Information Systems 36 (2011) 386–405

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Collection trees for event-monitoring queries

Antonios Deligiannakis [a], Yannis Kotidis [b,*], Vassilis Stoumpos [c], Alex Delis [c]

[a] Technical University of Crete, Greece
[b] Athens University of Economics and Business, Greece
[c] University of Athens, Greece

ARTICLE INFO

Article history:
Received 2 June 2009
Received in revised form
2 June 2010
Accepted 19 August 2010
Recommended by: B. Kemme

Keywords:
Aggregate queries
Sensor networks

ABSTRACT

In this paper we present algorithms for building and maintaining efficient collection trees that provide the conduit to disseminate data required for processing monitoring queries in a wireless sensor network. While prior techniques base their operation on the assumption that the sensor nodes that collect data relevant to a specified query need to include their measurements in the query result at every query epoch, in many event monitoring applications such an assumption is not valid. We introduce and formalize the notion of event monitoring queries and demonstrate that they can capture a large class of monitoring applications. We then show techniques which, using a small set of intuitive statistics, can compute collection trees that minimize important resources such as the number of messages exchanged among the nodes or the overall energy consumption. Our experiments demonstrate that our techniques can organize the data collection process while utilizing significantly lower resources than prior approaches.

1. Introduction

Many pervasive applications rely on sensory devices that are able to observe their environment and perform simple computational tasks. Driven by constant advances in microelectronics and the economy of scale, it is becoming increasingly clear that our future will incorporate a plethora of such sensing devices that will participate and help us in our daily activities. Even though each sensor node will be rather limited in terms of storage, processing and communication capabilities, they will be able to accomplish complex tasks through intelligent collaboration.

Nevertheless, building a viable sensory infrastructure cannot be achieved through mass production and deployment of such devices without addressing first the technical challenges of managing such networks. In this paper we focus on developing the necessary data collection infrastructure for supporting data-hungry applications that need to acquire and process readings from a large scale sensor network. While previous work has focused on optimizing specific types of queries such as aggregate [23], join [2], model-based [11,20] and select-all [8,29] queries, we propose a data dissemination framework that can address the needs of multiple, concurrent data acquisition requests in an efficient manner.

It is generally agreed that one cannot simply move the readings necessary for processing an application request out of the network and then perform the required processing in a designated node such as a *base station*. Wireless sensor nodes have limited energy capacity and such an approach will not only result in overburdening their radio links, but will also quickly drain their energy as radio transmission is by far the most important factor in energy consumption [24]. Thus, most recent proposals rely on building some type of ad hoc interconnect for answering a query such as the *aggregation tree* [23,34]. This is a paradigm of in-network processing that can be

* Corresponding author.
E-mail addresses: adeli@softnet.tuc.gr (A. Deligiannakis), kotidis@aueb.gr (Y. Kotidis), stoumpos@di.uoa.gr (V. Stoumpos), ad@di.uoa.gr (A. Delis).

0306-4379/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2010.08.003

applied to non-aggregate queries as well [8]. In this paper we concentrate on building and maintaining efficient *data collection trees* that will provide the conduit to disseminate all data required for processing many concurrent queries in a sensor network, including long-term and ad hoc type of queries, while minimizing important resources such as the number of messages exchanged among the nodes or the overall energy consumption.

While prior work [5,30,32] has also tackled similar problems, previous techniques base their operation on the assumption that the sensor nodes that collect data relevant to the specified query need to include their measurements (and, thus, perform transmissions) in the query result at every query *epoch*. However, in many monitoring applications such an assumption is not valid. Monitoring nodes are often interested in obtaining either the actual readings, or their aggregate values, from sensor nodes that detect interesting events. The detection of such events can often be identified by the readings of each sensor node. For example, in vehicle tracking and monitoring applications high noise levels may indicate the proximity of a vehicle. In military applications, high levels of detected chemicals can be used to warn nearby troops. In industrial settings, where the sensors monitor the condition of machines, high temperature readings may indicate overheating parts. In other applications, as in the case of approximate evaluation of queries over the sensor data [7,25,28], an event is defined when the current sensor reading deviates by more than a given threshold from the last transmitted value. In all of these scenarios, each sensor node is not forced to include its measurements in the query output at each epoch, but rather such a *query participation* is evaluated on a per epoch basis, depending on its readings and the definition of interesting events. In this paper we term the monitoring queries where the participation of a node is based on the detection of an event of interest as *event monitoring queries* (EMQs). It is important to note that typical monitoring queries, considered in the bulk of research so far, are a subclass of EMQs, as the former correspond to the case where the participation of sensor nodes in the query result at each epoch is fixed (either true, or not) throughout the query execution.

Our techniques base their operation on collecting simple statistics during the operation of the sensor nodes. The collected statistics involve the number of events (or, equivalently, their frequency) that each sensor detected in the recent past. Our algorithms utilize these statistics as hints for the behavior of each sensor in the near future and periodically reorganize the collection tree in order to minimize certain metrics of interest, such as the overall number of transmissions or the overall energy consumption in the network. The formation of the collection tree is based on the collection and local transmission of only a small set of values at each node termed as *cost factors* in our framework. Using these cost factors each sensor selects its parent node, through which it will forward its results towards the base station, based on the estimated corresponding *attachment cost*. In a nutshell, the attachment cost of a parent selection is the increase in the objective function (i.e., the number of transmitted messages) resulting from this selection. Given the estimates of attachment costs that our algorithms compute, our work demonstrates that they are able to design significantly better collection trees than existing techniques.

Our contributions are summarized as follows:

- We formally introduce the notion of EMQs in sensor networks. EMQs are a superset of existing monitoring queries, but are handled uniformly in our framework, irrespectively of the minimization metric of interest.
- We present detailed algorithms for minimizing important metrics such as the number of messages exchanged or the energy consumption during the execution of an EMQ. The presented algorithms are based on the collection and transmission of a small, and of constant size, set of statistics. We introduce our algorithms along with a succinct mathematical justification.
- We present alternative techniques that we considered in our work and discuss their intuition and drawbacks.
- We extend our framework for the case of multiple concurrent EMQs of different types.
- We present a detailed experimental evaluation of our algorithms. Our results demonstrate that our techniques can achieve a significant reduction in the number of transmitted messages, or the overall energy consumption, compared to alternative algorithms.

2. Related work

The database community has long been the advocate of using an embedded database management system for data acquisition in sensor networks [23,34]. The use of a declarative SQL-like query interface allows rapid development of applications in such systems without the need to manage hand-coded programs at each sensor node [24].

In the database community different types of popular queries have been discussed, such as aggregate [6,7,23,26,28], join [2], model-based [11,20] and select-all queries [8,29]. Tracking queries that seek to determine the spatial extent of a particular phenomenon have also been considered [12,33]. In [23] the nodes are first organized in a tree topology, termed the aggregation tree. During query execution, each epoch is subdivided into intervals and parent nodes in the aggregation tree listen for messages containing partial aggregates from their children nodes during pre-defined time-slots. The rest of the time the nodes may power-down their radios in order to reduce their energy and bandwidth consumption. Another notable method for synchronizing the transmission periods of nodes is the recently proposed wave scheduling approach of [10]. The work in [36] describes a framework that profiles recent data acquisition activity by the nodes and computes their waking window though an in-network execution of the critical path method. This technique is complementary to our work, as it helps identify a proper scheduling for data transmission by the

nodes, while our methods focus on optimizing the routing topology. Alternative techniques do not utilize an aggregation tree but rather they compute aggregation queries using decentralized algorithms [3,19].

Many of the low-level networking details have already been discussed in the networking community and, thus, can be utilized in our framework. As an example, nodes in unattended wireless networks must be able to self-configure [4] and discover their surrounding nodes [13]. Prior work on computing energy-efficient data routing paths (such as the aggregation tree) [16,30] have tackled similar problems, but these techniques base their operation on the assumption that the sensor nodes that collect data relevant to the specified query need to include their measurements in the query result at every query epoch. However, this assumption does not hold in event monitoring queries that are the scope of our framework. In the other end of the spectrum, the work in [21,14] discuss join and aggregation queries involving rare events. Thus, they follow an alternative path, which is to construct the data collection network on-the-fly when such events occur. However, this practice in unsuitable for out setting involving sensor nodes with both low and high participation frequencies, since it would incur a high overhead for frequently maintaining the collection network. Furthermore, the work in [14] assumes the existence of a high speed connection for all nodes at the boundaries of the network, through which the data that reaches the boundary nodes can be communicated.

Recently, there has been a realization that readings from sensor nodes are inherently dirty [9,15,22,31]. In [17] the authors present ESP, a data cleaning framework in support of pervasive applications. ESP allows programmers to specify five pipelined cleaning stages using high-level declarative queries over data streams produced by the sensors. The same issue arises in other domains, such as in data streams generated by RFID applications [18]. Our framework may also compliment this work, as often data cleansing can be performed by appropriate query rewriting [17] and, thus, results in a continuous query that can use our optimization framework for data gathering.

3. Motivational example

In Table 1 we present examples of the two main classes of monitoring queries in sensor networks. We borrow the syntax of TinyOS [23] to denote the epoch duration (e) and the lifetime of the query (t). The predicate *inclusionConditions* has been added in order to specify which sensor nodes will participate in the query evaluation per epoch. At each query epoch, all the sensor nodes that include their collected data in the query result are termed in our framework as *epoch participating nodes*. For queries that wish to collect data from all the sensor nodes at each epoch, the above predicate always evaluates to *true*.

When a monitoring query specifies inclusion predicates, these may contain either static or dynamic predicates (or both) regarding the sensor nodes. Examples of static predicates may involve, but are not limited to, the collection of measurements from: (i) sensors with specific identifiers; (ii) immobile sensors in a specific area; or (iii) sensors monitoring a specific quantity, in cases of sensor networks with diverse types of sensor nodes that monitor different quantities. Static predicates are very useful in a variety of applications and have received the focus of the bulk of past research [23,34]. Inclusion conditions that contain only static predicates result in a fixed subset of the sensor nodes participating in the query output at each epoch. This allows for simple data dissemination and collection protocols based on fixed collection trees that need to be altered only when either node or communication failures exist.

However, there exists a large class of monitoring queries that cannot be expressed using static inclusion conditions. Examples include vehicle tracking and equipment monitoring applications where inclusion predicates need to be conditioned on readings taken by the sensor nodes such as noise levels or temperature readings. In its most simple form a dynamic inclusion predicate may be a condition of the form "current reading $>$ threshold". More complex forms may require the evaluation of a user defined function over a history of accumulated readings. In the case of approximate evaluation of queries over the sensor data [7,25,28], the inclusion predicate is satisfied when the current sensor reading deviates by more than a given threshold from the last transmitted value. We call such predicates, whose evaluation depends also on the readings taken by the nodes, as dynamic predicates as they specify which nodes should include their response in the query evaluation at each epoch (i.e., nodes whose values exceed a given threshold, or deviate significantly from previous readings). We term those monitoring queries that contain dynamic predicates as *event monitoring queries* (EMQs).

Given a monitoring query, existing techniques seek to develop *collection trees* that specify the way that the data are forwarded from the sensor nodes to the `Root` node. Periodically these collection trees may be reorganized in order to adapt to evolving data characteristics [28].

An important characteristic of EMQs, which is not taken into account by existing algorithms that design collection trees, is that each sensor node may participate

Table 1
An aggregate and a non-aggregate query over the values collected by the sensor nodes.

Aggregate query		Non-aggregate query	
SELECT	AggrFun(s.value)	SELECT	s.id, s.value
FROM	Sensors s	FROM	Sensors s
WHERE	inclusionConditions(s)=true	WHERE	inclusionConditions(s)=true
SAMPLE	PERIOD e FOR t	SAMPLE	PERIOD e FOR t

in the query evaluation, by including its reading in the query result, only a limited number of times, based on how often the inclusion conditions are satisfied. We can thus associate an *epoch participation frequency* P_i with each sensor node S_i, which specifies the fraction of epochs that this node participated in the query result in the recent past.

