Processing Proximity Queries in Sensor Networks

Yannis Kotidis*
Athens University of Economics and Business
kotidis@aueb.gr

Abstract

Sensor networks are often used to perform monitoring tasks, such as in animal or vehicle tracking and in surveillance of enemy forces in military applications. In this paper we introduce the concept of *proximity queries* that allow us to report interesting *events* that are observed by nodes in the network that are within certain distance of each other. An event is triggered when a user-programmable predicate is satisfied on a sensor node. We study the problem of computing proximity queries in sensor networks using existing communication protocols and then propose an efficient algorithm that can process multiple proximity queries, involving several different event types. Our solution utilizes a distributed routing index, maintained by the nodes in the network that is dynamically updated as new observations are obtained by the nodes. We present an extensive experimental study to show the benefits of our techniques under different scenarios. Our results demonstrate that our algorithms scale better and require orders of magnitude fewer messages compared to a straightforward computation of the queries.

1 Introduction

Sensor networks are used in a variety of monitoring tasks. In this paper we initiate a study of new decentralized algorithms for the detection of events that are observed by nodes within certain distance of each other, i.e. events that are reported by sensor nodes in spatial proximity. An event is triggered when a user-programmable predicate is satisfied on a sensor node. The definition allows different types of events, depending on the sensing capabilities of the nodes in the network and on the application. For instance, in an application where nodes are used for collecting meteorological data, an event may be defined for when the temperature readings on a sensor exceed a certain threshold. Another type of event, in the same application may be defined when temperature readings fall below another, lower value. When both events are detected, each by a different node, and these two nodes are in proximity, this may indicate an abrupt, abnormal shift in temperature in the terrain. In a military surveillance application, events may be used to detect the movement of friendly and enemy forces. Proximity alerts then may be used for the early warning of approaching enemy forces. In another application of wild animal tracking, we may want to raise an alert when a predator in spotted in an area occupied by a flock that we observe, assuming that the presence of each animal can be detected by the use of some RFID technology or by matching the sensory data to stored patterns in the node [10].

In this paper we provide a generalized solution to the problem of detecting proximity among interesting types of events such as those presented in the aforementioned applications. Given a set of predefined event types $\mathcal{E}$=$\{A, B, C, \ldots\}$ the users can register continuous *proximity queries* of the form Q=(X, Y, d) where X and Y are members of $\mathcal{E}$ and d is a proximity threshold. In turn, the sensors will inform a base station by raising a *proximity alert* when two events of type X and Y are detected by sensor nodes S_1 and S_2 that are within distance d of each other. A response could be a quadtuple $(S1, X, S2, Y)$ indicating that node $S1$ (resp. $S2$) has observed event X (resp. Y). In the general case, X and Y may be of the same type. Similarly, the definition of a query can be extended to include multiple event types.

If the proximity threshold d is zero, an alert is raised when both events are detected at the same node (S_1=S_2). This is a trivial case in that each node may process the query independently using only localized information (i.e. its own readings) without further knowledge on the observations of other nodes in the network. When d is greater than zero, the query requires that the sensor nodes collaborate in order to share their observations. In a straightforward implementation, when a node S observes either event X or event Y, this needs to be announced to all its neighbors that are within distance d. Thus, the number of messages that need to be communicated in response to a proximity query rises rapidly with the value of threshold d and becomes prohibitively expensive for larger, dense networks. Similar observations hold, if we decide to simply forward the events to a base station and compute each proximity query outside the network.

As has been well documented by past research, communication constitutes the biggest source of energy drain in sensor networks. Thus, we need to devise more efficient ways of processing a proximity query Q by the nodes in the network. In this paper we propose a solution that utilizes a distributed routing index for the communication of interesting types of events in the network. When node S observes an interesting event X, or hears an announcement by some of its neighbors, it utilizes this index to propagate the announcement to other parts of the network that have the highest likelihood of witnessing an event of type Y, assuming X and Y are both used in the same proximity query. By carefully tuning the routing information available in the nodes we can simultaneously process a large number of proximity queries without the need of flooding portions of the sensor network that do not contain qualifying events. Of course, an optimization that prunes messages may not compute all matching proximity events in the network. In the spirit of past work that exploit approx-

*The project is co-funded by the European Social Fund and National Resources (Ministry of Education) Pythagoras II - EPEAEK.

