SenseSwarm: A Perimeter-based Data Acquisition Framework for Mobile Sensor Networks

Demetrios Zeinalipour-Yazti, Panayiotis Andreou[‡], Panos K. Chrysanthis[⋆], George Samaras[‡],

Open University of Cyprus, P.O. Box 24801, 1304 Nicosia, Cyprus
[‡] University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus
[⋆] University of Pittsburgh, Pittsburgh, PA 15260, USA
zeinalipour@ouc.ac.cy, cs98ap1@ucy.ac.cy, panos@cs.pitt.edu, cssamara@ucy.ac.cy

ABSTRACT

This paper assumes a set of n mobile sensors that move in the Euclidean plane as a swarm[1]. Our objectives are to explore a given geographic region by detecting and aggregating spatio-temporal events of interest and to store these events in the network until the user requests them. Such a setting finds applications in environments where the user (i.e., the *sink*) is infrequently within communication range from the field deployment. Our framework, coined *SenseSwarm*, dynamically partitions the sensing devices into *perimeter* and *core* nodes. Data acquisition is scheduled at the perimeter in order to minimize energy consumption while storage and replication takes place at the core nodes which are physically and logically shielded to threats and obstacles. To efficiently identify the perimeter of the swarm we devise the *Perimeter Algorithm (PA)*, an efficient distributed algorithm with a message complexity of $O(p+n)$, where p denotes the number of nodes on the perimeter and n the overall number of nodes. For storage and replication we devise a spatio-temporal in-network aggregation scheme based on minimum bounding rectangles and minimum bounding cuboids. Our trace-driven experimentation shows that our framework can offer significant energy reductions while maintaining high data availability rates.

1. INTRODUCTION

Stationary sensor networks have been predominantly used in applications ranging from environmental monitoring [20, 18] to seismic and structural monitoring [5] as well as industry manufacturing [14]. Recent advances in distributed robotics and low power embedded systems have enabled a new class of *Mobile Sensor Networks (MSNs)* that can be utilized for land [2, 6, 11], ocean [12] and air [7] exploration and monitoring. MSNs have a similar architecture to their stationary counterparts, thus are governed by the same energy and processing limitations, but are supplemented with implicit or explicit mechanisms that enable these devices to move in space (e.g., motor or sea/air current).

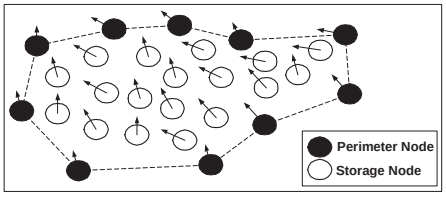

Figure 1: *SenseSwarm*: Data Acquisition takes place at the virtual perimeter while core nodes act as storage nodes for the acquired events.

The main advantages of MSNs over their stationary counterparts are that they can : i) control the deployment, thus provide optimal and flexible coverage of a given region; and ii) dynamically repair the network, thus eliminating bottleneck and low energy nodes. On the other hand, the absence of a stationary network structure in MSNs makes continuous data acquisition to some sink point a non-intuitive task. In particular, the absence of an always accessible sink mandates that acquisition has to be succeeded by in-network storage of the acquired events [23, 19, 16, 1], so that these events can later be retrieved by the user.

In this paper we propose *SenseSwarm*, a novel framework for the acquisition and storage of spatio-temporal events in MSNs. In SenseSwarm, nodes have the dual role of *perimeter* and *core* nodes (see Figure 1). Data acquisition is scheduled at the perimeter, in order to minimize energy consumption, while storage and replication takes place at the core nodes. Such a setting is suited well for applications in which new events are more prevalent at the periphery of the swarm (e.g., water and contamination detection) rather than for online monitoring applications (e.g., fire detection) or for applications where new events might occur anywhere in a given swarm. Note that the storage of the detected events takes place at the core nodes since these nodes are expected to feature a longer lifetime (due to their reduced sensing activity) but are also physically shielded to threats and obstacles that might immobilize the sensors.

In order to instantiate the problem setting and motivate our description assume the following *Mars Exploration* scenario: Spirit was one of the two rovers deployed by NASA

[1]The term *Swarm (or Flock)* in this paper refers to a group of objects that exhibit a polarized, non-colliding and aggregate motion.