Given estimates of the epoch participation frequencies, one can design significantly more efficient collection trees than prior approaches. Consider the sample scenario depicted in Fig. 1(a). In this figure, 100 sensor nodes are placed in a grid. The sensor identifiers appear next to each sensor node. We also distinguish the `Root` node at the lower left corner (depicted as a dark square), a monitoring node that performs queries over the data collected by the sensor nodes. In our sample network we assume that each sensor node can communicate with its immediate horizontal, vertical or diagonal neighbors, while only node S_{90} can communicate with the `Root` node. In Fig. 1(b) we depict sample estimates for the number of times each sensor node will participate in the query result within the next 100 epochs. Thus, the epoch participation frequencies for all the sensor nodes, can be derived by dividing these values by 100. In the above scenario, given the presented epoch participation frequencies, six interior nodes along with all the boundary nodes on the upper and rightmost edges of the network always detect events, while the remaining interior nodes detect events with a lower probability, whose average value is about 5%. For the aforementioned sample scenario, in Fig. 2(a) we depict a sample collection tree chosen by an algorithm, termed as MinHops that seeks to minimize the number of hops that each node's data needs to traverse until it reaches the `Root` node. TAG [23] utilizes such an algorithm for

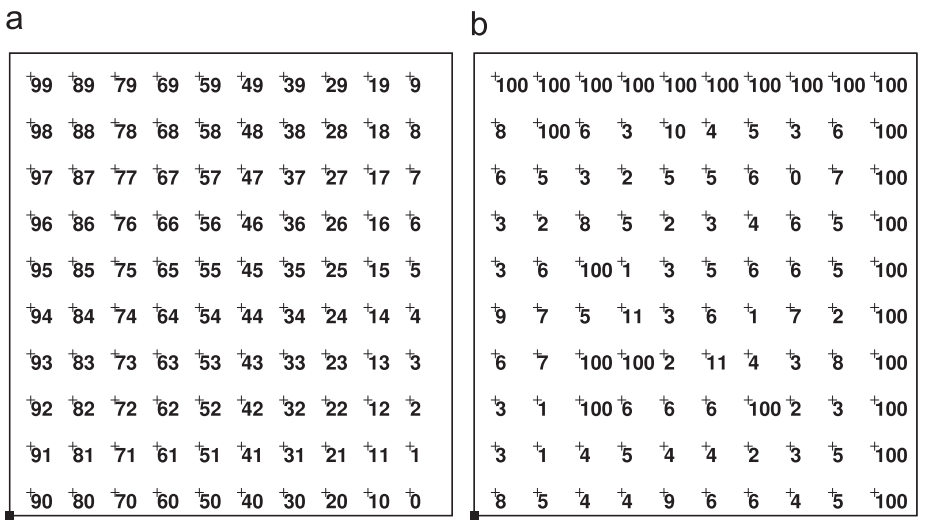

Fig. 1. (a) Identifiers of sensors in grid arrangement; (b) estimated number of participations in query result in 100 epochs.

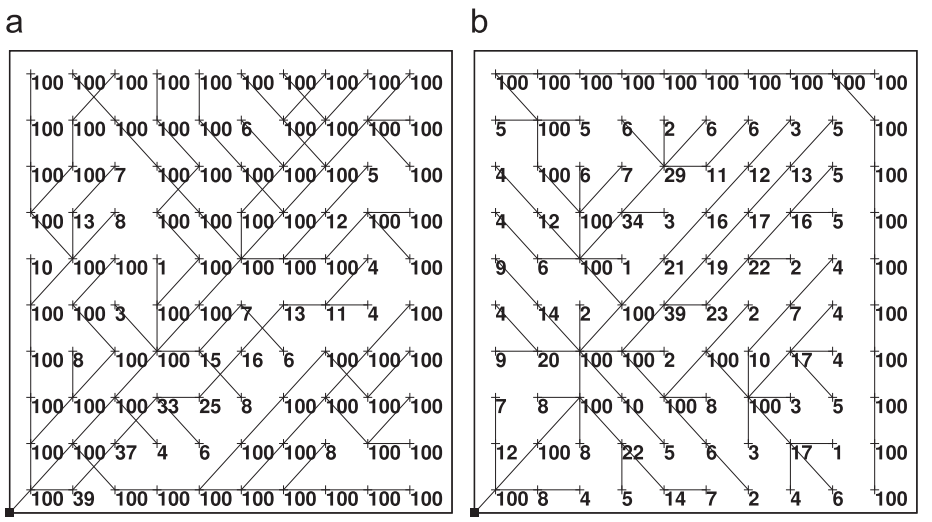

Fig. 2. (a) Collection tree for MinHops algorithm. Cost=7709 transmissions; (b) collection tree for our algorithm. Cost=3838 transmissions.

building the collection tree. Next to each node we depict the actual number of transmissions that each node performed within these 100 epochs. Similarly, in Fig. 2(b) we present the collection tree that our algorithms created for the evaluation of the SUM aggregate. One can easily establish the significant reduction in the number of transmissions that our algorithm achieved (3838 vs 7709, a twofold reduction).

A first observation is that our algorithm seeks to forward the query results from nodes with high epoch participation frequencies through a limited number of interior nodes, compared to the *MinHops* algorithm. Moreover, we note that our algorithm does not necessarily route its messages through the neighboring node with the highest epoch participation frequency. For example, node S_{14} has chosen to forward its results through node S_{23} and not through nodes S_{24} or S_{13}, even though P_{23}=3% is lower than both P_{24}=7% and P_{13}=8%. While, a detailed description of why this occurs is deferred until later in this paper, we note that what is important when deciding how to best form the collection tree is the estimated impact that each of the algorithm's decisions has on the desired minimization metric. For example, when selecting a *parent* node in the collection tree, the impact may involve (depending on the minimization metric) the estimated increase in the transmitted messages in the path from the parent node towards the `Root` node. Given this brief explanation, we note in our example that node S_{23} is an immediate neighbor of S_{32}, a node with a high epoch participation frequency. Thus, since the nodes in the path of S_{32} to the `Root` node will probably be forced in making several transmissions, due to the events transmitted by S_{32}, the additional messages transmitted by having S_{14} select S_{23} as its parent node will mostly influence the path between S_{23} and S_{32}. Such an observation cannot be made for S_{24} or S_{13}, since they lie one hop further from a node with a high epoch participation frequency.

4. Problem formulation

We first introduce the types of EMQs that our framework supports and then present the optimization problems tackled in this paper. We then discuss the cost model used in our algorithms in order to estimate the energy consumption of a sensor node during the transmission process.

4.1. Supported queries

In Table 1 we presented the two main classes of SQL queries that our framework supports. It is important to emphasize at this point that even non-participating nodes may take part in the query evaluation process by forwarding messages towards the `Root` node. However, the collected values of non-participating nodes influence neither the reported query result nor its size.

The first class of supported queries involve non-aggregate queries over the values of epoch participating sensor nodes. In this type of queries the amount of data transmitted by any node of the collection tree depends on the number of epoch-participating sensors that are descendants of that sensor node.

The second class of supported queries involves aggregate functions over the measurements collected by the participating sensor nodes. A good classification of aggregate functions is presented in [23], depending on the amount and type of state required in non-leaf nodes in order for them to calculate the aggregate result for the partition of descendant, in the collection tree, participating sensors. Table 2 summarizes this classification.

A crucial part of the operation of our algorithms is the estimation of the amount of data that will be transmitted in a given (or candidate) collection tree. In order to accurately estimate this information, the aggregate function being used needs to be distributive, algebraic or holistic (see Table 2). Unique and content-sensitive aggregate functions can only be supported by using a worst case estimate for the amount of transmitted data. Note that holistic aggregate queries share similar characteristics with non-aggregate queries and, thus, are treated in a similar way in our framework.

4.2. Problem definition

In this paper we seek to develop dissemination protocols for the classes of EMQs described in Section 4.1. The goal is, given the type of query at question, to design the collection tree so as to minimize either:

1. The number of transmitted messages in the network.
2. The overall energy consumption in the network.

The minimization of additional metrics of interest is discussed in Section 6. Our algorithms do not make any assumptions about the placement of the sensor nodes, their characteristics or their radio models. However, in order to simplify the presentation, in our discussion we will focus on networks where any communication between pairs of sensor nodes is either bidirectional or impossible.

Table 2
Characteristics and examples of aggregate function types.

Category	Type of partial state needed	State size	Examples
Distributive	Aggregate values for descendants	Constant	MAX, MIN COUNT, SUM
Algebraic	Aggregate values for descendants, but for different aggregate function	Constant	AVG
Holistic	Entire data of descendants	Proportional to #epoch-participating descendants	MEDIAN
Unique	Distinct values of descendants	Data-dependent	COUNT DISTINCT
Content-sensitive	Aggregate-specific	Data-dependent	Histogram of values

4.3. Energy consumption cost model

A sensor node consumes energy at all stages of its operation. However, this energy consumption is minimal when the sensor is in a sleep mode. Furthermore, the energy drain due to computations may, in some applications, be significant, but it is typically much smaller than the cost of communication [24]. Due to this fact and because our algorithms do not require any significant computational effort by the sensor nodes, we ignore in the cost model the power consumption when the sensor node is idle and the consumption due to computations. We will thus focus on capturing the energy drain due to data communication in data driven applications. More particular, we need to estimate the energy consumption of a node S_i when either transmitting, receiving or idle listening for data. The notation that will be used in our discussion here, and later in the description of our algorithms, is presented in Table 4. Additional definitions and explanations are presented in appropriate areas of the text.

We first describe the cost model used to estimate the energy consumption of a node S_i during the data transmission of $|aggr| > 0$ bits of data to node S_j, which lies in distance $dist_{i,j}$ from S_i. The energy cost can be estimated using a linear model [27] as

$$E_{tr_{i,j}} = SC_i + (H + |aggr|) \times (E_{TX_i} + E_{RF_i} \times dist_{i,j}^2) \quad (1)$$

where (i) SC_i denotes the energy startup cost for the data transmission of S_i. This cost depends on the radio used by the sensor node; (ii) H denotes the size of the packet's header; (iii) E_{TX_i} denotes the per bit power dissipation of the transmitter electronics; and (iv) E_{RF_i} denotes the per bit and squared distance power delivered by the power amplifier. This power depends on the maximum desired communication range and, thus, from the distance of the nodes with which S_i desires to communicate. Thus, the additional energy consumption required to augment an existing packet from S_i to S_j with additional $|aggr|$ bits can be calculated as: $DE_{tr_{i,j}} = |aggr| \times (E_{TX_i} + E_{RF_i} \times dist_{i,j}^2)$.

For the case when each sensor node receives data, we need to keep in mind that each sensor must open its radio in order to receive data or queries transmitted by neighboring nodes. This startup cost is incurred when the node wakes up from its sleep mode and, in contrast to the data transmission case, is not directly related to the reception of data (since the sensor may receive no data). Thus, this mandatory cost is not taken into account in our model.

When a sensor node S_i receives $H+b_j$ bits from node S_j, then the energy consumed by S_i is given by: $E_{rec_i} = E_{RX_i} \times (H + b_j)$, where the value of E_{RX_i} depends on the radio model. Some typical values [27] of SC, E_{TX}, E_{RX} and E_{RF} are presented in Table 3.

The energy consumed by a sensor node when idle listening for data is significant and often comparable to the energy of receiving data. For example, in the popular MICA2 nodes the ratios for radio power draw during idle-listening, receiving of a message and transmission are 1:1:1.41 at 433 MHz with RF signal power of 1 mW in transmission mode [35]. Thus, due to the similar energy consumption by a sensor while either receiving or idle listening for data, our algorithms focus on the energy drain during the transmission of data.

Table 3
Typical radio parameters.

Symbol	Typical value
SC	1 μJ
E_{TX}	50 nJ/bit
E_{RF}	100 pJ/bit/m^2
E_{RX}	50 nJ/bit

Table 4
Symbols used in our algorithm.

