Noname manuscript No.
(will be inserted by the editor)

In-Network Data Acquisition and Replication in Mobile Sensor Networks

Panayiotis Andreou · Demetrios Zeinalipour-Yazti · Panos K. Chrysanthis · George Samaras

the date of receipt and acceptance should be inserted later

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 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 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 values from the network in cases of failures. We also extend DRA with a spatio-temporal in-network aggregation scheme based on minimum bounding rectangles to form the *Hierarchical-DRA (HDRA)* algorithm, which enables the *approximate* retrieval of events from the network. 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 across all nodes are less than 60%, our framework can recover over 80% of detected values exactly.

P. Andreou, D. Zeinalipour-Yazti (contact author), and G. Samaras, Department of Computer Science, University of Cyprus, Nicosia, 1678, Cyprus; Tel.: +357-22-892755; Fax: +357-22-892701; E-mail: {panic,dzeina,cssamara}@cs.ucy.ac.cy

P.K. Chrysanthis, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 5213-4034, Tel.: +1-412-624-8924; Fax: +1-412-624-8854; E-mail: panos@cs.pitt.edu

Keywords

Mobile Sensor Networks, Data Management, Fault Tolerance.

1 Introduction

Stationary sensor networks have been predominantly used in applications ranging from environmental monitoring [33,30] to seismic and structural monitoring [7] as well as industry manufacturing [20]. Recent advances in distributed robotics and low power embedded systems have enabled a new class of *Mobile Sensor Networks (MSNs)* [8,38] that can be used in land [3,9,24], ocean [25] and air [11] exploration and monitoring, automobile applications [13,10], habitant monitoring [30] 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 [26,40]*, 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 infrastructure 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 [39,31, 28,1], 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.

There are numerous advantages of MSNs over their stationary counterparts. In particular, MSNs offer: i) *dynamic network coverage*, by reaching areas that have not been adequately sampled; ii) *data routing repair*, by replacing failed routing nodes and by calibrating the operation of the network; iii) *data muling*, by collecting and disseminating data/readings from stationary nodes out of range; iv) *staged data stream processing*, by conducting in-network processing of continuous and ad-hoc queries; and v) *user access points*, by enabling connection to handheld and other mobile devices that are out of range from the communication infrastructure.

In this paper we present *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. 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 applications where new events might occur anywhere in the network.

Mobile Sensor Networks are useful in an ever increasing number of applications and domains. Below, we motivate our discussion by describing two of these applications that are founded on the premise of MSNs. In particular, we present applications using the MBARI ocean observation system [22] and applications using People-centric Sensing [4].

Example 1 - MBARI ocean observation systems: The Monterey Bay Aquarium Research Institute (MBARI) [22] is one of the leading institutes in deep-sea exploration and under-water research. MBARI currently drives a number of ocean observation projects like the Monterey Ocean Observing System (MOOS), the Monterey Accelerated Research System (MARS) and the Autonomous Ocean Sampling Network (AOSN). The aforementioned ocean observation systems provide critical information for research on climate change, biogeochemical cycles, ecosystem assessment, and environmental hazards. To accomplish data acquisition, these systems utilize sensor devices attached on aerial and underwater vehicles that move in space and perform a coordinated task. Since most oceanographic instruments have no means of connecting with the surface, they have to rely on battery operation and local data storage while exploring the underwater terrain. This necessitates the use of energy efficient mobile sensor network infrastructures and especially data replication strategies that ensure data availability in cases of failures. These are characteristics offered by the Senseswarm Framework presented in this work.

Example 2 - People-Centric Sensing: People-centric sensing [4], aims to support sensor-enabled applications that engage the general public through the use of their own personal mobile devices. The recent miniaturization and integration of sensors into popular consumer mobile devices (e.g., iPhone, HTC Hero) has enabled a myriad of new sensor based applications for personal, social and public sensing. These applications can be utilized for increasing the sensing coverage of large public spaces and collect targeted information about their mobile device owners (e.g., human mobility patterns). The information can then be uploaded to a centralized database system or exchanged with neighboring mobile devices. What is really important, is that these environments allow new levels of data sharing among commodity devices. Specifically, a particular device can request sensor data from any available neighboring device through the establishment of an adhoc communication network (e.g., through Bluetooth or Wi-Fi). Assuming that the users of such a system move in a coordinated manner (e.g., a group of cyclists), highlights the distinct characteristics of the Senseswarm framework presented in this work.

In order to better frame the SenseSwarm framework, let us consider a phenomenon, described as an arbitrarily shaped sub-region of the terrain where the MSN has been deployed (Figure 1). We assume that this phenomenon and does not expand, shrink or move rapidly. When the MSN moves closer to the phenomenon (i.e., at T=3) it is easy to see that perimeter nodes will be the first ones capturing the event. In this setting, perimeter nodes continuously sample the events of the phenomenon and transmit their results to the MSN. The storage of these detected events takes place at the core nodes since these

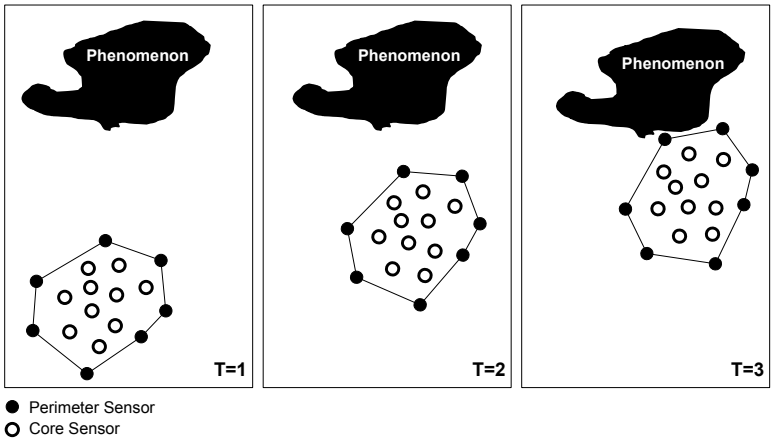

Fig. 1 Example Scenario: SenseSwarm detects physical phenomena (e.g., oil spills) by using a swarm of sensor nodes that are dynamically organized in perimeter and core nodes. Perimeter nodes continuously sample the events of the phenomenon and transmit their results to the core nodes. Storage and replication of detected events takes place at the core nodes since these are expected to feature a longer lifetime (due to their reduced sensing activity) but also because these are physically shielded to threats and obstacles.