Proceedings of the 4th International Workshop on Data Management for Sensor Networks (DMSN'07), Vienna, Austria, 2007.

in 2004 in order to perform geological analysis of the red planet. Instead of one rover, consider a design that consists of many cheaper rovers deployed as a swarm. Such a design avoids the peculiarities of individual rovers, is less prone to failures and is potentially much cheaper. The swarm moves together and attempts to detect events of interest (e.g., the presence of water). We assume that either an explicit algorithm [17] or an implicit mechanism (e.g., air current) provides the polarized behavior to the swarm. The operator (on earth) then infrequently posts the question: *"Has the swarm identified any water and where exactly?"*. Due to expensive communications to the remote planet such a query is not performed continuously. Thus, the swarm collects spatio-temporal events of interest and stores them in the swarm until the operator requests them. In order to increase the availability of the detected answers, in the presence of unpredictable failures, individual sensors perform replication of detected events to neighboring nodes.

Similarly to the above description, we could draw another example in the context of an ocean monitoring environment: assuming n independent surface drifters floating on the sea surface and equipped with either acoustic or radio communication capabilities, the operator seeks to answer the query: *"Has the swarm identified an area of contamination and where exactly?"*. Finally, one could utilize a swarm of car robots, such as CotsBots [2], Robomotes [6] or Millibots [11], to construct spatio-temporal acquisition and storage scenarios for land applications.

Contributions

In this paper we make the following contributions:

- We propose a novel data acquisition framework for MSNs that utilizes the notion of a virtual perimeter. We additionally devise a distributed algorithm for the efficient construction of such a perimeter in a MSN.
- We devise an in-network spatio-temporal aggregation scheme based on Minimum Bounding Rectangles and Minimum Bounding Cuboids which approximates compactly events of interest in MSNs.

The remainder of the paper is organized as follows: Section 2 formalizes our system model and assumptions, Section 3 overviews the related research work. Section 4 introduces the components and algorithms of the SenseSwarm Framework. Section 5 presents our experimental study and Section 6 concludes the paper.

2. SYSTEM MODEL AND ASSUMPTIONS

In this section we will formalize our basic terminology and assumptions upon which we will base our description. The main symbols and their respective definitions are summarized in Table 1.

Let $\Re \times \Re$ denote a two-dimensional grid of points in the Euclidean plane that discretizes a given geographic area. Also assume a Cartesian coordinate system to describe each point in the grid by the tuple (x, y). Without loss of generality, let us initially configure n sensing devices $\{s_1, s_2, ..., s_n\}$ in the lower-left $n^{\frac{1}{2}} \times n^{\frac{1}{2}}$ sub-grid of $\Re^2$. For ease of exposition let n be a perfect square such that each cell contains exactly one sensor.

Each s_i $(i \leq n)$ can derive its coordinates (s_i^x, s_i^y) through absolute (e.g., dedicated Geographic Positioning System hardware) or relative means (e.g., *localization techniques* which enable sensing devices to derive their coordinates using the signal strength, time difference of arrival or angle of arrival). For instance on Mars there is no satellite coverage thus engineers can opt for a relative coordinate system rather than an absolute one. Additionally, each s_i can be aware of its neighboring nodes, denoted as $NH(s_i)$, using a local 1-hop broadcast.

Symbol	Definition
n	Number of Sensors $\{s_1, s_2, ..., s_n\}$
m	Number of Attributes at each s_i $\{a_1, a_2, ..., a_m\}$
s_i	Sensor with identifier i.
(s_i^x, s_i^y)	X and Y coordinates of sensor s_i.
r	Radius of communication for s_i.
$NH(s_i)$	1-hop (in commun. range) neighbors of s_i
$V(s_i, s_j)$	A Vector defined as $(s_j^x - s_i^x, s_j^y - s_i^y)$
Q	An m-dimensional Query

Table 1: Definition of Symbols

The sensing devices are *coarsely synchronized* through some operating system mechanism (e.g., similarly to TinyOS [8]) or through the GPS and can communicate with other sensors in a uniform radius r, i.e., $1 \leq r \ll n^{\frac{1}{2}}$.

A sensor s_i $(i \leq n)$ can acquire m physical parameters A=$\{a_1, a_2, ..., a_m\}$ from its environment at every discrete chronon t. This generates spatio-temporal tuples of the form $\{t, x, y, a_1, a_2, ..., a_m\}$ locally at each sensor. The user can specify one or more m-dimensional Boolean queries of the type Q=$\{q_1 \odot q_2 \odot ... \odot q_m\}$, where q_i $(i \leq m)$ corresponds to some predicate such as q_1="$Temperature > 100$" and $\odot$ denotes some binary Boolean operator. These queries correspond to the user-defined local events of interest and are injected in each s_i either prior the deployment or during execution.

