Perimeter-Based Data Replication in Mobile Sensor Networks

Panayiotis Andreou*, Demetrios Zeinalipour-Yazti*, Maria Andreou†, Panos K. Chrysanthis‡, George Samaras*

*Department of Computer Science, University of Cyprus
P.O. Box 20537, 1678 Nicosia, Cyprus, {p.andreou, dzeina, cssamara}@cs.ucy.ac.cy

†School of Pure and Applied Sciences, Open University of Cyprus
P.O. Box 24801, 1304 Nicosia, Cyprus, andreou@ouc.ac.cy

‡Department of Computer Science, University of Pittsburgh
Pittsburgh, PA 15260, USA, panos@cs.pitt.edu

***Abstract*—This paper assumes a set of n mobile sensors that move in the Euclidean plane as a swarm. Our objectives are to explore a given geographic region by detecting spatio-temporal events of interest and to store these events in the network until the user requests them. Such a setting finds applications in mobile 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 nodes laying on the perimeter of the swarm we devise the *Perimeter Algorithm (PA)*, an efficient distributed algorithm with a low communication complexity. For storage and fault-tolerance we devise the *Data Replication Algorithm (DRA)*, a voting-based replication scheme that enables the exact retrieval of events from the network in cases of failures. Our trace-driven experimentation shows that our framework can offer significant energy reductions while maintaining high data availability rates. In particular, we found that when failures are less than 60% failure then we can recover over 80% of generated events exactly.**

I. Introduction

Stationary sensor networks have been predominantly used in applications ranging from environmental monitoring [25], [23] to seismic and structural monitoring [5] as well as industry manufacturing [19]. Recent advances in distributed robotics and low power embedded systems have enabled a new class of *Mobile Sensor Networks (MSNs)* [6], [29] that can be used in land [2], [7], [16], ocean [17] and air [9] exploration and monitoring, automobile applications [11], [8], habitant monitoring [23] and a wide range of other scenarios. 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). Additionally, MSN devices might derive their coordinates 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). The absence of a stationary network structure in MSNs makes continuous data acquisition to some sink point a non-intuitive task as data acquisition needs to be succeeded by in-network storage [30], [24], [1], such that these events can later be retrieved by the user. Additionally, the operation of MSNs is severely hampered by the fact that failures are omnipresent, thus fault-tolerance schemes become of prime importance in such environments.

In this paper we propose *SenseSwarm*[1], 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 applications where new events might occur anywhere in the network. In our setting, storage of 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 increase the overall fault-tolerance of our system, we propose a data replication scheme that increases the availability of data and thus also the accuracy of executed queries.

For ease of exposition, let us now consider a *Mars Exploration* scenario: Spirit was one of the two rovers deployed by NASA 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). Let the polarized behavior of the

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

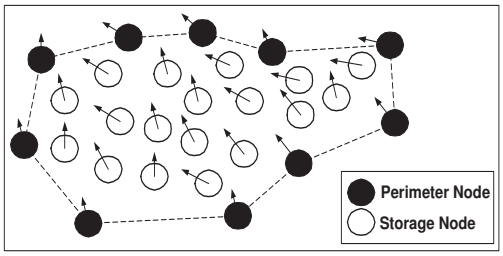

Fig. 1. *SenseSwarm*: Data Acquisition takes place at the virtual perimeter while core nodes act as storage nodes for the acquired events.

swarm be provided by an explicit algorithm [22] or an implicit mechanism (e.g., air current). The operator (on earth) then infrequently posts the question: *"Has the swarm identified any water and where exactly?"* Since the sink is located far away from the field deployment, the swarm collects spatio-temporal events of interest and stores them in the network until the operator requests them. In order to increase the availability of detected answers, in the presence of unpredictable failures, nodes perform data replication to neighboring nodes.

Similarly to the above visionary description, we could draw another more realistic 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 infrequently 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 [7] or Millibots [16], to construct spatio-temporal acquisition and storage scenarios for land applications.

