Detecting Proximity Events in Sensor Networks

Antonios Deligiannakis[a], Yannis Kotidis[*,b]

[a] *Technical University of Crete*
[b] *Athens University of Economics and Business*

Abstract

Sensor networks are often used to perform monitoring tasks, such as animal and vehicle tracking, or the surveillance of enemy forces in military applications. In this paper we introduce the concept of *proximity queries*, which allow us to report interesting *events*, observed by nodes in the network that lie within a certain distance from 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 and propose several alternative techniques that differ in the number of messages exchanged by the nodes and the quality of the returned answers. Our solutions utilize a distributed routing index, maintained by the nodes in the network, that is dynamically updated as new observations are obtained by the nodes. This distributed index allows us to efficiently process multiple proximity queries involving several different event types within a fraction of the cost that a straightforward evaluation requires. We present an extensive experimental study to show the benefits of our techniques under different scenarios using both synthetic and real data sets. Our results demonstrate that our algorithms scale better and require significantly fewer messages compared to a straightforward execution of the queries.

1. Introduction

Sensor networks, consisting of primitive wireless devices that are able to observe their environment and perform simple computational tasks, are becoming a popular paradigm in monitoring applications. While we are still years away from the smart dust vision, there is a consensus that our future will incorporate a plethora of sensing devices that will participate and help us in our daily activities. Such devices are expected to be rather limited in terms of storage, processing and communication capabilities, but will be able to achieve complex tasks by sheer numbers.

While we are still struggling to understand the technical difficulties of implementing and managing such networks, it is becoming increasingly clear that

*Corresponding author
Email addresses: adeli@softnet.tuc.gr (Antonios Deligiannakis), kotidids@aueb.gr (Yannis Kotidis)

we need to shift our focus from protocols that look at a single sensor at a time, to methods that are able to correlate multiple sensor readings and reason from them [18, 35, 55]. Such approaches will also help us cope with unexpected sensor failures and dirty data that are unavoidable when we operate on cheap, unreliable hardware [30]. In this paper we make a small step towards this direction by exploring techniques that allow us to correlate readings from neighboring sensors in monitoring applications. In particular, we investigate a new type of queries termed "proximity queries" that allow us to correlate interesting observations made by nodes in spatial proximity [36]. Monitoring applications trigger *events* that assert the evaluation of a user-defined Boolean predicate on a sensor. 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 to collect meteorological data, an event may be associated with a sensor's temperature readings exceeding 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 is 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 sensory data to stored patterns in the node [29].

In this paper we explore bandwidth-efficient algorithms for detecting proximity amongst interesting types of events such as those presented in the aforementioned applications. Our techniques are designed to handle *moving events*, i.e. events that do not randomly appear and disappear in the monitored area, but rather move along predefined or unknown paths in the terrain. The main technical challenge arises from the need to correlate observations on different sensors without communicating information about all events in the network, as this would quickly drain the nodes. We provide a comprehensive study of alternative methods to detect proximity events, including straightforward implementations based on localized flooding, or on external processing algorithms that seek to collect and process all sensor observations at a single node. We further discuss methods that intelligently route events through the nodes in the network in search of matching pairs, based on local indexing structures termed routing indices. We explore both deterministic and probabilistic routing schemes and study their trade-offs. Our findings indicate that the use of routing indices provides substantial savings, in terms of communication cost, when processing proximity queries. These savings, however, come at a small cost, in that the algorithms may, in certain situations, miss pairs of matching events. However, as it will be demonstrated in our experimental evaluation, many of these algorithms are able to capture most proximity events in the network (some with a median recall near 99%) and 100% precision, using a small fraction of the

messages that an exhaustive (i.e., centralized) implementation requires.

Our work builds upon the preliminary report we presented in [36]. In this paper we have modified the indexing structures proposed in [36] to allow us to filter events that cannot potentially join with a newly observed event. Moreover, the new algorithms that we introduce in this paper do not require a training process anymore due to the addition of a new step that handles situations when no local information on other events is available. Furthermore, we introduce and investigate alternative methods of propagating the announcements of events and also consider the case when nodes monitor moving objects and discuss the necessary alterations to our indexing scheme.

Our contributions are summarized as follows:

- We present the concept of proximity queries in sensor networks. Their definition captures a large number of interesting queries that may be used in a variety of monitoring applications.
- We propose the use of a distributed routing index for capturing the spatial distribution of interesting types of events in the sensor network. This index requires minimal resources at each node and is updated dynamically, when the nodes collaborate to provide answers to proximity queries. We consider both deterministic and probabilistic routing protocols based on the values maintained in the index, as well as alternative routing methods.
- We provide a detailed experimental evaluation where we study the effect of various parameters in the recall and the cost of our algorithms in terms of the number of transmitted messages. 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. These benefits increase as the size of the network increases.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we formally introduce the notion of proximity queries. In Section 4 we discuss several implementation issues and explore the benefits and drawbacks of a straightforward execution of the queries. In Section 5 we discuss our techniques in detail. Extensions of our basic techniques are presented in Section 6. Finally, in Section 7 we present an experimental evaluation of our proposed techniques, while in Section 8 we provide concluding remarks.

2. Related Work

Sensor networks consisting 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 [58] and TinyDB [40]. Most of the proposed techniques explore in-network processing to carefully synchronize the operation of the nodes [40, 62] and utilize the multihop communication links to leverage the computation of expensive queries, such

as those involving aggregation [12, 16, 23, 28, 46]. Continuous monitoring queries [15, 48], top-k [61] and distributed join algorithms [2] have also been considered. Alternative methods try to reduce the cost of data processing in sensor networks through probabilistic techniques [11], data modeling [18, 35] or through the use of decentralized algorithms [6, 33]. Our algorithms fall in the latter category. Application of existing methods for computing set-expressions in data streams [14] in the evaluation of proximity queries is an open research question due to the different settings and cost considerations. The networking aspects of wireless sensor networks have been extensively studied [4, 8, 9, 10, 24, 50, 60]. For example in [53] the authors discuss broadcasting and multicasting schemes that can provide the underlying communication primitives for our framework.

Random-walk techniques have been utilized for event discovery. For example, in [5] the authors utilize a random walk pattern for visiting sensor nodes until a sufficiently large percentage of sensor nodes have been visited. Techniques for biasing, at each step, the walk to unvisited neighboring nodes are also discussed. In [43] the authors consider the problem of target discovery when the target is stored at k different locations using a random walk technique. Using this technique, the random walk alternates its direction (moving either away from, or towards to, the sink node) until it locates one of the k targets. This is a slightly different problem than the one that we consider, since we are interested in locating *pairs* of matching events that occur within a maximum distance from each other. However, given an event announcement, a random walker can be easily modified to search for matching events instead. Please note that if the matching events are random (contrary to the applications considered in this paper, which involve moving events), then random walk techniques may be the appropriate choice. We explore simple adaptations to the random walk scheme in our experimental evaluation.

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 [29]. The work in [48, 49] discusses spatio-temporal suppression techniques for reducing the cost of monitoring queries in sensor networks. The authors of [55] propose an event detection mechanism based on matching the contour maps of in-network sensory data distributions. The work in [37, 56] discusses techniques that assert the coverage of the monitored area with minimum energy cost. In [42], kernel-based techniques are used to detect abnormal behavior in sensor readings. Detection of outlier observations by the nodes in a decentralized manner has been discussed in [17, 25]. In [27] the authors describe the implementation of a real system based on Mica2 motes for the surveillance of moving vehicles.

The use of some of our techniques, like the threaded algorithms TRI-k and $PTRI$-k share some common characteristics with techniques [39, 52] developed for Peer-to-Peer (P2P) applications. Local routing indices have already been proposed as an efficient method of locating content in P2P systems [13, 19, 57]. The idea of consulting a local routing table in order to compute the next hop to route a message is fundamental in the Border Gateway Protocol (BGP) [45],

Internet's global routing protocol. BGP routing differs from proximity search mainly in that the destination of a message (packet) in embedded, while this destination needs to be discovered in our framework. Moreover, the evaluation of proximity queries in sensor networks possesses characteristics that differ significantly from P2P searches. P2P networks searches initiated by a node often utilize feedback information (such as if the query succeeded or not) and adjust the weights in all the nodes that participated in the search accordingly [52]. This update process based on feedback information is unrealistic in a sensor network due to the overhead that it will impose on other nodes that overhear the exchanged messages. As an example, assuming that this feedback process is triggered when a search initiated by S_{src} fails, and assume that for some nodes this search failure occurs quite often simply because no matching events exist. In this case we are faced with a situation where a lot of messages are wasted when no matching events are in the neighborhood. If we decide to update the weights when searches succeed, then we are facing the same problem in the other extreme, when a lot of joins occur. Another important difference is the symmetry exhibited in the evaluation of proximity queries. Whenever a node S_{src} observes an event X, it initiates a search for matching events. However, at the same time the nodes that have observed these matching events initiate searches of their own for nodes that observed the X event. Thus, when a search from S_{src} successfully discovers matching events, the adjustment of the routing indices in the nodes belonging to the path of this search allows for these matching nodes to also discover S_{src} in the following epochs. This reverse step becomes easier because multiple nodes overhear these messages and update their routing indices at the same time. Thus, the use of a costly feedback mechanism, as in the case of P2P networks, is not required.

A similar approach (i.e., no feedback mechanism) was also followed in [7], where a rumor-based approach was proposed for locating events in a sensor network. The proposed technique utilizes agents that upon the detection of an event traverse the network propagating information about the event. To achieve this, an agent carries a list of all events that is has encountered and upon arriving at a node, it synchronizes this list with the node's local list. If we make the rough analogy that these lists correspond to the routing indices that we propose, there is a key difference between the works of our techniques and the proposal of [7]. In our approach, the routing indices are maintained locally and are never transmitted in the network. These indices are updated "incrementally" upon the announcement of an event. In contrast, the work of [7] utilizes an agent that traverses the network carrying full information about the events it has heard of from all locations it has visited. Since the amount of information is increased at every node the agent encounters, after several hops the generated lists can be prohibitively large, requiring multiple message exchanges between the nodes. As will be explained, in our technique, an announcement of an event is a short triplet containing information about the location, the type and the time of the event, and can be easily fitted withing a single message. Furthermore, information exchanges that require multiple messages are shown to increase the energy drain of the nodes, because of the increased probability of collisions that

result in retransmissions [25]. A final difference between [7] and our proposal is that it assumes synchronous stationary events (all events happen simultaneously at fixed locations). Thus, the technique cannot be applied to the problem that we consider, where events may be observed at any time during the operations of the algorithm, and, more importantly, these events may freely move inside the network.

In [31] the authors present Greedy Perimeter Stateless Routing (GPSR). GPSR allows point-to-point communication of distant wireless nodes using the positions of the nodes and a packet's destination to make packet forwarding decisions. Similar to our technique, GPSR makes greedy forwarding decisions using only information about a node's immediate neighbors in the network topology. However, in our context we do not seek to route messages towards a stationary destination; we assume that nodes monitor events with no specific knowledge of the process that generates these events and their dynamics. Thus, we are facing the challenging task of training the nodes to react quickly and try to route messages towards locations where we have greater chances of finding matching events. An important aspect of out framework is that, while routing announcements of events in the network, at the same time we use these announcements to train nodes in a path towards the destination point(s) so that they themselves can maintain an updated view of the dynamics in the network. This training process is, in our framework, fully integrated in the routing scheme we employ while announcing the events.