3. BACKGROUND AND RELATED WORK

This section provides an overview of other data acquisition frameworks and tools as well as background on the perimeter construction problem.

Traditional data acquisition frameworks for sensor networks, such as TinyDB [13] and Cougar [21], perform a combination of in-network aggregation and filtering in order to reduce the energy consumption while conveying data to the sink. The MINT View framework [22] additionally performs in-network top-k pruning in order to further reduce the consumption of energy.

In *data centric routing*, such as directed diffusion [9], low-latency paths are established between the sink and the sensors. Contrary to our approach, all the above frameworks have been proposed for stationary sensor networks while in this work we consider the challenges of a mobile sensor network setting.

In *data centric storage* [19, 16, 1] schemes, data with the same name (e.g., humidity readings) are stored at the same node in the network offering therefore efficient location and retrieval. Such an approach is supplementary to the perimeter-based data acquisition framework we propose in this paper. Supplementary to our framework are also the MicroHash [23] and TINX [15] local index structures, which provide $O(1)$ access to data stored on the local flash media of a sensor device. Such index structures can be deployed to speed up the retrieval of data whenever the operator performs a query.

The notion of a virtual perimeter is a main concept in the SenseSwarm framework as the identified core nodes can conserve energy by withdrawing temporarily from the acquisition (i.e., by operating in low-power mode). This is possible because perimeter nodes physically and logically circumvent the core nodes. Note that the perimeter construction problem we consider, is similar to the convex hull problem in computational geometry that finds application in pattern recognition, image processing and GIS [4]. The convex hull problem is defined as follows: *given a set of points, identify the boundary of the smallest convex region that encloses all the points either on the boundary or on its interior.* Such a boundary is both *non-intersecting* (i.e., no edge crosses any other edge) and *convex* (i.e., all internal angles are less than π). Note that there are numerous centralized algorithms for computing the convex hull with varying complexities. Two of the most popular convex hull algorithms are the *Jarvis March* [4] (or Gift Wrapping) and the Graham's scan [4].

The main difference between the convex hull and the perimeter problem we consider is that the latter defines non-convex cases (i.e., internal angles are up to 2π). Non-convex cases are typical for a sensor network context as convex angles might not be feasible due to communication radius constraints. Additionally, convex hull algorithms are centralized while we develop techniques to compute the perimeter in a distributed fashion minimizing communication and energy consumption without sacrificing correctness.

Related work in the context of sensor networks appears in [3], where the authors present localized techniques that enable sensors to determine whether they belong to the boundary of some phenomenon. The underlying assumption in the given work is that the edge sensors are not within communication range while we consider the perimeter to be continuous (which thus yields a different set of edge sensors).

4. THE SENSESWARM FRAMEWORK

In this section we describe the underlying algorithms of the SenseSwarm Framework. For ease of exposition, we present our framework in the following three conceptual phases:

A. The *Perimeter Construction Phase*, executed every σ chronons during which the n sensors execute our perimeter construction algorithm in order to conceptually divide S into perimeter sensors S^p and core sensors S^c;
B. The *Acquisition Phase*, executed continuously by S^p nodes which attempt to identify answers to Q and store these answers locally; *and*
C. The *Replication Phase*, executed every $\sigma'{=}\alpha * \sigma$ ($\alpha \geq 1$) chronons during which S^p nodes replicate local answers to their surrounding S^p and S^c nodes using an in-network spatio-temporal aggregation scheme.

4.1 Perimeter Construction Phase

This subsection describes algorithms for the construction of a perimeter in a sensor network. We first describe a centralized solution and then our Perimeter Algorithm.

Centralized Perimeter Algorithm (CPA): First note that the construction and dissemination of a perimeter can be performed in a centralized manner, i.e., a sink collects the coordinates of all nodes in S, using an ad-hoc spanning tree, and then identifies the perimeter nodes (S^p) using some straightforward geometric calculations. Finally, the sink disseminates the ordered set S^p to all nodes in S using a spanning tree. Clearly, the first and last phase of the CPA algorithm require the transfer of many (x, y)-pairs between nodes. Specifically, although both phases require O(n) messages the first phase requires the transfer of O(n^2) (x, y)-pairs (i.e., assume that the nodes are connected in a bus topology which yields $\sum_1^n(i){=}\frac{n(n+1)}{2}$ (x, y) pairs), while the last phase requires the transfer of O($p * n$) (x, y)-pairs (i.e., each edge transfers the complete perimeter of size p).

