Distrib Parallel Databases (2011) 29: 113–150
DOI 10.1007/s10619-010-7072-5

Power efficiency through tuple ranking in wireless sensor network monitoring

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

Published online: 24 November 2010

Abstract In this paper, we present an innovative framework for efficiently monitoring Wireless Sensor Networks (WSNs). Our framework, coined *KSpot*, utilizes a novel top-k query processing algorithm we developed, in conjunction with the concept of in-network views, in order to minimize the cost of query execution. For ease of exposition, consider a set of sensors acquiring data from their environment at a given time instance. The generated information can conceptually be thought as a horizontally fragmented base relation R. Furthermore, the results to a user-defined query Q, registered at some sink point, can conceptually be thought as a view V. Maintaining consistency between V and R is very expensive in terms of communication and energy. Thus, KSpot focuses on a subset $V'(\subseteq V)$ that unveils only the k highest-ranked answers at the sink, for some user defined parameter k.

To illustrate the efficiency of our framework, we have implemented a real system in nesC, which combines the traditional advantages of declarative acquisition frameworks, like TinyDB, with the ideas presented in this work. Extensive real-world testing and experimentation with traces from UC-Berkeley, the University of Washington and Intel Research Berkeley, show that KSpot provides an up to 66% of energy savings compared to TinyDB, minimizes both the size and number of packets transmitted

Communicated by Erik Buchmann.

P. Andreou · D. Zeinalipour-Yazti (✉) · G. Samaras
Department of Computer Science, University of Cyprus, Nicosia, 1678, Cyprus
e-mail: dzeina@cs.ucy.ac.cy

P. Andreou
e-mail: panic@cs.ucy.ac.cy

G. Samaras
e-mail: cssamara@cs.ucy.ac.cy

P.K. Chrysanthis
Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 5213-4034, USA
e-mail: panos@cs.pitt.edu

over the network (up to 77%), and prolongs the longevity of a WSN deployment to new scales.

Keywords Top-k query processing · In-network aggregation · Sensor networks

1 Introduction

The improvements in hardware design along with the wide availability of economically viable embedded sensor systems make it feasible today to interact and understand the physical world at an extremely high fidelity [30, 39, 52]. The applications of sensor networks range from environmental monitoring (such as atmosphere and habitant monitoring [46, 52]) to seismic and structural monitoring as well as industry manufacturing [17, 39]. Recently, Voltree Power [55] has engineered a bio-energy harvesting technology that allows sensor devices to recharge themselves by collecting the energy that is naturally produced by living trees or other large plants. This alternative, minimizes the cost of replacing batteries frequently, especially in large-scale deployments. Such networks have already been deployed by the United States Department of Agriculture (USDA) at 5 different sites [55].

Although sensor devices in a Voltree Climate Sensor Network can recharge themselves to a certain degree, the ratio between energy required/energy collected greatly depends on the executed monitoring query. Long running high frequency sampling queries can quickly deplete the energy reserves of the sensor device raising the need for energy-conscious algorithms that decrease both processing and communication. The ideas presented in this paper enable efficient and effective monitoring of important events using energy-aware algorithms.

In traditional data acquisition techniques [30, 40, 60], the sensor data is transmitted to the *sink* (also denoted as *base station* or *querying node*) immediately after it is acquired from the physical world. Although in-network aggregation significantly reduces the consumption of energy, the oblivious transmission of all query results from all sensors at every acquisition round is still the most energy demanding factor in such environments [46, 52, 65, 67]. Supplementary approaches to cope with the energy challenge during query processing have in the recent years appeared at numerous venues. These approaches range from efficient join processing algorithms in sensor networks [14, 26, 33, 51], to the underlying data management layer [1, 12, 21, 65] and network optimization [2, 6, 15, 42], among others. Yet, these approaches focus either on a different system model or a different problem formulation, than the work we present in this paper.

In this paper we model the retrieval of data on the presumption that the user is only interested in the k highest-ranked answers rather than all of them. A Top-K query [9, 24, 44, 66] focuses on the subset of most relevant answers for two reasons: (i) to minimize the cost metric that is associated with the retrieval of all answers; and (ii) to improve the quality of the answer set such that the user is not overwhelmed with irrelevant results. This assumption is quite reasonable and has been utilized in numerous other settings (e.g., consider a search engine that returns the 10 highest-ranked results to minimize the consumption of system resources and in order to improve the quality of the answer set).

Our framework, coined *KSpot*, utilizes a state-of-the-art top-k query processing algorithm, coined INT, in conjunction with materialized in-network views, in order to minimize the cost of query executions. A view V in relational databases is a virtual table that contains the results from an arbitrary query Q which is evaluated every time V is referred to. In order to avoid the unnecessary re-execution of Q it is beneficial to store V on secondary storage. This introduces the notion of a *materialized view* (referred to as *view* hereafter). Views have a clear *space* versus *time* trade-off: A *fully* materialized view V requires more space but also less time in evaluating Q, whereas a *partially* materialized view V' requires less space but also more time in evaluating Q. Materialized views can potentially conserve energy as the application can avoid the expensive re-evaluation of the in-network Q.

Materialized views have been studied in numerous seminal papers including [7, 11, 13, 34]. Although a fully materialized view V maintains the complete results of a query Q, the distributed nature of a sensor network environment, along with its distinct characteristics, imposes some fundamental limitations to this model:

i. Firstly, maintaining consistency between V and the underlying and distributed base relation R (defined by the sensor readings) is very expensive in terms of energy. Thus, we focus on maintaining a subset $V'(\subseteq V)$ that unveils only the k highest-ranked answers for some user defined k; and
ii. Secondly, V' is recursively defined using the results that are stored at the lower-levels of the multi-hop routing tree that interconnects the sink with the sensing devices. Thus, traditional view maintenance techniques are not directly applicable.

To illustrate the efficiency of our framework, we have implemented a real system in nesC, which combines the traditional advantages of declarative acquisition frameworks, like TinyDB, with the ideas presented in this work. Extensive real-world testing and experimentation with traces from UC-Berkeley, University of Washington and Intel Research Berkeley, show that KSpot presents an up to 66% of energy savings compared to TinyDB, minimizes both the size and number of packets transmitted onto the network (up to 77%), and prolongs the longevity of a WSN deployment to new scales.

At the foundation of KSpot lies MINT Views, a novel algorithm to minimize messaging and thus energy consumption in the execution of continuous monitoring queries. Like other frameworks, we support single-relation queries with the standard aggregate functions but our focus is to optimize top-k queries over *multi-tuple* answers. Such answers are very typical for queries with a `GROUP-BY` clause and for non-aggregate queries.

To facilitate our description, consider the scenario in Fig. 1, where we illustrate a deployment of 9 sensors in a 4-room building. We are interested in answering Query 1 at the sink (rooted above s_1). In particular we want to find the average temperature of each room every one minute.

Query 1

```
SELECT roomno, AVERAGE(temp)
FROM sensors
GROUP BY roomno
SAMPLE PERIOD 60000
```

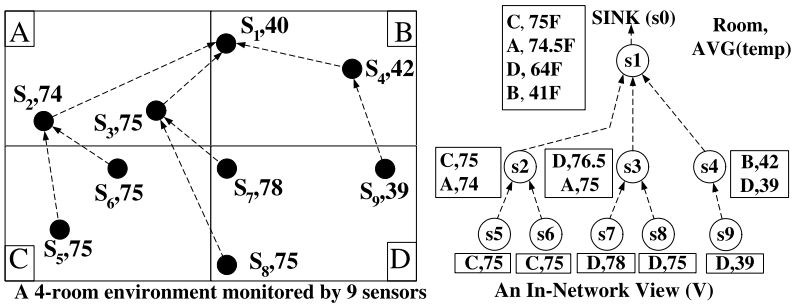

Fig. 1 The *left* figure illustrates a sensor network scenario that consists of 9 sensors $\{s_1, \ldots, s_9\}$ deployed in four rooms $\{A, B, C, D\}$. The label next to each sensor denotes the identifier of the node and the local temperature reading. The figure on the *right* presents a recursively defined In-Network View (V) to Query 1. The label next to each node indicate the local averages for each room

With the TinyDB-based [39, 40] in-network aggregation approach each node forwards tuples of the form (room,sum,count) to its parent every single time instance.[1] One alternative approach is the notion of an *In-Network View (V)* (Fig. 1 on the right). V materializes the result of Q and utilizes these results to speedup the next execution of Q. The performance of V largely relies on the premise of temporal coherence between consecutively acquired sensor readings as local changes will affect the intermediate views until the sink.

To improve the performance penalty of In-Network Views, we propose to prune the local views stored at each node and focus on the k highest-ranked answers rather than all of them. This turns out to be extremely useful because now sensors can discard view updates that do not refer to k highest-ranked answers. On the other hand, this also imposes an extremely challenging problem: "*a naive local greedy pruning strategy may easily discard tuples that will be finally among the k highest-ranked answers*".