Symbol	Description
`Root`	The node that initiates a query and which collects the relevant data of the sensor nodes
S_i	The i-th sensor node
P_i	The epoch participation frequency of S_i
D_i	The minimum distance, in number of hops, of S_i from the `Root`
$\|aggr\|$	The size of the (non-)aggregate values transmitted by a node
$E_{tr_{i,j}}$	Energy spent by S_i to transmit a new packet of $\|aggr\|$ bits to S_j
$DE_{tr_{i,j}}$	Energy spent by S_i to transmit additional $\|aggr\|$ bits to S_j (on an existing packet)
$AC_{i,j}$	Attachment cost of S_i to a candidate parent S_j
CF_i, DCF_i, HCF_i	Cost factors utilized by neighboring nodes of S_i when estimating their attachment cost to S_i

5. Algorithm overview

We now present our algorithms for creating and maintaining a collection tree that minimizes the desired metric (number of messages or energy consumption). We also provide detailed pseudocode in addition to a formal analysis. Our algorithms are based on a top-down formation of the collection tree. The intuition behind such an approach is that the epoch participation frequency of each node in the collection tree influences the transmission frequency of only nodes that lie in its path to the `Root`. We thus demonstrate in this section that estimating the magnitude of this influence can be easily achieved by a top-down construction of the collection tree, while requiring the transmission of only a small set of statistics.

5.1. Construction/update of the collection tree

The algorithm is initiated with the query propagation phase and periodically, when the collection tree is scheduled for reorganization. The query is propagated from the base station through the network using a flooding algorithm. In densely populated sensor networks, a node S_i may receive the announcement of the query from several of its neighbors. As in [23,34] the node will select one of these nodes as its *parent node*. The chosen parent will be the one that exhibits the lowest *attachment cost*,

meaning the lowest expected increase in the objective minimization function. For example, if our objective is to minimize the total number of transmitted messages, then the selection will be the node that is expected to result in the lowest increase in the number of transmitted messages in the *entire* path from that sensor until the `Root` node (and similarly for the rest of the minimization metrics). At this point we simply note that in order for other nodes to compute their attachment cost, node S_i transmits a small set of statistics $Stats_i$ and defer their exact definition for Section 5.2.

The result of this process is a collection tree towards the base station that initiated the flooding process. A key point in our framework is that the preliminary selection of a parent node may be revised in a second step where each node evaluates the cost of using one of its sibling nodes as an alternative parent. Due to the nature of the query propagation, and given simple synchronization protocols, such as those specified in [23], the nodes lying k hops from the `Root` node will receive the query announcement before the nodes that lie one hop further from the `Root` node. Let $RecS_k$ denote the set of nodes that receive the query announcement for the first time during the k-th step of the query propagation phase.

At step k of the query propagation phase, after the preliminary parent selection has been performed, each node S_i in set $RecS_k$, needs to consider whether it is preferable to alter its current selection and choose as its parent a *sibling node* within set $RecS_k - S_i$.[1] Each node calculates a new set of statistics $Stats_i$, based on its preliminary parent selection, and transmits an *invitation*, which also includes the node's newly calculated $Stats_i$ values, that other nodes in $RecS_k$ (and only these nodes) may accept. Of course, we need to be careful at this point and make sure that at least one node within $RecS_k$ will not accept any invitation, as this would create a disconnected network and prevent nodes from $RecS_k$ to forward their results to nodes belonging in $RecS_{k-1}$. We will achieve this by imposing a simple set of rules regarding when an invitation may be accepted by a sensor node.

Let $CandPar_i$ denote the set of nodes in $RecS_k$ that transmitted an invitation that S_i received. Let S_m be the preliminary parent node of S_i, as decided during query propagation. Amongst the nodes in $CandPar_i$, node S_i considers the node S_p such as the attachment cost $AC_{i,p}$ is minimized. If ties occur, then these are broken using the node identifiers (i.e., prefer the node with the highest id).[2] Then S_p is selected as the parent of S_i *instead of the preliminary choice* S_m only if all of the following conditions apply:

- $AC_{i,p} < AC_{i,m}$. This condition ensures that S_p seems as a better candidate parent than the current selection S_m.
- $AC_{i,p} \leq AC_{p,i}$. This condition ensures that it is better to select S_p as the parent of S_i, than to select S_i as the parent of S_p.
- If $AC_{i,p} = AC_{p,i}$, then the identifier of S_p is also larger than the identifier of S_i. This condition is useful in order to allow nodes to forward messages through neighbor nodes in $RecS_k$ and also helps break ties amongst nodes and to prevent the creation of loops.

The collection tree may periodically get updated, either because of a significant change in data distribution or because of the addition/termination of queries in a multi-query setup discussed in Section 7. Such updates are triggered by the base station using the same protocol used in the initial creation. In this case, the nodes compute and transmit their computed statistics in the same manner, but do not need to propagate the query itself.

5.2. Calculating the attachment cost

Determining the candidate parent with the lowest attachment cost is not an easy decision, as it depends on several parameters. For example, it is hard to quantify the resulting transmission probability of S_j, if a node S_i decides to select S_j as its parent node. In general, the transmission frequency of S_j (note that this is different than the epoch participation frequency of the node) may end up being as high as $\min\{P_i+P_j,1\}$ (when nodes transmit on different epochs) and as low as P_j (when transmissions happen on the same epochs and $P_i \leq P_j$). A commonly used technique that we have adopted in our work is to consider that the epoch participation by each node is determined by independent events. Using this independence assumption, node S_j will end up transmitting with a probability $P_i+P_j-P_iP_j$, an increase of $P_i(1-P_j)$ over P_j. Similarly, if S_{j-1} is the parent of S_j, this increase will also result in an increase in the transmission frequency of S_{j-1} by $P_i(1-P_j)(1-P_{j-1})$, etc. In our following discussion, for ease of presentation, when considering the attachment cost of S_i to a node S_j, we will assume that the nodes in the path from S_j to the `Root` node are the nodes S_{j-1}, $S_{j-2},\ldots,S_1$.

5.2.1. Minimizing the number of transmissions

The attachment cost of S_i when selecting S_j as its parent node can be calculated by the increase in the transmission frequency of each link from S_i to the `Root` node as

$$AC_{i,j} = P_i + P_i(1-P_j) + P_i(1-P_j)(1-P_{j-1}) + \cdots \quad (2)$$

A significant problem concerning the above estimation of $AC_{i,j}$ is that its value depends on the epoch participation frequencies of all the nodes in the path of S_j to the `Root` node. Since the number of these values depends on the actual distance, in number of hops, of S_j to the `Root` node, such a solution does not scale in large sensor networks.

Fortunately, there exists an alternative formula to calculate the above attachment cost. Our technique is based on a recursive calculation based on a single *cost factor* CF_i at each node S_i. In our example discussed above,

[1] Note that at this step any initially selected parent of a sibling node that lies within the transmission range of S_i has already been examined in the preliminary parent selection phase and does not need to be considered.

[2] Alternative choices are equally plausible. For example, prefer the nodes with the highest/lowest identifiers depending on whether this is an odd/even invocation of the collection tree formation algorithm.

the values of CF_i and $AC_{i,j}$ can be easily calculated as

$$CF_i = (1-P_i) \times (1+CF_j)$$

$$AC_{i,j} = P_i \times (1+CF_j) \quad (3)$$

One can verify that expanding the above recursive formula and setting as the boundary condition that the *CF* value of the `Root` node is zero gives the desired result. Thus, only the cost factor, which is a single statistic, is needed at each node S_j in order for all the other nodes to be able to estimate their attachment cost to S_j.

We also need to note that the formulas presented above also address the case of non-aggregate or holistic aggregate queries. In these cases the size of the transmitted data increases proportionally to the number of each node's epoch-participating descendants in the collection tree, as we approach the `Root` node. Thus, sometimes the transmitted data by a node may be split into multiple messages due to the maximum packet size. However, we first note that such cases typically occur in higher levels of the collection tree (and, thus, by a potentially small subset of the sensor nodes) and that, more importantly, our techniques seek to compute and utilize simple statistics. Our study of alternative cost models that incorporated this factor yielded only minor improvements while significantly increasing the communication cost during the collection tree formation. We thus omit such extensions from our presentation.

5.2.2. Minimizing total energy consumption, distributive and algebraic aggregates

This case is very similar to the case described above. When considering the attachment cost of S_i to a candidate parent S_j, we note that additional energy is consumed by nodes in the path of S_j to the `Root` node only if a new transmission takes place. This is because each node aggregates the partial results transmitted by its children nodes and transmits a new single partial aggregate for its sub-tree [23]. Thus, the size of the transmitted data is independent of the number of nodes in the subtree, and only the frequency of transmission may get affected. Let $E_{tr_{i,j}}$ denote the energy consumption when S_i transmits a message to S_j consisting of a header and the desired aggregate value(s)—based on whether this is a distributive or an algebraic aggregate function. The energy consumption follows the cost model presented in Section 4.3, where the E_{RF_i} value may depend on the distance between S_i and S_j (thus, the two indices used above). Using the above notation, and similarly to the previous discussion, the attachment cost $AC_{i,j}$ is calculated as

$$\begin{aligned} AC_{i,j} &= P_i \times E_{tr_{i,j}} + P_i \times (1-P_j) \times E_{tr_{j,j-1}} \\ &\quad + P_i \times (1-P_j) \times (1-P_{j-1}) \times E_{tr_{j-1,j-2}} + \cdots \\ &= P_i \times (E_{tr_{i,j}} + CF_j) \quad \text{where} \\ CF_i &= (1-P_i) \times (E_{tr_{i,j}} + CF_j) \end{aligned} \quad (4)$$

If one wishes to take the receiving cost of messages into account, all that is required is to replace in the above formulas the symbols of the form $E_{tr_{k,p}}$ with $(E_{tr_{k,p}} + E_{rec_p})$, since each message transmitted by S_k to S_p will consume energy during its reception by S_p.

5.2.3. Minimizing total energy consumption, holistic aggregate and non-aggregate queries

This case is slightly more complex than the case described above. When considering the attachment cost of S_i to a candidate parent S_j, we need not only consider the new messages generated in the path from S_j to the `Root` node, but also the energy consumption due to the increase in the length of messages that would have been transmitted anyway. Recall that the energy consumption for each transmission of $|aggr|$ bits by S_i to S_j is given by: $DE_{tr_{i,j}} = |aggr| \times (E_{TX_i} + E_{RF_i} \times dist_{i,j}^2)$. Calculating the aforementioned number of messages is simple, as we have already discovered a similar recursive formula that estimates the attachment cost when only considering the transmission of new messages. So, we will utilize two new recursively computed statistics. The *DCF* value of a node will be similar to the *CF* value, but will use the $DE_{tr_{*,*}}$ transmission costs, instead of the $E_{tr_{*,*}}$ transmission costs used in the *CF* formula. The *HCF* value of a node will be equal to the sum of the $DE_{tr_{*,*}}$ values in the nodes path to the `Root` node. One can verify that the energy consumption due to the enlargement of messages, because of the attachment of S_i to S_j, that would have been transmitted anyway is: $P_i \times (HCF_j - DCF_j)$. The required formulas are presented below:

$$CF_i = (1-P_i) \times (E_{tr_{i,j}} + CF_j)$$

$$HCF_i = DE_{tr_{i,j}} + HCF_j$$

$$DCF_i = (1-P_i) \times (DE_{tr_{i,j}} + DCF_j)$$

$$AC_{i,j} = P_i \times (E_{tr_{i,j}} + CF_j) + P_i \times (HCF_j - DCF_j) \quad (5)$$

5.2.4. Summary

Table 5 summarizes the statistics required to be transmitted by each node during the query propagation. Note that the invitation phase always requires one more transmitted statistic, as the nodes need to check whether it is more beneficial to be attached to another node or the reverse (see the last two rules in Section 5.1). As it can be clearly seen from this table, our algorithms utilize only a limited number of statistics, which are computed using only information transmitted by neighboring sensor nodes.