Algorithm 1 : Perimeter Algorithm (PA)

Input: A set of sensors $S = \{s_1, s_2, \cdots s_n\}$/
Output: Disjoint sets S^p (perimeter nodes) and S^c (core nodes)

```
1: procedure PERIMETER_ALGORITHM(S)
2:     minAngle=360°; // Variable initialization
3:     // Identify s_min (node w/ minimum y-coordinate in S).
4:     s_min = Find_Min_Coordinates(S);
5:     // Disseminate s_min to the network S.
6:     Disseminate(s_min, S); // ∀s_i ∈ S
7:     for i=1 to n do
8:         if (s_i = s_min) then
9:             LeftN(s_i)=s_min;
10:        else
11:            LeftN(s_i)=wait(); // Get token from LeftN(s_i).
12:        end if
13:        // Find neighbor with min. polar angle from s_i
14:        for j=1 to |NH(s_i)| do
15:            if (∠(LeftN(s_i), s_i, s_j)≤minAngle) then
16:                minAngle=∠(LeftN(s_i), s_i, s_j));
17:                RightN(s_i)=s_j
18:            end if
19:        end for
20:    end for
21:    Send(s_i, RightN(s_i)); // Send token to RightN(s_i).
22: end procedure
```

Perimeter Algorithm (PA): We shall next describe our distributed algorithm which minimizes the transfer of (x, y)-pairs, thus minimizing energy consumption. To simplify the description and w.l.o.g., assume that we have no *coincidents* (i.e., two points with the same (x, y) coordinates) and that no three points are *collinear* (i.e., lie on the same line). Although these assumptions make the discussion easier our implementation elaborately supports them.

Algorithm 1 presents the steps of the distributed PA process that is executed every σ chronons. In line 4, a randomly chosen sink identifies the minimum y-coordinate (denoted as s_{min}). This is achieved by constructing an aggregation tree rooted at the given sink using TAG [14]. In particular, each s_i identifies among its children and itself the minimum s^y_{min} value and then recursively forwards the triple $(s_{min}, s^x_{min}, s^y_{min})$ to s_i's parent. This step, has similarly to CPA, a message complexity of $O(n)$ but the overall number of (x, y)-pairs transmitted to the sink is only $O(n)$ rather than $O(n^2)$ (i.e., exactly one pair per edge). This improvement is due to the in-network aggregation that takes place in our approach.

Concurrently with the operation in line 4, each s_i updates its neighbor list $NH(s_i)$ as such an updated list will be necessary in the subsequent steps. Note that this update does not introduce any extra cost, as s_i simply adds to $NH(s_i)$ the neighbors that have participated in the calculation of s_{min}.

In line 6, we disseminate s_{min} to all the nodes in the network S from the sink. This has a message complexity of $O(n)$ and the overall number of (x, y)-pairs transmitted is $O(n)$, compared to $O(p * n)$ required by CPA. The next

task is to identify the nodes on the perimeter. Before we proceed, let us define the left and right neighbors of s_i:

Definition 1 [Left Neighbor of s_i ($LeftN(s_i)$)]: The predecessor of s_i on the perimeter. The termination condition of this recursive definition is as follows: $LeftN(s_{min}) = s_{min}$, where $s^y_{min} \leq s^y_j$ $(\forall s_j \in S, 1 \leq j \leq n)$.

Definition 2 [Right Neighbor of s_i ($RightN(s_i)$)]: The successor of s_i on the perimeter such that $LeftN(s_i) \neq RightN(s_i)$, if $|NH(s_i)| > 1$.

Continuing with the description of our algorithm in lines 8-12 each s_i, other than s_{min}, identifies its left neighbor. This is achieved by waiting for a token (i.e., the identifier of $LeftN(s_i)$) from $LeftN(s_i)$. When the token arrives, the node will execute the remaining steps of the algorithm (lines 13-21). In particular, in lines 13-19, s_i identifies the neighbors with the minimum polar angle from its x-axis. The x-axis of node s_i is defined in our context to be collinear with the vector $V(LeftN(s_i), s_i)$. This ensures the correctness of the algorithm although we omit a formal proof due to space limitations. In line 15 we utilize the notation $\measuredangle(a, b, c)$ to denote the angle between three arbitrary points a, b, c in the plane. Our objective in the given block (line 14-19), is to identify the neighbor with the minimum polar angle (which is then coined $RightN(s_i)$). Finally in line 21, s_i transmits a token to $RightN(s_i)$ notifying it that it is the next node on the perimeter. The procedure between lines 13-21 continues sequentially along the network perimeter until any s_i receives the token for a second time from its left neighbor. At the end, every node receiving the token knows that it belongs to S^p while the rest nodes continue to belong to S^c.