Proceedings of the 3rd International Workshop on Data Management for Sensor Networks (DMSN'06), Seoul, South Korea, 2006

imation [6, 7, 8, 12, 15] in order to reduce energy drain in sensor nodes, our algorithm will provide approximate answers to each proximity query. As will be demonstrated in our experimental evaluation, our algorithm is able to capture most proximity events in the network (with a median recall of 99%) and 100% precision, using a tiny fraction (that can be as low as 2%) of the messages that a straightforward implementation requires. We also discuss conditions under which the algorithm provably finds all answers to a proximity query, while still able to produce similar savings.

Our contributions are summarized as follows:

- We introduce the concept of proximity queries in sensor networks. Our definition captures a large number of interesting queries that may be used in a variety of monitoring applications in sensor networks.
- We propose the use of a distributed routing index for capturing the spatial distribution of interesting types of events in the sensor network. This routing index requires minimal resources at each node and is being updated dynamically, when the nodes collaborate to provide answers to proximity queries.
- We provide a detailed experimental evaluation where we study the effect of various parameters in the accuracy of our algorithms. Our results demonstrate that our techniques are very robust and can accurately process a variety of proximity queries, while substantially reducing the number of messages exchanged in the network.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we introduce the problem of proximity query answering, discuss several implementation issues and explore the benefits and drawbacks of a straightforward execution of the queries. In Section 4 we discuss in detail our techniques. Finally, in Section 5 we evaluate our techniques and in Section 6 we provide concluding remarks.

2 Related Work

Sensor networks consisted of wireless, battery-powered sensing devices, have introduced new challenges in data management and have spawn several recent proposals for embedded database systems, such as COUGAR [19] and TinyDB [13]. Most of the proposed techniques explore in-network processing to carefully synchronize the operation of the nodes [13] and utilize the multi-hop communication links to leverage the computation of expensive queries, such as those involving aggregation [4, 7, 15]. Continuous monitoring queries [6, 16] and distributed join algorithms [1] have also been considered. Alternative methods try to reduce the cost of data processing in sensor networks through probabilistic techniques [3], data modeling [8, 12] or through the use of decentralized algorithms [2, 11]. Our algorithms fall in the latter category. Application of existing methods for computing set-expressions in data streams [5] in the evaluation of proximity queries is an open research question due to the different settings and cost considerations.

Most of these fundamental techniques have been devised to support event-based monitoring applications. For example, in animal tracking, an event such as the presence of an animal can be determined by matching the sensor readings to stored patterns [10]. The authors of [18] propose an event detection mechanism based on matching the contour maps of in-network sensory data distributions. In [14], kernel-based techniques are used to detect abnormal behavior in sensor readings. In [9] the authors describe the implementation of a real system based on Mica2 motes for surveillance of moving vehicles.

3 Motivation

We consider the case of a proximity query $Q(X, Y, d)$ that requests the network to inform a base station whenever two events X and Y are observed by nodes $S1$ and $S2$ that are within distance d of each other. We note that the order of the events doesn't matter, i.e. $Q(X, Y, d)$ is equivalent to $Q(Y, X, d)$. It is also straightforward to generalize the definition of the query to include multiple event types such as $Q(A, B, C, \ldots, d)$ and modify the semantics to return *any pair* of events from the specified set that are in proximity.