Data centric techniques [3, 21, 22, 29, 44, 47, 54] are built on the premise that data is more important than the node that gathers them. In the data-centric approach, relevant data is identified by its name at nodes within the sensornet so that all data with the same name (e.g., event-types in our context) will be stored at the same location (not necessarily the one that originally gathered the data). Data-centric storage architectures have been extensively used in support of range or nearest-neighbor queries in wireless sensor networks [20, 26, 38]. Unfortunately, data-centric storage will not work well in our problem where the number of event types (not event instances) is rather limited. This will result in a few nodes receiving most of the announcements. This limitation is exemplified when multiple proximity queries with overlapping sets of events are registered in parallel. Furthermore, the data-centric approach in our problem will demonstrate poor performance when a lot of events are reported but only a few proximity matches exist in the network.

3. Problem Formulation

We assume that the nodes are able to observe events[1] drawn from a set $\mathcal{E}=\{A, B, C, \ldots\}$. The monitored events may correspond to arbitrary detected conditions. For instance, in a weather monitoring application, event A may indicate that the temperature readings have fallen below a predefined threshold.

[1] We will use the terms "events" and "event types" interchangeably hereafter.

Symbol	Description
d	The proximity threshold of a query.
d_{eff}	The effective threshold of a query.
S_{src}	A node that observed the processed event.
S_{cur}	The sensor node whose operation we are currently describing.
S_{nb}	A neighboring node of the current sensor
NH_{cur}	The next hop list of S_{cur}
$Q(\{X, Y\}, d, d_{eff})$	A proximity query satisfied if events X and Y are observed by nodes within a proximity threshold d. The query search may be propagated only by nodes within d_{eff} from S_{src}

Table 1: Notation used in this paper.

In a surveillance application an event may indicate the identification, possibly through the execution of a complex face-recognition algorithm, of a person near the sensor. It is not required that each node can detect (or compute) all events, as this depends on the node's sensing capabilities and the application at hand. We emphasize here that in an application there can be numerous concurrent *instances* of an event observed at the same or different nodes. For example, in a surveillance application multiple nodes may detect vehicles (or potentially the same vehicle), or a single node may detect multiple types of vehicles, and each such type may correspond to a different event from the set $\mathcal{E}$.

The notation used throughout this paper is summarized in Table 1. Additional definitions and comments about these symbols are introduced in appropriate areas of the text.

We assume that a user or the application communicates with the wireless sensor nodes using a special-purpose node called the *base station*. The base station is often assumed to have increased processing and communication capabilities than the rest of the network. A proximity query is initiated by the base station and is denoted as $Q(\mathcal{QS}, d, d_{eff})$, where

- $\mathcal{QS} \subseteq \mathcal{E}$ is a non-empty subset of known events.
- d is a *proximity threshold* denoting the maximum acceptable distance of nodes that observe matching instances of events.
- d_{eff} is the *effective threshold* of the query. While its exact definition is deferred for Section 4.1, in a nutshell d_{eff} denotes the maximum acceptable distance, from the sensor node that made the announcement, of a node that will propagate an event announcement. The value of d_{eff} is decided by the base station, based on the information that it possesses about the topology of the sensor nodes.

The semantics of the query are such that the network should inform the base station whenever two or more events from the set $\mathcal{QS}$ are detected in proximity.

When $\mathcal{QS}$ contains just one event, we are looking for nearby instances of the same event. From the definition it is clear that proximity queries generalize spatial joins, normally considered in centralized systems.

In this paper we mainly focus on the case where the set QS contains only two events (say X and Y) and the proximity query has the form $Q(\{X, Y\}, d, d_{eff})$. It is easy to see that when more than two events are given, the query can be written equivalently as a collection of proximity queries amongst two events.

Throughout this paper we will not make any assumptions on the distance functions being used. For example, the proximity and the effective threshold of a query may be computed by (but not limited to) any given L_k $(1 \leq k \leq \infty)$ distance norm. The extensions to where d refers to the distance in number of hops between two nodes are straightforward. Moreover, in our discussion we will assume that each node has precise information about its location.

Example 1. Let us now provide an example on the difference of d and d_{eff}. Figure 1 demonstrates the computation of d_{eff} for the case a long path is required to reach node S_{target} from S_{src} (note that the transmission range of the nodes, denoted by the length of the arrow at the lower right corner of the figure, is not sufficient so that S_{src} can directly reach S_{target}). A similar behavior could have been observed if an obstacle between S_{src} and S_{target} forced the messages to traverse a long path in order to circumvent the obstacle. Essentially, any node that lies more than d_{eff} distance units from S_{src} will not process or propagate any announcement corresponding to events observed at S_{src}. If the desired distance metric is the number of hops, then we can set $d_{eff} = d - 1$ (in this case, nodes that propagate the announcement are at most $d-1$ hops away from S_{src}, while nodes that process the announcement are at most d hops away). If, however, the distance metric is computed as the L_k distance of two nodes, then d_{eff} may need to be larger than d, similarly to the example in Figure 1. In the particular example, the value of d_{eff} is determined by the physical distance between S_{src} and S_m in that figure, since S_m is the node that lies the furthest from S_{src} in this path. The details on how to determine the node S_m and its distance from S_{src} are described in Section 4.1.

∎

Each proximity query may be registered and communicated to all nodes in the network using a flooding algorithm (see for example [12, 16]) initiated by the base station. In turn, the nodes should inform the base station whenever the two events X and Y are detected by nodes S_1 and S_2 that are located within distance d of each other. A strict requirement specifying that only S_1 or S_2 should inform the base station does not exist. This information can be, for instance, computed and communicated to the base station by any node in the network, such as a node somewhere in between nodes S_1 and S_2 that becomes aware of both events. In a sensor network application it is natural to consider proximity queries as *continuous* queries, meaning queries that are registered once and are then continuously evaluated by the nodes in the network, until they are explicitly terminated. An alternative would be to include during query

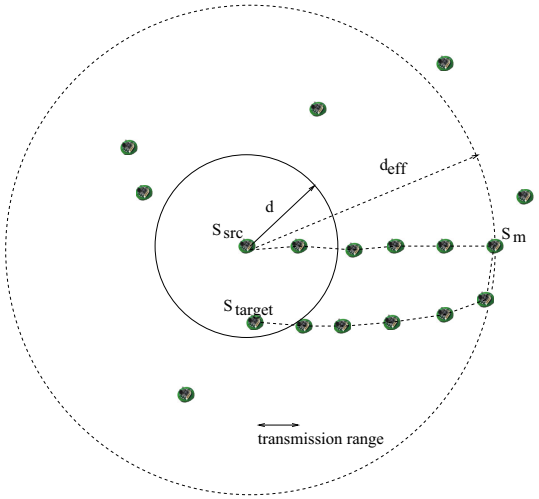

Figure 1: Computing the query effective threshold d_{eff}.

registration information about the lifetime of the query in the form of a time interval. The output of a proximity query is a stream of quadruples of the form (S_1, X, S_2, Y) indicating that node S_1 (resp. S_2) has observed event X (resp. Y). Additional information, such as the location of the nodes, can be easily included in the result.

If the proximity threshold d is zero, an alert is raised when an interesting pair of events is detected at the same node. This is a trivial case in that each node may process the query independently using only local 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.

4. Alternative Techniques based on Stateless Query Evaluation

Before presenting our algorithms for the evaluation of proximity queries, we first discuss in this section two straightforward algorithms for the evaluation of proximity queries, and explain their shortcomings. Both algorithms are agnostic on past observations made by the nodes (i.e., they do not maintain state) and simply react to new observations (events) in a deterministic manner. We assume that node S_{src} has just observed event X and there exists at least one continuous query of the form $Q(\{X, Y\}, d, d_{eff})$ registered in the network. The generalization where more than two events are selected by the query is straightforward.

4.1. Naive Algorithm - Use a Flooding Scheme

The first algorithm is based on flooding and simply broadcasts, in successive steps, the announcement of the event to an area large enough to contain every

node within distance d of S_{src}. We sketch the algorithm in what follows. Node S_{src} first checks whether an event of type Y is locally present, and if so, it notifies the base station. This can be accomplished, for example, by using the inverse routing path computed during query propagation [12, 40]. Then, if $d > 0$, the sensor node S_{src} broadcasts a message to its neighborhood including the type of the event (in our running example X), its location (i.e., the location of S_{src}) and a timestamp t_{obs} denoting the time that the observation took place.

Each neighboring node S_{nb} that receives this message behaves in a similar way. If $dist(S_{src}, S_{nb}) < d$, the node checks whether a locally observed event (i.e., an event Y) matches and should be reported to the base station. Then, if $dist(S_{src}, S_{nb}) \leq d_{eff}$ it broadcasts (X, S_{src}, t_{obs}) to its neighborhood. During this process a node S_i 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, its originator (node S_{src}) and the timestamp of the observation are included in the message. The cascading process terminates when all nodes that can be within distance d of S_{src} have been reached. This is accomplished by properly selecting the *e*ffective threshold d_{eff}, mentioned above, of the query. Its value depends on the physical layout of the network. Let $\mathcal{R}(S_{src}, d)$ denote the set of nodes that lie within distance d from S_{src}. Let $\mathcal{P}(S_{src}, d)$ denote the set of nodes that belong in the shortest paths, excluding the final node of each path, between S_{src} and at least one node in $\mathcal{R}(S_{src}, d)$. Thus,

$$\mathcal{P}(S_{src}, d) = \bigcup_{S_i \in \mathcal{R}(S_{src}, d)} \{minpath(S_{src}, S_i) - S_i\}$$

To ensure that all the nodes within distance d from S_{src} are reached, d_{eff} must be set to the maximum distance of any node in $\mathcal{P}(S_{src}, d)$ from S_{src}. Thus,

$$d_{eff} = max_{S_k \in \mathcal{P}(S_{src}, d)}(dist(S_{src}, S_k))$$

The proper value for d_{eff} could be determined by a base station that is aware of the location of the nodes and can be included in the message that initiates the query. While the case of mobile nodes is outside the scope of this paper, in such fast changing mobile networks, calculating the exact value of d_{eff} may be too costly, thus dictating the use either of a rough estimate of d_{eff} (i.e., one would expect d_{eff} to be close to d for dense networks in many cases), or of a conservative value $d_{eff} = \infty$.

In most practical cases d_{eff} will be roughly equal to d, unless we have a sparse network or many obstacles in the terrain that prohibit communication between certain nodes. We need to emphasize here that a larger value of d_{eff} has dramatic effects in the cost of the naive execution based on flooding. On the other hand, as it will be demonstrated in our experimental evaluation, our techniques can actually benefit from a marginally increased value of d_{eff} because, with a slightly increased number of messages, more nodes get the opportunity to better tune their indices.

The main drawback of using flooding to propagate the announcements of events is that, in a dense sensor network, these announcements will be received (and transmitted) by many nodes, even though these nodes do not contain, or are not in a path leading to, a matching event. Even though a node will eventually react to the first announcement only, duplicate announcements still drain energy during the reception of the messages. Since the cost of listening is often comparable to the cost of transmission (e.g., in the popular Mica2 motes the ratio amongst the power consumption while idle-listening, receiving of a message and transmitting is 1:1:1.41 at 433MHz with RF signal power of 1mW in transmission mode [59]), the aggregate power consumption during the flooding algorithm can be prohibitively large.

