other multi-dimensional queries over sensor networks. One such example could be environmental monitoring by running continuous queries asking temperature, etc. Each of the sensed attributes can be defined as a dimension in multi-dimensional space and the peer-tree can be constructed as a multi-dimensional index structure. The dynamic nature of the system should be carefully addressed while building such an index structure.

5 Related Work

In this section, we compare and contrast our work with related work on 1) database systems, 2) peer-to-peer systems, and 3) sensor networks.

Database systems. Nearest neighbor (NN) searching [4,18,19,28] is an important primitive operation in database systems and therefore has been extensively studied in the database community, especially in the context of CAD [3], multimedia [11], biomedical [22], and spatial databases [6]. The similarity metric of an NN query can be defined considering the underlying application: in spatial databases the commonly used distance is the geographical distance between objects. Many of the peer-to-peer and sensor network applications require spatial and/or similarity queries, and hence incorporation of NN queries to these applications is crucial. However, as we have discussed in the introduction, the solutions (e.g., indexing technologies) provided by the database systems are not readily applicable in the context of sensor networks and peer-to-peer systems due to the decentralized nature of these systems.

Peer-to-peer systems. Exact lookup operations are commonly used in the current peer-to-peer implementations, e.g., Napster, Gnutella, and Kazaa. Napster [26] was one of the first popular systems that allow data sharing through a wide area network of users. Napster utilized a centralized index where exact match queries can be performed. The central index has a directory of all data available in the system. Each peer in the system communicates with the central node. This simple approach is clearly not scalable, not suitable for dynamic systems, and prone to failures because of a central point of failure. Gnutella [13] follows a radically different approach where each peer keeps its own index. The local index has the local database information as well as the addresses of the neighbor peers. The former is used for lookup and the latter is used for routing purposes. In contrast to the centralized model, e.g., Napster, the distributed model, e.g., Gnutella, does not have a central point of failure. However, Gnutella does not use an intelligent indexing technique to process the queries, which obviously results in a high overhead in query processing and network maintenance. In contrast to these two models, i.e., centralized vs. distributed, Kazaa [20] follows a hybrid approach where supernodes are used inside the system to minimize the flood and the possibility of failures in the system. This idea is similar to the use of base stations in sensor networks.

Distributed hash tables (DHT) also have been successfully utilized in peer-to-peer query processing. For example, Chord [29] proposes a distributed hash lookup technique over a ring of peers. For routing purposes, each peer in the system maintains information about some of its neighbors. The neighbors are chosen in a way logarithmically increasing distance on the ring structure. By keeping some redundant information, the lookup can be performed even in the case of some node failures and re-joins. CAN [27] proposes to use a multi-dimensional space to index the data over the network. Each data object is mapped into a d-dimensional space, and stored in a peer that owns the corresponding part. The proposed index structure is similar to a grid structure that performs exact match queries. SkipNet [16] organizes data primarily by lexicographic key ordering (and only secondarily by DHT) to support controlled data placement, routing locality, and efficient range queries. And finally Gupta et al. [14] recently developed a peer-to-peer data sharing architecture for computing approximate answers for the complex queries by finding data ranges that are similar to the user query.

All of the indexing techniques discussed above have been proposed for exact match and 1-dimensional range queries for file sharing. Even though the idea of indexing on peer-to-peer systems has been explored in the context of exact match and 1-dimensional range queries, to the best of our knowledge our work is the first to explore the use of index structures for NN and spatial queries in peer-to-peer systems.

Since we have presented our peer-to-peer spatial query processing model in the context of sensor networks, our work focused also on energy conservation, e.g., lesser number of nodes are participating at the higher levels of indexing hierarchy. Due to the hierarchical nature of our indexing structure, the routing paths in our work scale logarithmically with respect to the number of nodes in the system. While adopting our work for the wired model, we can further improve the scalability of our routing paths by exploiting DHT functionality. More specifically, this can be achieved by inserting long links (i.e., remote or finger neighbors) at each node. These long links also improve the load balancing: traffic overhead would be approximately the same at all nodes regardless of the level of the node in the indexing hierarchy.

Sensor networks. Peer-to-peer query processing in sensor networks is a new concept. Traditionally, sensors are used as data gathering instruments, which continuously feed a central base station database. The queries are executed in this centralized base station database which continuously collates the data. For example, in TinyDB [23] queries are posed at a powered base station where they are flooded to sensors in the network. However, given the current trends (increase in numbers of sensors, together collecting gigabits of data, increase in processing power at sensors) it is not anymore feasible to use a centralized solution for querying the sensor networks. Therefore, there is a need for establishing an efficient access structure on sensor networks in order to contact only the relevant nodes for the execution of a query and hence achieve minimal energy consumption, minimal response time, and an accurate response. We achieve these goals with our peer-to-peer query processing model on top of a distributed index structure on wireless sensor networks.