To understand this problem, consider again Query 1 but assume that we are only interested in the top-1 result. Such a query should return room $(C, 75F)$. Assuming that each node naively eliminates anything below its local top-1 result will lead us to the erroneous answer $(D, 76.5F)$. In particular, the leaves $\{s_5, s_6, s_7, s_8, s_9\}$ will send their only tuple to their respective parent. The parents $\{s_2, s_3, s_4\}$ will then aggregate the results of their children along with their own result and forward this result to their own parent (i.e., s_1). In particular, s_2 will send $(C, 75F)$, s_3 the tuple $(D, 76.5F)$ and s_4 the tuple $(B, 42F)$. It is now easy to see that if s_1 aggregates the results of its children $\{s_2, s_3, s_4\}$ along with its own result $(B, 40F)$, then this will yield $V_0^{\text{wrong}} = \{(D, 76.5F), (C, 75F), (B, 41F)\}$, where room D is the top-1 answer rather than room C.

Our MINT algorithm utilizes an intelligent upper-bounding algorithm and a local parameter k to construct a subset of V, denoted as the *k-covered bound-set* V', to be materialized. We will show that any tuple outside V' can safely be eliminated during

[1] For clarity in Fig. 1, we only depict the average (i.e., sum/count).

the execution of a query because this tuple cannot be among the k highest-ranked results.

The key idea of the MINT pruning algorithm is to exploit a set of $|\gamma|$ descriptors ($\gamma = \{\gamma_1, \gamma_2, \ldots\}$), in order to bound above the score of tuples that are not known at a given level of the sensor network. The elements in γ are application specific: these can either be known in advance so they can be defined prior to setting up the execution of a query, or these can be learned and dynamically adjusted during query execution (as we will show in Section 4.5). Without loss of generality, in the rest of our discussion we will utilize the following instances: $\gamma_1 =$"*Maximum possible temperature value*" and $\gamma_2 =$"*Number of sensors in each room*". For instance, the temperature sensor on the TelosB Weather Board [52] might only record values between -40F to 250F and the barometric pressure module can only measure pressure in the range 300 mb to 1100 mb.

This paper builds on our previous work in [3, 63], in which: (i) we have presented the preliminary design and simulation results of the MINT [63] algorithm; and (ii) we have demonstrated the preliminary utility of the KSpot framework [3][2]. In this paper, we introduce several new improvements and extensions that are summarized as follows:

- We introduce an elaborate experimental study and solid experimental evidence that shows that the INT/MINT algorithms are indeed offering new levels of energy efficiency in WSN deployments. Our new study is carried out using real instances in TinyOS measuring energy with PowerTOSSIM, while previous studies were carried out on a proprietary simulator. Additionally, we compare our algorithms under different real sensor network traces, querysets and a real micro-benchmark on the CC2420 radio transceiver [53]. In our experiments we focus on a number of parameters including energy consumption and pruning magnitude as well as scalability and network lifetime. To accomplish this we introduce a series of new experiments that focus on the scalability of k, cardinality of GROUP-BY clause and network lifetime.
- We describe in detail how the γ descriptors can be learned during query execution and dynamically adjusted using a sliding window sampling prediction mechanism.
- We present a detailed description of the KSpot system architecture including insight information on all its components and internal procedures. In this description, we list our data structures and explain why such structures are beneficial for top-k query aggregation.
- We provide an extensive overview of related work and a taxonomy of related algorithms based on three different dimensions: *data fragmentation*, *input scores* (exact, approximate) and *output ranking* (exact, approximate). We also qualitatively explain the differences and similarities of these techniques compared to the KSpot framework.

The MINT/INT algorithms, presented at the foundation of this work, make the following overall contributions to the state-of-the-art:

[2]KSpot is currently publicly available under http://www.cs.ucy.ac.cy/~panic/kspot.

– We formulate the problem of constructing a hierarchy of recursively defined top-k views. We solve this problem by introducing MINT Views. We also present a stateless, non-materialized version of MINT, coined *INT* (In-Network Top-k) Views, that is appropriate for sensing device with limited memory.
– We introduce the notion of a *k-covered bound set* V' which only maintains the tuples of V that lead to the k highest ranked answers at the sink. We additionally provide energy-conscious techniques to incrementally and immediately update V'.

It is important to mention that these ideas span well beyond the scope of TinyDB and related technologies, and that these could easily be implemented in other types of systems that deal with a similar system model.

The remainder of the paper is organized as follows. Section 2 presents the architecture of the KSpot framework. Section 3 formalizes our system model and Section 4 presents the underlying algorithms of the KSpot framework. Next, in Section 5 we present our experimental methodology and in Section 6 the results of our evaluation. Finally, Section 7 overviews the related research work and Section 8 concludes our paper.

2 System architecture

KSpot features a two-tier architecture (see Fig. 2), which consists of server-side software written in JAVA and sensor-side software written in nesC.

The first tier (server-side), consists of a server, attached to a fixed sensor node (sink) that is responsible for propagating queries and acquiring results from the sen-

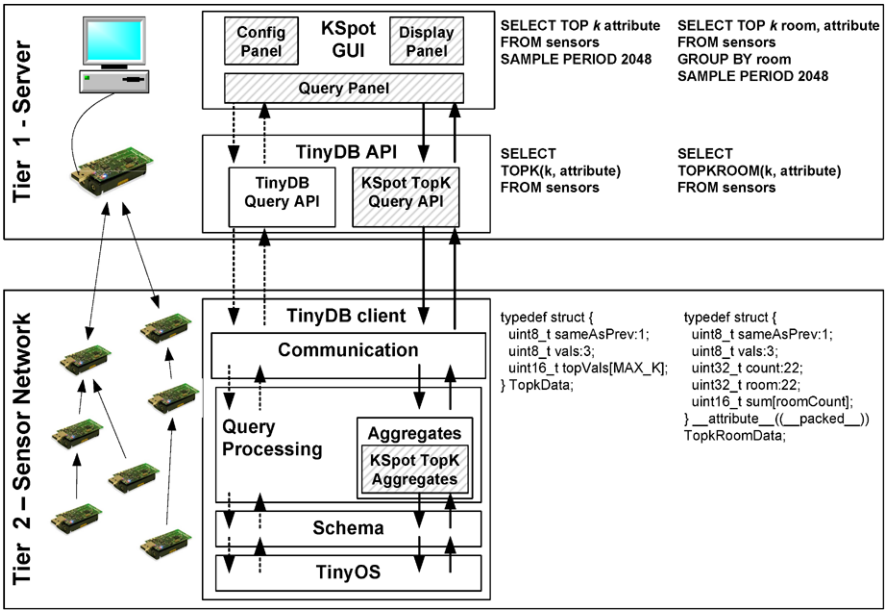

Fig. 2 Main data structures used in our nesC implementation of the KSpot client

sor network deployment. Part of this tier is also the *KSpot Graphical User Interface (KSpot GUI)*, which allows: (i) the declaration of Top-K queries in non-SQL and SQL mode, (ii) the visual representation of sensor network topology, (iii) the visual ranking of results using the KSpot Bullets; and several other administrative functions. Top-k queries generated through the KSpot GUI are translated into an extended-TinyDB query syntax and are then integrated to the KSpot TopK Query API. These queries are then injected into the network through the sink node. The KSpot Top-K Query API is also responsible for translating the raw data arriving through the sink node to the KSpot Display panel.

The second tier (sensor-side), consists of a number of sensor nodes that are positioned in predefined areas of interest. The sensor devices are loaded with the KSpot client software running on the TinyOS [12] operating system. The KSpot client currently extends the TinyDB base implementation by enabling the execution of Top-k queries in the form of aggregates. More specifically, at each epoch a sensor node acquires its local sensor reading and then merges all values acquired from its child nodes. As soon as this phase is completed, the sensor locally prunes a subset of results using the MINT/INT algorithms described later in Section 4. Finally, a node recursively transmits the aggregated result to its parent node until the expected result reaches the sink node.

We have selected TinyOS/TinyDB for the implementation of the KSpot framework for practical reasons as it already provides a kernel of declarative data acquisition functionalities (i.e., SQL query syntax). However, we could have similarly applied our ideas on top of other sensor network operating systems like Contiki [22] or LiteOS [10].

We describe each of the components of the KSpot architecture individually in the following sections.