The identification of s_{min} takes $O(n)$ messages and the token dissemination takes $O(p)$ messages, where p the number of the nodes in the perimeter. Thus the overall message complexity is $O(p + n)$. In the future we plan to devise techniques to incrementally compute the perimeter.

Example: Figure 2 illustrates the perimeter construction for eight nodes $\{s_1 \cdots s_8\}$. Assume that we have executed steps 2-6 of Algorithm 1 and that we continue with the execution of the perimeter construction at node s_{min} (i.e., s_1). s_{min} measures the polar angle of all the nodes in $NH(s_{min})$ to its x-axis and subsequently derives $RightN(s_{min})$=2 (s_3 is not within communication range from s_1). Next, s_{min} sends a token to s_2 informing it that it is the next node on the perimeter. Upon reception of the token, s_2 sets its x-axis collinear with $V(s_1, s_2)$. The same idea applies to all nodes on the perimeter until s_8 transmits the token to s_1.

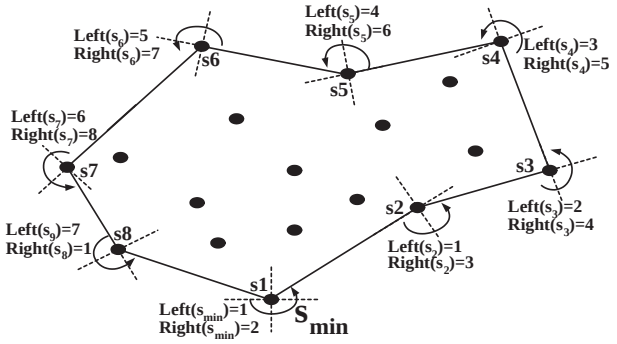

Figure 2: Execution of PA: The construction starts at s_{min} and proceeds counterclockwise.

4.2 Acquisition Phase

Once the perimeter partitioning has been conducted, S^p nodes start their sensing modules in order to detect an event that satisfies the predicate of the pre-registered Boolean query Q. The amount of time between two consecutive readings σ' is either a user-defined parameter (e.g., σ'=$\alpha * \sigma$, $\alpha \geq 1$) or is dynamically adjusted according to the dynamics of the swarm. In a sea oil-spill detection scenario σ' can be configured to several hours as surface drifters usually float slowly on the sea surface. Once a query answer Q^A_i is detected at node s_i, Q^A_i is stored on non-volatile storage of s_i using an efficient access method [23, 15]. Organizing answers using such a local structure will enable the efficient retrieval of records when requested by the user.

4.3 Data Replication Phase

After σ'=$\alpha * \sigma$ ($\alpha \geq 1$) chronons of acquisition, the framework proceeds to the data replication phase which ensures that a node failure will not subvert a detected event. In SenseSwarm we utilize a replication scheme that is based on approximations of where the events have occurred. In particular, we utilize 2-D and 3-D bounding rectangles and cuboids similar to those utilized in spatial index structures. In order to facilitate our presentation we proceed with an example that provides the intuition behind our approach.

Figure 3 (left) illustrates a segment of a MSN that consists of eight sensors $\{s_1, \cdots, s_8\}$. Assume that the sensors have been partitioned into S^p=$\{s_1, s_2, s_3\}$ and S^c=$\{s_4, \cdots, s_8\}$ nodes and that each s_i in S^p has detected exactly one answer to Q. Thus, we have the following conceptual set (horizontally segmented across the S^p nodes): $Q^A = \{(t, s^x_1, s^y_1), (t, s^x_2, s^y_2), (t, s^x_3, s^y_3)\}$.

If any of the S^p nodes gets corrupted then we will lose part of Q^A which is not desirable. The objective of the replication phase is to replicate Q^A in the network in an energy-efficient manner while offering the capability to later recover either an approximation of Q^A or the complete Q^A.

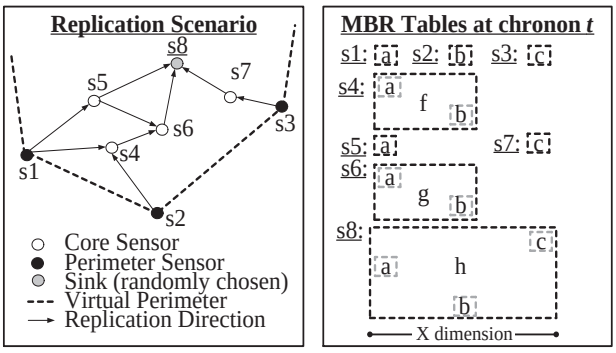

Figure 3: Replication Example.