A final and important note that we need to make at this point involves the estimation of the attachment cost when seeking to minimize the overall energy consumption in all the types of queries discussed in this paper. When each sensor node S_i examines the invitations of neighboring nodes (and only in this step) and estimates the attachment cost to any node S_j, in our implementation it utilizes the same E_{RF} value in order to determine the value of $E_{tr_{i,j}}$, independently on the distance of S_i to S_j. This is done so that the value of $E_{tr_{i,j}}$ is the same for all candidate parents of S_i, as desired by the proof of Theorem 1 in order to guarantee the lack of loops in the formed collection tree.

Theorem 1. *For sensor networks that satisfy the connectivity requirements of Section 4.2 our algorithm always creates a connected routing path that avoids loops.*

Table 5
Statistics attached to messages.

Minimization metric	Type of aggregate	Decision	Invitation
Transmissions	Aggregate Non-aggregate	CF_i	P_i, CF_i
Energy consumption	Distributive algebraic	CF_i	P_i, CF_i
Energy consumption	Holistic non-aggregate	CF_i, HCF_i, DCF_i	P_i, CF_i, HCF_i, DCF_i

Proof. We only sketch the proof here. It is obvious that any node that will receive the query announcement will select some node as its parent node. We first demonstrate that no loops can be introduced and prove this by contradiction. Assume that the parent relationships in the created loop are as follows: $S_1 \rightarrow S_2 \rightarrow \cdots S_p \rightarrow S_1$. Let D_i be the distance (in number of hops) of node S_i from the `Root`. Since each node can select as its parent node a node with equal or lower D value, the existence of a path from S_1 to S_2 means that $D_2 \leq D_1$ and the existence of a path from S_2 to S_1 means that $D_1 \leq D_2$. Therefore, $D_1 = D_2$.

The attachment cost $AC_{i,j}$ calculated using the aforementioned statistics is of the form: $P_i \times (a_i + CF_j)$,[3] where a_i is a constant for each node S_i. Considering that S_1 selected S_2 as its parent and not S_p, we get: $AC_{1,2} \leq AC_{1,p} \Longrightarrow CF_2 \leq CF_p$. By creating such inequalities between the current parent and child of each node, summing these up (note that because one of the nodes in the loop will exhibit the highest identifier, for at least one of the above inequalities the equality is not possible), we get that: $CF_1 + \cdots + CF_p < CF_1 + \cdots + CF_p$.

We therefore reached a contradiction, which means that our algorithm cannot create any loops. □

5.3. Algorithm implementation

In Algorithm 1 we present the complete algorithm for the decisions of a sensor node. This algorithm is invoked both at the query propagation phase and when updating the collection tree. Each node first waits to receive the decisions by nodes that lie one hop closer to the `Root` node (Line 2). Based on the received decisions it performs an initial parent selection using the *ProcessDecisions* subroutine described in Algorithm 2 (Lines 3–4). It then calculates some necessary statistics and transmits an invitation to neighboring nodes (Lines 5–6). The node then waits (Line 7) to receive invitations from neighboring nodes and makes a final decision on its parent selection using the *ProcessInvitations* subroutine presented in Algorithm 3 (Lines 8–9). The node then transmits its final decision (Line 10) to neighboring nodes and ignores any received decisions or invitations until the next update period when the collection tree will be reorganized (a counter denoting the reorganization period can be attached to the queries transmitted by the `Root` node in order to help the nodes understand the transition to a new update period). An interesting observation that we have not mentioned so far involves the nodes with zero epoch participation frequencies. For these nodes, the computed attachment costs to any neighboring node will also be zero. In such cases we select the candidate parent which produces the lowest value for the attachment cost if we ignore the node's epoch participation frequency (i.e., minimizes the $E_{tr_{i,j}} + CF_j + HCF_j - DCF_j$ value when referring to the minimization problem of Section 5.2.3). This decision is expected to minimize the attachment cost, if the sensor at some point starts observing events.

[3] In the case of energy minimization for holistic or non-aggregate functions, described in Section 5.2.3, it suffices to substitute CF_j with $CF_j + HCF_j - DCF_j$ in this proof.

Algorithm 1. `BuildCollectionTree()` Subroutine

1: {S_i is the node being examined}
2: Wait to receive decisions by neighboring nodes
3: Set $\overrightarrow{Dec}$ as the received decisions by the nodes with minimum D values (ignore other decisions).
4: k = ProcessDecisions ($\overrightarrow{Dec}$) {Returns index of selected parent}
5: $D_i = 1 + D_k$
6: Transmit invitation to neighboring nodes
7: Wait to receive invitations by neighboring nodes
8: Set $\overrightarrow{Inv}$ as the received invitations by the nodes with D values equal to D_i (ignore other invitations).
9: m=ProcessInvitations ($\overrightarrow{Inv}$) {Returns index of selected parent}
10: Transmit decision
11: Ignore received decisions and invitations until next reorganization

Algorithm 2. `ProcessDecisions` ($\overrightarrow{Dec}$) Subroutine

1: {S_i is the node being examined}
2: Select Dec_k as the decision with the minimum attachment cost. If $P_i = 0$ utilize in the calculations a non-zero value at this step to prevent all nodes from having the same (zero) attachment cost
3: Let S_k be the sender of Dec_k
4: Set $parent(S_i) = S_k$
5: Calculate statistics (cost factors) for current node based on current parent selection
6: Return k {Index of selected parent node}

Algorithm 3. `ProcessInvitations` ($\overrightarrow{Inv}$, k) Subroutine

1: {S_i is the node being examined}
2: {S_k is the current parent node}
3: In the following discussion, all estimations of the attachment cost utilize the same E_{RF_i} value, as discussed at the end of Section 5.2.3
4: Select Inv_m as the invitation with the minimum attachment cost. If $P_i = 0$ utilize in the calculations a non-zero value at this step to prevent all nodes from having the same (zero) attachment cost
5: Let S_m be the sender of Inv_m
6: **if** $AC_{i,m} \leq AC_{i,k}$ **then**
7: Return k {No benefit in changing parent node}
8: **end if**
9: Calculate $AC_{m,i}$ using information from Inv_m

10: **if** $AC_{i,m} > AC_{m,i}$ **then**
11: Return k {Reverse decision is better}
12: **else if** $AC_{i,m} == AC_{m,i}$ AND $i > m$ **then**
13: Return k {Base decision on identifier}
14: **end if**
15: Set $parent(S_i)=S_m$
16: Calculate statistics (cost factors) for current node based on current parent selection
17: Return m {Index of selected parent node}

6. Discussion and extensions to the basic algorithm

In this section we first provide alternative techniques that we considered for our optimization problem, as well as their drawbacks when compared to our approach. We also elaborate on some decisions made in our algorithms and discuss some extensions to our basic algorithms.

6.1. Alternative techniques considered

In our work we investigated several alternative ways of building collection tree, while utilizing statistics collected by the sensor nodes. The two most promising approaches were to select the parent node of each sensor based on either its epoch participation frequency or the actual transmission frequency of the node.

The first approach involved selecting as a parent node the neighboring node with the highest epoch participation frequency. This seemed as the most intuitive approach at the beginning. The problem with this approach is that we have no knowledge about the epoch participation frequencies of the sensor nodes on the path of the candidate parent towards the `Root` node. The locally best option may end up being a poor decision overall. Moreover, this algorithm could not perform good decisions even in the case of very simple scenarios. For example, consider the scenario of a sensor network with simply three sensors (besides the `Root` node) depicted in Fig. 3. Let the minimum possible distance of each node from the `Root` node be: $D_1=1$, $D_2=D_3=2$. An algorithm where the sensor nodes would select as parent node the neighboring node with the maximum P value would result in node S_2 selecting node S_3 as its parent node. Assuming the case of a distributive aggregate query (like the SUM of all measurements), we first observe that this decision would result in an increase on the expected number of transmissions by: $P_2 + P_2(1-P_3) + P_2(1-P_3)(1-P_1)$. On the other hand, if S_2 had selected S_1 as its parent node (as in Fig. 3(b)), then the corresponding expected number of transmissions due to this decision would have been: $P_2 + P_2(1-P_1)$. One can easily verify that for a large range of values (but not for all) for P_1, P_2, P_3 the second decision would have been more beneficial. For example, for $P_1=0.1$, $P_2=0.1$ and $P_3=0.2$, this algorithm would select a solution with expected cost per epoch equal to $0.1+0.1(1-0.2)+0.1(1-0.2)(1-0.1)=0.252$, while the second solution has an expected cost of only: $0.1+0.1(1-0.1)=0.19$.

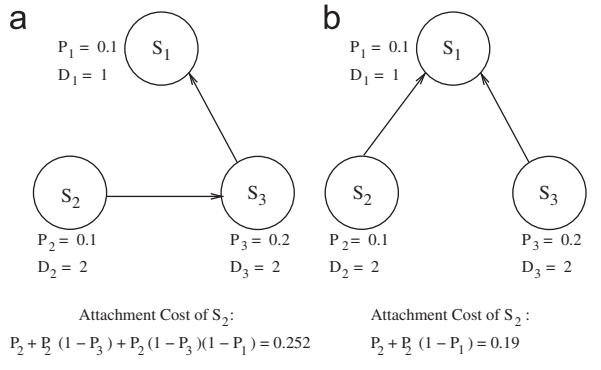

Fig. 3. Parent selection using the neighbor with the maximum P value leads to wrong decision in (a). In (b) the parent selection using the attachment cost leads to the best decision.

The second alternative approach was based on the idea that if a node performs a large number of transmissions, then selecting it as a parent node would result in only a small number of additional transmissions from nodes in its path to the `Root` node. This approach has multiple drawbacks. First of all, it makes no distinction on whether the transmissions by a node are due to corresponding transmissions by its descendants in the collection tree, or because of a high epoch participation frequency by the node itself. Our experimental evaluation of this technique indicated that in the case of moving phenomena (or moving sensor nodes) even in cases of large changes in the epoch participation of some sensor nodes the collection tree was not altered significantly (except perhaps near its leaves), as the sensor nodes seek to forward their results through nodes (and paths) that exhibited large transmission frequencies in previous epochs. This resulted in a large number of nodes with low epoch participation frequencies to transmit a large number of messages. In general, this second alternative approach performed significantly worse in our experimental evaluation that both our proposed techniques based on the attachment cost of a node and compared to the first alternative technique, discussed earlier in this section, that utilized solely the epoch participation frequencies of each sensor node.

A final alternative approach that we considered was to build the collection tree bottom-up, starting from the nodes that are most distant from the `Root`. Using such an approach, a node that performs a parent selection actually has no real idea of the incurred attachment cost, and thus whether the node is being directed towards a "desired" area of the collection tree, since the path from the node's parent to the `Root` has not been formed. The node is only aware of its transmission frequency and of the epoch participation frequency of its candidate parent. Our experience with such a bottom-up construction of the collection tree was that it consistently provided significantly worse results than our presented techniques and is, thus, omitted from our discussion.

6.2. Power of independence assumption

In order to calculate the attachment cost when selecting a parent node, in Section 5.2 we made the

simplification to consider that the epoch participation by each node is independently derived. Clearly, there are many practical scenarios where this assumption may not hold, for instance when nearby nodes observe the same event. From an algorithmic standpoint, removing this assumption does not allow us to compute and use local statistics in a recursive manner. One might be tempted to assume that the independence could have been raised by requiring each node to transmit, along with its locally calculated statistics, a bitmap specifying at which epoch the node participates in the query. Each node receiving this bitmap could make, when considering candidate parents, a better estimation of the increase in the epoch participation frequency that each parent selection would incur. However, note that this approach: (i) requires more information to be transmitted by each node. Since the extra information corresponds to a long bitmap, this information may require multiple messages for its transmission; (ii) this solution still has the pitfall that the attachment cost is overestimated, since it assumes that any increase in transmissions to the parent node is due to the node at question, ignoring the contribution by sibling subtrees; and (iii) this procedure can only utilize one-hop dependencies in epoch participation—multi-hop dependencies require the bitmaps of all nodes in the path to the root node, in order for the cost estimation to be more accurate. Clearly, this is a non-realistic scenario.

If no local statistics can be computed, then the other alternative solution to our problem is a centralized computation, where the root node would have to know which node transmitted in which epoch (thus, requiring a bitmap by each node in the network), something that would severely affect the bandwidth consumption in the network.