In order to ease presentation and without affecting the applicability of our techniques to other configurations we will assume that the sensor nodes are placed in a two-dimensional $n \times n$ grid. Then, $S_{i,j}$ will denote the sensor node at location i,j in the grid, where $0 \leq i, j < n$. For this arrangement we will use the L_∞ norm to compute distances

$$dist_\infty(S_{i,j}, S_{k,l}) = max(|i-k|, |j-l|)$$

Using this metric, nodes $S_{0,0}$, $S_{1,0}$, $S_{2,0}$, $S_{2,1}$, $S_{2,2}$, $S_{1,2}$, $S_{0,2}$ and $S_{0,1}$ are all within distance one from sensor node $S_{1,1}$. We will further assume that a node can transmit a message to any of the nodes that are in adjacent locations in the grid. In the above example node $S_{1,1}$ can reach any of its 8 immediate neighbors. We note that other distance metrics, placements of nodes and transmission ranges are possible without affecting the generality of our techniques.

We assume that nodes are able to observe events drawn from a set $\mathcal{E}$=$\{A, B, C, \ldots\}$. It is not required that each node can detect (or compute) all event types. This will depend of the node's sensing capabilities and the application. When a proximity query $Q(X, Y, d)$ is registered, it is communicated to all nodes in the network using a flooding algorithm (see for example [4, 7]). In turn, the nodes should inform the base station whenever the two events X and Y are detected by nodes $S_{i,j}$ and $S_{k,l}$ in proximity of each other. There is not a strict requirement that one of $S_{i,j}$ and $S_{k,l}$ should inform the base station. This information can be, for instance, computed and communicated to the base station by any node in the network, for instance a node somewhere in between nodes $S_{i,j}$ and $S_{k,l}$ that becomes aware of both events.

Before discussing our techniques for computing the query, we will first present a straightforward algorithm. The discussion will also help highlight some of the characteristics of sensor nodes and motivate our techniques. We will draw our examples using the characteristics of the Berkeley Mica2 motes radios. A mote can communicate with its neighboring nodes by sending either unicast or broadcast messages. At an abstract level, the two methods differ in that broadcast messages use a predefined broadcast

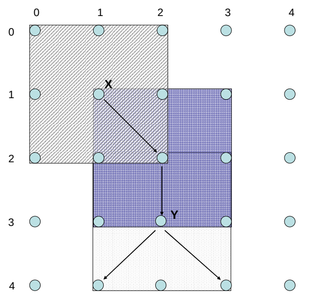

Figure 1: Proximity Query Processing

address (TOS_BCAST_ADDRS), while unicast messages encapsulate the local address of the recipient (roughly analogous to a node-id) in their header. When unicast communication is used, other motes in the neighborhood that have their radios turned-on will be *idle-listening* to a unicast message that is not directed towards them. On the Mica2 mote, the ratios for radio power draw during idle-listening, receiving of a message and transmission are 1:1:1.41 at 433MHz with RF signal power of 1mW in transmission mode [20]. Similar observations hold in networks consisted of other types of nodes [20]. Thus, idle listening is a dominant factor of energy drain in such networks making broadcast a attractive communication paradigm.

Based on the discussion above we can now derive a simple algorithm for computing proximity queries in sensor networks. Let's assume that sensor node $S_{i,j}$ observes an event of type X (resp. Y). The node first checks whether an event of type Y (resp. X) is also present and if so, it notifies the base station.[1] Then, assuming $d > 0$, sensor node $S_{i,j}$ broadcasts a message to its neighborhood including the type of the event and its location (for instance $(X, S_{i,j})$). Each node $S_{k,l}$ that receives this message makes similar checks. It firsts sees whether a locally observed event (e.g. Y) matches and should be reported to the base station. Then, if $dist(S_{i,j}, S_{k,l}) < d$ it broadcasts $(X, S_{i,j})$ to its neighborhood. During this process a node $S_{k,l}$ may receive the same message multiple times from different neighbors. In order to reduce unnecessary traffic the node only reacts to the first announcement it receives for the event and ignores subsequent messages. This is possible since the event X and the originator of the event (node $S_{i,j}$) are both included in the message. Cascading terminates at nodes that are in distance equal to the proximity threshold d from node $S_{i,j}$ that initiated the process.