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 data replication schemes that increase the availability of data and thus also the accuracy of executed queries. More specifically, the goals of the SenseSwarm framework are the following:

- Minimize the energy consumption required for defining the perimeter of the network. We accomplish this by introducing the distributed Perimeter Algorithm (PA).
- Maximize fault tolerance and recoverability in the presence of network failures according to application preferences. We accomplish this by introducing the DRA and HDRA algorithms.

This paper builds upon our previous works [37, 2] in which we presented the initial design of the SenseSwarm framework. In this paper we introduce several new improvements including a novel hierarchical voting-based fault-tolerance scheme as well as an in-network aggregation scheme, that in conjunction increases the availability of data and thus improves both fault tolerance and query execution. This is shown through additional experimental evaluation.

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 *data* (i.e., acquired events) in cases of system failures. In particular, we devise the *DRA* algorithm that replicates data using distributed read/write quorums.
- We additionally devise HDRA, a spatio-temporal in-network aggregation scheme based on minimum bounding rectangles that enables the retrieval of acquired events in an approximate form.
- 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: Sect. 2 overviews the related research work and provides background on our perimeter construction and fault-tolerance schemes we present. Sect. 3 formalizes our system model and assumptions, Sect. 4 the PA algorithm and Sect. 5 the DRA and HDRA algorithms. Sect. 6 presents our experimental study and Sect. 7 concludes the paper.

2 Related Work and Background

This section provides an overview of traditional 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 (e.g., TinyDB [19], Cougar [35]), 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 [36] performs in-network top-k pruning in order to further reduce the consumption of energy. In *data centric routing*, such as directed diffusion [14], 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 [31,28,1], 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 [39] and TINX [21] 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. Additionally, optimization query processing techniques like the works presented in [23,34] can be used in conjuction with our framework in order to speed up query execution.

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 [6]. 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* [6] (or Gift Wrapping) algorithm and the Graham's scan algorithm [6]. 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 [5], where the authors present localized techniques that enable the sensors to determine whether they belong 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 [27] the authors present an algorithm that can identify perimeter nodes without any location information but in the presence of specialized nodes, called bootstrap beacon nodes, which have long range antennas that enable them to broadcast messages to the entire network. The sensor nodes can then estimate their distance to these special nodes and decide if they are perimeter nodes. In SenseSwarm we do not assume that these specialized long-range bootstrap beacons are available. On the contrary, our assumption is that all sensor nodes have the same capabilities. However, the work in [27] is supplementary to SenseSwarm because if bootstrap beacons were available we could have utilized them to calculate the perimeter faster. In SenseSwarm, once perimeter nodes have been identified, the core nodes need not to know their coordinates (actual or virtual) since they forward their results to their parents. This routing scheme is different from [27,17] where virtual coordinates are necessary for maintaining the correct routing tables used for forwarding packets. In [17] nodes make forwarding decisions in a greedy manner by only using information about the immediate neighbors of the node. In SenseSwarm we do not perform routing decisions but instead we focus on sensing, aggregating and storing. In [32], 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 schemes consist 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 left part of Figure 2 we illustrate a segment of a MSN at a specific time τ. Assume that a copy of the

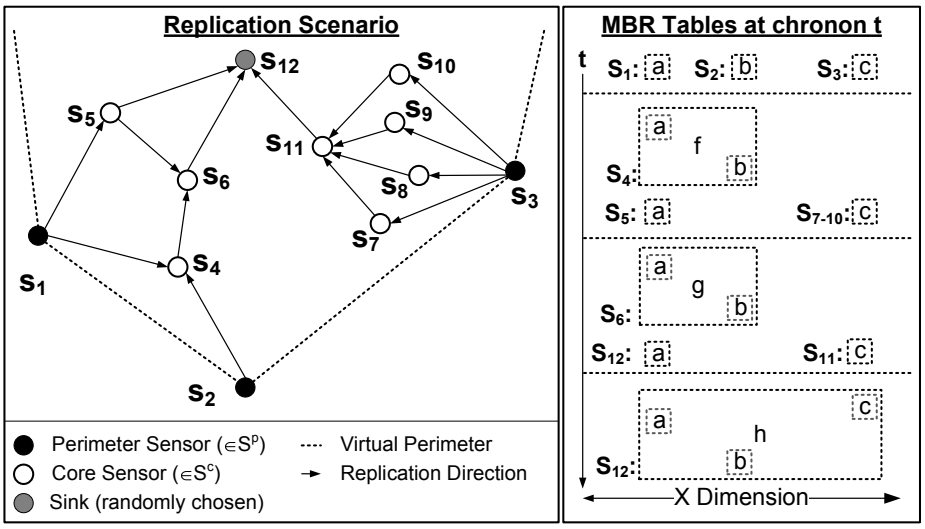

Fig. 2 Replication and Aggregation in SenseSwarm: In-network aggregates are constructed during replication by using Minimum Bounding Rectangles (MBRs).

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 node 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 [16,18] 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 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.