4.2. External Computation

A simple alternative to flooding is for node S_{src} to transmit the announcement of a new event X to the base station, which will then compute the results to all running proximity queries externally. This practice however violates one of the main principles in designing sensor networks, that is, to perform as much computation as possible inside the network, and to leverage the large number of nodes in order to reduce, as much as possible, unnecessary communication with the base station [29]. Consider for example the case where a lot of nodes locally observe events of type X but there is no event of type Y in the monitored area. Then, all the traffic towards the base station will be wasted, since the query $Q(\{X, Y\}, d, d_{eff})$ has an empty answer set. Since in the external algorithm the number of transmitted events per node depends on the number of events observed in its subtree, we expect that this algorithm will perform worse as the size of the network increases. In such cases, the base station and the nodes surrounding it end up receiving and/or transmitting a significant amount of messages, thus quickly draining their energy. This is also verified in our experimental evaluation.

5. Query Processing Using Routing Indices

In this section we describe algorithms that utilize state information for computing proximity queries. The presented algorithms are based on local indexing structures, termed routing indices. These indices are maintained by the nodes while processing new events announced in their neighborhood and, as it will be demonstrated by our experimental evaluation, drastically reduce the number of messages required to process a proximity query. We first present some important notes on the applicability of our algorithms to different topologies or organizations of the sensor nodes, and then proceed by presenting an overview of the operation of the nodes in our algorithms. We then describe the routing indices needed in the sensor nodes for the operation of our algorithms, describe how these indices are updated and, finally, how they are used by our algorithms in order to properly guide the searches for matching events.

5.1. Applicability to Different Topologies and Node Organizations

It is important to note that the algorithms that we introduce in this section can be applied to any given topology of the sensor nodes. Thus, the algorithms that we will present do not assume any given topology. However, some presented examples will use a grid arrangement for the sensor nodes, purely for ease of presentation.

Moreover, in our algorithms we do not assume any given organization of the sensor nodes. Our discussion will thus not assume any such organization, and will be applicable to any flat sensor topology. However, we need to emphasize that our algorithms can also be applied in cases when the sensor nodes are organized in clusters, but these clusters are not hierarchically organized. In such an organization, the observed events by any node can be propagated to the node's clusterhead. Each clusterhead can then detect whether some matching events have been observed by nodes within its cluster. However, not all matching events observed by nodes within a distance d can be detected in such a case. In the general case, the nodes within a distance d from a sensor that observed an event cannot be assumed to belong to the same cluster, especially for larger values of d. Thus, each clusterhead will need to communicate with other clusterheads, in order to discover all matching events. One can, thus, use for this communication phase the algorithms that we propose in this section, applied over only the clusterheads. We also need to note that there is no straightforward reason why restricting the non-cluster nodes to seek matching events only through their clusterhead would reduce the number of transmitted messages in our application, when compared to an alternative approach where each node is free to select where to direct and propagate its searches, amongst any of its neighbors.

On the other hand, our algorithms are not tailored to hierarchical organizations of clusters. Each node in such hierarchical organizations could be used to detect all matching events within its subtree. However, since in many cases matching events can also be observed by nodes in different subtrees, the observed events may need to be propagated all the way to the top of the hierarchy. Thus, such organizations are expected to have, in our application, similar characteristics and drawbacks with the external computation, presented in Section 4.2.

5.2. Node Operation Overview

Consider a given node S_{cur} that receives the announcement of an event X observed by node S_{src}. Note that the above assumption also covers the case when S_{cur} is a node that actually observed X (i.e., $S_{cur} = S_{src}$). Briefly, the sensor node will perform the following actions:

1. Determine if it has already responded to this announcement (i.e., has received the same announcement from more than one nodes).
2. If not, determine whether X and some locally observed or registered, but observed by other sensors, concurrent events (we describe later in this section when two events can be considered to have been observed concurrently) satisfy one or more registered proximity queries. If so, it notifies the base station about the relevant events.

3. Based on the node(s) through which it received the announcement, the sensor updates relevant information (i.e., the weights of neighboring nodes in its routing indices) regarding these nodes.
4. Based on the distance of the sensor from S_{src} and the effective query thresholds specified in registered proximity queries involving the event X, the node determines if it needs to forward the announcement to neighboring nodes. This step is performed only if the received message specified S_{cur} as a node that will forward the request (this is clarified in the next step).
5. If so, and the node has information on where matching events may be located, based on its routing indices, then a proper subset of the neighboring nodes is selected, denoted as the node's *next hop list* NH_{cur}, to forward the announcement, and the node broadcasts the announcement of the event X along with the list of the nodes in NH_{cur}. If, however, the node does not have any information on where to locate matching events, then the node initiates a search to locate nodes with matching events.

We now describe in more detail the above operations. In Algorithm 1 we sketch an implementation of the Accept() subroutine at sensor node S_{cur} for processing an incoming message during the course of the algorithm. In the algorithm we can clearly distinguish the five actions mentioned above. In Lines 3–5, the node first checks whether it has already responded to an announcement for the same event and, if so, ignores the message. Duplicate announcements can be easily distinguished based on (i) The node S_{src} that observed the event; (ii) The observed event X; and (iii) The timestamp t_{obs} of the observation. Then, in Lines 7–9, if a local or a registered (but observed by some other sensor) concurrent event Y has been detected at node $S_{X'}$, for which a query $Q(\{X, Y\}, d, d_{eff})$ is also registered and the distance of node $S_{X'}$ is less or equal to d from the originator of event X (node S_{src}), then the node informs the base station of the two matching events.

A question that naturally arises at this point is when two events observed by two different sensors are considered to satisfy a proximity query, since the timestamps of the observations may differ, even slightly. A simple solution involves utilizing the notion of the *query epoch*. In many sensor network applications the sensor nodes are asked to collect measurements at specified time periods (i.e., once every second). The time between two consecutive observations is termed as the *query epoch*. If in our application the timestamp of the measurements is expressed in terms of the epochs since the beginning of the query evaluation, then two observations can be considered to refer to concurrent events if they were collected within the same epoch. Otherwise, in case more complex timestamps are used, we may consider that two observations are concurrent if their difference is less than a query epoch. Notice that it is straightforward to expand our algorithm to answer arbitrary spatio-temporal queries where the base station specifies the maximum acceptable difference between the timestamps of two matching events.

Algorithm 1 `Accept` Subroutine

Require: $(X, S_{src}, t_{obs}, NH)$

1: $\{X$ is the event type reported by node S_{src} at time $t_{obs}\}$
2: $\{NH$ is a list of nodes, specified in the received message, that should continue further the messaging process$\}$
3: **if** HasSeen(X, S_{src}, t_{obs}) **then**
4: return {Has already processed an announcement for this event}
5: **end if**
6: $\{S_{cur}$ is the current node$\}$
7: **if** Exists local or registered concurrent event Y observed at node $S_{X'}$ and query $Q(\{X, Y\}, d, d_{eff})$ and dist$(S_{X'}, S_{src}) \leq d$ **then**
8: InformBaseStation$(X, S_{src}, Y, S_{X'})$
9: **end if**
10: UpdateLocalIndices(X, S_{tr}, t_{obs}) $\{S_{tr}$ is the node that transmitted this processed message$\}$
11: $\{maxd_{eff}$ is the largest effective threshold for any registered query $Q(\{X, Y\}, d, d_{eff})\}$
12: **if** S_{cur} in NH and $dist(S_{cur}, S_{src}) \leq maxd_{eff}$ **then**
13: {A tuple in the routing index $T(S_{nb}, e, S_{tar}, t)$ means that the e event was observed by S_{tar} at time t. S_{tar} can be reached through neighboring node S_{nb}.}
14: $\mathcal{E}_{cur}$=$\{e \in \mathcal{E}$: $\exists Q(\{X, e\}, d, d_{eff}) \wedge \exists T(S_{nb}, e, S_{tar}, t) \wedge dist(S_{src}, S_{tar}) \leq d\} \wedge (|t_{obs} - t| \geq 1\ epoch)$ {Matching events, but not matching timestamps}
15: **if** Routing index contains info about any event in $\mathcal{E}_{cur}$ **then**
16: NH_{cur} = PickHops(X, S_{cur}, t_{obs}){Pick next hops for event X and originating node S_{cur}}
17: Broadcast$(X, S_{cur}, t_{obs}, NH_{cur})$
18: **else**
19: InitiateSearch(X, S_{src}, t_{obs})
20: **end if**
21: **end if**

In Line 10 of Algorithm 1, a call to function UpdateLocalIndices() is made, so that the routing indices of the node can be updated. This process is discussed in Section 5.3.1. Finally, in Lines 11–20, if the node belongs to the next hop list NH that is encapsulated in the received message and its distance from S_{src} does not exceed the largest effective threshold amongst all matching proximity queries registered at the node, then this node is responsible for propagating the announcement further in the network. To accomplish this operation, the node S_{cur} first computes, as it will be explained shortly, a new list of next hops for the message and finally broadcasts this list, along with the original event X, the identifier of the node S_{src} and the timestamp t_{obs} of the observation. The method through which the list of the next hops is selected varies on each of our algorithms, and is presented in Section 5.3.3. Please note that the message should intuitively be propagated only to nodes through which matching proximity events can be detected. To achieve this, the algorithm removes in Line 14 from consideration events that have been observed by nodes S_{tar} that lie further than the specified proximity threshold d from S_{src}. The algorithm also removes

(in the last conjunctive term of Line 14) from consideration events (events that are either locally observed, or registered events that were observed at different nodes) that the current node matched with the X event and reported to the base station in order to avoid unnecessary traffic and to avoid duplicating reported results. An important point is that if the node does not have any information on where to locate matching events (either because it has never received the announcement of such events or because the relevant information in its routing indices has become outdated and evicted from them), then the node initiates a search to locate matching events. This process is also described in Section 5.3.3.

It is worth noting that nodes that are not in the list of the next hops NH are still able to check for local matching events and also to update their routing information, since broadcast communication is used. 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.

Example 2. In order to ease presentation and without affecting the applicability of our techniques to other configurations, we provide an example where the sensor nodes are placed in a two-dimensional $n \times n$ grid. We also slightly deviate from the symbols used so far to denote the sensor nodes and rather utilize two subscripts to denote the position of the sensors in the grid. Using this notation, $S_{i,j}$ will denote the sensor node at location i,j in the grid, where $0 \leq i, j < n$. For this grid arrangement of the sensor nodes we will use the L_∞ norm to compute distances between nodes.[2] Thus, in our example:

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

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 1 from sensor node $S_{1,1}$. For this example we 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 thus reach any of its 8 immediate neighbors. We note again that other distance metrics, placements of nodes and transmission ranges are possible without affecting the generality of our techniques.

In order to build our running example, we consider the scenario depicted in Figure 2. Node $S_{1,1}$ has just observed event X (that may indicate the presence of say enemy forces) and assume that the continuous proximity queries $Q_1(\{X, Y\}, 3, 2)$ and $Q_2(\{X, W\}, 3, 2)$ have been previously registered in the network. Note that in the aforementioned grid topology the effective threshold d_{eff} actually has a smaller value than d. This occurs because in the described grid topology any node at distance d from S_{src} can be reached through nodes at distance $d-1$ from the same node. In the straightforward flooding algorithm, node $S_{1,1}$ will start a flooding process that will announce the triplet $(X, S_{1,1}, t_{obs})$ to all nodes in the 5×5 area depicted in Figure 2. The number of messages transmitted in

[2]Please note that in our discussion so far we have not made any assumptions on the distance function being used.