This paper builts upon our previous work in [28], in which we presented the initial design of the SenseSwarm framework. In this paper we introduce several new improvements including a voting-based fault-tolerance scheme that increases the availability of data and thus improves fault tolerance. In particular, our work makes the following contributions:

- We present the *Perimeter Algorithm (PA)*, which efficiently constructs a perimeter of a MSN using a two-phase protocol. Our algorithm has a $O(n)$ message complexity, where n is the total number of sensors instead of $O(n^2)$, featured by the centralized algorithm.
- We devise a voting-based replication scheme to preserve the *datums* (i.e., acquired events) in cases of system failures. In particular, we devise the *DRA* algorithm that replicates datums using distributed *read/write quorums*.
- We experimentally validate the efficiency of our propositions using a trace-driven experimental study that utilizes real sensor readings.

The remainder of the paper is organized as follows: Section II overviews the related research work and provides background on our perimeter construction and fault-tolerance scheme we present. Section III formalizes our system model and assumptions, Section IV the PA algorithm and Section V the DRA algorithm. Section VI presents our experimental study and Section VII concludes the paper.

II. Related Work and Background

This section provides an overview of predominant data acquisition frameworks in order to highlight the unique characteristics of the SenseSwarm framework. It also provides background on the two main problems our framework addresses (i.e., the perimeter construction and the data replication processes).

Traditional data acquisition frameworks for sensor networks, such as TinyDB [18] and Cougar [26], 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 [27] performs in-network top-k pruning in order to further reduce the consumption of energy. In *data centric routing*, such as directed diffusion [12], 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 this work considers the challenges of a mobile sensor network setting. In *data centric storage* schemes [24], [1], data with the same attribute (e.g., humidity readings) is 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 [30] and TINX [20] local index structures, which provide $O(1)$ access to data stored on the local flash media of a sensor device. Such structures can be deployed to speed up the retrieval of data whenever required.

The first problem our framework investigates is that of partitioning the network into perimeter and core nodes. The perimeter construction problem we consider has similarities to the convex hull problem in computational geometry, which finds applications 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). 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) algorithm and the Graham's scan algorithm [4]. The main difference between the convex hull and the perimeter problem we consider in this work, 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 boundaries 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 the sensors to determine whether they belong

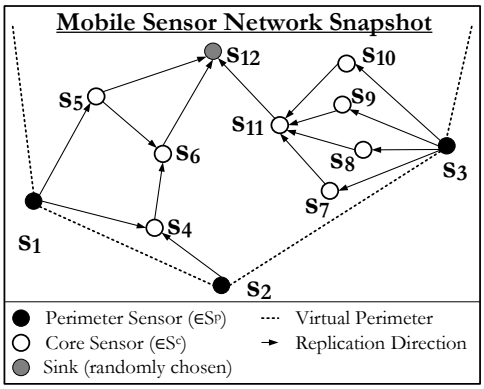

Fig. 2. In SenseSwarm, data acquisition is scheduled at the perimeter while storage and replication takes place at the core nodes.

TABLE I
Definition of Symbols

Symbol	Definition
n	Number of Sensors $S = \{s_1, s_2, ..., s_n\}$
m	Number of attributes at each s_i $\{a_1, a_2, ..., a_m\}$
(s_i^x, s_i^y)	x and y coordinates of each s_i
r	The communication radius of each s_i
$NH(s_i)$	1-hop (in commun. range) neighbors of s_i
$V(s_i, s_j)$	A Vector defined as $(s_j^x - s_i^x, s_j^y - s_i^y)$
$LeftN(s_i)$	The predecessor of s_i on the perimeter
$RightN(s_i)$	The successor of s_i on the perimeter
S^p, S^c	The set of Perimeter nodes, Core nodes
Q	An m-dimensional Query
e	Epoch Duration (i.e., data acquisition interval)
σ, σ'	Perimeter Reconstruction, Replication interval
d_i	The datum of node s_i
v_i^j, v_i	The vote (preference) of s_i to replicate d_i to node s_j, All votes from s_i

to the boundary of some phenomenon. Yet, the underlying assumption in the given work is that the edge sensors are not within communication range while we consider the perimeter to be a continuous chain of nodes. In [21], the authors devise an algorithm that combines current and historic measurements to trace a contour of a given value in the field (e.g., an oil spill). The presented ideas (e.g., that of quickly arriving at the contour) are supplementary to ideas presented in this paper.