The first step of the replication phase is to request from all sensors to transmit their local readings to a randomly identified sink that is part of S (i.e., s_8 in the example). While S^p nodes transmit their local answers towards their parent, intermediate S^c and S^p nodes identify the region that encloses all the answers from their children. Let us next provide a formal definition of the enclosed area:

Definition 3 [Minimum Bounding Rectangle of S']: A rectangle that encloses all points in S'. The Cartesian coordinates of the bounding box $MBR(S')$ are defined by

the following quadruple:

$$(min\{s_i^x\}, min\{s_j^y\}, max\{s_k^x\}, max\{s_l^y\}), [i, j, k, l \leq n]$$

The MBR is an approximation for a set of detected events S' and might encapsulate $|S'|$ events using only five numbers, i.e., $(ts, MBR(S'))$, as opposed to $(|S'|*2 + 1)$ numbers. That makes MBRs highly compact structures, enabling huge energy savings during replication. This is particularly true when $5 \ll |S'|$. Finally, note that when Q specifies some aggregate function, such as MIN, MAX, SUM or COUNT, then the aggregate answer ($aggr$) can easily be stored with the bounding box as $(t, x_1, y_1, x_2, y_2, aggr)$.

Figure 3 (right) illustrates the MBRs developed locally at each of the eight sensors. We observe that s_1 through s_3 know precisely where their events happened, thus the MBRs a, b and c are actually point coordinates. On the contrary, s_4 has an approximation of s_1's and s_2's answer (this is denoted as MBR f). The intuition is that even if both s_1 and s_2 fail, then the user will still be able to recover an approximation of where the event has occurred (i.e., through s_4 or some other node). On the same figure, we also notice that s_8 has an MBR which encapsulates all the events that have occurred.

Although our discussion has so far assumed that aggregation takes place only in space, it is straightforward to support spatio-temporal aggregation as well (i.e., using (x, y, ts)). In particular, we extend the definition of MBRs to *Minimum Bounding Cuboids (MBC)* (i.e., rectangular boxes). A MBC contains the coordinates of an event in space and time. Note that the MBC structure is not fundamentally different than the MBR structure, as it is represented again using two coordinates (i.e., 3D coordinates).

When a user performs a query, we collect the MBRs (or MBCs respectively) from all the nodes for the user-specified interval and intersect these boxes. This allows us to derive the coordinates of the points at which events have occurred.

5. EXPERIMENTAL EVALUATION

In this section we present our experimental evaluation of the SenseSwarm framework.

5.1 Experimental Methodology

We adopt a trace-driven experimental methodology in which a real dataset from n sensors is fed into our custom-built simulator. Our methodology is as follows:

Sensing Device: We use the energy model of Crossbow's new generation TelosB [5] sensor device to validate our ideas. Our performance measure is $Energy(Joules) = Volts \times Amperes \times Seconds$ that is required at each discrete chronon to resolve the query.

Dataset: We utilize a real dataset from Intel Berkeley Research [10]. This dataset contains data that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley between February 28th and April 5th, 2004. The motes utilized in the deployment were equipped with weather boards and collected time-stamped topology information along with humidity, temperature, light and voltage values once every 31 seconds. The dataset includes 2.3 million readings collected from these sensors.

Swarm Simulation: In order to introduce a movement into our sensor network we have derived synthetic temporal coordinates for the n sensors using the Craig Reynolds algorithm [17], which is widely used in the computer graphics community. Using this algorithm we generated 100 individual scenes and during each scene a sensor obtains 100 readings (i.e., σ=σ'=100). In order to simulate failures we make the assumption that there is a 0.2 independent probability that a node fails at any given timestamp.

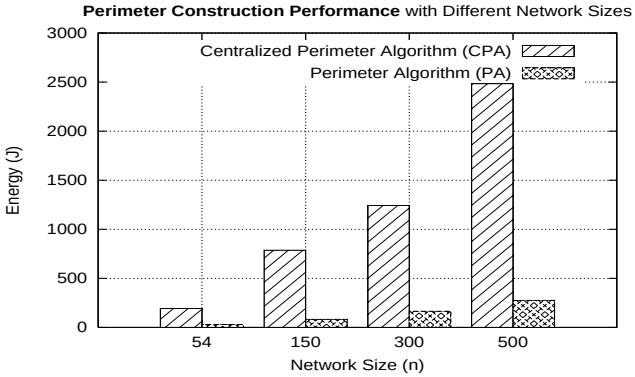

Figure 4: Evaluating the perimeter construction.