The KSpot Graphical User Interface (KSpot GUI), is used for: (i) configuring the number of sensors/rooms displayed in the scenario, (ii) execute Top-K queries, and (iii) for displaying the query results in a manner that highlights the ranking properties of the executed query. In particular, the KSpot GUI consists of three panels (see Fig. 3):

i. The *Configuration Panel* (Fig. 3, top-left), which enables the user to load a new scenario from a configuration file or to create a new scenario that can be stored in a configuration file. Through this panel the user can specify which nodes belong to (are *clustered*) in the same physical region. Additionally, the user can assign values to the $|\gamma|$ descriptors mentioned in Section 1. If no specific values are assigned, KSpot assigns the maximum values for each attribute as these were found in the sensorboard manual. Note, that both the cluster configuration and γ descriptors are translated to KSpot commands which are transmitted to the sensor nodes prior the execution of a top-k query.
ii. The *Query Panel* (Fig. 3, bottom-left), which enables the user to specify aggregate (AVG, MIN and MAX) and non-aggregate SQL-like queries either graphically or manually. The constructed query is parsed and translated to the KSpot Query API if the query is a Top-k query or to the TinyDB Query API otherwise.
iii. The *Display Panel* (Fig. 3, right), which allows a user to load a JPG image representation of the scenario map. Subsequently, the user can drag-and-drop the

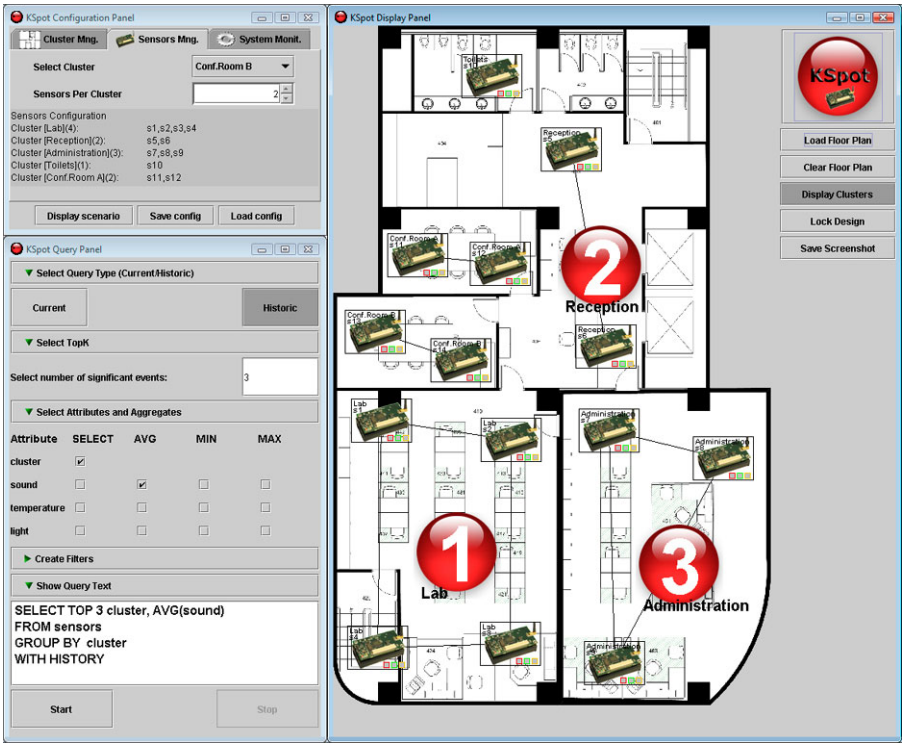

Fig. 3 KSpot's Graphical User Interface (GUI) allows users to administer the execution of Top-K Queries through an intuitive and declarative user interface. The above scenario conducts a Top-3 query over a 14-node sensor network organized in 6 clusters. The Display Panel (on the *right*) illustrates the three KSpot-Bullets for the three highest-ranked sensor clusters

sensing devices to the respective positions on the map. Our system allows the user to choose among a wide range of sensor devices, coming in various shapes and sizes, in order to accommodate crowded map configurations. Note that the Display Panel links together nodes of the same cluster using a black line. Additionally, the panel highlights the K-highest ranked clusters by utilizing a red bullet, coined the *KSpot Bullet*, which projects the rank of the given cluster at any given time instance. Subsequently, the KSpot bullets are continuously re-ranked such that the user is informed about the K highest ranked answers instantaneously.

KSpot Top-K Query API: The KSpot Query API has two functions: (i) to translate KSpot Top-K queries into TinyDB query messages with the aid of the TinyDB parser, and (ii) to asynchronously receive the results of a KSpot Top-K query and deliver these results both to the TinyDB and KSpot GUIs according to which is making the request. As illustrated in Fig. 2, the Query API supports two new types of queries, the TopK and TopKRoom queries that have been added to the TinyDB catalog.

KSpot Aggregates: Currently, KSpot supports two different aggregates, TopK and TopKRoom. Both aggregates are implemented in the TopkM and TopkRoomM modules, which are wired with the AggOperator configuration component of the TinyDB

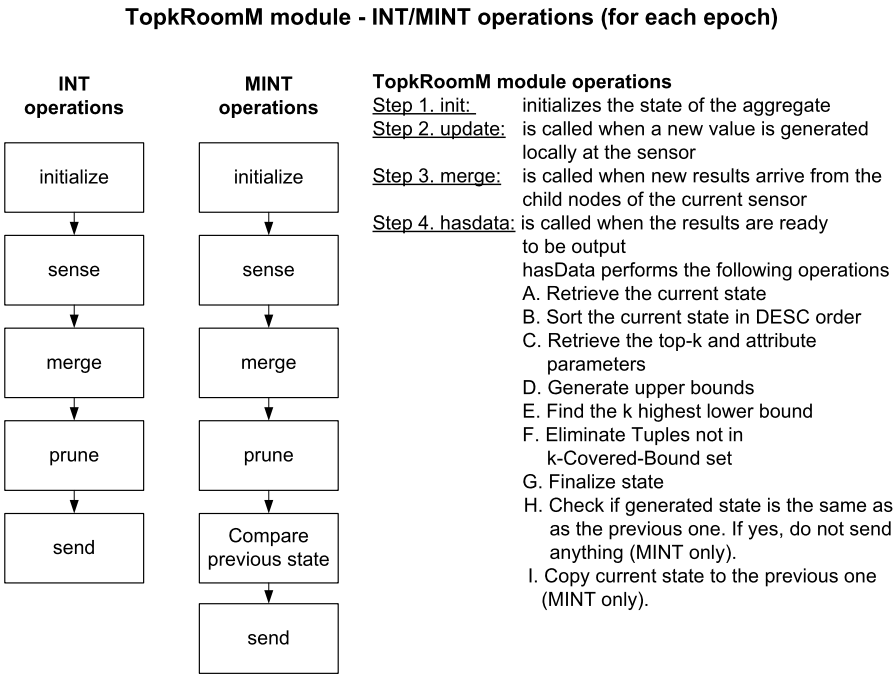

Fig. 4 Internal operations of the INT and MINT Views algorithms

client system. The KSpot data structures presented in Fig. 2 will be thoroughly described in the Experimental methodology section. An abstract representation of the internal mechanisms of the INT and MINT algorithms that operate inside the TopkRoomM module is illustrated in Fig. 4 (left). Similar to all aggregate operations supported by the TinyDB framework, both the INT and MINT algorithms follow a linear procedure to compute the result of each epoch. This procedure includes the following steps: (i) initialize, where all aggregate state variables are reset, (ii) sense, where each sensor generates its local measurement, (iii) merge, where each sensor acquires measurements from its child sensors and merges them with its own, (iv) prune, where the k-Covered-BoundSet is generated by pruning results that will not appear in the final top-k result, and (v) send, where the sensor transmits its results to its parent. The corresponding nesC functions that implement this procedure are illustrated in Fig. 4 (right). The pruning procedure of the underlying INT and MINT algorithms that operate inside the TopkM and TopkRoomM modules are thoroughly described in Section 4.

3 System model and definitions

In this section we will formalize our basic terminology upon which we will build the description of the algorithms that comprise the foundation of the KSpot Framework.

Table 1 Definition of symbols

Symbol	Definition
Q	A Query
k	Number of requested results
s_i	Sensor number i (s_0 denotes the sink).
n	Number of Sensors $\{s_1, s_2, \ldots, s_n\}$
m	Number of Attributes at each sensors $\{a_1, a_2, \ldots, a_m\}$
V_i	Local View (the results to Q) at sensor s_i ($i \leq n$)
V_i'	Pruned View at s_i (unveils the top-k answers at s_i)

We will then outline the motivation behind the phases of these algorithms. The main symbols and their respective definitions are summarized in Table 1.

Let S denote a set of n sensing devices $S = \{s_1, s_2, \ldots, s_n\}$. Assume that s_i ($i \leq n$) is able to acquire m physical attributes $A = \{a_1, a_2, \ldots, a_m\}$ from its environment at every discrete time instance t. This generates tuples of the form $\{t, a_1, a_2, \ldots, a_m\}$ at each sensor. At any given time instance, the aforementioned scenario yields an $n \times m$ matrix of readings $R := (s_{ij})_{n \times m}$. This matrix is *horizontally fragmented* across the n sensing devices (i.e., row i contains the readings of sensor s_i and $R = \bigcup_{i \in n} R_i$).

A user submits a query Q at some centralized querying node (denoted as s_0, or sink node) prior deployment and the system then initiates the execution of Q by disseminating it to the n sensors. In particular, the sink sends Q to one sensor s_1. Subsequently, s_1 recursively forwards Q to all of its neighbors until all n sensors have received the given query. Without loss of generality, we adopt the *First Heard From* (*FHF*) mechanism which is utilized in a variety of data acquisition frameworks such as [40, 60, 63, 67] and where each sensor s_i selects as its parent the first node from which Q was received. This creates an acyclic subset of the communication graph G (i.e., a spanning tree) which is denoted as $T = (S, E')$, where $E' \subset E$. Each s_i also maintains a *Child Node List* (denoted as children(s_i)), which is trivially constructed during the creation of T (i.e., using an acknowledgment from each child to its parent).