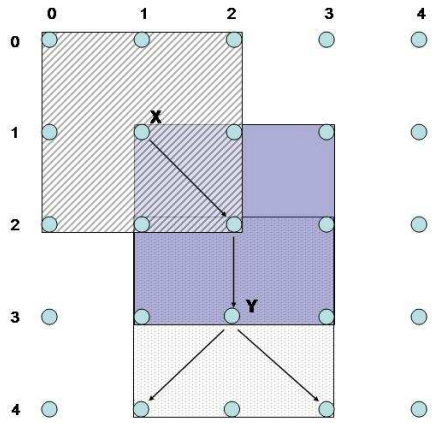

Figure 2: Grid Arrangement of Sensors in our running example. The 3 shaded rectangles denote the nodes reachable by sensors $S_{1,1}$,$S_{2,2}$ and $S_{2,3}$. The arrows denote which nodes are selected to propagate the announcement.

that case will be equal to 16 and the number of messages received (including nodes that are idle-listening) equal to 105.

In the same figure we also observe that an event of type Y was previously detected by node $S_{2,3}$. Thus, both events need to be reported in response to query Q_1. If node $S_{1,1}$ had a way of knowing the position of event Y at grid location (2,3) 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 $S_{2,3}$ and have the latter confirm the proximity alert by responding to the base station with the tuple $(X, S_{1,1}, Y, S_{2,3})$.

In order to accomplish the task as described, node $S_{1,1}$ that first observed event X sends a broadcast message to its adjacent nodes indicating the location of the event and the timestamp t_{obs} of the observation (i.e., the tuple $(X, S_{1,1}, t_{obs})$) and that its south-east neighbor, node $S_{2,2}$ should carry over propagating the event to the rest of the network.[3] All the other nodes in its neighborhood can hear the announcement, and 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 $(X, S_{1,1}, t_{obs})$ as well as the next "hop", node $S_{2,3}$. Node $S_{2,3}$ that has observed event Y reacts by informing the base station of the new result to the proximity query (the required messages are not depicted in the figure). The distance of node $S_{2,3}$ from $S_{1,1}$ is 2, which is equal to the query effective threshold and the node further propagates the

[3]Please note that this example simply illustrates some steps of our algorithm. Section 5.3 describes how the next hop list NH - i.e., node $S_{2,2}$ in our example - is formed.

announcement, indicating this time that both nodes $S_{1,4}$ and $S_{3,4}$ should be the next "hops". Nodes $S_{1,4}$ and $S_{3,4}$ do not need to propagate the announcement further, because the effective threshold of the query has been reached. In all, just three messages have been transmitted and the number of received messages was 24, a significant reduction compared to the straightforward algorithm. ∎

Notice that, in general, a node may indicate that more than one nodes should further propagate the announcement of an event, in search of results to the proximity query. In Section 5.3.3 we will describe algorithms for both of these modes of operation, namely the *threaded* and the *webnet* modes. In the threaded mode each node that simply propagates (and, thus, was not the node that observed the event itself) the announcement of an event selects just one node to include in its next hop list NH. On the contrary, in the webnet mode each node can select multiple neighboring nodes to include in its next hop list. This has the visual effect that the links between the nodes that propagate the query form paths that look like threads or a spider web, respectively.

5.3. 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 that stores information about observed events. Using this routing index a node S_{cur} associates with each neighboring node S_{nb} a weight, based on the matching events that were routed to S_{cur} through S_{nb} and the timestamp of these events (i.e., how old or current are they?). There are 3 major operations associated with the routing indices of a node:

1. How to update the routing index when the node receives information on new events.
2. How to determine the weights for neighboring nodes given the entries in the routing index.
3. Given the routing index, how to determine the next hop list NH of the node.

We now discuss the above operations in detail.

5.3.1. Updating the Routing Index

At each node S_{cur}, the index is instantiated as a list of quadruplets $T(S_{nb}, e, S_{src}, t_{obs})$, where S_{nb} is a neighbor of node S_{cur}, e is an event (from the set $\mathcal{E}$), S_{src} is the node that observed the corresponding event e and t_{obs} denotes the timestamp of the observation. We assume that the size of the index is bounded. Thus, when the routing index becomes full older entries, based on their timestamp of observation, need to be evicted and replaced by new observations.

An open question is, given the tuples in the routing index, how to associate a weight with each link and event-type. The larger the weight that is associated with a link, the more likely it 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_{src} 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_{src}. Therefore, when node S_{cur} hears about event (X, S_{src}, t_{obs}) from its neighbor S_{nb}, the timestamp t_{obs} can be used as a weight for the pair $\langle S_{nb}, X\rangle$. Intuitively, this process generates a reverse routing tree that is rooted at node S_{src} that originally announced the event. This conceptual reverse routing tree depends on the nodes that are included at each step in the NH list. Thus, one cannot provide any guarantees on its optimality, with respect to the distance (in hops) of each node from the root of this reverse tree.

Example 3. In our running example, node $S_{2,3}$ will insert a new row with values $(S_{2,2}, X, S_{1,1}, t_{obs})$ 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 (see the two registered proximity queries in Example 2) is announced in the neighborhood of $S_{2,3}$ 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. ∎

5.3.2. Determining the Weights of Neighbors

For each neighbor S_{nb} of node S_{cur} we compute the compound weight $W_{S_{nb},X}$ for routing a message regarding X through S_{nb} using the following equation:

$$W_{S_{nb},X,S_{cur}} = \max_{\substack{e, S_{tar}, t' : \exists Q(\{X,e\}, d, d_{eff}) \\ AND\ \exists T(S_{nb}, e, S_{tar}, t') \\ AND\ dist(S_{src}, S_{tar}) \leq d \\ AND\ |t_{obs} - t'| \geq 1\ epoch}} (t') \quad (1)$$

The compound weight $W_{S_{nb},X}$ is computed as the maximum of all registered timestamps t' (the maximization term), in the routing index entries, involving events that are used in proximity queries of the form $Q(\{X, e\}, d, d_{eff})$ (Line 1 of the conditioned term of Equation 1). An important point is that the distance of the node S_{src}, which detected the X event, from nodes that have detected matching events should not exceed the distance specified in the proximity query (Lines 2-3 of the conditioned term of Equation 1; also see Line 14 in Algorithm 1). Several techniques [34] have been proposed for determining the location of sensor nodes, with perhaps the most popular techniques being: (i) Deploying GPS enabled sensor nodes; or (ii) Deploying anchor points in the network. Thus, in such scenarios nodes can piggyback information about the location of S_{src} in their transmitted messages and store such information in the routing indices. Of course, if the sensors are placed in predetermined positions (i.e., in the grid arrangement of Example 2), then the nodes can determine the

NextHop (n)	Event(e)	Obs	Time
$S_{3,4}$	W	$S_{4,4}$	55
$S_{1,4}$	Y	$S_{1,5}$	56
$S_{3,4}$	Y	$S_{3,5}$	57
$S_{1,4}$	W	$S_{1,4}$	58
$S_{1,3}$	V	$S_{2,5}$	60

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

location of other sensor nodes based simply on their identifier. Another requirement is that entries in the routing index that helped the current node S_{cur} detect and announce to the base station results to registered proximity queries at the current epoch should not be used when calculating the weights of links (Line 4 of the conditioned term of Equation 1). Routing index entries corresponding to the current epoch represent events that the node is aware of, and for which it has already tested for matching proximity queries. Thus, entries corresponding to the current timestamp (i.e., epoch) must be ignored, in order to direct the search towards other nodes/areas where matching events may be detected.

The operation of determining the compound weights of neighboring nodes dominates the running time of the Accept subroutine at each node. For each entry in the routing table, the algorithm examines whether the recorded event e matches X. Thus, the running time is $O(|Q| \times |RoutingIndex|)$, where $|Q|$ denotes the set of registered queries and $|RoutingIndex|$ denotes the size of the routing index. Please note that the compound weight of a neighboring node regarding an event X is not affected if no matching events are received through that node, an observation that allows us in some cases to reduce the computation time by using precomputed values.

Example 4. Continuing the scenario described in Example 2 and illustrated in Figure 2, in Table 2 we show a snapshot of the routing index for node $S_{2,3}$. Assume that the node has just received the announcement for event $(X, S_{1,1}, 62)$ and recall that the following two proximity queries are registered: $Q_1(\{X, Y\}, 3, 2)$ and $Q_2(\{X, W\}, 3, 2)$. Thus, only tuples involving the events Y and W are relevant in this case. In this example the ranked list of weights will be (we note that $W_{S_{1,3},X}$ is not defined in this example since V is not a matching event): (i) $W_{S_{1,4},X}$=58 and (ii) $W_{S_{3,4},X}$=55, since $dist(S_{3,5}, S_{1,1}) > 3$. Also note the first tuple involves the most outdated observation and is, thus, the first candidate for eviction if new events are observed and the index reaches its maximum size. ∎

Based on our discussion above, the compound weight of each neighboring node is determined by the latest timestamp of a matching event received through that neighboring node. We now analyze why we opted for such a decision, and did not settle on using different weighting methods, such as using the total number of events heard through each neighboring node, or the distance of the received events.

We first argue that the timestamp of received events should be a factor in determining the compound weight of neighboring nodes. Please recall that our assumption is that events do not randomly appear, but rather move around in the space. A neighboring node through which several/close events were received in the distant past may not be a proper choice, as these events may be currently placed at entirely different/distant areas of the network. On the other hand, for recently received event announcements we stand a better chance of directing the search either directly to, or at least close to, the location of a matching event. We also note that utilizing combinations of different metrics can be easily incorporated in our framework. For example, it is straightforward to break ties between neighboring nodes that have propagated the announcement of an event at the same epoch, based on the distance of these events from the current node S_{cur}. We plan to explore such extensions in our future work.

5.3.3. Selecting the nodes in the NH list

We now describe how our algorithms determine which neighboring nodes will be included in their NH list when transmitting a message. In this paper we studied several variations that arise by the following cases:

- Restricting the maximum number k of neighbors included in the NH list of the node(s) that observed an event and which initiate a search for matching events. We note that a value of $k = \infty$ will result in these node(s) always including all of their candidate neighboring nodes (we will explain shortly that not all the neighboring nodes will be included in the list) in their NH list.

- Varying the type of message propagation. We mentioned at the end of Section 5.2 that a node that did not initiate a search (please note the difference with the previous case) may include either only one neighboring node in its NH list (in the *threaded* propagation mode), or include up to k neighboring nodes (in the webnet propagation mode), where k is the same parameter used by the nodes that initiated the search.

- Selecting the nodes in the sensor's NH list using deterministic or probabilistic decisions.

All the variations that we will describe in this section share the common characteristics that they do not include in a node's NH list neighboring sensors through which the node received the announcement. This is why the set of candidate neighboring nodes is typically smaller than the set of neighboring nodes. We examined all the variations that may arise from the possible combinations of the algorithm's characteristics. The most interesting cases involved are:

1. *RI-k* algorithm: This is our main algorithm where any node that propagates the announcement of an event (whether it is the node that observed the event or not) selects up to k neighboring nodes to include in its NH list. The decisions are made deterministically, based on the compound weight (see Section 5.3.2) of each neighboring node. Neighboring nodes

whose compound weight is zero are never selected. This means that fewer than k neighboring nodes may be selected. The initials "RI" denote the use of *routing indices*.