In conclusion, using the independence assumption seemed as the only viable solution in order to help us reach our decisions, while utilizing an in-network construction of the collection tree. Our experimental evaluation with data sets that exhibit significant dependency between the epoch participation frequencies of neighboring nodes reveals that our techniques still manage to produce significantly better collection trees (with respect to the number of transmitted messages and the overall energy consumption) when compared to existing techniques.

6.3. Refining the utilized statistics

Our presented techniques for constructing the collection tree utilize a two-step approach, where each sensor node makes a preliminary parent selection, addresses an invitation to neighboring nodes with equal minimum distance from the `Root` node, awaits for the corresponding invitation of the neighboring nodes and then finalizes its parent decision and broadcasts it. A potential problem with this approach is that multiple neighboring nodes may decide to alter their parent selection at the same step (i.e., when reaching their final decision) and, thus, create a long routing path through nodes having the same minimum distance D from the `Root` node. However, note that when these nodes reached their decision, they were not aware that something like this might happen. The statistics that they utilized in order to reach their decision involved statistics included in the invitations of neighboring nodes. Unfortunately, statistics were calculated based on the preliminary parent selection of each node. We can improve our algorithm by refining the statistics that each node uses when making its final decision as follows.

After each node receives the invitations of neighboring nodes, it does not make its decision immediately. Instead, it selects a subset of these nodes termed the *candidate set*. This set includes the set of neighboring nodes that seem as better candidate parents (using the attachment cost of each selection, as described so far) than the preliminary choice performed. The node then waits until it receives the final decision from these nodes, or until a timeout occurs. This timeout can be specified at the query setup phase. Having received the final decisions from the nodes in its candidate set (with their updated statistics included in the message), the node then makes its final decision. This modification of the algorithm is slightly more complicated, as it involves a timeout step.

6.4. Minimizing other metrics

Our techniques can be easily adapted to incorporate different minimization metrics, than the ones presented in Section 4.2. For example, the formulas for minimizing the number of transmitted bits can be derived using the formulas for the energy minimization for the corresponding type of query (i.e., distributive, non-aggregate). In these formulas one simply has to substitute the term $E_{tr_{i,j}}$ with the size of a packet (including the packet's header) and to substitute the term $DE_{tr_{i,j}}$ with the size of each transmitted aggregate value (thus, ignoring the header size). In the case where the goal is to maximize the minimum energy amongst the sensor nodes, the attachment cost can be derived from the minimum energy, amongst the nodes in a sensor's path to the `Root` node, raised to -1 (since our algorithms select the candidate parent with the *minimum* attachment cost).

6.5. Reducing the load of sensors

Our presented algorithms sometimes lead to cases where some sensor nodes with cost factors significantly lower than those of their neighboring nodes will potentially have a large number of children nodes in the resulting collection tree. In situations where the epoch participation frequency does not vary significantly over time, this scenario will result in a quick drain of the sensors' energy. One way to solve this problem is by having each node select its parent in a *probabilistic* manner. We can thus associate, for each sensor S_i, with each candidate parent node S_j (amongst the candidate parents that satisfy the conditions presented in Section 5.1) of S_i a cost value $CV_{i,j}$, where $CV_{i,j} = \min\{s, 1/AC_{i,j}\}$. In this equation the value of CV_i is capped by a sanity bound s (i.e., a maximum value). The sanity bound is used to ensure that nodes with zero attachment cost will not

exhibit an infinite value of $CV_{i,j}$. We then associate a selection probability $SP_{i,j}$ to each candidate parent S_j as

$$SP_{i,j} = \frac{CV_{i,j}}{\sum_j CV_{i,j}} \quad (6)$$

Using this probabilistic process for selecting a parent node will result in a distribution of the load of individual nodes that would be otherwise overburden by a deterministic selection.

One may be tempted to further reduce the load of individual nodes by taking into account, for example, the amount of available energy at each node. One way to accommodate this in our calculations, would have been to divide the cost factors by the parent node's remaining energy level.[4] While this approach does not minimize a specific cost metric, it allows us to prefer parent nodes with low attachment costs and higher levels of residual energy. However, we believe that such an extension would be inappropriate and contradicts a foundational principle in building viable sensor networks: an application should not be concerned at serving the needs of individual nodes. What matters in the network as a whole. For example, cluster formations regularly place more load at specific nodes (the cluster leaders) at an attempt to reduce the cumulative drain of network resources. As a result, certain nodes "take the hit" for the good of their local community. Our focus on reducing global metrics (like the total number of messages or the overall energy consumption) stems from the same principle. In our prior work [20] we have demonstrated that network lifetime is increased when emphasis is put on minimizing global metrics, at the expense of the operation of certain nodes in a network. Our proposed techniques are built around the same premise.

Of course, if certain nodes are deemed to be critical for the application at hand, or when certain sensors face severe energy limitations, we can easily modify our algorithm so as not to consider these nodes when selecting a parent node.

6.6. Taking into account the quality of links

A question that naturally arises is whether our presented algorithms can take into account the existence of message failures, which is common in sensor networks. When such failures occur, reliable communication can be achieved through a protocol that includes a mechanism for acknowledgments and retransmissions. Let $fp_{i,j}$ denote the probability that a message transmitted by S_i will not be received by S_j. Using a protocol that performs retransmissions, the expected number of transmissions for the successful reception of the aforementioned message is $1/(1-fp_{i,j})$. Thus, our algorithms can easily be adapted as follows (the presentation here focuses on the *CF* factor, but the adaptation to the other cost factors is straightforward): (i) any CF_i value previously computed as $(1-P_i)\times(\alpha+CF_j)$ (where α may represent in the equations of this paper the values of 1 or $E_{tr_{i,j}}$) will now be computed as $(1-P_i)\times(\alpha/(1-fp_{i,j})+CF_j)$; and (ii) any attachment cost previously computed as $P_i\times(\alpha+CF_j)$ will now be computed as $P_i\times(\alpha/(1-fp_{i,j})+CF_j)$.

[4] Note that this approach is different than the case of Section 6.4, where only the minimum energy of a node in the entire path to the `Root` is used to derive the cost factors.

7. Multi-query optimization

In the multi-query scenario, each node S_i may choose different parent nodes for each posed query. Thus, the resulting network topology may not be a tree after-all but a directed acyclic graph. In the case of multiple concurrent queries we need to introduce some additional (or augmented) notation for the presentation of our algorithms. Let P_i^k denote the epoch participation frequency of S_i regarding the k-th query. Let $f_i(k)$ denote the index of the selected parent node of S_i for the k-th posed query.

In order to be able to derive recursive formulas for the estimation of the attachment cost, in our approach we break the posed queries into two groups. The first group of queries contains the distributive and algebraic aggregate queries, while the second group contains the holistic and non-aggregate queries. In our discussion below we assume that the group of queries handled in each case contains a total of M queries of similar type.

7.1. Minimizing total number of transmissions

If we process the queries in order based on their identifier (i.e., from 1 to M) and consider the attachment cost $AC_{i,j}^k$ of S_i to S_j regarding the k-th query, we need to estimate how many new messages will be transmitted by S_j and, recursively, by any node in the path from S_j to the `Root` node. Note that because we may have selected S_j as the parent node for other queries as well, the transmission of results for the k-th query does not always generate new messages in a link. Therefore, we consider the additional cost imposed on the link between S_i and S_j only for query k as: $P_i^k \times \prod_{\substack{x<k \\ f_i(x)=S_j}} (1-P_i^x)$.

Before the attachment that we are considering for the k-th query, the joint epoch participation frequency of S_j was: $JP_j = 1-\prod_{x=1}^{M}(1-P_j^x)$. After the attachment, aggregates regarding the k-th query will be transmitted by S_j with a frequency $P_j'k$, which is increased over P_j^k by $P_i^k(1-P_j^k)$. Thus, the resulting increase in the overall (i.e., for all queries) transmission frequency of S_j becomes

$$\left[1-(1-P_j'^k)\times\prod_{\substack{x=1\\x\neq k}}^{M}(1-P_j^x)\right]-\left(1-\prod_{x=1}^{M}(1-P_j^x)\right)$$
$$=\prod_{\substack{x=1\\x\neq k}}^{M}(1-P_j^x)\times(P_j'^k-P_j^k)=P_i^k\times\prod_{x=1}^{M}(1-P_j^x) \quad (7)$$

Using the notation $PROD_i^k = \prod_{\substack{x=1\\f_i(x)=f_i(k)}}^{M}(1-P_i^x)$ and $partialPROD_{i,j}^k = \prod_{\substack{x<k\\f_i(x)=S_j}}(1-P_i^x)$, one can follow similar arguments as in Section 5.2.1 and calculate the attachment cost $AC_{i,j}^k$ as

$$AC_{i,j}^k = P_i^k\times(JCF_j^k + partialPROD_{i,j}^k)$$

$$JCF_i^k = PROD_i^k + (1-P_i^k)\times JCF_j^k \quad (8)$$

7.2. Minimizing total energy consumption

7.2.1. Distributive and algebraic aggregates

In the multi-query optimization scenario, when estimating the attachment cost $AC^k_{i,j}$ of having S_i select S_j as its parent node for the k-th query, one needs to consider that both new messages and/or new aggregate values in existing messages (transmitted due to some other query) may be generated. In particular:

- New messages (first summand of $AC^k_{i,j}$ presented below) may be transmitted from S_i to S_j if the queries with index lower than k do not generate a message by themselves (recall that the queries are processed in order in our algorithm). The frequency of these new messages is: $P^k_i \times partialPROD^k_{i,j}$.
- In the same link, new aggregate values (only second summand of $AC^k_{i,j}$ presented below) but in messages that would have been transmitted anyway due to other queries are included with frequency: $P^k_i \times (1-partialPROD^k_{i,j})$.
- In the path of S_j to the root node, new messages (third summand of $AC^k_{i,j}$ presented below) and aggregate values (fourth summand of $AC^k_{i,j}$ presented below) in existing messages are generated. The number of total aggregate values for the k-th query can be calculated as in the single query case described in Section 5.2.1. Out of these values, a fraction of them (the number is derived in Section 7.1) involves new messages not containing aggregates about other measures. In the remaining cases the aggregates are attached to messages that would have been transmitted anyway.

Using similar arguments as in Sections 5.2.3 and 7.1 (due to the generation of both new messages and additional aggregate values in existing messages the formulas below resemble strongly the ones in Section 5.2.3), the attachment cost $AC^k_{i,j}$ can be estimated as

$$\begin{aligned} AC^k_{i,j} &= P^k_i \times partialPROD^k_{i,j} \times E_{tr_{i,j}} \\ &\quad + P^k_i \times (1-partialPROD^k_{i,j}) \times DE_{tr_{i,j}} \\ &\quad + P^k_i \times JCF^k_j + P^k_i \times (JECF^k_j - JDCF^k_j) \end{aligned}$$

$$JCF^k_i = PROD^k_i \times E_{tr_{i,f_i(k)}} + (1-P^k_i) \times JCF^k_{f_i(k)}$$

$$JECF^k_i = (1-P^k_i) \times (DE_{tr_{i,f_i(k)}} + JECF^k_{f_i(k)})$$

$$JDCF^k_i = PROD^k_i \times DE_{tr_{i,f_i(k)}} + (1-P^k_i) \times JDCF^k_{f_i(k)} \quad (9)$$

Note that in the single query scenario, the second and fourth summands of $AC^k_{i,j}$ always evaluate to zero, as $partialPROD^1_{i,j} = 0$ and $JDCF^k_i$ is always equal to $JECF^k_i$, since $PROD^k_i$ expands to $(1-P^k_i)$.