3 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 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 the position of each point in the grid with coordinates (x, y). In order to be able to introduce movement patterns to the sensor net-

Table 1 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

work we uniformly distribute the n sensing devices in an area $n^{\frac{1}{2}} \times n^{\frac{1}{2}}$ approximately in the middle of $\Re^2$. 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 [12]) 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 as presented in [37]. 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 to Q) are stored in some local vector, referred to as d_i (i.e., *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 5. 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

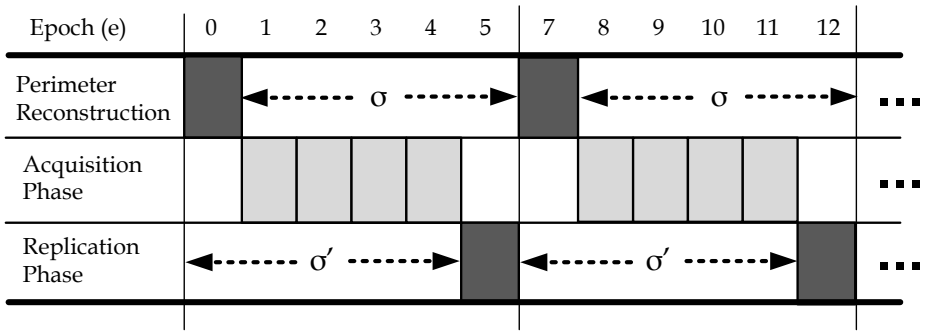

Fig. 3 Outline of the SenseSwarm framework operation.

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.

4 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.

4.1 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).

4.2 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

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
```

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 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 equal to s^y_{min}, 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 [20]. 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 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 $s^y_{min} \leq s^y_j$ $(\forall s_j \in S, 1 \leq j \leq n)$.

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 8-10 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 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 is the number of the nodes on 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 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.

5 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.

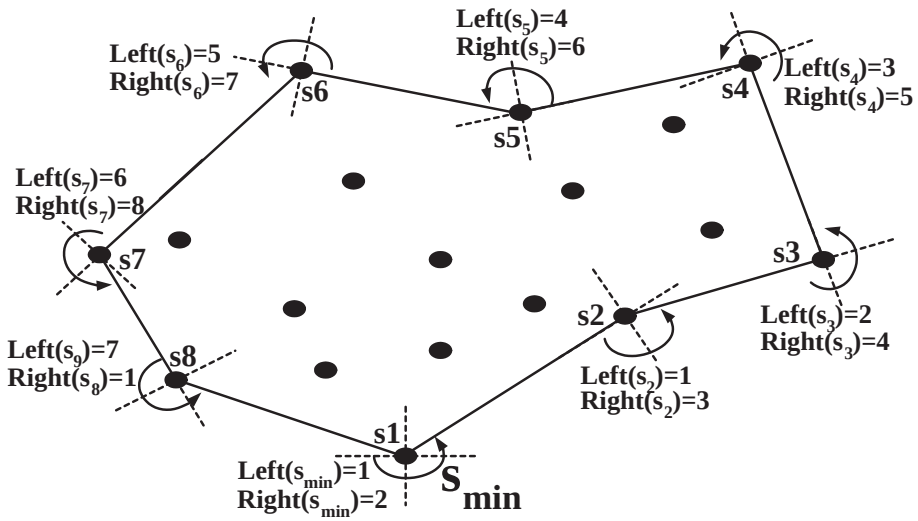

Fig. 4 Execution of PA: The construction starts at s_{min} and proceeds counterclockwise starting from π.

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 algorithms we propose in this section. In particular, we propose a data replication scheme based on *votes* and a replication scheme based on spatial approximations.

The first presented algorithm, DRA, 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 any replicated 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 second presented algorithm, HDRA, extends the basic DRA idea by additionally constructing the Minimum Bounding Rectangles (MBRs) of tuples in d_i (see Figure 2 right). The system then replicates the $MBR(d_i)$ vector, rather than d_i, to its parent node in a virtual spanning tree. That significantly increases the availability of d_is in cases of failures. Additionally, the HDRA approach will return an approximate answer, rather than an exact answer, in cases the algorithm can not proceed otherwise. The details of the above two algorithms follow next.

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**

5.1 Data Replication Algorithm (DRA)

The objective of the DRA algorithm is to construct a data replication configuration that 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 (left) 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 selecting 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:

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

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 $r > v - w$ the following (r,w)-combinations are valid combinations: 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 one hand, by 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 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).

Consequently, in this fourth step 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. Our ranking function tries to balance the fault tolerance and replication overhead (i.e., message complexity). This is accomplished by examining the effect of both parameters in each combination and then opt for the one that maximizes both. However, this ranking function can be easily adapted to the requirements of the MSN application developer. For example, in an MSN with extremely limited energy reserves, an application may choose to sacrifice high levels of fault tolerance in order to minimize the communication overhead.

The local ranking process presented in this paper 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 $w/max(\forall w \in RW')$.

Table 2 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	0.6	1.0	1.6

– Calculate the rank of each (r,w)-combination by summing the number of broadcast messages and replication spreading factor parameters: $f_{(r,w)} = nbm'_{(r,w)} + rsf_{(r,w)}$. [1]

The results of the ranking on our example are summarized in Table 2. 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 s_i's datum.

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.

A question that now arises is how to retrieve (i.e., read) the d_i structures from the network during the execution of a query. Fortunately, this is a straightforward procedure as the querying node can proceed by querying r_x neighbors, which are defined in the same manner the w_x neighbors were constructed, and be sure that a copy of d_i has been recovered.

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 □

5.2 Hierarchical Data Replication Algorithm (HDRA)

In this section we describe an extension of the original DRA algorithm which attempts to replicate d_i structures at an even coarser representation through-

[1] $nbm'_{(r,w)}$ 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 data and recovery performance.

out the network such that this information survives in cases of high failure rates and disconnections.

At a high level, the HDRA algorithm proceeds as follows: When the DRA algorithm completes its operation, some arbitrary node s_{sink} (e.g., the one with the minimum (x,y) coordinates), identifies itself as the sink node. s_{sink} then recursively disseminates a request to its 1-hop neighbors, using a typical tree-based query dissemination mechanism [12], asking them to conduct an aggregation of their local datum results (i.e., both their own d_i result and those data that have been replicated to s_i). The aggregated result is forwarded to s_{sink} through the parents of each node s_i, as those parents are identified during the tree construction process. The above procedure continues recursively until all n sensors have received the aggregation request and forwarded their answers to s_{sink}.

When the above procedure terminates, nodes farther away from a node s_i will contain a coarser representation of the information stored locally on s_i. That has two advantages: i) Even if s_i is completely eliminated from the system then the user will still be able to recover a coarser representation of d_i from the j-hop neighbors of s_i (where $j \geq 1$); ii) The network can speedup query execution as certain queries can be answered at no extra cost. For instance a query that aims to answer the question: *"Has the swarm detected any water,"* can be answered even if the system preserves only a very coarse representation of the generated d_i structures.

Before proceeding with the details of the HDRA algorithm let us define the notion of an MBR which is utilized during the in-network aggregation process.

Definition 3 [Minimum Bounding Rectangle]: A rectangle that encloses all points in a given area V. The Cartesian coordinates of the bounding box $MBR(V)$ 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 in the area V and might encapsulate $|V|$ events using only five real numbers, i.e., (ts, $MBR(V)$), as opposed to ($|V|$*2 + 1) real numbers. That makes MBRs highly compact structures, enabling huge energy savings during their replication. This is particularly true when $5 \ll |V|$. Finally, note that an MBR can easily incorporate aggregate answers ($aggr$) with the bounding box as $(t, x_1, y_1, x_2, y_2, aggr)$.

The specifics of the HDRA algorithm are shown in Algorithm 3. In line 3, node s_i waits in standby mode until it receives an *Aggregate_Request* from its parent, which is a message that initiates the construction of the in-network aggregation tree. In line 4, it immediately broadcasts *Aggregate_Request* to its own neighborhood. Each node then waits for the MBRs of its children nodes. Without loss of generality, we adopt the *child anchor* mechanism used in [35], where a sensor s_j confirms to exactly one of its parent s_i that it wants to be its child. This provides s_i with a list of children so that s_i can know when all the answers from its children have arrived. Whenever an MBR is received

Algorithm 3 : **Hierarchical Data Replication Algorithm (HDRA)**

Input: A set of sensors $S = \{s_1, s_2, \cdots s_n\}$, a randomly selected sink s_{sink}
Output: A set of n distributed MBRs organized in a Querying Routing Tree.