For the construction of the peer-tree structure, we have presented a self-stabilizing clustering program that constructs a hierarchical partitioning of a sensor network based on the number of nodes contained at each cluster. We could have alternatively employed LOCI [25] clustering program, that uses the geometric radii of the clusters as the criteria for constructing a hierarchical partitioning of a sensor network. As analogous to $[n, N]$ (where $N \geq 2*n$) tolerance factor in our clustering program, LOCI constructs clusters with radius $[r, R]$ where $R \geq 2*r$. Both programs exploit the tolerance factor of the clusters for achieving local (in the sense that each node needs information only about nearby nodes) self-stabilization in the face of arbitrary transient faults.

6 Conclusion

In this paper, we have investigated the problem of efficient execution of spatial and similarity queries in sensor networks. To this end, we have presented a self-stabilizing peer-to-peer indexing structure: peer-tree. Using peer-tree, we have presented efficient methods for performing nearest neighbor queries, and discussed an extension of these methods for constrained nearest neighbor, and reverse nearest neighbor queries, as well as multi-dimensional data queries in sensor networks.

References

[1] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[2] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, pages 78–86, Tuscon, Arizona, June 1997.

[3] S. Berchtold and H.-P. Kriegel. S3: Similarity search in CAD database systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 564–567, Tuscon, Arizona, 1997.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is ”nearest neighbor” meaningful. In *Int. Conf. on Database Theory*, pages 217–225, Jerusalem, Israel, January 1999.

[5] N. Bulusu, J. Heidemann, and D. Estrin. Adaptive beacon placement. *In Proceedings of the 21st International Con-*

ference on Distributed Computing Systems, pages 489–498, 2001.

[6] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, A. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *IEEE Proceedings of the International Conference on Advances in Digital Libraries, ADL*, pages 70–81, Washington, D.C., 1997.

[7] Y. Choi, M. Gouda, M. C. Kim, and A. Arora. The mote connectivity protocol. Technical Report TR03-08, Department of Computer Sciences, The University of Texas at Austin, 2003.

[8] A. Cournier, A.K. Datta, F. Petit, and V. Villain. Self-stabilizing PIF algorithms in arbitrary networks. *International Conference on Distributed Computing Systems (ICDCS)*, pages 91–98, 2001.

[9] M. Demirbas, A. Arora, and M. Gouda. A pursuer-evader game for sensor networks. *Sixth Symposium on Self-Stabilizing Systems(SSS'03)*, pages 1–16, 2003.

[10] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11), 1974.

[11] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[12] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi. Constrained nearest neighbor queries. In *Proc. of the 7th International Symposium on Spatial and Temporal Databases (SSTD)*, pages 257–276, Los Angeles, CA, July 2001.

[13] Gnutella. http://www.gnutella.com.

[14] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.

[15] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[16] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems*, 2003.

[17] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS*, pages 93–104, 2000.

[18] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.

[19] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. of 4th Int. Symp. on Large Spatial Databases*, pages 83–95, Portland,ME, 1995.

[20] Kazaa. http://www.kazaa.com.

[21] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Dallas, USA, May 2000.

[22] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 215–226, Mumbai, India, 1996.

[23] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, June 2003. To appear.

[24] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. *ACM Int. Workshop on Wireless Sensor Networks and Applications*, September 2002.

[25] V. Mittal, M. Demirbas, and A. Arora. LOCI: Local clustering in large scale wireless networks. Technical Report OSU-CISRC-2/03-TR07, The Ohio State University, February 2003.

[26] Napster. http://www.napster.com.

[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[28] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, California, May 1995.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[30] Y.-M. Wang, W. Russell, A. Arora, J. Xu, and R. Jagannathan. Towards dependable home networking: An experience report. *International Conference on Dependable Systems and Networks*, 2000.

[31] C. Yang and K.-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *Proceedings of the 17th International Conference on Data Engineering*, Rome, Italy, 2001.

[32] H. Zhang and A. Arora. Gs^3: Scalable self-configuration and self-healing in wireless networks. In *21st ACM Symposium on Principles of Distributed Computing*, July 2002.

[33] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, U. C. Berkeley, April 2001.