In other frameworks, like GANC [48] and Multi-Criteria Routing [38], T can be constructed based on query semantics, power consumption, remaining energy and others. In more unstable topologies a node can maintain several parents [16] in order to achieve fault tolerance but this might impose some limitations on the type of supported queries. Each sensor s_i is additionally supplemented with an *Alternate Parents List*, that is constructed locally at each sensor by *snooping* (i.e., monitoring the radio channel while other nodes transmit and recording neighboring nodes). This list is utilized in cases of network failures or low-quality links to the parent node.

4 KSpot framework algorithmics

In this section we describe the underlying algorithms of the KSpot framework. As already mentioned in the System Architecture section, that KSpot Framework operates on two new types of aggregate queries, `TopK` and `TopKRoom`.

The TopK query dictates that each sensor node must return at most k results (highest or lowest depending on the query) during each epoch (i.e., $|V_i'| \leq k$). The procedure for this is the following: (i) at each epoch, a sensor s_i collects the results from its child sensors, (ii) merges these results with its local results, generating V_i; and finally (iii) selects the k highest-ranked answers, generating in that way V_i'. As soon as this process is complete, the sensor s_i forwards V_i' to its own parent node. As the above procedure is conceptually not very complex, we do not devote any additional description to the internal mechanisms needed to realize this Top-k aggregate.

On the other hand, the TopKRoom query, which is responsible for GROUP-BY queries, features a much more complex pruning procedure that we will outline next. In this type of query, it is not always possible to discard tuples from V_i' because these may appear in the final k highest-ranked answers (recall the example that appeared in Section 1). To overcome this problem, we propose the MINT Views algorithm that utilizes an upper-bounding mechanism, which ensures that no tuples appearing in the final result will be omitted from V_i' during the pruning phase. Additionally, the MINT Views algorithm employs a temporal coherence filter that allows the suppression of results, if these do not change between subsequent epochs.

4.1 Overview of operation

In this section, we overview the three phases of the MINT Views algorithm, which addresses the TopKRoom-types of queries (i.e., group-by queries). We also present the INT Views algorithm, MINT's stateless version, which is appropriate for sensing devices of limited main memory.

The MINT Views algorithm consists of three phases:

A. *The Creation Phase*, executed during the first acquisition of readings from the distributed sensors. This phase results in n distributed views V_i ($i \leq n$);
B. *The Pruning Phase*, during which each sensor s_i locally prunes V_i and generates V_i' ($\subseteq V_i$). V_i' contains only the tuples that might be located among the final top-k results; and
C. *The Update Phase*, executed once per epoch, during which s_i updates its parent node with V_i'.

The above conceptual phases are executed in a distributed manner using the tree-based query routing protocol established by the operating system layer [29] after the query has been disseminated to the n sensors. In the following sections we thoroughly describe each phase of the MINT Views algorithm.

4.2 MINT creation phase

The first phase of the algorithm is a recursive execution of Algorithm 1 at all sensors in a given network. Recall that a sensor generates an $(m+1)$-tuple of the form $v = \{t, a_1, a_2, \ldots, a_m\}$ at each timestamp t. A sensor starts out by performing the selection σ_Q that retains the tuples that satisfy the selection criterion (e.g., temperature > 60). Note that a sensor can acquire concurrently several readings, all of which might not be of interest to a particular query. For example, the Xbow Weather board which was utilized in the Great Duck Island study [52] supplements the motes with

Algorithm 1 : **Construct MINT/INT View**

Input: A distributed sensor s_i ($\forall s_i \in S$) that generates m attributes $\{a_1, a_2, \ldots, a_m\}$, a query Q, an empty buffer $V_i = \{\}$
Output: A set of n distributed views $V = \{V_1, V_2, \ldots, V_n\}$.

```
1:  procedure CONSTRUCT_MINT_VIEW(s_i, Q)
2:      // Execute Q and store the answer in V_i (takes O(1) time).
3:      insert(π_Q(σ_Q(current_reading())), V_i);
4:      for j = 1 to |children(s_i)| do
5:          c = child(s_i, j); // c is the jth child of node s_i
6:          // w is a list of tuples returned to query Q.
7:          w = Construct_Mint_View(c, Q);
8:          for l = 1 to |w| do
9:              // w_l is the lth entry of table w.
10:             // Inserts tuple w_l into local table V_i in O(1) time.
11:             insert(w_l, V_i);
12:         end for
13:     end for
14:     send(V_i, parent(s_i));
15: end procedure
```

14 physical parameters. Thus, we only project the attributes related to Q prior to storing the result in the in-memory buffer V_i (line 3). The next step of the algorithm merges the tuples that arrive from the children of s_i into V_i (lines 4–13). This yields an in-network view similar to Fig. 1 (right).

If the various values at each node of the depicted tree do not change across consecutive timestamps, then V can efficiently provide the answer to the subsequent re-execution of Q. On the contrary, whenever we have a deviation, or a change, in a parameter at s_i, this change has to cascade all the way up to the sink. A change at all sensors has a worst-case message complexity of $O(n)$ for every single timestamp of the *epoch* duration, thus we seek to optimize this process through the proposition of the pruning phase.

4.3 MINT pruning phase

Algorithm 1 constructs a hierarchy of views, where ancestor nodes in the routing hierarchy maintain a superset view of their descendants. Before we explain the details of the pruning phase which minimizes messaging between sensors consider the following query:

Query 2 (Q2)

```
SELECT TOP k room,avg(temp)
FROM SENSORS
GROUP BY room
SAMPLE PERIOD 60000
```

which returns the k rooms with the highest average temperature. If s_i could locally define the k-highest answers to Q2 (at s_0), then s_i could use this information to prune

room	sum	count	sum'
2	200	4	320
5	270	4	390
6	500	5	500
11	460	4	580
12	290	3	530
15	130	2	490

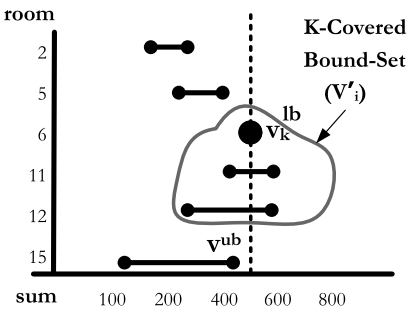

Fig. 5 The *left table* illustrates the V_i of a given node during the execution of query Q2. The *right figure* illustrates the intuition of the pruning algorithm. In particular, we plot the (lb,ub) ranges for the various returned tuples at some arbitrary node. We then generate a k-covered bound set V_i' using Algorithm 2. We only propagate a tuple u to the parent of s_i, if $u \in V_i'$

its local view V_i. However, this is a recursively defined problem that can only be solved once all tuples percolate up to the sink s_0. In order to avoid this, we utilize a set of descriptors γ which are utilized to bound above the attributes in V_0 and subsequently enable a powerful pruning framework.

Consider the example of Fig. 5 (left), where we illustrate the V_i for a given sensor. Prior to the execution of $Q2$ we established that $\gamma_1 =$"*Maximum possible temperature value*"=120 and $\gamma_2 =$"*Number of sensors in each room*"=5. The figure indicates the *sum* and *count* for several room numbers. By observing column 3 (i.e., count), it becomes evident that the *sum* for the rooms $\{2, 5, 11, 12, 15\}$ is a partial value of the *sum* returned at the sink (since $\gamma_2 = 5$).

On the contrary, the tuple of room 6 is already in its final form (i.e., 500). In this example the *sum* of each row is bounded above using the following formula $sum' = sum + (\gamma_2 - count) * \gamma_1$ and bounded below using the actual attribute *sum*. This creates six lower-bound (lb) and upper-bound (ub) pairs which precisely show the range of possible values for the *sum* attribute at the sink.

Having such knowledge locally, it can now help us to prune (lb, ub) pairs which will not be in the final top-k result. The intuition behind our algorithm is to identify the k^{th} highest lower bound (i.e., v_k^{lb}) and then eliminate all the tuples that have an upper bound (i.e., v^{ub}) below v_k^{lb}. Figure 5 (right), visually depicts this idea. We will prove that by applying locally such an operation yields at the end the correct top-k tuples at the sink. In order to achieve this we define the notion of a *k-Covered Bound-Set* as following:

Definition 1 (k-Covered Bound-Set) (V_i') is the subset of V_i that satisfies the following condition: If there is some $v \notin V_i'$, then $v^{ub} < v_k^{lb}$, where v_k^{lb} is the kth highest lower bound.[3]

[3]Due to contraposition, the condition could also be expressed using the implication *if* $v^{ub} \geq v_k^{lb}$, *then* $v \in V_i'$.