```
1: procedure HDRA(S, s_i)
2:     MBR_i = NULL;
3:     receive(Aggregate_Request, parent(s_i));
4:     broadcast(Aggregate_Request);
5:     for j = 1 to |children(s_i)| do
6:         receive(MBR_j, child(s_j));
7:         MBR_i = merge(MBR_i, MBR_j);
8:     end for
9:     send(MBR_i, parent(s_i));
10: end procedure
```

from some child s_j (line 6), this MBR_j is merged with the local MBR_i (line 7) and when all children have answered then MBR_i is forwarded to the parent node of s_i (line 9).

Example: Figure 2 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_{12} has an MBR which encapsulates all the events that have occurred. When a user performs a query, we collect the MBRs 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.

Discussion: Although the MBR aggregation ideas are only conducted in space, a similar logic could also be applied in order to conduct spatio-temporal aggregation (i.e., using (x, y, ts)). In particular, we could 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) but the discussion of this extension is outside the scope of this paper.

6 Experimental Evaluation

In this section we present the experimental evaluation of the SenseSwarm framework. Using a trace-driven methodology, we measured the time and energy behavior of our proposed algorithms as well as the robustness of our SenseSwarm framework in the presence of failures.

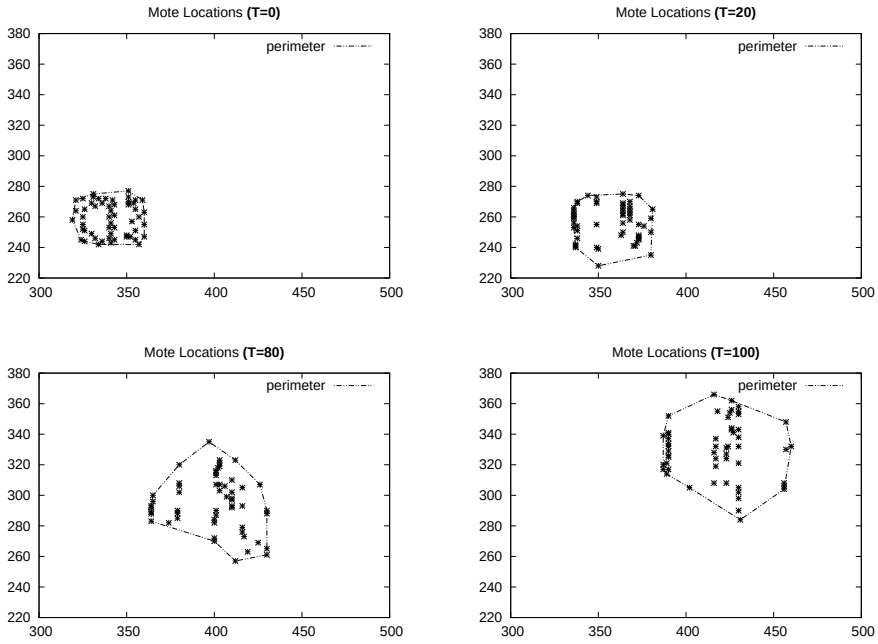

Fig. 5 Sample simulator output for individual scenes at timestamps 0,20,80 and 100. Perimeter nodes are connected using dashed lines.

6.1 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:

Swarm Simulation: In order to introduce motion to our sensor network we have derived synthetic spatial coordinates for the n sensors using the Craig Reynold's algorithm [29], 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). Our simulator has the ability to visual representations of the swarm simulation as illustrated in Figure 5. Additionally, in order to simulate failures we make the assumption that there is a X% independent probability that a node fails at any given timestamp.

Dataset: We utilize a real dataset from Intel Berkeley Research [15]. 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

Table 3 Configuration parameters for all experimental series.

Section	Objective	n	Failures	Scenes
6.2	Energy Cost	54,150,300,500	20%	1000
6.3	Time Overhead	54	0%	1000
6.4	Coverage	54	10%-50%	1000
6.5	Acquisition Cost	54	20%	1000
6.6	Fault Tolerance	54	20-90%	100
6.7	Scalability	54,150,300,500	50%	100

54 sensors that had the largest amount of local readings since some of them had many missing values.

Sensing Device: We use the energy model of Crossbow's research sensor device TelosB [7] to validate our ideas. TelosB is a ultra-low power wireless sensor equipped with a 8 MHz 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: $1.8V \times 23 * 10^{-3}A \times 30 * 8bits/250kbps = 39\mu J$.

Perimeter Performance Metrics: In order to evaluate the coverage efficiency of the perimeter algorithm (PA) under failures, we introduce the *Coverage ratio* metric, which is defined as the ratio of the area generated by perimeter nodes under failures over the area generated by perimeter nodes under no failures.

Replication Performance Metrics: In order to evaluate the accuracy performance of our two replication algorithms, we introduce two metrics i) absolute fault-tolerance accuracy, and ii) approximate fault-tolerance accuracy. *Absolute fault-tolerance accuracy* is the percentage of discovered events over the *total number of events requested by a query* and will be utilized for the evaluation of the DRA algorithm which attempts to uncover exact answers to queries. *Approximate fault-tolerance accuracy* measures the proximity penalty that occurs when the MSN returns an MBR that encloses an event instead of the actual coordinates of a specific event. We will provide a more thorough description of this performance metric in Section 6.6. Note that in either experiment each node only propagates correct results to the sink.

Table 3 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

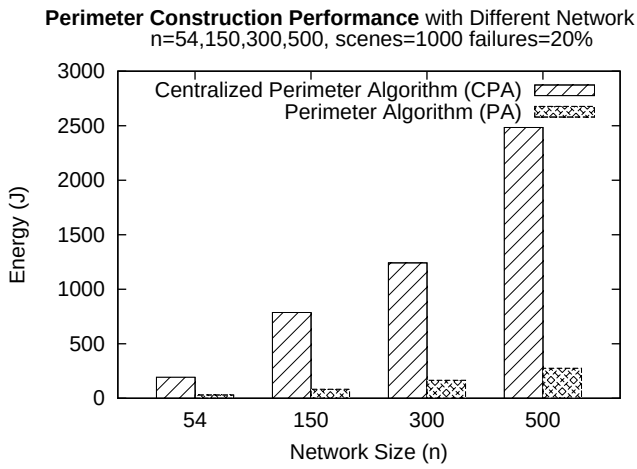

Fig. 6 Evaluating the energy consumption of the Perimeter Algorithm.