4 Our Solution

While the straightforward algorithm for computing a proximity query is very simple to implement, it is rather expensive in terms of the number of messages exchanged in the network. As has been already discussed, each message sent by a node in the network drains energy not only from the sender but also from any other node that is listening on the channel. Thus, we need to derive ways to reduce unnecessary traffic during computation of a proximity query.

Consider for example the scenario depicted in Figure 1. Node $S_{1,1}$ has just observed event X (that may indicate the presence of say enemy forces) and assume that continuous proximity query $Q(X, Y, 3)$ has been previously registered in the network. In the straightforward algorithm, node $S_{1,1}$ will start a flooding process that will announce pair $(X, S_{1,1})$ to all nodes in the 5×5 area depicted in the Figure. The number of messages transmitted in that case will be equal to 16 and the number of messages received equal to 105.

In the same Figure we also observe that an event of type Y was previously detected by node $S_{3,2}$. Thus, both events need to be reported in response to query Q. If node $S_{1,1}$ had a way of knowing the position of event Y at grid location (3,2) it would transmit a message indicating the presence of event X at its location to its south-east neighbor $S_{2,2}$, which would in turn pass this information to node $S_{3,2}$ and have the latter confirm the proximity alert by responding to the base station with tuple $(X, S_{1,1}, Y, S_{3,2})$.

An open question is how this information should be passed from node to node, using unicast or broadcast? As long as the cost of idle-listening is comparable to the cost of receiving a message, the unicast approach provides no real benefits. An additional advantage of broadcasting is that additional nodes in the neighborhood will be listening to the announcements and can use this information for tuning their local indices (discussed below) at no additional cost. We will assume at this point that broadcast messages will be used in the process of informing adjacent nodes for the presence of an event and revisit this issue later in this section.

Using these assumptions, node $S_{1,1}$ that first observed event X sends a broadcast message to its adjacent nodes indicating the location of the event $(X, S_{1,1})$ and that its south-east neighbor, node $S_{2,2}$ should carry over propagating the event to the rest of the network. All other nodes in its neighborhood can hear the announcement, and can use the information to update locally stored information but do not further react to it. Similarly, node $S_{2,2}$ transmits a second message that is heard by all nodes in the second shaded area and includes in the message the location of the event $(X, S_{1,1})$ as well as the next "hop", node $S_{3,2}$. Node $S_{3,2}$ that has observed event Y reacts by informing the base station of the new result to the proximity query (the required messaging is not depicted in the figure). The distance of node $S_{3,2}$ from $S_{1,1}$ is 2 that is less that the proximity threshold (3) and the node further propagates the announcement, indicating this time that both nodes $S_{4,1}$ and $S_{4,3}$ should be the next "hops". Notice that, in general, a node may indicate that more than one nodes should further propagate the announcement of an event, in search for results to the proximity query. Nodes $S_{4,1}$ and $S_{4,3}$ do not need to propagate the announcement further, because the proximity threshold has been reached. In all, just three messages have been transmitted and the number of nodes that received them was 24, a significant reduction compared to the straightforward algorithm.

In Algorithm 1 we sketch an implementation of the accept() subroutine at sensor node $S_{k,l}$ for processing an incoming message during the course of the algorithm. There are fours major tasks in this process. In lines 3–5, the node first checks whether it has already responded to an announcement for the same event, and if so it ignores the

[1]This can be accomplished, for example, by using the inverse routing tree computed during query propagation [4, 13].