7.2.2. Holistic aggregate and non-aggregate queries

The attachment cost $AC^k_{i,j}$ can in this case be derived based on the analysis presented in Sections 5.2.3 and 7.2.1. The main difference compared to the case of algebraic or non-aggregate queries involves the estimation of the energy consumption due to additional aggregates that are transmitted in existing messages. The complete formulas become (note the similarity with the single query scenario):

$$\begin{aligned} AC^k_{i,j} &= P^k_i \times partialPROD^k_{i,j} \times E_{tr_{i,j}} \\ &\quad + P^k_i \times (1-partialPROD^k_{i,j}) \times DE_{tr_{i,j}} \\ &\quad + P^k_i \times JCF^k_j + P^k_i \times (JHCF^k_j - JDCF^k_j) \end{aligned}$$

$$JCF^k_i = PROD^k_i \times E_{tr_{i,f_i(k)}} + (1-P^k_i) \times JCF^k_{f_i(k)}$$

$$JHCF^k_i = DE_{tr_{i,f(k)}} + JHCF^k_{f_i(k)}$$

$$JDCF^k_i = PROD^k_i \times DE_{tr_{i,f_i(k)}} + (1-P^k_i) \times JDCF^k_{f_i(k)} \quad (10)$$

8. Experiments

We developed a simulator for testing the algorithms proposed in this paper under various conditions. In our discussion we term our algorithm for minimizing the number of transmissions as MinMesg, and our algorithm for minimizing the overall energy consumption as MinEnergy. Our techniques are compared against two intuitive algorithms. In the MinHops algorithm, each sensor node that receives the query announcement randomly selects as its parent node a sensor amongst those with the minimum distance, in number of hops, from the `Root` node [23]. In the MinCost algorithm, each sensor seeks to minimize the sum of the squared distances amongst the sensors in its path to the `Root` node, when selecting its parent node. Since the energy consumed by the power amplifier in many radio models depends on the square of the communication range, the MinCost algorithm aims at selecting paths with low communication cost.

In all sets of experiments we place the sensor nodes on random locations over a rectangular area. The radio parameters were set accordingly to the values in Table 3. The message header was set to 32 bits, similarly to the size of each statistic and half the size of each aggregate value.

Our algorithms require for their operation an estimate of the epoch participation frequencies by each node. This is achieved by maintaining a single counter at each node that is increased whenever a transmission is made. We used a periodic schedule that reorganizes the collection tree every 200 epochs. Thus, every 200 epochs the MinMegs or the MinEnergy algorithm was invoked utilizing the epoch participation frequencies computed in the previous 200 epochs by the nodes. Of course, the MinHops and the MinCost algorithms are only executed once, since they never need to reorganize the collection tree. In all figures we account for the overhead of transmitting statistics and invitation messages during the creation of the collection tree in our algorithms. All numbers presented are averages from a set of five independent experiments with different random seeds.

8.1. Experiments with synthetic data sets

We initially placed 36 sensor nodes in a 300×300 area, and then scaled up to the point of having 900 sensors. We set the maximum broadcast range of each sensor to 90 m. In all cases the `Root` node was placed on the lower left part of the sensor field. We set the epoch participation frequency of the sensor nodes with the maximum distance, in hop count, from the `Root` to 1. Unless specified otherwise, with probability 8% some interior node assumed an epoch participation frequency of 1, while the epoch participation frequency of the remaining interior nodes was set to 5%.

We considered two setups for selecting which interior nodes assume an epoch participation frequency of 1. In the *INDEPENDENT* setting, these nodes are selected randomly. In the *DEPENDENT* setting, the selected interior nodes that have an epoch participation frequency of 1 are spatially correlated. In particular: (1) an initial proper (this will be defined shortly) set of interior nodes that have an epoch participation frequency of 1 is selected randomly. (2) Then, for each interior node X in the aforementioned set, with a probability $P_{neighbor}$ (with probability $1-P_{neighbor}$ no node is selected) we decide to include in the set a random neighbor (which did not already have a high epoch participation frequency) of X. (3) If a neighbor was selected in the previous step, we follow the same procedure recursively in that node as well (flip a coin, select another neighbor to activate, etc.).

In the *DEPENDENT* setting, the process results in the formation of clusters of interior nodes with an epoch participation frequency of 1. The number of nodes in each cluster depends on the probability $P_{neighbor}$. High values of this probability results in our settings in fewer, but larger clusters. Low values of this probability result in more, smaller clusters. The expected number of nodes in each cluster approaches $1/(1-P_{neighbor})$. Using this estimate, we may properly determine the number of interior nodes that we initially (before starting to select neighbors of these nodes, etc.) select to have an epoch participation frequency of 1. The *INDEPENDENT* setting can be viewed as a special case of the *DEPENDENT* setting, where $P_{neighbor} = 0$.

8.1.1. Experiments with the INDEPENDENT setting

We first evaluated for the INDEPENDENT setting a non-aggregate "SELECT *" query over the measurements obtained by the epoch participating sensor nodes. Since in this query the measurements of each epoch participating node need to be propagated all the way to the `Root` node, and the only sharing that can be achieved in combined messages involves the message's header, we expect little energy savings in this case. We also evaluated for the same setting a SUM aggregate query over the values of epoch participating sensor nodes using all algorithms. We present the total number of transmissions for each type of query, algorithm and number of sensors in Fig. 4. Note that the MinEnergy algorithm built very different collection trees for the two types of queries. For the remaining algorithms, the number of transmitted messages was the same for both types of queries. The corresponding average energy consumption by the sensor nodes for each case is presented in Tables 6 and 7.

As we can see, our MinMesg algorithm achieves a significant reduction in the number of transmitted messages compared to the MinHops and MinCost algorithms. The increase in messages induced by the

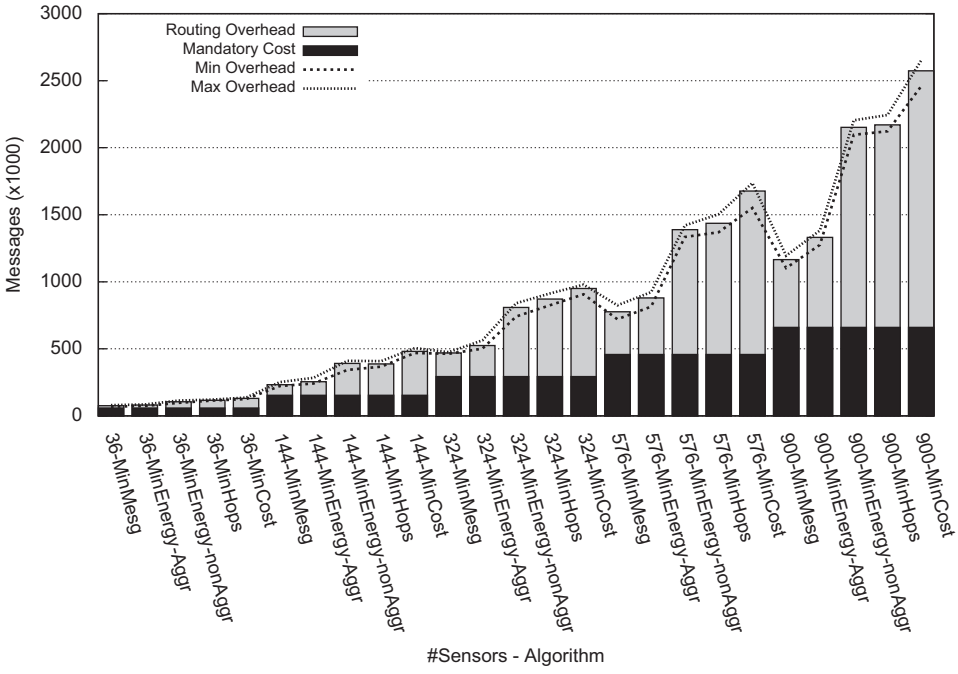

Fig. 4. Messages and average message overhead for INDEPENDENT synthetic data set. Results for MinEnergy presented for both aggregate SUM and non-aggregate query.

MinHops and MinCost algorithms compared to our approach is up to 86% and 120%, respectively, with an average increase of 66% and 93%, respectively.

However, since these gains depend on the number of transmissions that epoch-participating nodes perform, it is perhaps more interesting to measure the *routing overhead* of each technique. We define the routing overhead of each algorithm as the relative increase in the number of transmissions when compared to the number of epoch participations by the sensor nodes. Note that the latter number is a *mandatory* cost that represents the transmissions in the network if each sensor could communicate directly with the `Root` node. For example, if the total number of epoch participations by the sensor nodes was 1000, but the overall number of transmissions was 1700, then the routing overhead would have been equal to $(1700-1000)/1000=70\%$. As we observe from Fig. 4, our MinMesg algorithm often results in 3 times smaller routing overhead compared to the alternative algorithms considered. We also observe that the MinEnergy algorithm in the aggregate case produced results very close to the ones of MinMesg. A main difference between these two algorithms is that amongst candidate parents with similar cost factors, the MinEnergy algorithm is less likely to select a distant neighbor than the MinMesg algorithm, which only considers epoch participation frequencies. This is a trend that we observed in all our experiments. However, in the case of non-aggregate queries, the MinEnergy algorithm formed very different collection trees, as it avoided routing measurements through very long paths.

Table 6
Average power consumption (in mJ) for INDEPENDENT synthetic data set with error bounds for aggregate SUM query.

Sensors	Aggregate SUM query			
	MinMesg	MinEnergy	MinHops	MinCost
36	94.38	87.20	166.12	125.78
144	72.84	67.56	140.81	117.18
324	66.06	61.83	141.46	103.22
576	62.52	58.73	133.71	101.84
900	61.29	56.68	127.79	99.93
±	7.51%	10.94%	5.66%	6.8%

Table 7
Average power consumption (in mJ) for INDEPENDENT synthetic data set with error bounds for non-aggregate "SELECT *" query.

Sensors	Non-aggregate "SELECT *" query			
	MinMesg	MinEnergy	MinHops	MinCost
36	492.48	283.20	323.51	286.51
144	585.04	357.27	414.65	358.02
324	695.86	428.49	494.26	428.12
576	781.93	492.43	578.37	493.21
900	869.54	554.36	645.14	555.97
±	16.09%	7.62%	7.73%	8.54%

The MinEnergy algorithm performs very well in both types of queries. Compared to the MinHops algorithm, it achieves up to a twofold reduction in the power drain for aggregate queries and up to 17% for non-aggregate queries. Compared to the MinCost algorithm the energy savings are smaller but still significant (i.e., up to 76% in the aggregate query). The MinMesg algorithm is obviously a very poor choice, with respect to the energy consumption, for non-aggregate queries.

We expect that the more the epoch participation frequencies of sensor nodes increase, the less likely that out techniques will be able to provide substantial savings compared to the MinHops and MinCost algorithms. In Fig. 5 we repeat the aggregate query of Fig. 4 at the sensor network with 324 nodes, but vary the epoch participation frequency P_i of those nodes that do not make a transmission at each epoch (i.e., of those nodes with $P_i < 1$). While Fig. 5 validates our intuition, it also demonstrates that significant savings can be achieved even when sensor nodes have large P_i values (i.e., $P_i \geq 0.5$).

A novel feature of our technique is the 2-step parent selection phase. In Table 8 we compare the performance of our MinEnergy algorithm in the aggregate SUM query described above versus a variant that was not allowed to select a node's sibling as its parent node in the collection tree. As we can see, the benefits from utilizing the 2-step process are important in all aspects (transmitted

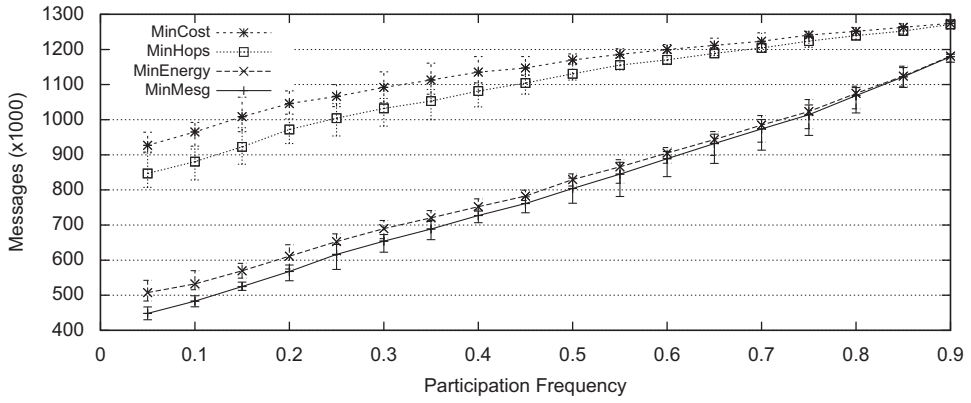

Fig. 5. Transmissions varying the epoch participation frequency, INDEPENDENT setting.

messages and power consumption). Note that the 1-step process constructs minimum-hop trees, similarly to MinHops. However, each node selects its parent in an informed manner (utilizing the estimated attachment cost). The improvements that the 1-step process achieves over MinHops can be viewed by comparing Table 8 with the corresponding numbers of MinHops from Fig. 4 and Table 6.