2. TRI-k algorithm: This algorithm is similar to the RI-k algorithm. The main difference is that it is a *threaded* algorithm, meaning that each node that propagates the announcement of an event not observed by the node itself will only include one node in its NH list. Moreover, unlike the RI-k algorithm, a node that observes (resp., propagates the announcement of) an event will always include k (resp., 1) neighboring nodes in its NH list (assuming that at least k candidate neighboring nodes exist), even if fewer than k (resp., 1) candidate neighboring nodes with non-zero compound weights exist. In such a case the node first includes in its NH list all the candidate neighboring nodes with non-zero compound weights and then adds an appropriate number of neighboring nodes. The selection amongst these nodes is performed in a random manner.
3. PRI-k algorithm: This algorithm is similar to the RI-k algorithm but makes the decisions on which candidate neighboring nodes will be selected in its NH list probabilistically. Given that after several epochs the ratio amongst any two timestamps stored in a node's routing index is likely to be close to 1 (i.e., in Table 2, $60/55 \approx 1.09$), we need a way to assign significantly larger probabilities to nodes with more recent observations. In our implementations we, thus, first determine the compound weights of all neighboring sensor nodes. If $W_{min,X}$ denotes the minimum non-zero such compound weight, the probability of each neighboring node S_{nb} with a non-zero compound weight was set proportional to $W_{nb,X} - W_{min,X} + 1$ (after proper normalization - see Example 5).
4. $PTRI$-k algorithm: This algorithm is similar to the TRI-k algorithm but makes its decisions on which candidate neighboring nodes will be selected in its NH list probabilistically, similarly to the PRI-k algorithm. Thus, a node that observes (resp., propagates the announcement of) an event first examines whether it has at least k candidates neighbors with non-zero compound weights. If this is true, then the choice of which candidate neighboring nodes will be selected in its NH list is performed probabilistically, similarly to the PRI-k algorithm. Otherwise, the NH list will contain not only all candidate neighbors with non-zero compound weights, but also an additional proper (such that the number of nodes in NH becomes equal to k) number of random neighboring nodes with zero compound weights.

Example 5. Continuing Example 4, for k=1 the $S_{2,3}$ node will only include in its NH list node $S_{1,4}$, which exhibits the largest compound weight, when using either the RI-k or the TRI-k algorithm. Similarly, when using the PRI-k or the $PTRI$-k algorithms, the $S_{2,3}$ node will include in its NH list only $S_{1,4}$ with a probability $\frac{58-55+1}{(58-55+1)+(55-55+1)} = 0.8$ and include only $S_{3,4}$ with a probability $\frac{55-55+1}{(58-55+1)+(55-55+1)} = 0.2$. ∎

Algorithm	State	Propagation	Deterministic
Flooding	N	Webnet	Y
External	N	Threaded	Y
RI-k	Y	Webnet	Y
PRI-k	Y	Webnet	N
TRI-k	Y	Threaded	Y
PTRI-k	Y	Threaded	N

Table 3: Categorization of algorithms discussed in this paper.

An important observation that we have not clarified yet is what occurs in Line 19 of the Accept() subroutine. Recall that the execution of the code reaches this line when the node has no local information on how to route the announcement of event X. Function InitiateSearch will initiate a threaded search using the $PTRI$-k algorithm. We used the threaded algorithm in that scenario in order (i) not to overburden the nodes around S_{cur} in case they do not themselves contain entries on matching events in their routing indices; and (ii) to allow these nodes to update their routing indices using the information of the X event even if no matching event exists in the neighborhood of the node. This call also serves to essentially train the routing indices of a node when it first joins the network. The way to easily implement the potentially different search behavior of a node is by adding another argument to the transmitted messages, namely the maximum number k' of nodes in the node's next hop list NH. For the nodes that propagate an announcement about an event that they observed, $k' = k$. Let us now consider the case of nodes that received an announcement about an event observed at some other sensor. For such nodes, the value of k for their transmitted messages is computed (as a function of the k' value in the received announcement) as follows:

- In all transmissions up to k' sensors are included in a node's NH list.
- However, if the transmission occurs at Line 17 of the Accept() Algorithm and for webnet algorithms (i.e., RI-k or PRI-k), the k value specified in the transmitted message is set to k'.
- Otherwise, for messages at Line 19 or for threaded algorithms, the transmitted k value is set to 1.

Table 3 presents an overview of the characteristics of the algorithms discussed in this paper.

5.3.4. Memory Requirements

Up to this point he have not concerned ourselves with the memory requirements for maintaining the routing indices. Please recall that the entries of the index are utilized when computing the compound weight $W_{S_{nb},X}$ for routing a message regarding X through neighboring node S_{nb} based on Equation 1. This

equation, as explained, utilizes the most recent observation regarding X. Let $|\mathcal{E}|$ indicate the number of event types relevant to all running queries and N_{nbrs} indicate the maximum number of neighbors that a node has, then we need at most $|\mathcal{E}|$ * N_{nbrs} entries in the routing index. Furthermore, especially for the deterministic versions RI/TRI of our algorithms, assuming that the value of k is selected by the application, only the top-k values per event are essential for selecting the NH list members, thus upper-bounding in these cases the size of the index by $|\mathcal{E}|$ * min(k,N_{nbrs}), which is typically a small value. In our experiments we have used a modest maximum value of 20 entries for storing the routing index. With the current trend in increasing the available memory at each sensor via flash memory, we believe that the size of the index is not a concern since we can easily store thousands of entries in the flash. Thus, in our work, we do not explore the effect of limited memory (less than 20 entries) in the performance of the index, as we believe such a study will be of limited value for real application scenarios.

6. Extensions

We now discuss some useful extensions to our algorithms. We first discuss the case when the communication between some sensor nodes is not symmetric. We then describe how moving events are handled in our framework in the case when the sensor nodes can collect additional information about these events.

6.1. Dealing with Asymmetric Communication

Many cases may arise when the communication between pairs of sensor nodes is not symmetric. This means that a sensor S_{cur} is able to receive messages transmitted by another node S_{nb}, but the reverse is not true. This may occur due to either the different types of deployed sensors, or the need of some sensors to limit their energy drain by decreasing their transmission range. In such cases the weights associated with each neighboring node need to be assigned with some care.

In particular, consider the aforementioned case where S_{cur} receives an announcement by S_{nb} about the observation of event X from node S_{src} at time t_{obs}. Since we assume that S_{nb} cannot receive messages from S_{cur}, the weight $W_{nb,e}$ about any event e is not defined. If S_{cur} becomes aware of any matching event and needs to forward messages towards S_{nb}, then it needs to use its routing tables and determine a routing path to S_{nb}. Let S_i denote the first node (i.e., the next hop) in that path. Then the aforementioned received announcement from S_{nb} needs to be used in determining the weight $W_{i,e}$.

6.2. Dealing with Moving Events

Up to this point we have only discussed the case where sensor nodes can only detect the existence of an event, without being able to collect any additional information about this event. However, when the event at question involves the observation of possibly moving objects, then additional information, when

available, such as the direction or speed of the event [32] could help better tune the routing index.

Consider the case, for instance, when each node, in addition to the location of an object, can also determine its speed and direction. In this case we can estimate, using this information along with the timestamp of this observation (essentially the epoch lag between the current epoch and the epoch of the observation) and the position of the sensor node, the likely future location of this event. In this case the PickNextHops() function of Algorithm 1 can be modified to take into account this information by adjusting the weights of neighboring sensors if (i) a matching event has been moved in the area of their observation; or (ii) they can be used to route information towards the moved events (if these events are estimated to lie more than one hops away). Please note that for sensor nodes without any such sensing capabilities simply setting the speed of the object to zero results in the setup that we have considered so far.

Example 6. Consider the scenario in Example 4 where node $S_{2,3}$ has just received the announcement for the event $(X,\ S_{1,1}, t_{obs})$. This node initially determines that $dist_\infty(S_{2,3},\ S_{1,1}) = 2$. Recall that in the example $Q_1(\{X, Y\}, 3, 2)$ and $Q_2(\{X, W\}, 3, 2)$ are the sample proximity queries that are registered at the node. Table 4 represents the augmented routing table stored at node $S_{2,3}$. When compared to Table 2, one can detect that additional information is recorded such as the direction and the speed of the event. Based on Table 4, the node first seeks to determine the weights $W_{nb,X}$ for each of its neighbors. For the $S_{1,3}$ node the observed event V does not match any of the proximity queries involving X. However, note that at the third line of Table 4 the event Y was observed at node $S_{3,5}$. Based on the recorded statistics, the estimated new position of the event at the current epoch $t = 62$ has moved $(62 - 57) \times 0.4 = 2$ nodes northwest, at node $S_{1,3}$. Therefore, the weight $W_{S_{1,3},X}$ becomes equal to 57, based on the timestamp of this observation. The neighboring node $S_{3,4}$ can only be used to find the matching event W, based on the first line of the routing index. Thus, $W_{S_{3,4},X} = 55$. Similarly, for node $S_{1,4}$ the event W is estimated to have moved $(62-58) \times 0.25 = 1$ nodes to the south (to node $S_{1,5}$). However, the distance of $S_{1,1}$ from $S_{1,5}$ exceeds the distance of the proximity query. The same is also true for the estimate of the Y event listed at line 2 of the index. Thus, $W_{S_{1,4},X} = 0$, since no matching event is estimated to be reachable through $S_{1,4}$ given the specified proximity thresholds. ∎

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 [51] 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.

7. Experiments

In this section we study the performance of the various techniques that we discussed in Sections 4 and 5 using a simulator that we developed. Unless

NextHop (n)	Event (e)	Obs	Time	Direction	Speed (nodes/epoch)
$S_{3,4}$	W	$S_{4,4}$	55	-	0
$S_{1,4}$	Y	$S_{1,5}$	56	-	0
$S_{3,4}$	Y	$S_{3,5}$	57	NW	0.4
$S_{1,4}$	W	$S_{1,4}$	58	S	0.25
$S_{1,3}$	V	$S_{2,5}$	60	SE	0.2

Table 4: Sample routing index at node $S_{2,3}$ and epoch $t = 62$.

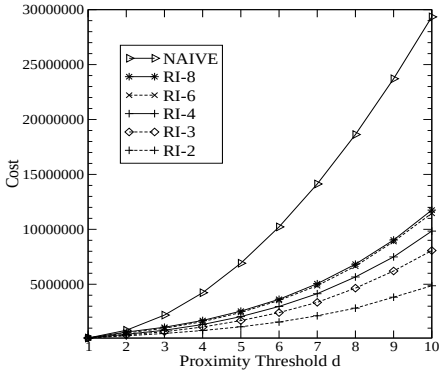

Figure 3: Cost of a query, varying d (RI-k).

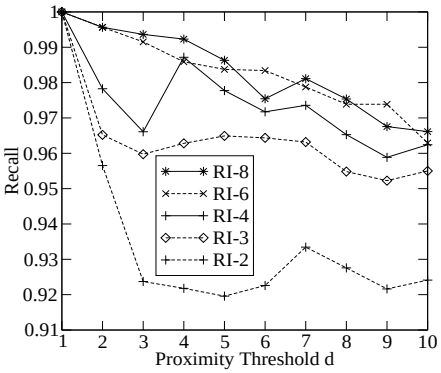

Figure 4: Recall of the RI algorithm.

specified otherwise, our basic node topology was modeled as a 100×100 grid of nodes (10K total). Due to the grid topology, in most of our experiments (except from Figure 6 discussed later in this section) the proximity and the effective thresholds are set to differ by 1 (i.e., $d_{eff} = d - 1$). Moreover, the L_∞ norm is used in order to express the distances between nodes, similarly to Example 2. We tested our techniques using the following data sets:

- **RandMoves:** This is a synthetic data set of moving objects of various types, depending on the configuration parameters. The objects are free to move in the monitored space by performing random walks. The walks are modeled so that when object X is at grid location (i, j) at time t, its next position at $t+1$ is randomly chosen from one of the adjacent cells of (i, j).
- **Trucks:** This is a real trace containing trajectories of moving trucks [1]. We randomly labeled 10 trucks as type X and 10 as type Y and overlaid a network grid topology over the monitored area.
- **SchoolBuses:** This is a real trace containing trajectories of moving schoolbuses [1]. We randomly labeled 10 buses as type X and 10 as type Y and overlaid a network grid topology over the monitored area.