Algorithm 1 `Accept` Subroutine

```
Require: $(X, S_{i,j}, ListNextHops)$
1: {$X$ is the event type reported by node $S_{i,j}$}
2: {$ListNextHops$ is a list of nodes that should continue further the messaging process}
3: if HasSeen($X, S_{i,j}$) then
4:     return {Have already processed an announcement for this event}
5: end if
6: {$S_{k,l}$ is the current node}
7: if Exists Local Event $Y$ and query $Q(X,Y,d)$ and dist($S_{i,j}, S_{k,l}$) $\leq d$ then
8:     InformBaseStation($X, S_{i,j}, Y, S_{k,l}$)
9: end if
10: UpdateLocalIndices($X, S_{i,j}$)
11: {$d$ is the largest proximity threshold for any registered query $Q(X, Y', d')$}
12: if $S_{k,l}$ in ListNextHops and $dist(S_{i,j}, S_{k,l}) < d$ then
13:     MyNextHops=PickHops(X){Pick next hops for event $X$}
14:     Broadcast($X, S_{i,j}, MyNextHops$)
15: end if
```

message. Then, lines 6–8, if a local event Y has been detected, for which a query $Q(X,Y,d)$ is also registered and the distance of node $S_{k,l}$ is less or equal to d from the originator of event X (node $S_{i,j}$), the node informs the base station of the two matching events. In line 10, a call to function UpdateLocalIndices() is made, so that the routing information can be updated. This process is discussed in more detail below. Finally, in lines 11–15, if the node belongs to the list ListNextHops, encapsulated in the message, it is responsible for propagating the announcement further. For that, it first computes, as will be explained, a new list of next hops for the message and finally broadcasts this list, along with the original event.

It is worth noting that nodes that are not in the NextHopList are still able to check for local matching events and also to update their routing information. This is another advantage of using broadcast instead of unicast communication. Intuitively, there is a band of nodes around the routing path that overhears the communication and is able to either contribute to the query or tune its indices for future queries and events. A implementation based on unicast (or multicast) messaging would not be able to support this. Effectively, in our algorithm, we simulate a multicast protocol, embedded over a broadcast communication channel in order to benefit from both worlds.

4.1 Routing Indices

Up to this point we have assumed that the nodes have the ability to properly route the announcement of an event to one or more of their neighboring nodes in the topology. This is accomplished by the use of a local routing index. At each node s, the index is instantiated as a list of triplets $(nb, e, w_{nb,e})$, where nb is a neighbor of node s, e is an event (from set $\mathcal{E}$), and $w_{nb,e}$ is a weight associated with the link between s and nb. The size of the full index is equal to the number of event-types times the number of neighbors of node s. Thus, in our running example, it is bounded by $8 \times |\mathcal{E}|$. Nevertheless, not all entries are required for routing a message as is explained below.

Routing of Messages. Function PickHops(X) is used in Line 13 of the Algorithm for selecting the next hops for routing a message that contains an event of type X. This function returns the top num_hops entries from the index using the following process. For each neighbor nb of node s we compute the compound weight $W_{nb,X}$ for routing a message regarding X through nb as the maximum of all weights $w_{nb,e}$ for all events e that are used in a proximity query of the form $Q(X, e, d)$. Thus,

$$W_{nb,X} = max_{e:\exists Q(X,e,d)}(w_{nb,e})$$

Given these aggregate weights, function PickHops(X) returns the top-num_hops neighbors of s, ranked in descended order of their values.[2] Parameter num_hops is a built-in value and we will study its effect in the experimental evaluation of Section 5.

NextHop (n)	Event(e)	Weight ($w_{n,e}$)
$S_{3,1}$	V	3
$S_{4,1}$	Y	3
$S_{4,1}$	W	5
$S_{4,3}$	X	5
$S_{4,3}$	Y	4

Table 1: Sample routing information at node $S_{3,2}$

In Table 1 we show a snapshot of the routing index for node $S_{3,2}$ used in the example of Figure 1. Assume that the node has just received the announcement for event $(X, S_{1,1})$ and that the following two proximity queries are registered: $Q_1(X, Y, 3)$ and $Q_2(X, W, 3)$. In this example the ranked list of weights will be (we note that $W_{S_{3,1},X}$ is not defined in this example): (i) $W_{S_{4,1},X}$=5 and (ii) $W_{S_{4,3},X}$=4.