Algorithm 2 : **Prune MINT/INT View**

Input: A distributed sensor s_i ($\forall s_i \in S$), a buffer V_i that contains the local view, a set of descriptors $\gamma = \{\gamma_1, \gamma_2, \ldots\}$, a query result parameter k.
Output: A locally pruned view V_i', such that V_0' can be utilized to answer a top-k query Q.

```
1:  procedure PRUNE_MINT_VIEW(V_i)
2:      for j = 1 to |V_i| do // Identify the pruning threshold v_k^lb.
3:          v_j = V_i[j] // v_j=(v_j^lb, v_j^ub) pair.
4:          kHighest(v_j^lb, kBuff)
5:          bucketinsert(v_j^ub, sortedUBs)
6:      end for
7:      v_k^lb = min(kBuff);
8:      for j = 1 to |sortedUBs| do
9:          v_j^ub = sortedUBs[j]
10:         If (v_j^ub < v_k^lb) then break; end if
11:         add_to_candidates(v_j, V_i');
12:     end for
13: end procedure
```

Algorithm 2 illustrates the pruning of V_i at some arbitrary node s_i and the construction of the candidate set V_i'. This algorithm applies to both the MINT View and the INT View algorithms. The first step of the algorithm (lines 2–6) identifies the pruning threshold v_k^{lb}. This threshold allows the algorithm to prune-away tuples that will not be in the result.

Although V_i physically resides in main memory, we want to minimize the running time of our algorithms in order to accommodate the scarce energy budget. In particular, we utilize similarly to the well known *selection algorithm*, a k-element buffer *kBuff* in order to locate v_k^{lb} in linear time (i.e., $O(k)$ per tuple). This procedure takes place inside the *kHighest* function which inserts v_j^{lb} into *kBuff*, if the former is larger than the minimum item in *kBuff*.

The next step of the algorithm is to locate the tuples that have an upper bound v^{ub} below the threshold v_k^{lb}. By visually examining Fig. 5, it is easy to see that an efficient way to do so is to create an ordered list of upper bounds and then perform a linear scan in descending order until a tuple v_j^{ub} ($< v_k^{lb}$) is located. Any upper bound below or equal to v_j^{ub} can be safely eliminated.

The ordered list can be constructed in parallel with the location of the pruning threshold v_k^{lb}. In particular, while scanning for v_k^{lb}, we insert each upper bound v_j^{ub} into a new table *sortedUBs* (line 5). This takes only O(1) per tuple as we utilize an idea similar to *bucketsort*. However, if memory is limited then this optimization can be avoided without any consequence on the correctness of our approach.

In lines 8–12, we finally perform a linear scan of the *sortedUBs* table in descending order and stop when we find a tuple v_j^{ub} that is below v_k^{lb}. The correctness of our algorithm is established by Theorem 1.

Theorem 1 *The k-Covered Bound-Set V_i' correctly identifies the k-highest ranked answers to Q.*

Algorithm 3 : **Update MINT View**

Input: A buffer T' that contains the V_i' of the previous time instance, the v_k^{lb} of T', a tuple update x from some child.
Output: A locally pruned view V_i', such that V_0' can be utilized to answer a top-k query Q.

```
1: procedure UPDATE_MINT_VIEW(T', v_k^lb, x)
2:     V_i' = T';
3:     if (v_k^lb ≤ x^ub) then
4:         add_to_candidates(x, V_i');
5:         if (x^lb ≤ v_k^lb) then
6:             send(x, parent(s_i)); // Single tuple x update
7:         else  // x^lb > v_k^lb
8:             Prune_MINT_View(V_i'); // Using Algorithm 2
9:             send(V_i', parent(s_i)); // Complete V_i' update
10:        end if
11:    end if
12:    T' = V_i';
13: end procedure
```

Proof (by contradiction): Let v denote an arbitrary tuple which is not included in the k-Covered Bound-Set V_i'. We have to show that v will have a smaller value than any of the k highest-ranked tuples w (i.e., $v < w$). Assume that $v \geq w$. It always holds that $v^{ub} \geq v$ which consequently yields $v^{ub} \geq w$ (by using the assumption). However if $v^{ub} \geq w$, then v would have been included in V_i', by Definition 1, a contradiction. □

4.4 MINT update phase

In the previous step, we transformed V_i into a pruned subset V_i'. We shall now describe how to incrementally and recursively update V_i'. Let T' denote the V_i' taken at the last execution of Q. The below description only applies to the MINT View algorithm, for which T' is available. The update phase of the INT View algorithm is simply a re-execution of Algorithm 1 which re-constructs V_i' from the beginning.

Since our objective is to identify the correct results at the sink, we utilize an *immediate* view maintenance mechanism: "*As soon as a new tuple is generated at s_i, this update is reflected in V_i'*". In order to minimize communication, s_i only re-transmits V_i' to its parent, if V_i' has changed (*temporal coherence filter as in TINA*). Additionally, in order to minimize energy consumption even further, we seek to minimize processing consumption as well. Therefore, our objective is to construct V_i' by avoiding the re-executing of Algorithm 2.

Algorithm 3 presents the MINT Update Algorithm and Fig. 6 illustrates the respective steps of the algorithm. In particular, line 3 of Algorithm 3 shows that any tuple update x with an upper bound (denoted as x^{ub}) less than the v_k^{lb} can be *ignored* (also see respective Example 1 of Fig. 6). In the opposite case, we add the tuple x to the set of candidates V_i' (line 4 of Algorithm 3 and Example 2 of Fig. 6).

Now the remaining question is whether v_k^{lb} has changed by this addition of x. If $x^{lb} \leq v_k^{lb}$ is true then v_k^{lb} has not changed. Consequently, s_i only propagates the update x towards its parent rather than a complete view update. In the implementation

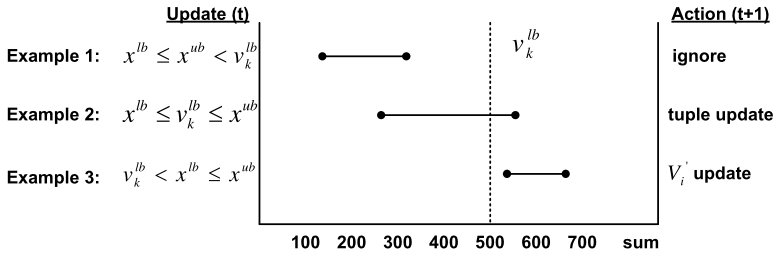

Fig. 6 The figure illustrates how different tuples will be handled during the update phase

we buffer these updates until all children send their updates to their parents. If on the contrary $v_k^{lb} < x^{lb}$, then v_k^{lb} might have changed. As a result s_i has to reconstruct V_i' using Algorithm 2 and transmit the complete V_i' to its parent (also see respective Example 3 of Fig. 6). This re-construction procedure is necessary to guarantee the correctness of our algorithm. Note that the reconstruction only happens for $|V_i'|$ elements rather than all the elements (i.e., $|V_i|$), had we executed Algorithm 2 for the first time.

4.5 Dynamically tuning the γ descriptors

The gamma descriptors are used for bounding above the maximum possible value of tuples in the INT and MINT Views algorithms. In our examples so far, γ_1 denoted the "*Maximum possible temperature value*" and γ_2 denoted the "*Number of sensors in each room*". While the static (fixed) configuration of these descriptors is general enough to fit different application scenarios (e.g., using humidity, light, sound, etc.), this could lead to a sub-optimal pruning power of our framework when these are overestimates. While our experimental evaluation in Section 6 shows that this will not be very typical, in this section we discuss for completeness how these descriptors can be adjusted dynamically with runtime knowledge.

Tuning γ_1 (Maximum Possible Sensed Value): Assume that we need to determine the maximum value for a sensed parameter (e.g., temperature) over the past. Using the running maximum (i.e., highest value seen so far), is certainly not efficient as some outlier, or some abnormal past recording, will set the running maximum to a high value. Subsequently, this will limit the pruning power of the KSpot framework. However, since the majority of sensor readings (e.g., temperature, humidity, light, voltage, etc.) usually follow the Gaussian distribution, the maximum possible value for an attribute can be predicted using a sliding window sampling mechanism. Given the limited memory and processing capabilities of sensor devices, the size of the sliding window must be relatively small, for memory and processing reasons, but also large enough to accurately predict the next maximum value.