The second problem our framework investigates is that of data replication to improve fault-tolerance. At a high level, our proposed scheme consists of maintaining a set of identical copies of each datum at several nodes in the network. For ease of exposition, let us consider the example network of Figure 2, which will be utilized throughout this paper. On the given figure we illustrate a segment of a MSN at a specific time τ. Assume that a copy of the datum d_1 (i.e., data published by node s_1), has been replicated to nodes s_4, s_5, s_6, s_{12}. Now let nodes s_1 permanently fail along with its one hop neighbors (i.e., s_4 and s_5) at time instance $\tau+1$. Since d_1 has been replicated beyond these nodes then it will be feasible to recover d_1 if necessary.

Our proposed solution is based on a voting-based data replication scheme. Voting algorithms [14], [15] have been among the most popular techniques to offer fault-tolerant properties in distributed systems. *A vote denotes the preference of some node to replicate a specific piece of information (i.e., a datum) to another node.* Voting schemes consist of first selecting a set of nodes where a specific datum will be replicated (i.e., the *write quorum*) and another set of nodes where a query will be conducted at, to search for that specific datum (i.e., the *read quorum*). One of the major challenges is to effectively choose the correct quorums so that the replication process will produce consistent results in an efficient manner. SenseSwarm's data replication algorithm utilizes the basic ideas of voting in conjunction with the unique characteristics of MSN systems.

III. System Model and Assumptions

In this section we will formalize our basic terminology and assumptions. The main symbols and their respective definitions are summarized in Table I.

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 the position of each point in the grid with coordinates (x, y). W.l.o.g, let us initialize the n sensing devices $S = \{s_1, s_2, ..., s_n\}$ at 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 some absolute or relative mechanism. Additionally, each s_i can be aware of its neighboring nodes, denoted as $NH(s_i)$, using a local 1-hop broadcast. The sensing devices are *coarsely synchronized* through some operating system mechanism (e.g., similarly to TinyOS [10]) or through the GPS and can communicate with other sensors in a uniform radius r, i.e., $1 \leq r \ll n^{\frac{1}{2}}$.

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 registered at each s_i either prior the deployment or during execution. The discussion of more complex query types is outside the scope of this paper.

A SenseSwarm network is initiated by conceptually dividing S into perimeter nodes S^p and core nodes S^c using the algorithms we present in Section IV. This operation is periodic and will be repeated after σ time instances (see Figure 3). Each perimeter sensor s_i $(i \leq n)$ then acquires m physical parameters A=$\{a_1, a_2, ..., a_m\}$ from its environment during every epoch e, which defines the interval after which data acquisition re-occurs. The value for e is either dynamically adjusted according to the dynamics of the swarm or prespecified. In a sea oil-spill detection scenario, e can be configured to several hours as surface drifters usually float very slowly on the sea surface. The above procedure generates spatio-temporal tuples of the form $\{t, x, y, a_1, a_2, ..., a_m\}$ locally at each sensor. The generated tuples of interest (with respect

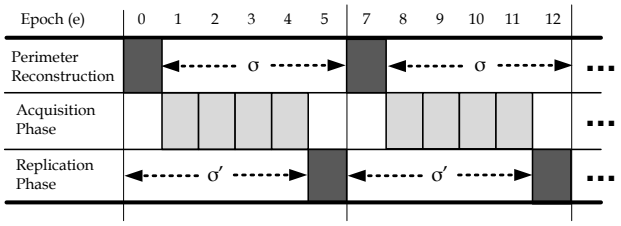

Fig. 3. Outline of the SenseSwarm framework operation.

to Q) are stored in some local vector structure that will be referred to as d_i (i.e., the $datum$ of node s_i).

In order to increase the availability of d_i structures, we adopt a data replication scheme based on *votes* that will be presented in Section V. A vote v_i^j denotes the preference of sensor s_i (i.e., the publisher of some datum d_i), to replicate d_i to node s_j ($i \neq j$) at a given time instance. Additionally, we define v_i as the set of all votes by node s_i on the given time instance.