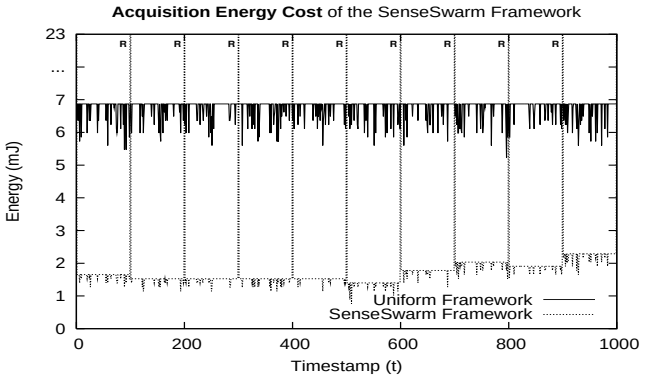

Figure 5: Evaluating the SenseSwarm framework.

5.2 Perimeter Cost Evaluation

In the first experimental series we investigate the efficiency of our distributed PA algorithm compared to the centralized CPA algorithm. Figure 4 presents the aggregate cost (i.e., for the whole network for all 10,000 timestamps) of the two algorithms for 4 different network sizes 54, 150, 300 and 500. These networks were derived from the initial dataset of 54 nodes using replication of the sensor readings to different initial coordinates. We observe that the PA algorithm consumes in all cases between 85%-89% less energy than the CPA algorithm. This is attributed to the fact that during the computation of s_{min}, the PA algorithm intelligently percolates only one (x, y)-pair to the sink rather than all of them. Additionally, we observe that the performance gap between the two algorithms grows substantially with the size of the network. Specifically, for n=54 the total energy difference between the two algorithms was 163 Joules while for n=500 the total energy difference was 2,208 Joules.

5.3 Acquisition Cost Evaluation

In the next experiment, we measure the cost of operating a SenseSwarm network. As a baseline of comparison we utilize the *Uniform framework*, one in which all 54 sensing

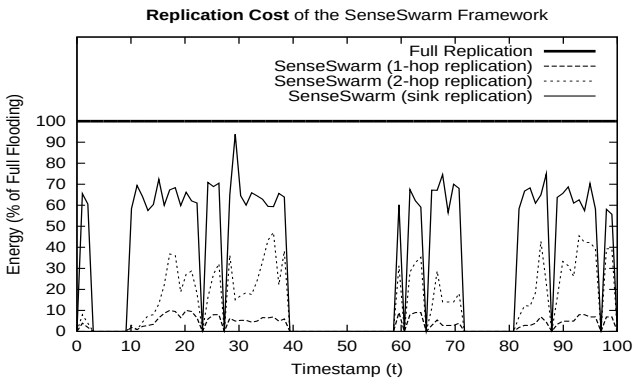

Figure 6: Evaluating the cost of Replication.

devices operate at any given moment. The figure shows that the cost of the SenseSwarm framework is almost 75% less than the energy cost of the Uniform framework. We also observe that every σ timestamps, a reconstruction of the perimeter is triggered in PA. This yields a non-uniform cost equivalent to 23 mJ. Although this cost is quite high, the average cost is still well below the overall cost of the Uniform framework. Particularly, the SenseSwarm network still consumes on average 1.7 ± 2.2 mJ while the Uniform framework consumes 6.7 ± 0.3mJ.

5.4 Replication Cost Evaluation

In the final experiment, we measure the cost for replicating the identified events of a query Q. We evaluate three different replication strategies: i) *Full replication*, where each detected event is replicated to all nodes in the network using flooding; ii) *SenseSwarm k-hop replication*, where each node forwards a detected event for k hops using the spatio-temporal aggregation scheme we described; and iii) *SenseSwarm sink replication*, where the events are replicated to a predetermined sink point (in essence an m-hop replication scheme, where m is the depth of the tree rooted at the sink).

Figure 6 shows the energy cost as a percentage of full flooding's cost. In all the cases the replication scheme manages to retain either the complete Q^A or an approximation of it. We observe that SenseSwarm 1-hop and 2-hop consume on average only 3% and 13% of full flooding's energy consumption. We also observe that in many cases the cost is equal to zero. This is attributed to the fact that no events occur on the given chronons, thus no replication takes place.