In our setting, we have implemented the sliding window sampling mechanism using a circular buffer (CB) of size 40 bytes (10×4 bytes). CB records the requested sensor measurement (*val*) for the previous 10 epochs. In the case where the CB structure is full, the oldest value is omitted. We can configure the γ_1 descriptor using the

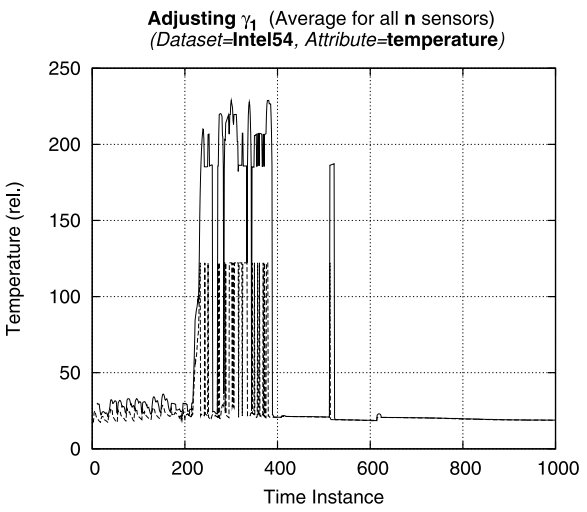

Fig. 7 Dynamic adaptation of the γ_1 descriptor using a sliding window prediction mechanism

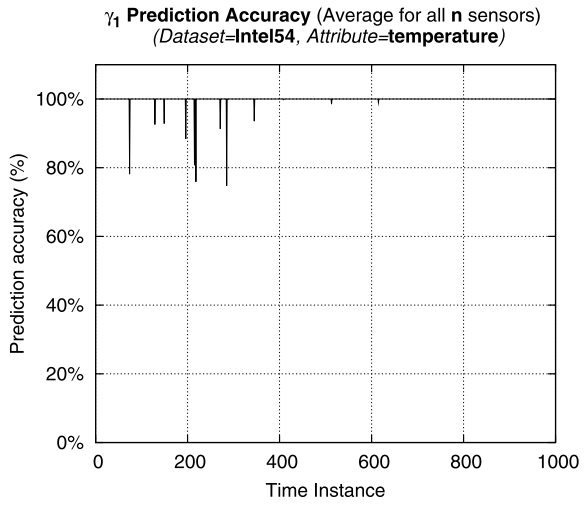

Fig. 8 Prediction accuracy (γ_1) of the sliding window prediction mechanism

following formula: $\gamma_1 = MAX_{i=0}^{|\text{CB}|}(val_i \in \text{CB}) + 2 \times \sigma_{i=0}^{N}(val_i \in \text{CB})$, where σ is the standard deviation.

Figure 7 shows how γ_1 is dynamically adapted through 1000 timestamps using the Intel49 dataset presented in Section 5. We observe that γ_1 is tightly bounding the real recorded value, i.e., it is approximately ≈4% higher than the recorded value. Additionally, Fig. 8 shows that this prediction is correct in 95% of the cases and that the incorrect situation is usually corrected in the immediately next epoch. The above discussion shows that one can easily achieve higher pruning power with acceptable levels of accuracy.

Tuning γ_2 (Number of Sensors Per Room): Now assume that we need to dynamically determine the γ_2 descriptor, which refers to the number of sensors per *room*.

A room in our description is a "*conceptual region that needs to be monitored using several sensors such that a group-by aggregate per region—e.g., average—can be determined.*" In case the sensor board features a GPS (e.g., Crossbow's MTS420), then the conceptual partitioning can easily be conducted at the sink, by the human operator, after acquiring over the air the coordinates of the participating nodes. If on the other hand absolute positioning techniques are not available, then sensor devices can derive their coordinates through relative means.

In particular, several localization technologies have been discussed in the literature including methods based on Infrared, Bluetooth, RFID, UWB, ultrasound and WLAN [28]. The underlying positioning algorithms may utilize different types of measurements, such as *Angle of Arrival (AOA)*, *Time of Arrival (TOA)*, *Time Difference of Arrival (TDOA)* and *Received Signal Strength (RSS)*. These techniques could have been utilized for localizing nodes and for dynamically tuning the γ_2 descriptor, but a more extensive exploration of these techniques remains outside the scope of this paper.

4.6 Discussion

MINT vs. INT: The differences of the two algorithms are summarized as following: (i) MINT exploits a temporal coherence in order to suppress view updates that do not change between consecutive time instances, while INT has to re-send these updates, because it is stateless. (ii) In MINT, we only have to update V_i' using Algorithm 3 (in $O(|V_i'|)$ time), while in INT we have to construct it every time from the beginning, in $O(|V_i|)$ time, using Algorithm 2. (iii) INT has the advantage of not requiring any extra storage thus is more appropriate for sensors for which the storage is at premium.

Deferred View Updates: In order to minimize communication even more in the MINT/INT Views, we could have opted for a *deferred* view maintenance mechanism, rather than a *immediate* one. A *deferred* mechanism could propagate changes periodically, after a certain number updates or even randomly. In all cases this would produce probabilistic answers at the sink, as the sink would not have at its disposal the most up-to-date view. Although deferred view maintenance mechanisms are extremely interesting in the context of sensor networks, as these allow us to trade accuracy versus energy consumption, in this paper we only focus on exact answers.

In-Memory Buffering: The materialized views and temporary results of all algorithms, can either reside in an SRAM-based buffer or a Flash-based buffer. For instance, a typical MICA mote with a 2 KB SRAM might need to exploit the 512 KB on-chip flash memory, while Intel's i-mote might easily store these results in the 64 KB SRAM. There is a growing trend for more available local storage in sensor devices [65] and therefore local buffering of results is not a threat to our model.

Supported Query Types: We support single-relation queries with the standard aggregate functions (i.e., `SUM, MIN, MAX` and `AVERAGE`). In contrast with other frameworks, we optimize queries with *multi-tuple* answers. Such answers could be generated by a `GROUP-BY` clause, or by a non-aggregate query. Note that for *single-tuple* answers, such as those generated by an aggregate query without a GROUP-BY clause, there is no notion of a top-k result.

5 Experimental evaluation methodology

In this section we describe our experimental methodology which involves a set of trace-driven simulations with real datasets from the Department of Atmospheric Sciences at the University of Washington, Intel Research Berkeley and UC-Berkeley and a real micro-benchmark on the CC2420 radio chip [53], utilized on MICAz, TelosB and IMote2 sensing devices. Our testbed is a publicly available real system that has been demonstrated at [3].

The experimental evaluation described in this section focuses on five parameters: (i) the **Energy Consumption Cost**, for the INT and MINT Views algorithms proposed in this paper compared to two other popular query processing algorithms namely, TAG and TINA, (ii) the **Pruning Magnitude**, of the k-Covered Bound-Set V_i^i of the INT and MINT Views algorithms, (iii) the **Scalability with respect to** k, were we evaluate the efficiency of the MINT Views algorithm with different values of the k parameter, (iv) the **Cardinality of the GROUP-BY clause**, were we evaluate the effect of different cardinalities on the energy consumption of the MINT Views algorithm, and (v) the **Network Lifetime**, of all algorithms presented in this paper.

5.1 Experimental testbed

In order to fairly compare the INT and MINT Views algorithms we have implemented, or ported, all algorithms discussed in this paper under the KSpot Framework. It is important to mention that the TAG algorithm is already implemented as part of the TinyDB framework (that lies at the kernel of KSpot) and has been used as a baseline for comparison. The rest algorithms, TINA, INT and MINT Views, were implemented from scratch in nesC [27], the programming language of TinyOS [29].

TinyOS is an open-source operating system designed for wireless embedded sensor nodes. It was initially developed at UC-Berkeley and has been deployed successfully on a wide range of sensor devices (e.g., Mica, Telos, IMote2 mote, etc.). TinyOS uses a component-based architecture that enables programmers to wire together the minimum required components in on-demand basis. This minimizes the final code size and energy consumption as sensor nodes have extremely limited power and memory. nesC [27] is the programming language of TinyOS and it realizes its structuring concepts as well as its execution model.

We utilize the TOSSIM [37] environment to conduct realistic trace-driven simulations of our code with a variety of input datasets described next. TOSSIM [37] provides a scalable, high fidelity simulation environment of TinyOS sensor networks. It simulates the TinyOS network stack, allowing experimentation with low-level protocols in addition to top-level application systems. In order to conduct fine-grained power modeling in TOSSIM, we use PowerTOSSIM [49], a popular power modeling extension of TOSSIM. PowerTOSSIM has been shown [49, 65], to be more than 90% accurate. In particular, the authors in [49] measure the energy for executing the demonstration examples bundled with TinyOS both using PowerTOSSIM and on real sensors (measured with a multi-meter). The authors show that this yielded an average error of only 4.7%. Similar observations also apply for more complex applications

Fig. 9 Sample execution scenario for the MINT Views algorithm on the GDI dataset

```
Step 1: Create lossy radio model
   java net.tinyos.sim.LossyBuilder-d 7 2
   -s 20 -o 7x2_20.nss

Step 2: Run experiment with TOSSIM and
        collect power statistics
   DBG=power ./build/pc/main.exe -b=10 -seed=10 -t=1000
   -r=lossy -rf= 7x2-20.nss -p 14 > mintGDI.trace

Step 3: Get energy results from power
        statistics
   ./postprocess.py --detail --sb=1 -em
   telos_energy_model.txt mintGDI.trace
   >mintTotalEnergy.txt
```

Fig. 10 Trace from the PowerTOSSIM log file

```
...
38: POWER: Mote 38 ADC ON at 2741220
38: POWER: Mote 38 ADC ON at 2741220
38: POWER: Mote 38 ADC ON at 2741220
38: POWER: Mote 38 RADIO_STATE ON at 2741220
8: POWER: Mote 8 RADIO_STATE TX at 2791414
38: POWER: Mote 38 RADIO_STATE RX at 2842220
8: POWER: Mote 8 RADIO_STATE RX at 2850614
8: POWER: Mote 8 RADIO_STATE RX at 2851464
8: POWER: Mote 8 RADIO_STATE RX at 2852264
8: POWER: Mote 8 RADIO_STATE RX at 2853064
8: POWER: Mote 8 RADIO_STATE RX at 2853864
8: POWER: Mote 8 RADIO_STATE RX at 2853864
38: POWER: Mote 38 RADIO_STATE TX at 2862733
38: POWER: Mote 38 RADIO_STATE TX at 2863533
...
```

like TinyDB and Surge that were shown to have an error of 9.5% on average. Consequently, we expect that the accuracy will remain at the same high levels with our integrated TelosB power model.