Computing/Updating the Index. An open question, is how the weights $w_{n,e}$ associated with each link and event-type will be determined. The larger the weight that is associated with a link, the more likely is to route a message using this link. Arguably, there are many ways to determine the weights and different choices may work better for certain applications. In our current implementation we target applications where events represent moving objects such as in vehicle and animal tracking, in military surveillance etc. In such applications, events do not randomly appear and disappear in the monitored area. They rather move along predefined or unknown paths in the terrain. Thus, when a node s reports an event of type X (for instance the presence of an animal) at time t, our best guess for the location of the object at time t+1 will be the neighborhood of s. Therefore, when node $S_{k,l}$ hears about event $(X, S_{i,j})$ by its neighbor n, it assigns a value for weight $w_{n,X}$ equal to the current timestamp t. Notice that in this scheme t can be simply a local variable that will be incremented by each call to function UpdateLocalIndices(). Intuitively, this process generates a reverse routing tree that is rooted at node $S_{i,j}$ that originally announced the event. In our running example, node $S_{3,2}$ will insert a new row with values $(S_{2,2}, X, t)$ when it receives the announcement from node $S_{2,2}$. In turn this entry will be used when an event of type Y or W is announced in the neighborhood of $S_{3,2}$ and will properly route the announcement back to node $S_{1,1}$ to produce the proximity alert, unless newer observations of the event X, in other locations, get higher priority.

[2]When computing this list, we exclude the node, which sent the message that node s is currently processing.

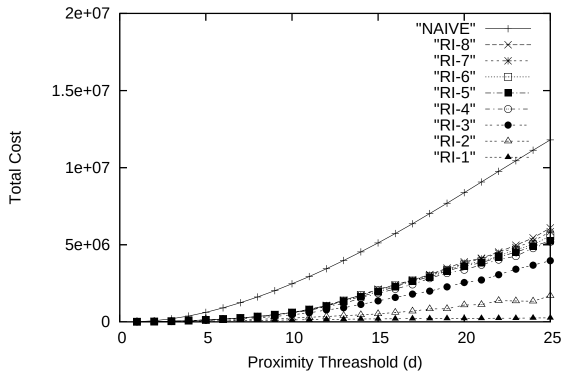

Figure 2: Cost of a query, varying d

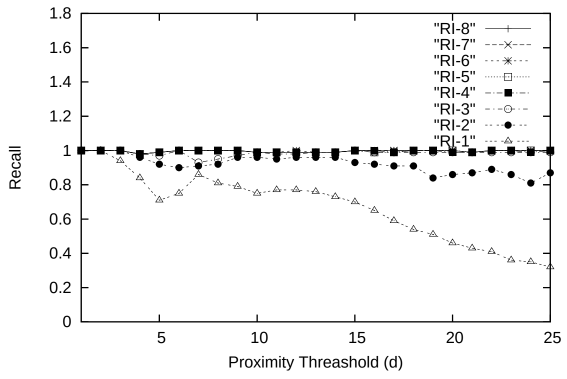

Figure 3: Recall of the RI algorithm

Of course, one can better tune the routing index, when more information is available on the movement of each event X in the area covered by the sensors. For instance, if the nodes, in addition to the location of an object, can also determine its speed and direction, we can easily modify function PickNextHops() to compute the most likely position of the event at time present t, given a past observation. When nodes with advanced sensing and computation capabilities are available, we can take this process further and design a more effective routing index using techniques such as those described in [17] that can capture a variety of unknown motion patterns. Such extensions do not change the core logic of our algorithm and are left for future work, due to lack of space.

5 Experiments

In this section we study the performance of our algorithm and alternative implementations, using a simulator that we developed. For the first experiment we modeled the network using a 50×50 grid of nodes. Given our preference for surveillance applications, as discussed in Section 4, we modeled 10 objects, 5 of type "X" and 5 of type "Y" that were free to move in the monitored space making random walks. The walks were modeled so that when object X is at grid location (i, j) at time t, its next position at t+1 was randomly chosen from one of the adjacent cells of (i, j). In Figure 2 we compute the communication cost of computing a single continuous proximity query $Q(X, Y, d)$ for 2,000 epochs, varying the proximity threshold d. The total cost, following the suggestions in [20] was computed as cost=1.41$\times$(messages sent)+(messages received)+(messages idle listening). The costs also account for the messages required to route a matching tuple back to the base station, which in this run was placed in the center of the area (results for other placements were analogous and are omitted due to lack of space).