8.1.2. Experiments with the DEPENDENT setting

We now present a similar set of experiments using the DEPENDENT setting. We provide results for two different values of $P_{neighbor}$, a medium value of 0.5 and a high value of 0.9. For brevity we present results for the SUM aggregate query, as it will become obvious that the results are very similar to the INDEPENDENT setting. For the "SELECT *" query, the MinCost algorithm is again competitive in terms of energy consumption, due to the few opportunities (i.e., only the message header can be shared) of merging information by different sensors.

In Fig. 6 we show the total number of transmitted messages for the same setting as in Fig. 4, but for the DEPENDENT setting. One can observe that the results are very similar to the ones of the INDEPENDENT setting, and qualitatively similar in both cases. A small reduction (observe that the maximum value of the y-axis is different in the two subgraphs) in the number of transmitted messages is observed when we use a high value $P_{neighbor}$=0.9 for the competitive MinHops and MinCost techniques. With higher values of $P_{neighbor}$, each internal node with a high epoch participation frequency has more neighbors with the same behavior. Thus, the non-informed parent selection that MinHops and MinCost perform has a higher chance of being a correct choice (i.e., select a parent with a high participation frequency) as the value of $P_{neighbor}$ increases. However, the benefits and the behavior of all techniques is similar to the ones of the INDEPENDENT setting.

The corresponding energy consumption in each case is presented in Table 9. Again, the results are very similar to the INDEPENDENT case.

In Fig. 7 we plot for each scale of the sensor field the ratio of the average tree height of each algorithm to the

Table 8
Comparison of 1-step and 2-step parent selection for MinEnergy algorithm.

# Sensors	Transmissions		Avg. energy consumption	
	1-Step	2-Step	1-Step	2-Step
36	115,963	80,813	153.40	87.20
144	388,171	255,122	124.36	67.57
324	790,751	524,317	115.21	61.83
576	1,222,320	879,019	99.15	58.74
900	1,833,270	1,330,900	93.88	56.68
±	22.44%	22.45%	10.94%	5.43%

Table 9
Energy consumption (in mJ) in DEPENDENT setting.

Scale	MinMesg	MinEnergy	MinHops	MinCost
$P_{neighbor}$=0.5				
1	91.8808	81.305	167.251	120.289
2	68.9294	63.0658	136.705	114.274
3	65.4574	59.9246	137.12	110.17
4	60.9268	55.7408	131.656	100.26
5	58.301	53.9292	125.828	98.3386
$P_{neighbor}$=0.9				
1	90.713	80.885	166.802	118.472
2	67.5194	61.7968	133.186	109.065
3	65.1932	59.6316	135.6	106.086
4	62.8632	58.4016	127.949	97.732
5	58.474	54.6552	121.062	94.7246

Number of transmissions and average power consumption (in mJ) for INDEPENDENT synthetic data set.

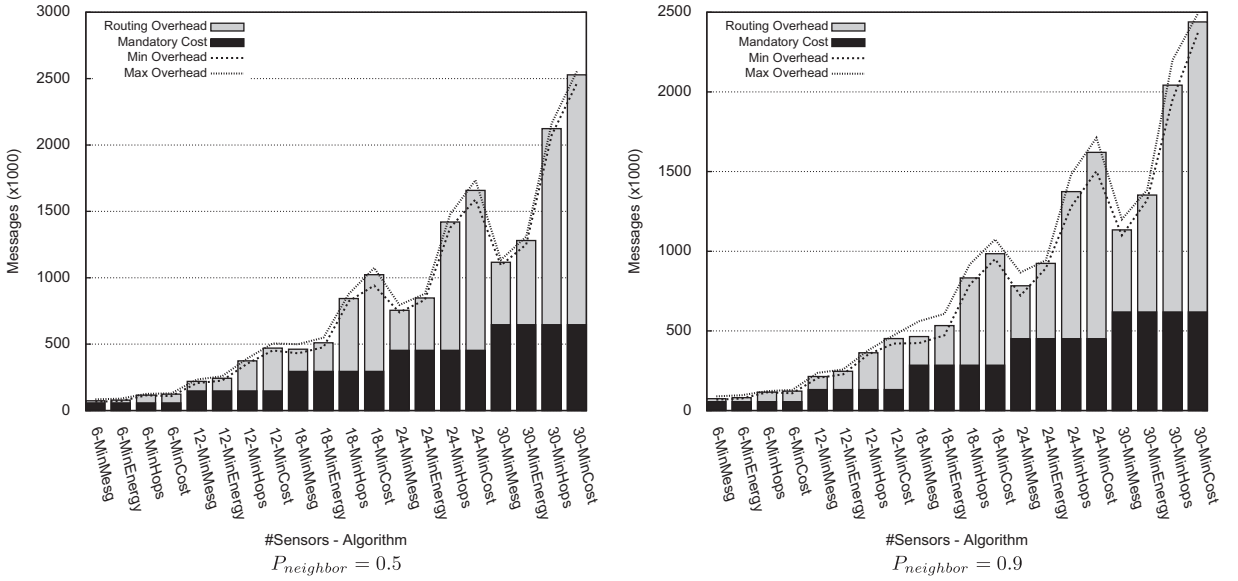

Fig. 6. Messages and average message overhead for SUM query in DEPENDENT synthetic data set.

corresponding tree height of MinHops, for values of $P_{similar}$ of 0.5 and 0.9. Obviously, this ratio can only be larger than 1, since MinHops builds a minimum height tree. For the INDEPENDENT setting the results are very similar to (actually, slightly lower than) the ones of the DEPENDENT setting for $P_{neighbor}$=0.5 and are, thus, omitted for brevity. The trees that MinHops and MinCost build do not depend on the epoch participation frequencies and, thus, do not depend on the value of $P_{similar}$. As we can see from Fig. 7, MinMesg and MinEnergy build trees that are deeper than MinHops (MinMesg actually built more shallow trees in both cases), with this ratio increasing as the value of $P_{similar}$ increases. This is to be expected, since with larger values of $P_{similar}$, it is more likely to create larger paths of nodes belonging to the same level. Of course, the main reason for the larger tree height of MinMesg and MinEnergy is the sensors with large epoch participation frequencies at the external part of the network. Thus, MinMesg and MinEnergy may reduce the number of transmitted messages, or the overall energy consumption, but may increase the length of the constructed collection trees.

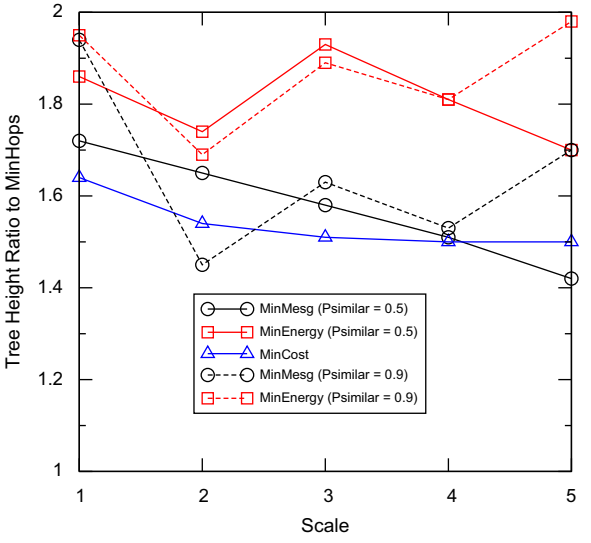

Fig. 7. Ratio of tree height for all methods when compared to MinHops, DEPENDENT setting.

Fig. 8 presents the results when we vary the epoch participation frequency of the interior nodes which were not selected to exhibit an epoch participation frequency of 1, for the DEPENDENT setting. The results are again similar to the corresponding results of the INDEPENDENT setting (Fig. 5).

8.2. Experiments with real data sets

We also experimented with the following two real data sets. The *Trucks* data set contains trajectories of 276 moving trucks [1]. Similarly, the *SchoolBuses* data set contains trajectories of 145 moving schoolbuses [1]. For each data set we initially overlaid a sensor network of 150 nodes over the monitored area. We set the broadcast range such that interior sensor nodes could communicate with at least 5 more sensor nodes. Moreover, each sensor could detect objects within a circle centered at the node and with radius equal to 60% of the broadcast range. We then scaled the data set up to a network of 1350 sensors, while keeping the sensing range steady. In Figs. 9 and 10 we depict the total number of transmissions by all algorithms for the Trucks and SchoolBuses data sets, correspondingly, when computing the SUM of the number of detected objects. In our scenario, nodes that do not observe an event make a transmission only if they need to propagate measurements/aggregates by descendant nodes. We present the average energy consumption of the sensor nodes in the same experiment for the SchoolBuses data set in Table 10. As it is evident, our algorithms achieve significant savings in both metrics. For example, the MinCost algorithm, which exhibits lower power consumption than the MinHops algorithm, still drains about 15% more energy than our MinEnergy algorithm. Moreover, both our MinMesg and MinEnergy algorithms significantly reduce the amount of transmitted messages by up to 31% and 53% when compared to the MinHops and MinCost algorithms, respectively.

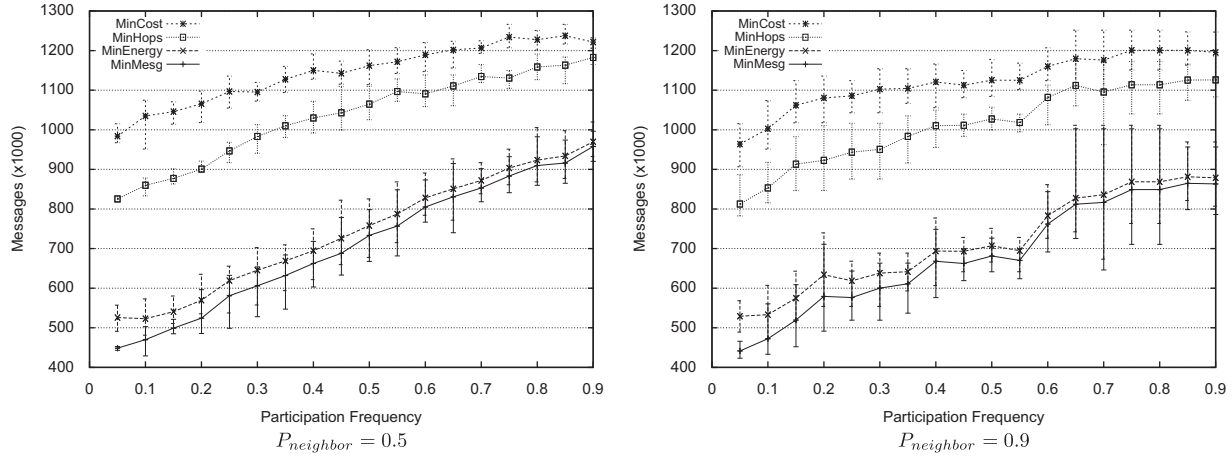

Fig. 8. Transmissions varying the epoch participation frequency, DEPENDENT setting.