6. CONCLUSIONS AND FUTURE WORK

This paper introduces and formalizes a novel perimeter-based data acquisition framework for mobile sensor networks, coined SenseSwarm. In the future we plan to study other geometric shapes besides MBRs, techniques to provide strong fault tolerance properties, investigate sink selection strategies for optimal in-network replication and also investigate techniques to incrementally maintain the perimeter rather than reconstructing it in every iteration.

Acknowledgements: This work was supported in part by the US NSF under projects S-CITI (#ANI-0325353) and AQSIOS (#IIS-0534531), the EU under the project mPower (#034707) and the Cyprus Research Foundation under the project GEITONIA (#PLYP-0506).

7. REFERENCES

[1] Aly M., Pruhs K., Chrysanthis P.K., "KDDCS: a load-balanced in-network data-centric storage scheme for sensor networks", In *CIKM*, 2006.

[2] Bergbreiter, S.; Pister, K.S.J., "CotsBots: An Off-the-Shelf Platform for Distributed Robotics,", In *IROS*, Las Vegas, NV, 2003.

[3] Chintalapudi K. and Govindan R., "Localized Edge Detection In Sensor Fields", Ad-hoc Networks, 2003.

[4] Cormen T.H., Leiserson C.E., Rivest R.L., and Stein C., "Introduction to Algorithms", second edition. The MIT Press and McGraw-Hill, 2001.

[5] Crossbow Technology Inc. http://www.xbow.com/

[6] Dantu K., Rahimi M.H., Shah H., Babel S., Dhariwal A., and Sukhatme G.S., "Robomote: Enabling mobility in sensor networks", In *IPSN-SPOTS*, 2005.

[7] Hasan A., Pisano W., Panichsakul S., Gray P., Huang J-H., Han R., Lawrence D. and Mohseni K., "SensorFlock: A Mobile System of Networked Micro-Air Vehicles", TR-CU-CS-1018-06, U. of Colorado at Boulder, 2006

[8] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., "System Architecture Directions for Networked Sensors", In *SIGOPS Operating Systems Review*, Vol.34, No.5, pp.93-104, 2000.

[9] Intanagonwiwat C., Govindan R. Estrin D., "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In *MOBICOM*, 2000.

[10] Intel Lab Data http://db.csail.mit.edu/labdata/labdata.html

[11] Navarro-Serment, L.E., Grabowski, R., Paredis, C.J.J., and Khosla, P.K. "Millibots: The Development of a Framework and Algorithms for a Distributed Heterogeneous Robot Team,", *IEEE Robotics and Automation Magazine*, Vol. 9, No. 4, December 2002.

[12] Nittel S., Trigoni N., Ferentinos K., Neville F., Nural A., Pettigrew N., "A drift-tolerant model for data management in ocean sensor networks", In *MobiDE*, 2007.

[13] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "The Design of an Acquisitional Query Processor for Sensor Networks", In *SIGMOD*, 2003.

[14] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In *OSDI*, Vol.36, pp.131-146, 2002.

[15] Mani A., Rajashekhar M., Levis P. "TINX: a tiny index design for flash memory on wireless sensor devices", In *Sensys*, 2006.

[16] Ratnasamy S., Karp B., Shenker S. Estrin D., Govindan R., Yin L., Yu F., "Data centric storage in sensornets with GHT, a geographic hash table", In MONET, Vol. 8, Iss. 4, pp. 427-442, 2003.

[17] Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model", In *SIGGRAPH*, 1987.

[18] Sadler C., Zhang P., Martonosi M., Lyon S., "Hardware Design Experiences in ZebraNet", In *SenSys*, 2004.

[19] Shenker S., Ratnasamy S., Karp B., Govindan R., Estrin D., "Data-centric storage in sensornets", In *SIGCOMM Computer Communication Review*, Vol. 33 , Iss. 1, pp.137-142, 2003.

[20] Szewczyk R., Mainwaring A., Polastre J., Anderson J., Culler D., "An Analysis of a Large Scale Habitat Monitoring Application", In *SenSys*, 2004.

[21] Yao Y., Gehrke J.E., "The cougar approach to in-network query processing in sensor networks", In *SIGMOD Record*, Vol.32, No.3, pp.9-18, 2002.

[22] Zeinalipour-Yazti D., Andreou P., Chrysanthis P. and Samaras G., "MINT Views: Materialized In-Network Top-k Views in Sensor Networks", In *MDM*, 2007.

[23] Zeinalipour-Yazti D., Lin S., Kalogeraki V., Gunopulos D., Najjar W., "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices", In *FAST*, 2005.