The line labeled NAIVE depicts the performance of the flooding algorithm discussed in Section 3. Results of our algorithm are shown as RI-k (standing for Routing Index), where k is the value of parameter num_hops, see Section 4. In all cases we used the first 10% of the epochs to train the indices using the NAIVE method and started counting after that point. As expected, a smaller value of parameter num_hops results in fewer messages in the network. Compared with the naive execution, we note that even a value of 8 (=max number of neighbors) for num_hops results in a reduction of the total cost by a factor of up to 5, depending on the proximity threshold. This is because, our algorithm does not route messages towards neighbors that have never announced a matching event before. Performance of RI-8, can be improved further by expiring old observations from the routing index. We defer details to the full version of this paper. For num_hops=1, the savings are even higher, by a factor of up to 45, increasing again with the selected value of d.

A potential drawback of using a small value for num_hops is that, under certain conditions, we might not be able to detect proximity events, because of the pruning of messages towards some of the neighbors of a node. In Figure 3 we plot the *recall* (=percentage of answers returned) of the algorithm varying d and for the different values of num_hops. We note that precision is a perfect 100% as we never return spurious tuples. The graphs show that a value of 3 or larger for num_hops returns at least 93% of the answers, while the median value of recall is 99%.

In the case of RI-8 one can show that the algorithm provably finds all answers to a proximity query, under the assumptions that (i) objects do not make abrupt jumps (i.e. when X is observed by node s at time t, it will be within the area covered by s and its neighbors at time t+1), (ii) the necessary communication among nodes within an area of "radius" d around the event can be carried out between t and t+1 without loss of messages (iii) the initial weights of the indices point towards the true location of the events at time t=0.

In Figure 4 we show the effect of scaling the size of the monitored area by increasing the grid from 10×10 up to 400×400. We used a single proximity query $Q(X, Y, 5)$. In the graph we introduce an additional implementation for the computation of the queries that simply unicasts the observation of an event to the base station (using a minimum cost path), which in turn computes the proximity events. This algorithm is denoted as EXTERNAL in the Figure, indicating that processing of the events is done outside the network and has the best performance for the two smaller grid sizes. This is explained as, for instance in the case of the 10×10 grid with a proximity threshold d=5 for the query, we essentially compute an all pairs cartesian product of the events. Thus, no filtering can be accomplished via in-network processing of the events, as in the case of