In our approach, we assume that every σ' time instances every sensor $s_i \in S^p$ proceeds with the replication of its local datum d_i to the votes of s_i.

IV. Perimeter Construction Phase

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

A. 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).

B. 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 supports them.

Algorithm 1 presents the steps of the distributed PA process that is executed by each sensor every σ time instances. In line 4, procedure $Find\_Min\_Coordinares(S)$ identifies the sensor with the minimum y-coordinate and returns its *id* to the variable s_{min}. If more than one sensors have the y-coordinate

Algorithm 1 : Perimeter Algorithm (PA)

Input: Sensor s_i $(1 \leq i \leq n)$, the set of sensors S
Output: An update of the set S^p

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

equal to s_{min}^y, then the above procedure returns the one with the minimum value in its x-coordinate. The above procedure is achieved by constructing an aggregation tree rooted at the given sink using TAG [19]. In particular, each s_i identifies among its children and itself the minimum s_{min}^y value and then recursively forwards the triple $(s_{min}, s_{min}^x, s_{min}^y)$ 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 above 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 5, 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 proceeding, let us provide the following definitions:

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 $s_{min}^y \leq s_j^y$ $(\forall s_j \in S, 1 \leq j \leq n)$.

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 8-10 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

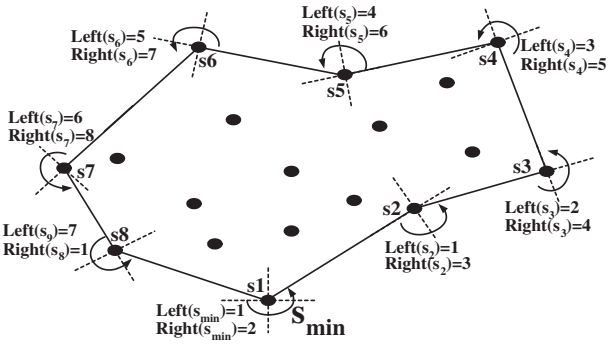

Fig. 4. Execution of PA: The construction starts at s_{min} and proceeds counterclockwise starting from π.

node will execute the remaining steps of the algorithm (lines 12-19). In particular, in lines 12-17, 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 13-18), is to identify the neighbor with the minimum polar angle (which is then coined $RightN(s_i)$), counterclockwise starting from π. Finally in line 19, s_i transmits a token to $RightN(s_i)$ notifying it that it is the next node on the perimeter. The procedure between lines 12-20 continues sequentially along the network perimeter until any s_i receives the token for a second time from its left neighbor or a timeout period expires. 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 on the perimeter. Thus the overall message complexity is $O(n)$, as $p \leq n$. In the future we plan to devise techniques to incrementally compute the perimeter.

Example: Figure 4, illustrates the perimeter construction for eight nodes $\{s_1 \cdots s_8\}$. Assume that we have executed steps 2-5 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.

V. Acquisition and Data Replication Phase

In this section we describe the second phase of the SenseSwarm Framework during which the perimeter nodes S^p start acquiring information from their environment and then replicate this information to their neighboring nodes.

Recall that the acquisition step proceeds every e time instances during which each s_i generates spatio-temporal tuples of the form $\{t, x, y, a_1, a_2, ..., a_m\}$. The generated tuples of interest (i.e., the tuples that satisfy the predicates of Q) are recorded in the local d_i (datum) structure of each s_i. Next, d_i structures are replicated to neighboring nodes according to the algorithm we propose in this section. In particular, we propose a data replication scheme based on *votes*.

The presented DRA algorithm replicates the d_i structures to w neighboring nodes (for any $w \geq 1$). If it is necessary to recover d_i then it is required to read d_i structures from at least $r = v - w + 1$ votes of s_i, where v is the total number of votes of s_i. For instance when $w = 2$ and $v = 4$ then $r = 4 - 2 + 1 = 3$ (i.e., 3 reads) are adequate to recover d_i in its exact form. When $w = 1$ and $v = 4$ then $r = 4 - 1 + 1 = 4$ reads are necessary to recover any replicated d_i. The details of the DRA algorithm follow next.