6.2 Perimeter Phase Evaluation: Energy Cost

In the first experimental series, we investigate the efficiency of our distributed PA algorithm compared to the centralized CPA algorithm. Figure 6 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.

6.3 Perimeter Phase Evaluation: Time Overhead

In the second experimental series, we measure the time overhead for each phase of the PA algorithm. We chose to present the time in simulated CPU ticks, as opposed to milliseconds, because the conversion would sometimes lead us to very small (close to zero) quantities. We record the time ticks at the start and end of each phase and show the duration for all 1000 timestamps.

In Figure 7, we observe that the time overhead for the first phase of the PA algorithm (i.e., initialization and discovery of the node with min y-coordinate) is quite low. This happens as the discovery and dissemination process for

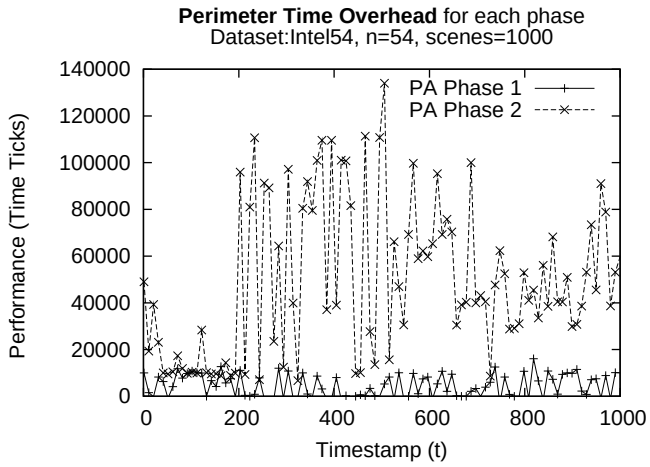

Fig. 7 Evaluating the time overhead of each phase of the PA algorithm.

identifying the s_{min} node requires minimal processing at each node (i.e., in the discovery process each node transmits its coordinates and in the dissemination process each node only processes messages if it is s_{min}.) On the other hand, the second phase of the PA algorithm is somehow more expensive. This is attributed to the fact that each node s_i has to discover its neighboring nodes and then process their coordinates in order to identify the next perimeter node (i.e., $RightN(s_i)$). The time overhead for the second phase is also augmented by the number of perimeter nodes (i.e., the larger the number of perimeter nodes, the larger the overall time overhead).

6.4 Perimeter Phase Evaluation: Coverage Under Failures

In the third experimental series, we investigate the area coverage generated by the PA algorithm under different failure settings, ranging from 0% (no failures) to 50% (high failure rate). We ran each experiment 10 times and record the average coverage ratio, defined as the ratio of the area generated by perimeter nodes under failures over the area generated under no failures, for each respective execution. The results of these experiments are depicted in Figures 8 and 9.

Figure 8 illustrates the coverage ratio for each of the failure scenarios. In order to display the results of the experiment more efficiently, we have applied a spline interpolation smoothing between consecutive timestamps. We observe that even with 50% failures the average coverage ratio for all experiments is above 70%. In Figure 9 we investigate the distribution of results in all experiments using a box plot. We observe that for experiments with failures $\leq$30% the majority of the coverage ratio results fall in the 3^{rd} quartile (i.e.,

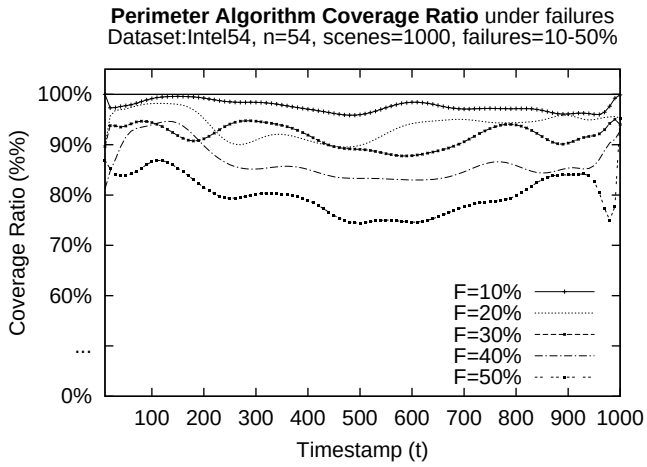

Fig. 8 Evaluating the coverage ratio of the Perimeter Algorithm.

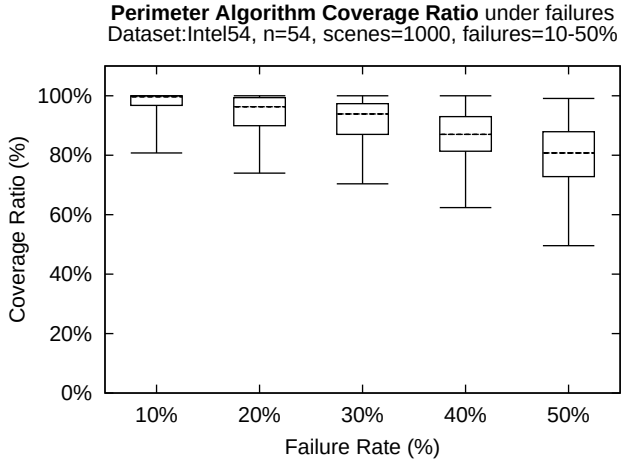

Fig. 9 Analysis of coverage ratio under different failure scenarios.

the perimeter coverage area generated by the PA algorithm is very close to the area generated under normal execution). This is more evident in experiments with 10% and 20% failures where the maximum value for each experiment is identical to the highest value of the 3^{rd} quartile. Finally, we observe that in all experiments there are scenarios (5% of the cases) where the coverage ratio is 20-25% below the average (illustrated by the bottom whisker lines). Investigating the individual scenes, we found out that this occurs when 3 or more perimeter nodes fail. However, in the majority of cases (95%) the PA algorithm maintains a competitive coverage ratio under node failures.

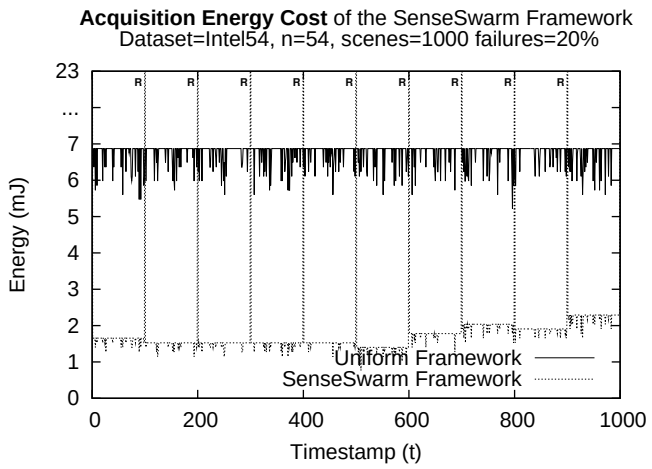

Fig. 10 Evaluating the energy cost of acquiring data at the perimeter of the swarm (SenseSwarm) versus the cost of acquiring information throughout the complete swarm (Uniform).

6.5 Acquisition Cost Evaluation

In the fourth experimental series, we measure the cost of operating a SenseSwarm network in which nodes suspend their sensing activity. As a baseline of comparison we utilize the *Uniform framework*, one in which all 54 sensing devices sense at any given moment. Figure 10 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 23mJ. 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 ± 2.2mJ while the Uniform framework consumes 6.7 ± 0.3mJ.