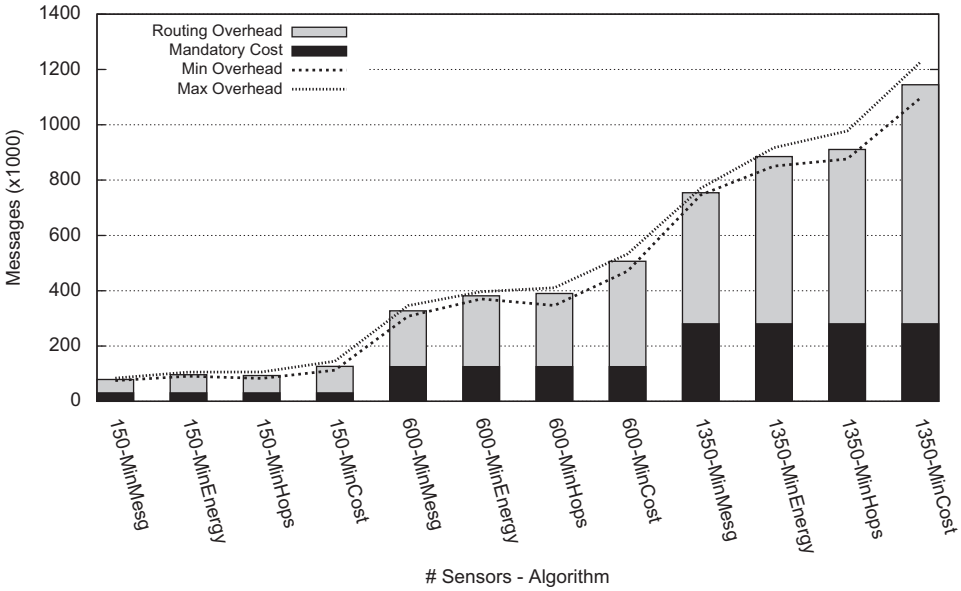

Fig. 9. Transmissions—Trucks data.

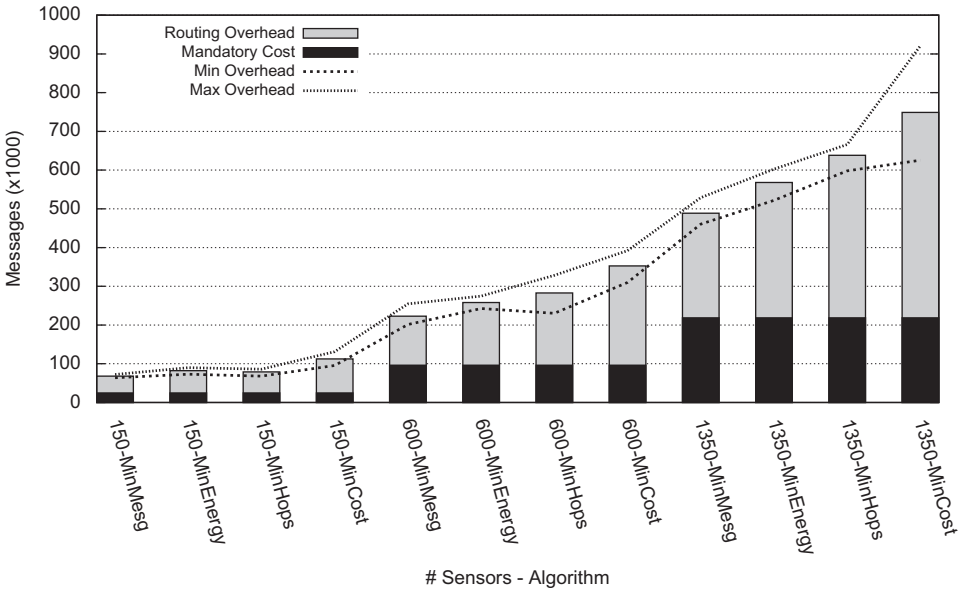

Fig. 10. Transmissions—SchoolBuses data.

Table 10
Average power consumption (in mJ) for schoolbuses data set with error bounds.

# Sensors	MinMesg	MinEnergy	MinHops	MinCost
150	75.65	67.64	94.33	75.51
600	61.19	51.10	84.67	58.74
1350	58.87	47.86	85.89	55.48
±	9.58%	5.5%	15.01%	17.77%

8.3. Multi-query experiments

We then decided to mix the data sets. We separated the SchoolBuses into two categories (by randomly coloring each schoolbus as either color *A* and *B*) and overlaid this data set with the trucks data set. We then performed three simultaneous queries requesting the total number of trucks, schoolbuses of color *A* and schoolbuses of color *B* observed in the network. We used a similar topology,

Table 11
Kilobytes transmitted for multi-query scenario.

# Sensors	MinMesg	MinEnergy	MinHops	MinCost
150	944.59	1075.20	1259.52	1720.32
600	3604.48	3983.36	4945.92	6318.08
1350	7823.36	8611.84	10,977.30	13,250.60

Table 12
Average power consumption (in mJ) for multi-query scenario.

# Sensors	MinMesg	MinEnergy	MinHops	MinCost
150	125.99	113.20	172.78	160.56
600	119.26	99.59	172.33	155.69
1350	115.75	93.55	169.95	146.02

network scale and placement of the `Root` node as above and compared our MinMesg and MinEnergy algorithm with the MinHops and MinCost algorithms, which were modified to select a single parent node for all queries (this produced the best results for them). In Table 12 we depict the energy consumption for all algorithms, with error bounds below 10% in all cases, and find our MinEnergy to yield up to 56% and 81% energy savings when compared to MinHops and MinCost. In addition, our MinMesg algorithm, which does not optimize on energy consumption, drains more power than MinEnergy but still uses less power than MinHops and even MinCost. In the multi-query scenario the size of the transmitted messages varies because not all messages carry the same number of query responses. Therefore, instead of the number of transmitted messages, we consider the total size of all transmissions and list the experimentation results in Table 11 with error bounds under 10%. Although MinHops transmitted marginally fewer messages than the MinMesg algorithm, it is until we look into the size of transmitted messages that it becomes clear that MinHops induces a 27%, or 40%, overhead to our MinEnergy, or MinMesg, respectively.

9. Conclusions

In this paper we presented algorithms for building and maintaining efficient collection trees in support of event monitoring queries in wireless sensor networks. We demonstrated that is it possible to create efficient collection trees that minimize important network resources using a small set of statistics that are communicated in a localized manner during the construction of the tree topology. Furthermore, our techniques utilize a novel 2-step refinement process that significantly increases the quality of the created trees. We have also demonstrated that our algorithms can handle a mix of event monitoring queries (EMQs) including aggregate and non-aggregate queries.

References

[1] Rtree Portal ⟨http://www.rtreeportal.org⟩.
[2] D.J. Abadi, S. Madden, W. Lindenr, REED: robust, efficient filtering and event detection in sensor networks, in: VLDB, 2005.
[3] M. Bawa, H. Garcia-Molina, A. Gionis, R. Motwani, Estimating aggregates on a peer-to-peer network, Technical Report, Stanford, 2003.
[4] A. Cerpa, D. Estrin, ASCENT: adaptive self-configuring sensor network topologies, in: INFOCOM, 2002.
[5] J.-H. Chang, L. Tassiulas, Energy conserving routing in wireless ad-hoc networks, in: INFOCOM, 2000.
[6] J. Considine, F. Li, G. Kollios, J. Byers, Approximate aggregation techniques for sensor databases, in: ICDE, 2004.
[7] A. Deligiannakis, Y. Kotidis, N. Roussopoulosm, Hierarchical in-network data aggregation with quality guarantees, in: EDBT, 2004.
[8] A. Deligiannakis, Y. Kotidis, N. Roussopoulos, Dissemination of compressed historical information in sensor networks, VLDB Journal 16 (4) (2007).
[9] A. Deligiannakis, Y. Kotidis, V. Vassalos, V. Stoumpos, A. Delis, Another outlier bites the dust: computing meaningful aggregates in sensor networks, in: ICDE, 2009.
[10] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, Y. Yao, The cougar project: a work in progress report, SIGMOD Record 32 (4) (2003) 53–59.
[11] A. Deshpande, C. Guestrin, S. Madden, J.M. Hellerstein, W. Hong, Model-driven data acquisition in sensor networks, in: VLDB, 2004.
[12] M. Duckham, S. Nittel, M. Worboys, Monitoring dynamic spatial fields using responsive geosensor networks, in: GIS, 2005.
[13] D. Estrin, R. Govindan, J. Heidermann, S. Kumar, Next century challenges: scalable coordination in sensor networks, in: MobiCOM, 1999.
[14] J. Gao, L.J. Guibas, N. Milosavljevic, J. Hershberger, Sparse data aggregation in sensor networks, in: IPSN, 2007.
[15] N. Giatrakos, Y. Kotidis, A. Deligiannakis, V. Vassalos, Y. Theodoridis, TACO: tunable approximate computation of outliers in wireless sensor networks, in: SIGMOD, 2010.
[16] C. Intanagonwiwat, D. Estrin, R. Govindan, J. Heidermann, Impact of network density on data aggregation in wireless sensor networks, in: ICDCS, 2002.
[17] S.R. Jeffery, G. Alonso, M.J. Franklin, W. Hong, J. Widom, Declarative support for sensor data cleaning, in: Pervasive, 2006, pp. 83–100.
[18] S.R. Jeffery, M.N. Garofalakis, M.J. Franklin, Adaptive cleaning for RFID data streams, in: VLDB, 2006.
[19] D. Kempe, A. Dobra, J. Gehrke, Gossip-based computation of aggregate information, in: FOCS, 2003.
[20] Y. Kotidis, Snapshot queries: towards data-centric sensor networks, in: ICDE, 2005.
[21] Y. Kotidis, Processing proximity queries in sensor networks, in: Proceedings of the 3rd International VLDB Workshop on Data Management for Sensor Networks (DMSN), 2006.
[22] Y. Kotidis, A. Deligiannakis, V. Stoumpos, V. Vassalos, A. Delis, Robust management of outliers in sensor network aggregate queries, in: Proceedings of MobiDE, 2007, pp. 17–24.
[23] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, Tag: a tiny aggregation service for ad hoc sensor networks, in: OSDI Conference, 2002.
[24] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, The design of an acquisitional query processor for sensor networks, in: ACM SIGMOD, 2003.
[25] C. Olston, J. Widom, Offering a precision-performance tradeoff for aggregation queries over replicated data, in: VLDB, 2000.
[26] S. Pattem, B. Krishnamachari, R. Govindan, The impact of spatial correlation on routing with compression in wireless sensor networks, in: IPSN, 2004.
[27] V. Raghunathan, C. Schurgers, S. Park, M. Srivastava, Energy aware wireless microsensor networks, IEEE Signal Processing Magazine 19 (2) (2002).
[28] A. Sharaf, J. Beaver, A. Labrinidis, P. Chrysanthis, Balancing energy efficiency and quality of aggregate data in sensor networks, VLDB Journal (2004).
[29] A. Silberstein, R. Braynard, J. Yang, Constraint chaining: on energy-efficient continuous monitoring in sensor networks, in: SIGMOD, 2006.
[30] S. Singh, M. Woo, C.S. Raghavendra, Power-aware routing in mobile ad hoc networks, in: ACM/IEEE International Conference on Mobile Computing and Networking, 1998.
[31] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, D. Gunopulos, Online outlier detection in sensor data using non-parametric models, in: Proceedings of VLDB, 2006, pp. 187–198.

[32] N. Trigoni, Y. Yao, A.J. Demers, J. Gehrke, R. Rajaraman, Multi-query optimization for sensor networks, in: DCOSS, 2005.
[33] W. Xue, Q. Luo, L. Chen, Y. Liu, Contour map matching for event detection in sensor networks, in: SIGMOD, 2006.
[34] Y. Yao, J. Gehrke, The cougar approach to in-network query processing in sensor networks, SIGMOD Record 31 (3) (2002) 9–18.
[35] W. Ye, J. Heidermann, Medium access control in wireless sensor networks, Technical Report, USC/ISI, 2003.
[36] D. Zeinalipour-Yazti, P. Andreou, P.K. Chrysanthis, G. Samaras, A. Pitsillides, The micropulse framework for adaptive waking windows in sensor networks, in: MDM, 2007, pp. 351–355.