A. Data Replication Algorithm (DRA)

The objective of the DRA algorithm is to construct a data replication configuration which will present to each s_i an energy efficient plan on how to replicate its local d_i structures. A *data replication configuration* is an energy efficient *(read,write)-combination* that dictates how many read and writes operations are necessary per d_i, such that a d_i structure can be preserved in cases of failures. It is important to notice that if energy conservation was not important then we could have opted for a scheme that replicates each d_i to the entire network.

Algorithm 2 presents the details of the DRA algorithm. For ease of exposition, we will again utilize Figure 2 to demonstrate the operation of DRA. Let us focus on the perimeter sensor s_1 (although a similar discussion applies to the other perimeter nodes as well.) The DRA algorithm starts in the first step by discovering an adequate number of votes (candidate neighbors) for each perimeter sensor s_i (lines 2-6). This is done by probing the 1-hop core node neighbors of s_1, ($NH(s_1)$), which are s_4 and s_5 (line 3). If the number of neighboring nodes, $|NH(s_1)|$ is lower than a user-defined threshold $vmin$ (for our discussion let $vmin$=4) then s_1 expands its neighbors by incorporating more multi-hop nodes (line 5). That results in the increase of the $NH(s_1)$ set (i.e., s_6 and s_{12} are added to $NH(s_1)$). Besides the identifier of each neighbor, s_1 also stores the hop count for each of them (i.e., (s_4,1), (s_5,1), (s_6,2), (s_{12},2)) so that it can later decide which set of neighbors will produce the most energy-efficient replication strategy. Since the number of candidates in $NH(s_1)$ is 4, thus the $vmin$ requirement has been satisfied, s_1 utilizes all of these 4 nodes including itself (i.e., v_i=5). Next, s_1 proceeds with the selection of a subset of v_i for data replication. This is done by utilizing a voting process that operates as follows (we denote $|v_i|$ as v for brevity):

In Step 2 we define two integers, r (number of read operations) and w (number of write/replicate operations) with the following properties:

$$r + w > v, \quad v \geq r \geq 1, \quad v \geq w > v/2$$

Algorithm 2 : Data Replication Algorithm (DRA)

Input: A sensor $s_i \in S^p$, a threshold parameter $vmin$, representing the minimum number of votes a sensor must register.
Output: The data replication configuration (r,w) of s_i.

1: **procedure** DRA($s_i \in S^p$)
2: ▷ **Step 1:** Find neighbors of $s_i \in S^c$
3: $NH(s_i) \leftarrow$ Find hop-1 neighbors of s_i that belong to S^c
4: **if** ($|NH(s_i)| < vmin$) **then**
5: $NH(s_i) \leftarrow$ recursively expand neighbors
6: **end if**
7: ▷ **Step 2:** Define possible read write (r,w)-combinations
8: RW={$(r,\ w)$: $v \geq w > v/2,\ v \geq r \geq 1,\ r+w > v$}, where $v = |NH(s_i)|$
9: ▷ **Step 3:** Eliminate redundant (r,w)-combinations
10: RW'={(r,w): $(r,w) \in RW,\ r+w=v+1$}
11: ▷ **Step 4:** Rank the (r,w) in RW' according to f
12: $(r_x,w_x) \leftarrow max_{i \leq |RW'|} f(r_i, w_i)$
13: ▷ **Step 5:** Replicate the information to neighbors
14: $v_i = select(NH(s_i), w_x)$ // select a set of w_x neighbors
15: notify$_{s \in v_i}(s, d_i)$ // replicate d_i to these w_x neighbors
16: **end procedure**

We then create the RW-set of eligible (r,w)-combinations (line 8). In our example, since w needs to be in the range $5 \geq w > 2.5$ then $w \in \{3,4,5\}$. Furthermore, since $r+w > v$ then $r > v-w$ and consequently the following (r,w)-combinations are valid: RW={(1,5), (2,5), (3,5), (4,5), (5,5), (2,4), (3,4), (4,4), (5,4), (3,3), (4,3), (5,3)}.