We first use the RandMoves data set with 20 objects, 10 of type "X" and 10 of type "Y" in the 100x100 grid. In Figure 3 we compute the communication cost of computing a single continuous proximity query $Q(\{X, Y\}, d, d_{eff} = d-1)$ for 2,000 epochs using the RI-k algorithm, varying the proximity threshold d (x axis). The total cost, following the suggestions in [59] is computed as cost=1.41×(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 thus omitted). In each node, we allotted space for 20 entries for the routing index and evicted older observations using the time-stamps of the events.

The line labeled NAIVE depicts the performance of the flooding algorithm discussed in Section 3. For RI-k we show results varying the k parameter (see Section 5) from 2 up to 8. As expected, a smaller value of the k parameter results in fewer messages in the network. Compared to the naive execution, we note that even a value of $k = 8$ (i.e., max number of neighbors for this grid topology) results in a reduction of the total cost by a factor of up to 4, depending on the proximity threshold. This is because, our algorithm does not route messages towards neighbors that have never announced a matching event before. In comparison, the cost of RI-2 was up to 9 times smaller than that of the naive flooding algorithm.

A potential drawback of using a small value for k is that, under certain conditions, we might not be able to detect a few proximity events, because of the pruning of messages towards some of the neighbors of a node. In Figure 4 we plot the *recall* of the RI-k algorithm varying d and for the different values of k. We note that precision is a perfect 100% as we never return spurious tuples. Thus, the recall metric, which is computed as the ratio

$$\frac{\{relevant\ answers\} \cap \{retrieved\ answers\}}{\{relevant\ answers\}}$$

is simplified to computing the percentage of answers returned (i.e., all returned answers are relevant). The lines show that a value of 2 or larger for k returns at least 92% of the answers. The median value of recall is 93% and 96% for k=2,3 respectively, while it reaches 99% when $k = 8$.

We next repeat the same experiment, considering additional algorithms for routing the events in the network. In Figure 5 we plot the recall of RI-k, PRI-k as well as their threaded versions (TRI-k and PTRI-k respectively) varying the k parameter when the proximity threshold d is set to 5. Figure 5 also includes two baseline algorithms that we consider in the evaluation. The first algorithm, termed as RW-k, initiates k random walks, whenever an event is observed. A more sophisticated version of this algorithm (denoted as informed Random Walks, or iRW-k, in Figure 5), performs random walks by only considering neighboring nodes that have observed a matching event in the past. The difference between iRW and RW is, thus, that the former avoids areas of the network where a matching event has never been located. The x-axis in the graph is the cost of the respective algorithm over the cost of NAIVE, while the

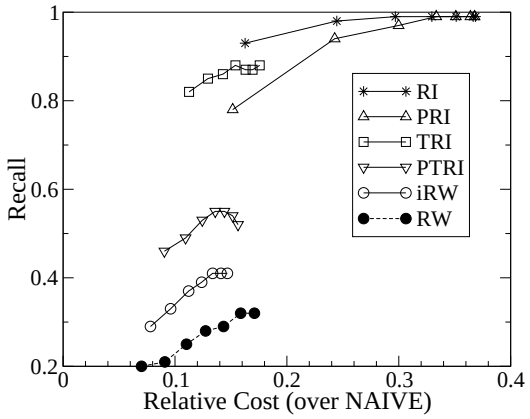

Figure 5: Comparison of different algorithms, varying k.

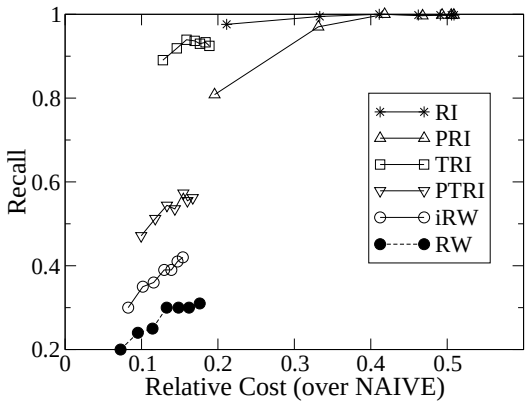

Figure 6: Extending the query effective range by 2.

y-axis shows the recall. Please note that seven different points for each algorithm are depicted, due to the different values of k being used (i.e., $k = 2, \ldots, 8$). RI-k provides between 92%-99% recall with a relative cost of 14%-29%. TRI-k, depending on k, provides additional savings (up to 1-10 over naive) in query execution with a small penalty in recall, which is in the range 83%-87%.

The probabilistic algorithms do not perform as well - in fact we can always obtain much higher recall with the same cost, using a deterministic algorithm, especially for small values of k. The differences in performance between the deterministic version of an algorithm and its probabilistic alternative are more evident in the threaded algorithms (TRI and PTRI). During a threaded execution, the probabilistic algorithm, which considers alternative routes through other neighboring nodes than the one with the most recent observation, often misses the target (the matching event), since, in most cases, the direction with the most recent observation is the best one to follow. However, we need to note

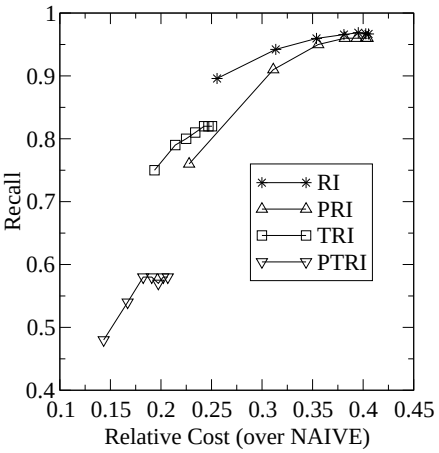

Figure 7: Comparison of different algorithms, ratio 3:1 of event frequencies (15 events of type "X", 5 events of type "Y").

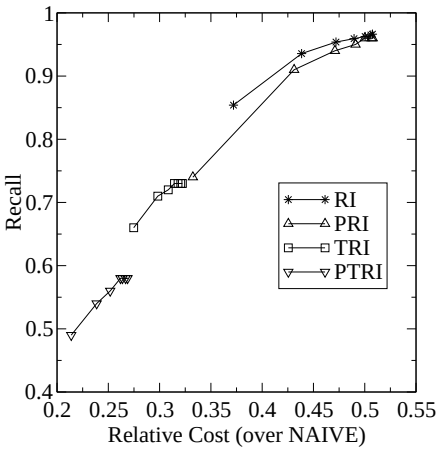

Figure 8: Comparison of different algorithms, increasing frequencies by a factor of 10

that for $k \geq 4$ the PRI-k algorithm provides results comparable to its deterministic counterpart. We also note that the Random Walk algorithms do not perform well. In particular, iRW performs worse than PTRI, as their main difference is that the latter considers the computed weights on the routing indices when selecting the next hop, while the former looks at the routing indexes simply for removing neighbors with non-existing entries from consideration. The even simpler RW algorithm fails to provide competitive performance.

By examining the traces of the algorithms we observe that most of the result tuples that are missing during the query evaluation are actually discovered during subsequent epochs, due to the continuous tuning of the indices. This means that in fact, these algorithms discover more matching tuples than the above values of recall indicate, with a delay of a few epochs. An easy workaround is to slightly increase the effective threshold d_{eff}, whose default value, as mentioned at the beginning of this section, was set equal to the proximity threshold minus 1, by 1 or 2. Increasing the effective threshold results in more nodes updating their indices when a new event is discovered and has no impact on the precision of the algorithm, since the proximity threshold defined by the user is used in Line 7 of the algorithm to inform the base station of matching events. In Figure 6 we repeat the experiment setting the query effective threshold equal to the proximity threshold plus 1 (d_{eff}=$d+1$). From this figure we can see that the change has a dramatic effect in increasing the recall, especially for the threaded algorithms For RI-k the minimum recall is now 97.3% when k=2, while is stays above 98.4% for larger values of k. Comparing these number with the case when $d_{eff} = d - 1$ we see that we can obtain better recall with fewer messages by switching to a smaller value of k, when we increase the effective threshold. In the rest of the experiments we again revert to conservatively using the default

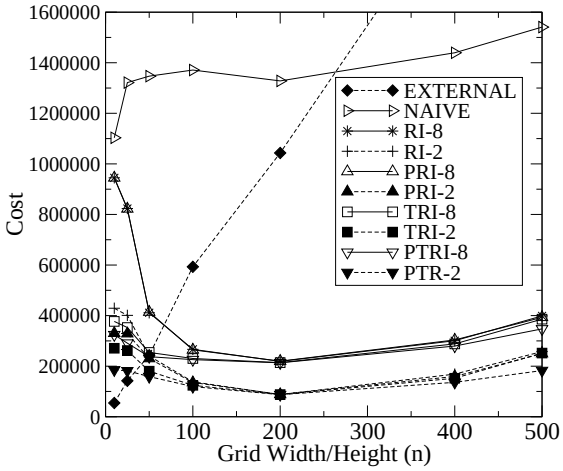

Figure 9: Cost, varying the size of the monitored area.

value for the effective threshold (i.e., equal to the proximity threshold minus 1) and note that a larger value would have resulted in larger recall values for all algorithms. Because of the poor performance of RW and iRW, we do not consider these two algorithms in the remainder of this section.

We now investigate the performance of the algorithms when the ratio of the observed events changes. In Figure 7 we repeat the experiment of Figure 5, but now use a different ratio 3:1 of the observed events (15 events of type "X" and 5 events of type "Y"). Compared to the case where all event types are equally frequent, we do not see significant differences in the results. The threaded algorithms perform slightly worse, as they have a harder time locating matching Y events that are less frequent. Similar observations hold for different frequency ratios we tested. However, we omit these results due to lack of space. In Figure 8 we used event types with the same frequency, but increase their number to 100, while using the same network area. Again the differences to the original set up are small.

In Figure 9 we show the effect of scaling the size of the monitored area by increasing the grid from 10×10 up to 500×500. We used a single proximity query $Q(\{X, Y\}, 5, 5)$ and the RandMoves data set with 10 objects of each type. In the graph we also show performance of the external processing algorithm (see Section 4.2) that simply transmits 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 because the cost of the EXTERNAL algorithm depends on (i) The number of detected events, which is not altered as the network size increases; and (ii) The average number of messages required to transmit the events to the base station. Thus, the linear dependency to the length/width n of our $n \times n$ grid that is clearly depicted

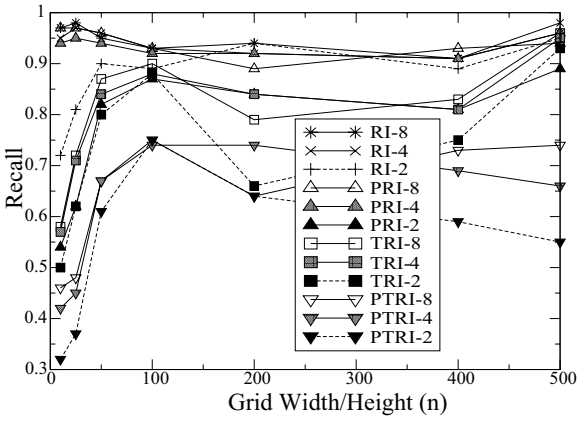

Figure 10: Recall, varying the size of the monitored area

in the figure is not surprising. As expected, the cost of NAIVE is bounded by $min(n^2, d^2)$. It is interesting that the class of algorithms that use routing indices (RI,PRI,TRI,PTRI) become more effective in pruning messages as the size of the network increases. Compared to NAIVE, we see a reduction in cost of up to 14-1 as the network size increases. This is explained as the increased sparsity of the space often resulted in routing indices with fewer than k entries, and thus smaller number of neighboring nodes included in each sensor's next hop list NH. In turn, the cost of the query actually drops slightly with larger grid sizes. For brevity, in the figure we only include numbers for k=2,4,8.