6.6 Replication Phase Evaluation: Fault Tolerance

In the fifth experimental series, we evaluate the fault-tolerance accuracy of our two replication algorithms using the metrics described in Section 6.1.

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

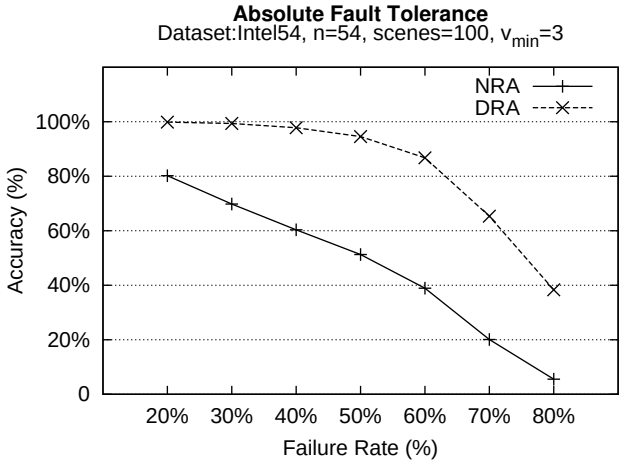

Fig. 11 Evaluating the absolute fault-tolerance accuracy (that measures the percentage of data that can be recovered) for the DRA and NRA algorithms.

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 11 illustrates the absolute 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 $O(n)$).

In the second experiment, we measure the approximate fault-tolerance accuracy of the HDRA algorithm over an increasing failure rate. Similar to the first experiment, we register 10 random queries at each individual scene requesting events captured at the perimeter nodes. This experiment differentiates from the previous one in the sense that sensor nodes participating in the query are able to return a MBR in the cases where the event requested by the query is not discovered in the sensors local storage. Note that an MBR is only returned if its rectangle/area encloses the event requested by the query. In the worst case example, the network will return the MBR stored at the sink

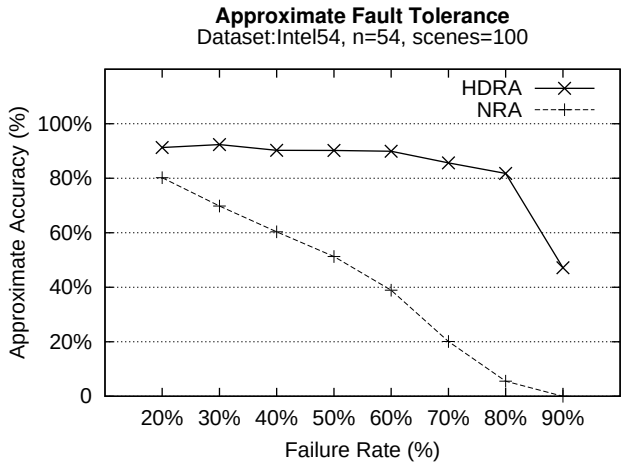

Fig. 12 Evaluating the approximate fault-tolerance accuracy (that penalizes recovered answers with large MBRs) for the HDRA and NRA algorithms.

(i.e., the area that encloses all events). Consequently, in order to measure the *approximate fault-tolerance accuracy* Φ, we use the following formula:

$$\Phi = 1 - \frac{E_Q}{E_{sink}}$$

where E_Q is the area defined by the MBR returned by some query Q, and E_{sink} is the area defined by the MBR stored at the sink. Simply put, the above formula favors results that are more precise (i.e., E_Q is small).

Figure 12 illustrates the approximate fault-tolerance accuracy of the HDRA algorithm over an increasing failure rate. We observe that HDRA is able to capture requested events with very high approximate fault-tolerance accuracy, even at failure rates as high as 80%. This is due to the fact that in HDRA, detected events are not only replicated to near-by core nodes but are also hierarchically stored to many more nodes in the form of MBRs. As a result, a query requesting these events will most likely receive either the exact events or a close MBR approximation to them. Finally, note that in the extreme case where all perimeter notes detect new events, the message complexity of HDRA is $O(n)$ (i.e., nodes will recursively transmit their data and MBRs to their parent nodes until all results arrive at the sink node).

6.7 Replication Phase Evaluation: Scalability

In the final experimental series, we evaluate the scalability of our DRA and HDRA algorithms. We measure the Absolute (DRA) and Approximate (HDRA) fault tolerance accuracy using 4 networks with different number of nodes. We utilize a 50% failure rate in all experiments in order to test our algorithms

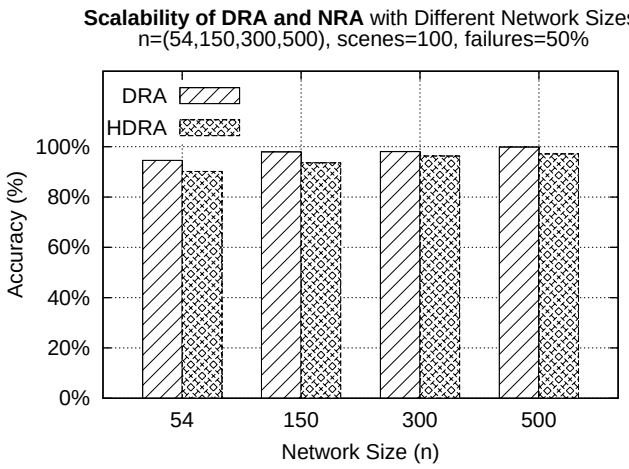

Fig. 13 Evaluating the scalability of the DRA and HDRA algorithms.

accuracy in a high risk scenario. Figure 13 illustrates the results of this experiment.

We observe that both the DRA and HDRA algorithms maintain a high degree of accuracy in all experiments. Additionally, we observe that as the network size increases, both of the algorithms present increased accuracy. The reason behind this is that since the number of sensors increases the results are distributed farther into the network. This rapidly decreases the probability of losing results which can only occur if a number of neighboring nodes fail simultaneously.

7 Conclusions and Future Work

This paper presents 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 and HDRA algorithms. Our trace-driven experimentation with realistic data shows that our framework offers singnificant energy reductions while maintaining high data availability rates. In particular, we found that even with 60% system failures we can recover the 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. We additionally plan to develop a real people-centric application founded on the ideas presented in this work.