In Step 3 of the voting process, we aim to eliminate redundant (r,w)-combinations in the RW set. To understand the intuition behind this elimination consider the (1,5)-combination. Since w=5 (i.e., all sensors hold a replica of datum d_1) then it is redundant to read more replicas than one (i.e., $(2,5), (3,5), \cdots, (5,5)$ are redundant). Although all of these combinations can recover d_i in cases of failures, they do not have the same energy requirements and should thus be excluded from the RW set. For instance the (2,5)-combination requires 1 read more than the (1,5)-combination and should thus be eliminated. The elimination of redundant combinations yields RW'={(1,5), (2,4), (3,3)}.

The objective of Step 4 is to further prune the RW' set in order to derive the (r,w)-combination that requires the least possible energy, but this operation is not straightforward. On the one hand, having more w operations involved in the replication process increases the overall fault-tolerance. On the other hand, more w operations would also incur additional messaging and consequently would require more energy. The negative effect of more w operations is particularly more apparent in cases where nodes have a hop distance from s_i that is larger than 1 (i.e., are not 1-hop neighbors).

In the fourth step of the DRA algorithm, we rank the remaining RW'={(1,5), (2,4), (3,3)} combinations using a ranking function $f_{(r,w)}$ and choose the one with the highest score. In particular, the local ranking proceeds as follows:

- Calculate the *number of broadcast messages* ($nbm_{(r,w)}$) that would be required for the replication process of the remaining (r,w)-combinations $\in RW'$ using the hop-count information gathered during lines 2-6 of DRA. Normalize $nbm_{(r,w)}$ to [0..1] using the following function: $nbm'_{(r,w)} = min(nbm_{\forall(r,w)})/nbm_{(r,w)}$.
- Calculate the *replication spreading factor* ($rsf_{(r,w)}$) by normalizing the w of each combination to [0..1] using formula $w/max(\forall w \in RW')$.
- Calculate the rank of each (r,w)-combination by summing the number of broadcast messages and replication spreading factor parameters: $f_{(r,w)} = nbm'_{(r,w)} + rsf_{(r,w)}$. [2]

The results of the ranking on our example are summarized in Table II. The presented results indicate that the (1,5)-combination has the highest rank in the f function and consequently that plan is utilized for the replication of d_i.

TABLE II
Ranking the (r,w)-combinations of RW' during the fourth step of DRA

(r,w)	$nbm_{(r,w)}$	$nbm'_{(r,w)}$	$rsf_{(r,w)}$	$f_{(r,w)}$
(1,5)	**4**	**1.0**	**1.0**	**2.0**
(2,4)	5	0.8	0.8	1.6
(3,3)	4	1.0	0.6	1.6

In the final fifth step of DRA, s_i proceeds with the replication of d_i to the identified neighboring nodes. In particular, in line 14 s_i selects w_x neighbors from its $NH(s_i)$ list and stores these results in the v_i set. Each s_i then proceeds with the replication of d_i to the identified w_x nodes in line 15. This completes the operation of the DRA algorithm.

Theorem 1: *The DRA algorithm guarantees that a datum d_i can be recovered if the number of reads (r_x) from the votes of s_i is at least $v - w_x + 1$ ($v \geq w_x$), where v denotes the number of all votes and w_x the number of writes during the replication of d_i.*

Proof: Let us select first two sets, R and W, such that $|R| = r_x$ and $|W| = w_x$ ($R, W \subset v_i$) as dictated by DRA. Since $w_x > v/2$ then d_i has been replicated to more than half of the nodes assigned a vote by node i. Now, considering that $r_x + w_x > v$, we must have $R \cap W \neq \emptyset$. Hence any read operation is guaranteed to read the value of at least one copy which has been updated by the latest write □

VI. Experimental Evaluation

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

A. Experimental Methodology

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

Sensing Device: We use the energy model of Crossbow's research sensor device TelosB [5] to validate our ideas. TelosB is a ultra-low power wireless sensor equipped with a 8 MHz

[2] $nbm'_{(r,w)}$ and $rsf_{(r,w)}$ are the two most prominent parameters for selecting the best (r,w)-combination. However, one could also consider parameters like capacity required to store the datums and recovery performance.