In Figure 10 we show the recall of these algorithms (recall is 100% for NAIVE and EXTERNAL) in the same run. We notice again that the deterministic algorithms perform better than their probabilistic counterparts.

In Figure 11 we depict the cost of evaluating multiple concurrent queries during 10,000 epochs for an 100×100 grid. The number of event types was 26. We used 10 event instances of each type and 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 and NAIVE are unaffected by the number of queries, as the first depends on the number of events and the second on the proximity threshold, which are both constant in this setup. The cost of the other algorithms increases slowly up to the point where the increased number of queries results in most events pairing up in a query. At that point, EXTERNAL is a viable alternative.

7.1. Experiments with Non-Grid Topologies

In all previous experiments, the sensor network topology was modeled as a grid. We also performed experiments using two alternative placements of the sensor nodes. In the first configuration, denoted as Random in Figure 12, we placed 6,000 nodes at random locations. In the second configuration, denoted as DisasterArea in the same Figure, we model a monitoring area with three

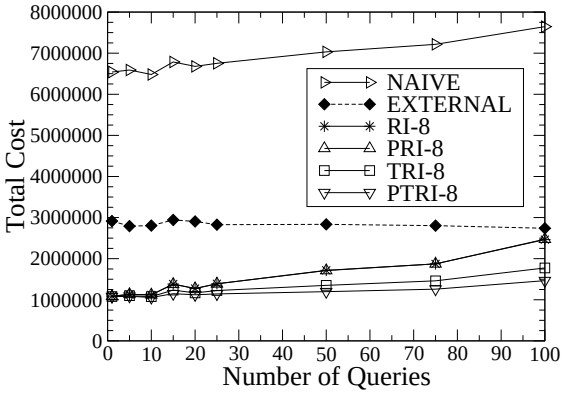

Figure 11: Varying the number of concurrent queries

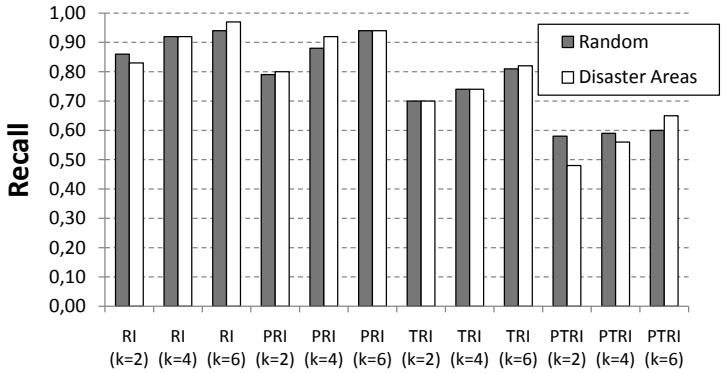

Figure 12: Utilizing a non-grid topology (Random) and a topology with failures (Disaster Areas)

disaster zones. Each disaster zone is centered around a disaster point (three random locations were selected for that purpose) and starting from a uniform allocation of 10,000 nodes, we randomly removed 4,000 sensor nodes selected with 80% probability from around a disaster point and with 20% from the rest of the network. We tested the RandMoves data set, however, in order to provide a meaningful comparison, we only allowed events in areas where at least one sensor node was present. The results in Figure 12 once again demonstrate that the RI-k algorithm provides the higher accuracy, which is increased with higher values of k.

7.2. Experiments with Real Data Sets

In Figure 13 we compare the various algorithms, varying the parameter k for the Trucks data set and for d=10. The network consisted of 10,000 nodes randomly spread over the monitored area. The same run is repeated in Figure 14

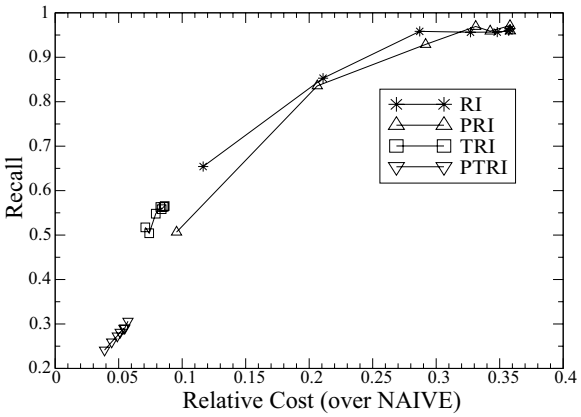

Figure 13: Trucks data set: Comparison of different algorithms, varying k

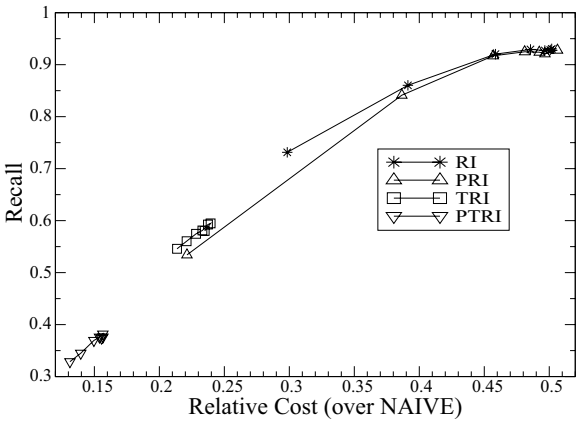

Figure 14: SchoolBuses data set: Comparison of different algorithms, varying k

for the SchoolBuses data set. Again, the different points in each line correspond to different values of parameter k from 2 up to 8, left to right. Again we observe that the RI and PRI algorithms achieve high recall rates while using only a fraction of the bandwidth required by a straightforward implementation.

7.3. Energy Drain During Query Evaluation

Up to this point we have not concerned ourselves with the energy drain on the nodes during the processing of the queries. In all experiments we assumed that the nodes have enough power supply to process and communicate events during the course of the query. In practice, battery-powered sensor nodes often face severe energy restrictions since they have to operate unattended for long periods. In the next experiment we modeled the energy drain on the nodes as follows. At network initialization, each node was given a fixed amount of

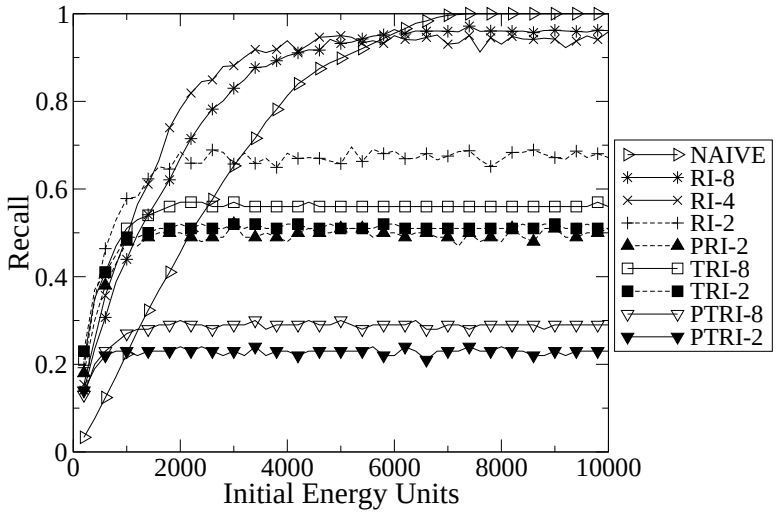

Figure 15: Recall varying the initial energy of nodes

energy units. During processing of the queries we drained the node's supply with 1 energy unit for each message received or overheard and 1.4 energy units for each message transmitted. We also accounted for the energy drain associated with detecting/processing an event and for managing the routing indices with one tenth of an energy unit.[4]

In Figure 15 we repeat the experiment varying the initial amount of energy allotted per node (x axis) for the trucks data set. The y axis shows the cumulative recall computed for 2,000 epochs. For clarity purposes the PTRI-8 algorithm was omitted because it closely matched the RI-8 algorithm, while the TRI-4, PRI-4 and PTRI-4 algorithms were omitted because their behavior can be easily understood based on the behavior of the corresponding algorithms for k values of 2 and 8. The following observations are made from the graph:

1. Threaded algorithms are less energy demanding and thus work better in extremely constrained scenarios.
2. In contrast, the NAIVE algorithm requires significant resources. One needs to allocate 6,000 energy units (or equivalently 3 units/epoch) for it to exceed the average recall achieved by other algorithms. This amount of energy is proportional to receiving 3 messages per epoch, or, equivalently making 2 transmissions per epoch.
3. The RI-k algorithms seem to better utilize energy drain in a larger spectrum of initial energy values and dominate all other techniques.

[4]This is probably an overestimate. For the slow-CPU Mica motes, the cost of sending one bit equals the cost of 1,000 operations [41]. For faster CPUs the ratio is a lot higher.

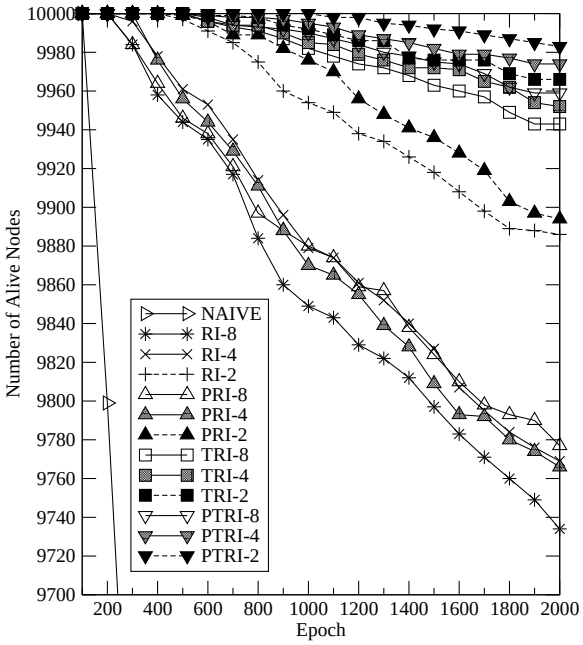

Figure 16: Number of sensors with non-empty battery cells during execution of the query

In Figure 16 we plot the number of nodes (the initial network has 10,000 sensor nodes) over time that have not yet drained their battery during the execution of the query. The X axis depicts the time, ranging from epoch 0 (query initialization) up to 2,000 (last epoch). The initial amount of energy allotted per node was 500 units. As expected the NAIVE flooding algorithm drains more energy and results in a quick drop in the number of sensors even from the first few epochs. While its entire behavior is not depicted in the figure (for clarity reasons - in order to distinguish the differences amongst the other algorithms), the number of alive nodes using this technique quickly falls to 9471, for 400 epochs, and then gradually decreases (at a smaller rate) to 8899 nodes, for 1300 epochs, and 8640 alive nodes for 2000 epochs.