Figure 9 illustrates an example of the process that we utilized in order to collect power statistics for our experiments. In the first step, we create a lossy model for the topology and store it in an `.nss` file. We then execute the experiment for a fixed time period (e.g., 1000 s ($-t = 1000$)) and collect power statistics in the `.trace` file. An example of the PowerTOSSIM trace file is depicted in Fig. 10. Finally, we process the power statistics file in order to generate the energy results for each sensor.

Our simulation experiments were performed on a Lenovo Thinkpad T61p PC with an Intel Core 2 Duo CPU running at 2.4 GHz and 2.0 GB of RAM. In order for us to collect realistic results for a large period of time, we collect statistics for 1000 epochs in each experiment. To increase the fidelity of our measurements we repeated each experiment 5 times and present the average energy consumption for each type of plot. The above process, resulted in quite long simulation runs for each type of plot as the simulation time required for completing an experiment and generating the power trace file, ranged from 2.5 to 8.5 hours. Furthermore, the generated power trace file size ranged from 20–250 MB, depending on the dataset. This led to an additional time overhead of 30–100 minutes for processing each power trace file, in order to collect the energy results. Our simulation statistics are depicted in Fig. 11.

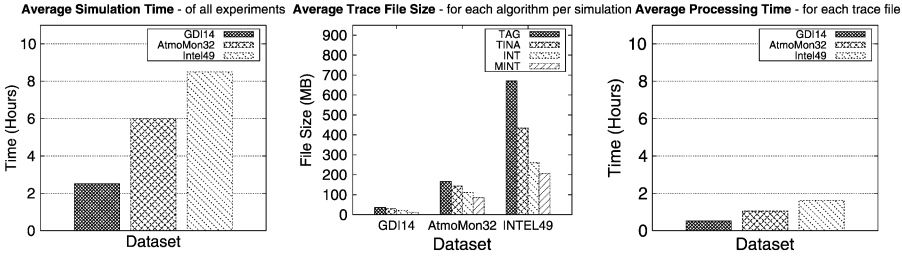

Fig. 11 Simulation Statistics: Average Simulation Time required for each experiment (Left); Average File Size required for storing the power statistics for each algorithm (Middle). Average Processing Time for each power trace file (Right)

5.2 Datasets

We utilize the following three real datasets in our trace-driven experiments in order simulate different network sizes:

i. **Great Duck Island (GDI14):** This is a real dataset from the habitat monitoring project deployed in 2002 on the Great Duck Island which is 15 km off the coast of Maine [52], USA. We utilize readings from the 14 sensors that had the largest amount of local readings. The GDI dataset includes readings such as: light, temperature, thermopile, thermistor, humidity and voltage.

ii. **Washington State Climate (AtmoMon32):** This is a real dataset of atmospheric data collected at 32 sensors in the Washington and Oregon states, by the Department of Atmospheric Sciences at the University of Washington [23]. More specifically, each of the 32 sensors maintains the average temperature and wind-speed on an hourly basis for 208 days between June 2003 and June 2004 (i.e., 4990 time moments).

iii. **Intel Research Berkeley (Intel49):** This is a real dataset that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley [31] between February 28th and April 5th, 2004. The sensors utilized in the deployment were equipped with weather boards and collected time-stamped topology information along with humidity, temperature, light and voltage values once every 31 seconds (i.e., the epoch). The dataset includes 2.3 million readings collected from these sensors. We use readings from the 49 sensors that had the largest amount of local readings since some of them had many missing values.

5.3 Sensing device

We use the energy model of Crossbow's TelosB [17, 45] research sensor device to validate our ideas (see Fig. 12). TelosB is a ultra-low power wireless sensor equipped with a 8 MHz MSP430 core, 1 MB of external flash storage, and a 250 Kbps Chipcon (now Texas Instruments) CC2420 RF Transceiver that consumes 23 mA in receive mode (Rx), 19.5 mA in transmit mode (Tx), 7.8 mA in active mode (MCU active) with the radio off and 5.1 μA in sleep mode. Our performance measure is *Energy*, in *Joules*, that is required at each discrete time instance to resolve the query.

Fig. 12 Crossbow's TelosB Mote (TPR2420). Our micro-benchmark and trace-driven experiments utilize the energy model of the TelosB sensor device and the CC2420 radio transceiver

As TelosB is not part of the PowerTOSSIM module, we had to extend PowerTOSSIM by incorporating a new energy model for TelosB. In particular, the process of configuring different hardware platforms in PowerTOSSIM, boils down to the customization of the configuration file that enumerates the power consumption of the individual operations (e.g., RADIO_RX, RADIO_OFF, CPU_ACTIVE, etc.).

5.4 Multi-hop topologies

In order to create a multi-hop network topology, we have utilized the LossyBuilder component of TOSSIM. LossyBuilder allows the creation of "lossy" radio models for the topologies utilized in all datasets. These lossy models position sensors at various distances from the sink node and generate a Gaussian packet loss probability distribution for each distance. TOSSIM then generates packet loss rates for each sensor pair by sampling these distributions and translating them into independent bit error rates. An example of the LossyBuilder output is depicted on Fig. 13 where we list some of the bit error for sensor mote (with id = 0) on a topology of 10 nodes. For example, line 3 (0 2 0.5) of Fig. 13 states that mote 0 listens to mote 2 with a bit error rate of 50%. This process allows the creation of multi-hop network topologies required for all of our experiments.

5.5 Communication protocol

Our communication protocol is based on the ubiquitous for sensor networks IEEE standard 802.15.4 (the basis for the ZigBee [68] specification used by most sensor devices including the TelosB sensor device). ZigBee uses the CSMA/CA collision avoidance scheme where a node employs a random exponential back-off algorithm that backs-off for a random interval of 0.25–0.5 s before retransmission.

The TinyDB message frames are structured as follows [37]: Each message is associated with a 7 Bytes TinyDB application layer header that includes: (i) the source identifier (2B), the query source identifier (2B), the sequence number (2B) and the hop count (1B). In the remaining payload (29B) we allocate our KSpot structures according to the query being executed:

i. **For the `TopkData` data structure:** we allocate 1 bit for identifying if the current state is the same as the previous state, 3 bits for identifying the number of tuples in the current state and 2B for the attribute value.

Fig. 13 Trace from the LossyBuilder output

```
<mote id> <mote id> <error rate>
...
0 0 0.0
0 1 8.99E-4
0 2 0.5
0 3 0.5
0 4 8.99E-4
0 5 0.012694
0 6 0.5
0 7 0.002147
0 8 0.5
0 9 0.5
0 10 0.00965
...
```

ii. **For the `TopkRoomData` data structure:** which is used for Top-k `GROUP-BY` queries, we allocate a number of variables for storing results for each room. In order to maximize the number of rooms that can be supported by this query we utilized a packed data structure that consists of the following information: (i) *sameAsPrevious (1 bit)*: is a bit flag that indicates whether the current result is the same as the previous result and thus should not be transmitted, (ii) *vals (3 bits)*: the number of values in the topk result. Note that the number of values is identical to the number of rooms that have reported their values. The maximum number of values (i.e., maximum number of rooms is 7), (iii) *count (22 bits)*: This attribute records the number of results for each room using a 3-bit counter for each room (maximum number of rooms $= 7 \times$ 3-bits $= 21$), (iv) *room (22 bits)*: This attribute records the room id of each room, and (v) *sum (16 bits* $\times$ *maximum number of rooms)*: stores the cumulative total of the sensor's result for each room. Since the maximum number of bytes available in the TinyDB message payload is 25 bytes, our packed data structure supports a maximum of 7 rooms.

Both of our data structures are illustrated in Fig. 2.