Acknowledgements: We would like to thank Polys Kourousides for the insightful discussions regarding the perimeter construction algorithm. This work was supported in part by the University of Cyprus under a Startup Grant of the second author, the Open University of Cyprus under project SenseView, the US National Science Foundation under the project AQSIOS (#IIS-0534531), the European Union under the projects IPAC (#224395) and CONET (#224053), and the project FireWatch (#0609-BIE/09), sponsored by the Cyprus Research Promotion foundation.

References

1. Aly M., Pruhs K., Chrysanthis P.K., "KDDCS: a load-balanced in-network data-centric storage scheme for sensor networks", In Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM), Arlington, Virginia, USA, November 6-11, pp.317-326, 2006.
2. Andreou P., Zeinalipour-Yazti D., Andreou M., Chrysanthis P.K., Samaras G., "Perimeter-Based Data Replication and Aggregation in Mobile Sensor Networks" In Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM), Taipei, Taiwan, May 18-20, pp.244-251, 2009.
3. Bergbreiter, S.; Pister, K.S.J., "CotsBots: An Off-the-Shelf Platform for Distributed Robotics,", In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, October 28-30, pp.27-31, 2003.
4. Campbell A.T., Eisenman S.B., Lane N.D., Miluzzo E., Peterson R.A., Lu H., Zheng X., Musolesi M., Fodor K., Ahn G.S., "The Rise of People-Centric Sensing", In IEEE Internet Computing Vol. 12, No. 4, pp.12-21, 2008.
5. Chintalapudi K. and Govindan R., "Localized Edge Detection In Sensor Fields", In Ad Hoc Networks, Vol. 1, No. 1, pp. 273-291, 2003.
6. 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.
7. Crossbow Technology Inc, http://www.xbow.com/
8. Chrysanthis P.K. and Labrinidis A., "NSF Workshop on Data Management for Mobile Sensor Networks Report", Pittsburgh, USA, Jan 16-17, 2007.
9. Dantu K., Rahimi M.H., Shah H., Babel S., Dhariwal A., and Sukhatme G.S., "Robomote: Enabling mobility in sensor networks", In Proceedings of the 4th international symposium on Information Processing in Sensor Networks (IPSN-SPOTS), Los Angeles, California, April 25-27, No.55, 2005.
10. 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 Proceeding of the 6th international conference on Mobile Systems, applications, and services (MobiSys), Breckenridge, CO, USA, June 17-20, pp.29-39, 2008.
11. 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 Proceedings of the 5th international conference on Embedded Networked Sensor Systems (SenSys), Sydney, Australia, Noveber 6-9, pp.117-129, 2007.
12. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., "System Architecture Directions for Networked Sensors", In ACM SIGPLAN Notices, Vol.34, No.5, pp.93-104, 2000.
13. 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 Proceedings of the 4th international conference on Embedded Networked Sensor Systems (SenSys), Boulder, Colorado, USA, October 31 - November 3, pp.125-138, 2006.
14. Intanagonwiwat C., Govindan R. Estrin D., "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In Proceedings of the 6th annual international conference on Mobile Computing and Networking (MobiCom), Boston, Massachusetts, USA, August 6-11, pp. 56-67, 2000.

15. Intel Lab Data, http://db.csail.mit.edu/labdata/labdata.html
16. Jalodia S., Mutchler D., "Dynamic Voting Algorithms for Maintaining the Consistency of a Replicated Database", In ACM Transactions on Database Systems (TODS), Vol.15, pp.230-280, June, 1990.
17. Karp B., Kung H.T., "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", In Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom), Boston, Massachusetts, USA, August 6-11, pp.243-254, 2000.
18. Koren I., Krishna C.M., "Fault-Tolerant Systems", Elsevier, ISBN: 978-0-12-088525-1, 2007.
19. Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "The Design of an Acquisitional Query Processor for Sensor Networks", In Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD), San Diego, California, USA, June 9-12, pp.491-502, 2003.
20. Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In Proceedings of the 5th symposium on Operating systems design and implementation (OSDI), Vol.36, Issue.SI, pp.131-146, 2002.