It is interesting to note that the probabilistic algorithms perform better than their deterministic counterparts. The selection of neighboring nodes using probabilistic techniques results in a more uniform energy consumption by the sensor nodes and, thus, in increased lifetime of the network. Thus, we need to note that since the *PRI-k* algorithm performs close to the *RI-k* algorithm, in terms of recall, for k values equal or larger than 4, it is a more plausible alternative for energy-constrained environments.

7.4. Probabilistic v.s. Deterministic Algorithms - a Compromise

In the experiments in Figures 15 and 16 we observed that the deterministic algorithms provide higher values of recall, at the cost of increased energy

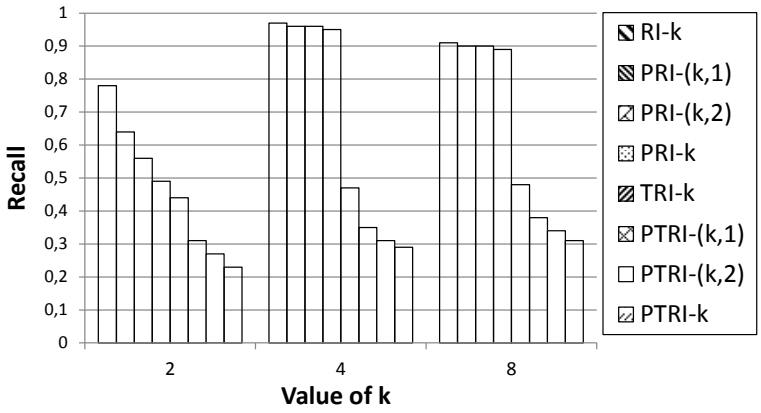

Figure 17: Recall - SchoolBuses Data Set

consumption, when compared to their probabilistic alternatives. In order to provide a compromise between the two approaches, we introduce a restricted version of the probabilistic techniques, termed as PRI-(k,l) and PTRI-(k,l) (for the webnet and threaded algorithms, respectively). Parameter l is used to restrict the choices that the probabilistic algorithms make among the top-(k+l) neighbors of a node, where the neighbors are sorted in descending order of their compound weights. Please note that for l=0, the algorithms behave similarly to their deterministic counterparts, while for $l = \infty$ (in practice, for a value of l greater than the maximum number of neighbors that a node has minus k) their selections are similar to the ones of their unrestricted versions. For brevity, we focus our discussion in the threaded algorithms - the results are similar for PRI-(k,l).

In Figure 17 we depict the recall of the techniques (y-axis), varying k for the SchoolBuses data set. As expected, for l=0 the recall of PTRI-(k,0) is similar to the recall of TRI-k and is omitted from the Figure. By increasing the value of parameter l, the probabilistic algorithm selects the next hop(s) for announcing an event from a larger pool of neighbors and its recall values drop. This happens because in this data set, these extra neighbors considered, are less likely to lead to a matching event since events (trucks) in our data set move along predefined routes and do not appear/disappear randomly.

Figure 18 plots the number of sensor nodes that are alive at the end of the experiment when the initial amount of energy allotted to each node was 500. By comparing Figures 17 and 18, the following trend appears: using the l parameter, we can balance the desired level of accuracy and the energy drain. In particular, a smaller value of l increases the accuracy at the cost of higher energy drain, while a larger value of l reduces accuracy but also reduces energy consumption.

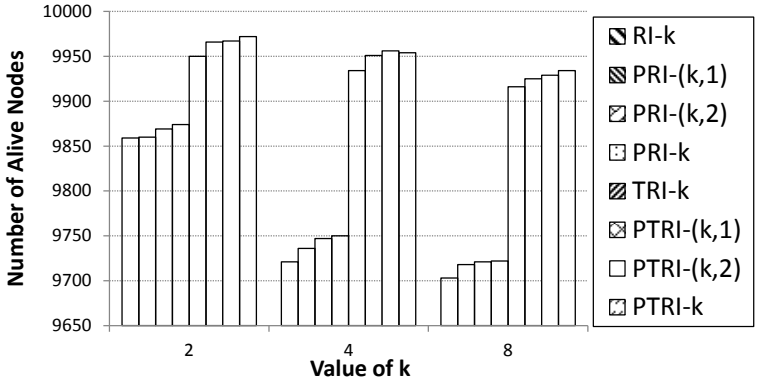

Figure 18: Number of sensors with non-empty battery cells - SchoolBuses Data Set

8. Conclusions

In this paper we introduced proximity queries as a means of detecting interesting events, which 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 consisting of battery-powered wireless nodes and proposed efficient distributed algorithms that utilize routing information computed accordingly at each node in the network. A key point in our framework is that the routing indexing weights are populated during past observation of the events and do not require expensive maintenance. Moreover, our algorithms do not assume any specific sensor topology and can also be applied to cases of clustered organizations.

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. Furthermore, by increasing the query effective threshold we can achieve recall values exceeding 99.9% while still being able to provide substantial savings in the cost of the query. An important observation from our experimental evaluation is that our techniques scale better when the size of the network increases. This means that our algorithms favor large scale surveillance applications including tens or hundreds of thousands of nodes and can handle many concurrent parallel investigations.

References

[1] Rtree Pportal.
http://www.rtreeportal.org.

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

[3] M. Aly, P. K. Chrysanthis, and K. Pruhs. Decomposing Data-Centric Storage Query Hot-spots in Sensor Networks. In *MOBIQUITOUS*, 2006.

[4] P. Andreou, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras. Workload-Aware Query Routing Trees in Wireless Sensor Networks. In *MDM*, pages 189–196, 2008.

[5] C. Avin and C. Brito. Efficient and robust query processing in dynamic environments using random walk techniques. In *Proceedings of the 3rd international symposium on Information processing in sensor networks (IPSN)*, pages 277–286, 2004.

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

[7] D. Braginsky and D. Estrin. Rumor Routing Algorithms for Sensor Networks. In *First International Workshop on Sensor Networks and Applications*, 2002.

[8] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *INFOCOM*, 2002.

[9] Jae-Hwan Chang and Leandros Tassiulas. Energy Conserving Routing in Wireless Ad-hoc Networks. In *INFOCOM*, 2000.

[10] I. Chatzigiannakis, A. Kinalis, and S. E. Nikoletseas. Fault-tolerant and efficient data propagation in wireless sensor networks using local, additional network information. *J. Parallel Distrib. Comput.*, 67(4):456–473, 2007.

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

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

[13] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. In *ICDCS*, 2002.

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

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

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

[17] A. Deligiannakis, Y. Kotidis, V. Vassalos, V. Stoumpos, and A. Delis. Another Outlier Bites the Dust: Computing Meaningful Aggregates in Sensor Networks. In *ICDE*, pages 988–999, 2009.

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

[19] C. Doulkeridis, A. Vlachou, Y. Kotidis, and M. Vazirgiannis. Peer-to-Peer Similarity Search in Metric Spaces. In *VLDB*, pages 986–997, 2007.

[20] D. Dudkowski, P. J. Marrón, and K. Rothermel. Efficient Algorithms for Probabilistic Spatial Queries in Mobile Ad Hoc Networks. In *COMSWARE*, 2006.

[21] D. Dudkowski, P. J. Marrón, and K. Rothermel. An Efficient Resilience Mechanism for Data Centric Storage in Mobile Ad Hoc Networks. In *MDM*, 2006.

[22] C. T. Ee, S. Ratnasamy, and S. Shenker. Practical Data-centric Storage. In *NSDI*, pages 24–24, 2006.

[23] F. Emekçi, S. E. Tuna, D. Agrawal, and A. E. Abbadi. Using Linear Models to Monitor the Physical World with Sensors. In *SSDBM*, 2005.

[24] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *MobiCOM*, 1999.

[25] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, and Y. Theodoridis. TACO: Tunable Approximate Computation of Outliers in Wireless Sensor Networks. In *SIGMOD Conference*, pages 279–290, 2010.

[26] B. Greenstein, S. Ratnasamy, S. Shenker, R. Govindan, and D. Estrin. DIFS: A Distributed Index for Features in Sensor Networks. *Ad Hoc Networks*, 1(2-3):333–349, 2003.

[27] 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.

[28] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidermann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *ICDCS*, 2002.

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

[30] S. R. Jeffery, M. Garofalakis, and M. J. Franklin. Adaptive Cleaning for RFID Data Streams. In *Proc. of VLDB*, 2006.

[31] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MOBICOM*, pages 243–254, 2000.

[32] P. Karras and N. Mamoulis. Detecting the Direction of Motion in a Binary Sensor Network. In *SUTC*, 2006.

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

[34] H.M. Khan, S. Olariu, and M. Eltoweissy. Efficient Single-Anchor Localization in Sensor Networks. In *Proc. of the 2nd IEEE Workshop on Dependability and Security in Sensor Networks and Systems*, 2006.

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

[36] Y. Kotidis. Processing proximity queries in sensor networks. In *Proceedings of the 3rd International VLDB Workshop on Data Management for Sensor Networks (DMSN)*, September 2006.

[37] L. Lazos and R. Poovendran. Stochastic Coverage in Heterogeneous Sensor Networks. *ACM Trans. Sen. Netw.*, 2(3):325–358, 2006.

[38] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional Range Queries in Sensor Networks. In *SenSys*, pages 63–75, 2003.

[39] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS*, 2002.

[40] 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.

[41] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *ACM SIGMOD*, 2003.

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

[43] K. K. Rachuri and C. S. Ran Murthy. Level biased random walk for information discovery in wireless sensor networks. In *Proceedings of the 2009 IEEE international conference on Communications (ICC)*, pages 5036–5041, Piscataway, NJ, USA, 2009.

[44] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[45] Y. Rekhter and T. Li. Border Gateway Protocol 4. RFC 1771, July 1995.

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

[47] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric Storage in Sensornets. *Computer Communication Review*, 33(1):137–142, 2003.

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

[49] A. Silberstein, A. Gelfand, K. Munagala, G. Puggioni, and J. Yang. Making Sense of Suppressions and Failures in Sensor Data: A Bayesian Approach. In *VLDB*, pages 842–853, 2007.

[50] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *ACM/IEEE International Conference on Mobile Computing and Networking*, 1998.

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

[52] D. Tsoumakos and N. Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. In *Proc. of the 3rd IEEE International Conference on P2P Computing*, 2003.

[53] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *INFOCOM*, pages 585–594, 2000.

[54] P. Xia, P. K. Chrysanthis, and A. Labrinidis. Similarity-aware Query Processing in Sensor Networks. In *IPDPS*, 2006.

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

[56] T. Yan, Y. Gu, T. He, and J. A. Stankovic. Design and Optimization of Distributed Sensing Coverage in Wireless Sensor Networks. *Trans. on Embedded Computing Sys.*, 7(3):1–40, 2008.

[57] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *ICDCS*, pages 5–14, 2002.

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

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

[60] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks. In *INFOCOM*, 2004.

[61] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, and G. Samaras. MINT Views: Materialized In-Network Top-k Views in Sensor Networks. In *Proceedings of the 7th International Conference in Mobile Data Management*, pages 182–189, May 2007.

[62] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, G. Samaras, and A. Pitsillides. The Micropulse Framework for Adaptive Waking Windows in Sensor Networks. In *MDM*, pages 351–355, 2007.