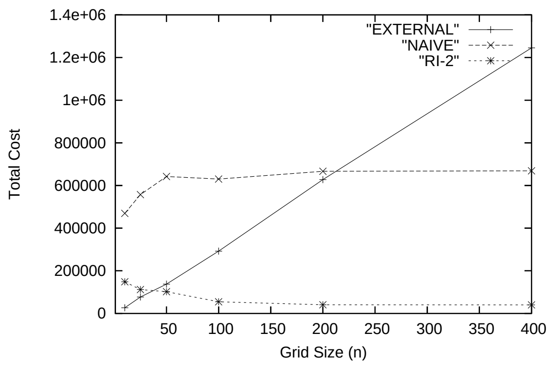

Figure 4: Varying the size of the monitored area

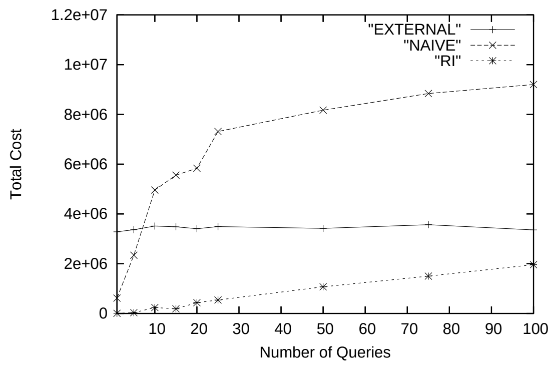

Figure 5: Varying the number of concurrent queries

the NAIVE or RI algorithms. Of course, with larger grid sizes, performance of the EXTERNAL method quickly deteriorates. Because of the selection of the distance metric (L_∞) the cost of EXTERNAL is asymptotically linear with n. As expected, the cost of NAIVE is bounded by d^2 while RI becomes even more effective in larger grid sizes, as the sparsity of the space helps prune more messages.

In Figure 5 we depict the cost of evaluating multiple concurrent queries during 12000 epochs for an 100×100 grid. The number of event types was 26, we used 5 event instances of each type and we varied the number of queries from 1 up to 100. Each query was randomly chosen to select two of the 26 event types and the proximity threshold was 5. As expected, the cost of EXTERNAL is unaffected by the number of queries, as it depends on the number of events and the size of the grid, which are constant in this setup. The cost of NAIVE increases rapidly, up to the point where is bounded by the size of the network (since a node transmits at most 1 message). The RI algorithm had the best performance (num_hops was 2 in this example).

6 Conclusions

In this paper we introduced proximity queries as a means of detecting interesting events that are observed by nodes in the network that are within certain distance of each other. Proximity queries can be used to set up alerts or even to reduce the cost of collecting continuous measurements from the nodes, since they allow us to define which events are interesting based on the observations at several nodes. We investigated the issues of computing proximity queries in networks consisted of battery-powered wireless nodes and proposed an efficient distributed algorithm that utilizes routing information computed accordingly at each node in the network. Our results demonstrate that our techniques are very effective and provide substantial savings compared to straightforward executions of the queries using in-network processing or to algorithms that relay the events to a base station for further processing.

There are many interesting extensions to our techniques that require further consideration. For example, when the nodes have the ability to provide more information about the events such as their speed, direction or even their trajectories, we should be able to exploit this extra information to achieve even higher benefits. Similarly, one should be able to also define temporal proximity thresholds, in addition to spatial proximity considered in this work. We plan to investigate such extensions in the future.

References

[1] D.J. Abadi, S. Madden, and W. Lindenr. REED: Robust, Efficient Filtering and Event Detection in Sensor Networks. In *VLDB*, 2005.

[2] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. Technical report, Stanford, 2003.

[3] R. Cheng and S. Prabhakar. Managing Uncertainty in Sensor Databases. *SIGMOD Record*, 32(4):41–46, 2003.

[4] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate Aggregation Techniques for Sensor Databases. In *ICDE*, 2004.

[5] A. Das, S. Ganguly, M. Garofalakis, and R. Rastogi. Distributed Set-Expression Cardinality Estimation. In *VLDB*, 2004.

[6] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *ACM SIGMOD*, 2004.

[7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *EDBT*, 2004.

[8] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB*, 2004.

[9] T. He, S. Krishnamurthy, J. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. An Energy-Efficient Surveillance System Using Wireless Sensor Networks. In *MobiSys*, 2004.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *MOBICOM*, 2000.

[11] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *FOCS*, 2003.

[12] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *ICDE*, 2005.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

[14] T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Distributed Deviation Detection in Sensor Networks. *SIGMOD Rec.*, 32(4), 2003.

[15] A. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB Journal*, 2004.

[16] A. Silberstein, R. Braynard, and J. Yang. Constraint Chaining: On EnergyEfficient Continuous Monitoring in Sensor Networks. In *SIGMOD*, 2006.

[17] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In *SIGMOD*, 2004.

[18] W. Xue, Q. Luo, L. Chen, and Y. Liu. Contour Map Matching for Event Detection in Sensor Networks. In *SIGMOD*, 2006.

[19] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.

[20] W. Ye and J. Heidermann. Medium Access Control in Wireless Sensor Networks. Technical report, USC/ISI, 2003.