21. Mani A., Rajashekhar M., Levis P. "TINX: a tiny index design for flash memory on wireless sensor devices", In Proceedings of the 4th international conference on Embedded networked sensor systems (Sensys), Boulder, Colorado, USA, October 31 - November 3, pp.425-426, 2006.
22. Monterey Bay Aquarium Research Institute (MBARI), http://www.mbari.org/rd/
23. Nascimento M.A., Alencar R.A.E., Brayner A., "Optimizing Query Processing in Cache-Aware Wireless Sensor Networks", In SpringerLink, Lecture Notes in Computer Science, Vol. 6187, pp.60-77, 2010.
24. 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", In IEEE Robotics and Automation Magazine, Vol. 9, No. 4, December, 2002.
25. Nittel S., Trigoni N., Ferentinos K., Neville F., Nural A., Pettigrew N., "A drift-tolerant model for data management in ocean sensor networks", In Proceedings of the 6th ACM international workshop on Data engineering for wireless and mobile access (MobiDE), Beijing, China, June 10, pp.49-58, 2007.
26. Purohit A., Zhang P., "SensorFly: a controlled-mobile aerial sensor network", In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, pp.327-328, 2009.
27. Rao A., Ratnasamy S., Papadimitriou C., Shenker S., Stoica I., "Geographic Routing without Location Information", In Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom), San Diego, CA, USA, September 14-19, pp.96-108, 2003.
28. 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 Mobile Networks and Applications (MONET), Vol. 8, No. 4, pp.427-442, 2003.
29. Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model", In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH), pp.25-34, 1987.
30. Sadler C., Zhang P., Martonosi M., Lyon S., "Hardware Design Experiences in ZebraNet", In Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys), Baltimore, MD, USA, November 3-5, pp.227-238, 2004.
31. Shenker S., Ratnasamy S., Karp B., Govindan R., Estrin D., "Data-centric storage in sensornets", In ACM SIGCOMM Computer Communication Review, Vol. 33, No. 1, pp.137-142, 2003.
32. Srinivasan S., Ramamritham K., Kulkarni P., "ACE in Hole: Adaptive Contour Estimation Using Collaborating Mobile Sensors", In Proceedings of the 7th international conference on Information processing in sensor networks (IPSN), St. Louis, Missouri, USA, April 22-24, pp.147-158, 2008.
33. Szewczyk R., Mainwaring A., Polastre J., Anderson J., Culler D., "An Analysis of a Large Scale Habitat Monitoring Application", In Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys), Baltimore, MD, USA, November 3-5, pp.214-226, 2004.

34. Wu S-H., Chuang K-T., Chen C-M., Chen M-S., "DIKNN: An Itinerary-based KNN Query Processing Algorithm for Mobile Sensor Networks", In Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey, April 15-20, pp.456-465, 2007.
35. 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.
36. Zeinalipour-Yazti D., Andreou P., Chrysanthis P. and Samaras G., "MINT Views: Materialized In-Network Top-k Views in Sensor Networks", In Proceedings of the 8th International Conference on Mobile Data Management, Mannheim, Germany, May 7 - 11, pp.182-189, 2007.
37. Zeinalipour-Yazti D., Andreou P., Chrysanthis P.K., Samaras G., "SenseSwarm: a perimeter-based data acquisition framework for mobile sensor networks", In Proceedings of the 4th workshop on Data management for sensor networks: in conjunction with 33rd International Conference on Very Large Data Bases (DMSN), Vienna, Austria, September 24, pp.13-18, 2007.
38. Zeinalipour-Yazti D., Chrysanthis P.K., "Mobile Sensor Network Data Management" Book Chapter in the Encyclopedia of Database Systems (EDBS), Editors: Ozsu, M. Tamer; Liu, Ling (Eds.), ISBN: 978-0-387-49616-0, 2009.
39. Zeinalipour-Yazti D., Lin S., Kalogeraki V., Gunopulos D., Najjar W., "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices", In Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies (FAST), San Francisco, CA, USA, December 13-16, pp.3, 2005.
40. Zhang P., Martonosi M., "LOCALE: Collaborative Localization Estimation for Sparse Mobile Sensor Networks", In Proceedings of the 7th international conference on Information processing in sensor networks (IPSN), St. Louis, Missouri, USA, April 22-24, pp.195-206, 2008.