MSP430 core, 1MB of external flash storage, and a 250Kbps Chipcon (now Texas Instruments) CC2420 RF Transceiver that consumes 23mA in receive mode (Rx), 19.5mA in transmit mode (Tx), $7.8mA$ in active mode (MCU active) with the radio off and $5.1\mu A$ in sleep mode. Our performance measure is *Energy*, in *Joules*, that is required at each discrete time instance to resolve the query. The energy formula is as following: $Energy(Joules) = Volts \times Amperes \times Seconds$. For instance the energy to transmit 30 bytes at 1.8V is: $1.8V \times 23 * 10^{-3}A \times 30 * 8bits/250kbps = 39\mu J$.

Dataset: We utilize a real dataset from Intel Berkeley Research [13]. 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. We use 10,000 readings from the 54 sensors that had the largest amount of local readings since some of them had many missing values.

Swarm Simulation: In order to introduce motion to our sensor network we have derived synthetic spatial coordinates for the n sensors using the Craig Reynolds algorithm [22], 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 $X\%$ independent probability that a node fails at any given timestamp.

B. Perimeter Cost Evaluation

In the first experimental series we investigate the efficiency of our distributed PA algorithm compared to the centralized CPA algorithm. Figure 5, presents the aggregate cost (i.e., for the whole network and 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.

C. Replication Phase Evaluation

In the final experimental series, we evaluate the fault-tolerance accuracy of our two replication algorithms.

In the first experiment we measure the absolute fault-tolerance accuracy of the Data Replication Algorithm (DRA).

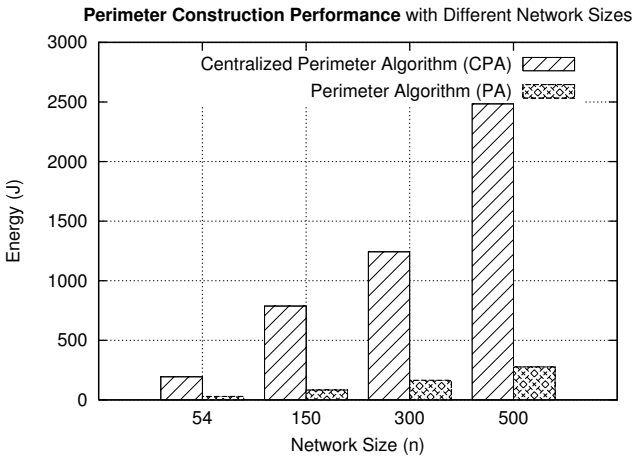

Fig. 5. Evaluating the energy consumption of the Perimeter Algorithm.

To accomplish this, we compare DRA against a version that does not employ any replication strategy, coined *No-Replication Algorithm (NRA)*. We execute both algorithms on each of the individual scenes generated by our swarm simulator. During each one of the 100 individual scenes, we randomly select a sensor node to be the sink. As soon as the sink is selected, it registers 10 random queries each of which requesting events detected by different sets of perimeter sensors. In order to measure the accuracy of each of the algorithms, we measure the average *ratio of detected events* over the *total number of events requested by the 10 queries*.

Figure 7, illustrates the fault-tolerance accuracy of the two algorithms over an increasing failure rate. We observe that in all cases DRA maintains a competitive advantage of ≈19-48% over NRA. This is due to the voting-based replication strategy utilized by DRA. Note that we have configured DRA with $vmin$=3 (i.e., 3 votes). Since, in DRA, detected events are replicated to 3 neighboring nodes, even if a node fails, its detected events are easily obtained by its votes thus ensuring a higher level of accuracy. We also observe that with a 60% failure rate the accuracy of both algorithms starts to decrease rapidly. This is expected at such high failure rates as large segments of the query routing tree become inaccessible by the sink.

We have finally measured the number of extra communication messages that DRA requires during replication. We discovered that on average, DRA requires approximately 90±32 extra messages (i.e., has a message complexity of $O(n)$).

VII. Conclusions and Future Work

This paper introduces and formalizes a novel perimeter-based data acquisition framework for mobile sensor networks, coined SenseSwarm. SenseSwarm dynamically partitions the sensing devices into *perimeter* and *core* nodes. Data acquisition is scheduled at the perimeter, with the invocation of the PA algorithm, while storage and replication takes place at the core nodes, with the invocation of the DRA algorithm.

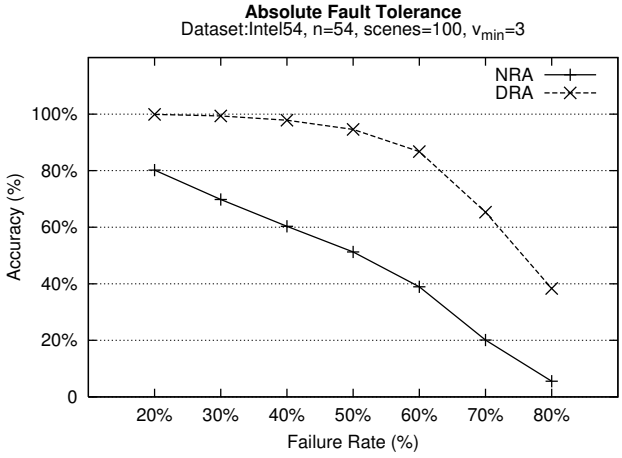

Fig. 6. Evaluating the absolute fault-tolerance accuracy (that measures the percentage of datums that can be recovered) for the DRA and NRA algorithms.

Our trace-driven experimentation with realistic data shows that our framework offers tremendous energy reductions while maintaining high data availability rates. In particular, we found that even with 60% system failures we can recover over 80% of generated events exactly. In the future we plan to study other geometric shapes besides MBRs, different sink selection strategies for in-network replication and also techniques to incrementally maintain the perimeter rather than reconstructing it in every iteration.

Acknowledgements: This work was supported in part by the Open University of Cyprus under the project SenseView, the US National Science Foundation under the project AQSIOS (#IIS-0534531), the European Union under the projects mPower (#034707) and IPAC (#224395).

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*, 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", 2^{nd} edition. The MIT Press and McGraw-Hill, 2001.

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

[6] Chrysanthis P.K. and Labrinidis A., "NSF Workshop on Data Management for Mobile Sensor Networks Report", Jan. 16-17, 2007, Pittsburgh, USA, http://www.mobisensors.org

[7] 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.

[8] Eriksson, J., Girod, L., Hull, B., Newton, R., Madden, S. and Balakrishnan H., "The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring", In *ACM Int. Conf. on Mobile Systems, Applications And Services* 2008.

[9] Hasan A., Pisano W., Panichsakul S., Gray P., Huang J-H., Han R., Lawrence D. and Mohseni K., "SensorFlock: An Airborne Wireless Sensor Network of Micro-Air Vehicles", In ACM *Sensys*, 2007

[10] 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.

[11] Hull B., Bychkovsky V., Chen K., Goraczko M., Miu A., Shih E., Zhang Y., Balakrishnan H., and Madden S., "CarTel: A Distributed Mobile Sensor Computing System", In *ACM Int. Conf. on Embedded Networked Sensor Systems*, 2006.

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

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

[14] Jalodia S., Mutchler D., "Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database", IN ACM TODS, vol.15, pp.230-280, June, 1990.

[15] Koren I., Krishna C.M., "Fault-Tolerant Systems", Elsevier, 2007.

[16] 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.

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

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

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

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

[21] Srinivasan S., Ramamritham K., Kulkarni P., "ACE in the Hole: Adaptive Contour Estimation Using Collaborating Mobile Sensors", In IEEE *IPSN*, 2008.

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

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

[24] 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.

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

[26] 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.

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

[28] Zeinalipour-Yazti D., Andreou P., Chrysanthis P.K., Samaras G., "SenseSwarm: a perimeter-based data acquisition framework for mobile sensor networks", In VLDB's *DMSN*, 2007.

[29] Zeinalipour-Yazti D., Chrysanthis P.K., "Mobile Sensor Network Data Management" Book Chapter in the Encyclopedia of Database Systems (EDBS), Ozsu, M. Tamer; Liu, Ling (Eds.), ISBN: 978-0-387-49616-0, 2009.

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