5.6 Query syntax

We utilize the following query syntax:

```
SELECT TOP K attribute [,aggregate]
FROM sensors
[WHERE filter]
[GROUP BY attribute]
[ORDER BY [attribute|aggregate] [ASC|DESC]]
[SAMPLE PERIOD time (ms)]
```

The attribute statement mentioned in the query syntax refers to all measurements that can be acquired from the sensorboard as well as variables stored locally at each sensor node. In the KSpot framework, when a TOP k attribute query is executed over the network we only return the k highest results for that attribute, if no ORDER BY clause is used. However we could have easily returned the k lowest results in

a similar way. The aggregate statement mentioned in the query class form refers to all aggregates supported by the TinyDB framework. Roughly, these aggregates can be distinguished in: (i) *distributive aggregates*, where records can be aggregated in-network without compromising correctness (e.g., `Max, Min, Sum, Count`), and (ii) *holistic aggregates*, where in-network aggregation might compromise the result correctness (e.g., `Median`), thus all tuples have to be transmitted to the sink before the query can be executed. If a GROUP-BY query is posted to the network, results are grouped by the attribute statement and aggregates are calculated for each group individually. In our experiments we utilize the following query:

```
SELECT TOP K room, AVG(temp)
FROM sensors
GROUP BY room
ORDER BY AVG(temp) [DESC]
SAMPLE PERIOD 4096
```

where k is configured as the 5% of the complete answer set. We also use the same epoch duration for all our experiments which specifies the amount of time that sensors have to wait before re-computing the continuous query. More specifically we set the epoch duration to be equal to 4096 ms.

6 Experimental evaluation results

In order to assess the efficiency of the algorithms presented in this paper we have conducted six experimental series. In the first experimental series we have compared the energy consumption of the INT and MINT Views algorithms to the TAG and TINA algorithms, showing that the former algorithms present significant energy savings compared to their competitors. In the second experimental series, we study the pruning magnitude of the INT and MINT Views algorithms. In the third experimental series, where we investigate the scalability of the parameter k, we manually test the efficiency of the MINT Views algorithm with different values for k. In the fourth experimental series we investigate the effect of the GROUP-BY cardinality. Note that in all datasets, we randomly and uniformly divide the sensors into areas (rooms). In this experiment, we distribute the sensors in different room configurations and study the energy consumption of all algorithms. In the fifth experimental series, we evaluate the efficiency of the overall KSpot framework focusing on energy consumption and system lifetime. Finally, in the sixth series, we have conducted one micro-benchmark on the CC2420 radio transceiver in order to quantify its reception inefficiencies in a real setting.

Table 2 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

6.1 Experimental series 1: energy consumption

In the first experimental series, we evaluate the energy consumption of INT and MINT Views algorithms compared to the popular TAG and TINA acquisition frameworks. We execute query Q on the three datasets and measure the energy consumption for each dataset separately.

Table 2 Configuration parameters for all experimental series

Section	Objective	Dataset	k	Rooms (R)
6.1	Energy Consumption	GDI14, AtmoMon32, Intel49	5%	4–7
6.2	Pruning Magnitude	AtmoMon32	5%	7
6.3	Scalability of k	GDI14	5%–100%	4
6.4	GROUP-BY cardinality	GDI14	5%	1–7
6.5	Network Lifetime	GDI14	5%	4

In Fig. 14 (top-left), we illustrate the energy consumption of the four algorithms (MINT, INT, TINA and TAG) using the GDI14 dataset. Let us mention the preliminary observation that the energy scale among consecutive time instances fluctuates greatly. This happens due to the arbitrariness of when and under which condition top-k pruning and temporal coherence filtering takes place. In order to correct this situation in the subsequent graphs, we apply a spline interpolation smoothing between consecutive data points. We shall next also mention the real observations we determine from the given execution.

In Fig. 14 (top-right), we plot the results using the GDI14 dataset. Since we utilize TAG as the baseline of comparison, it always has a value of 100%. The TAG line accounts for approximately 57 ± 2.52 J average energy for all 14 nodes of the network. Recall that in TAG, a sensor always transmits all aggregated tuples to the sink. Although TINA still returns all answers to the sink, it takes the average energy consumption down to 48 ± 1.57 J. This validates that exploiting temporal coherence can be beneficial in most cases. The INT Views approach on the other hand, performs in-network pruning of the results which reduces the energy consumption to 34 ± 1 J (i.e., $\approx$41% less than TAG).

Finally, the MINT Views algorithm exploits temporal coherence in addition to top-k pruning and only consumes an average of 19 ± 0.56 J which is equivalent to a 66% energy reduction from TAG, 49% energy reduction from TINA and 25% from INT. The reason why the TINA and MINT Views follow a similar pattern is because in both curves, the energy reduction is dominated by the savings that are due to the temporal coherence between consecutive time points.

In this figure, we also observe surges (deviations) for the TINA, INT and MINT Views algorithms in all experiments. In the case of the TINA algorithm, the surges attribute to the fact that, at some time instances, the sensors exploit the temporal coherence and do not report their results to their parents. This decreases the overall energy consumption of the network. In the case of the INT algorithm, the surges correlate with the fact that, at some time instances, the amount of results pruned from V_i is decreased or increased because of the deviation of values in the dataset. This is an indication that the top-k answer has changed at the particular timestamp and that this has brought some increase in energy consumption, until the updates propagate to the sink. On the other hand, the surges of the MINT Views algorithm correlate to both of the aforementioned attributes as MINT exploits both temporal coherence and top-k pruning.

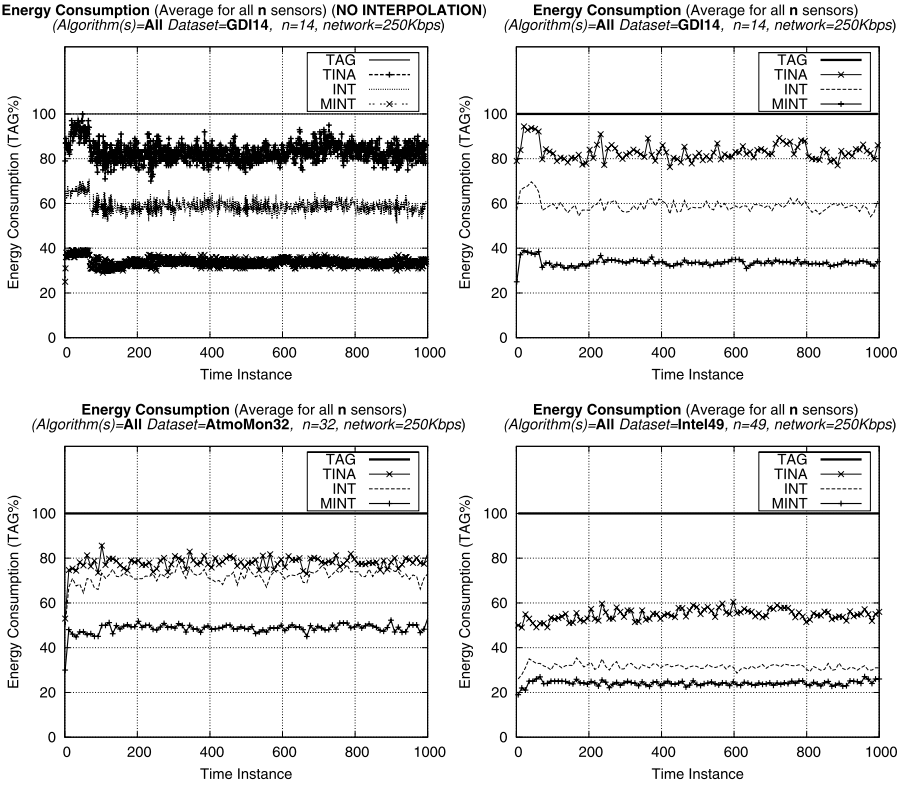

Fig. 14 Energy Consumption for the TAG, TINA, INT View and MINT View algorithms using the TelosB sensor energy model

By repeating the same experiment on the AtmoMon32 dataset, we observe in Fig. 14 (bottom-left), that MINT continuous to maintain a competitive advantage over TAG and TINA. In particular, we observe that MINT consumes 50% less energy than TAG (i.e. 115 ± 4 J versus 234 ± 2 J). The same conclusion applies for the INT Views algorithm although we observe that the performance difference compared to TINA has decreased. This happens because in the AtmoMon32 dataset, the temperature values do not change frequently and this allows the temporal coherence filter to significantly reduce the number of tuples transmitted over the network. However, the top-k pruning filter of the INT algorithm still manages to considerably decrease the size of packets that are transmitted through the network thus maintaining an advantage over TINA.

In the final dataset, Intel49 (Fig. 14 (bottom-right)) we observe that all algorithms behave in a similar manner to the previous experiments. The difference is that the energy performance of all algorithms has increased compared to TAG. One reason that this happens, is the fact that like the AtmoMon32 dataset the temperature values of the Intel49 do not change frequently, which is exploited by the temporal coherence filter of the TINA and MINT Views algorithms. On the other-hand, the INT Views algorithm which does not employ a temporal coherence filter, outperforms signifi-

Table 3 Average Energy Consumption for all sensors in experimental series 1: Evaluation of the TAG, TINA, INT and MINT Views algorithms under different datasets

Algor.	Dataset		
	GDI14	AtmoMon32	Intel49
TAG	57 ± 2.52 J	234 ± 2 J	523 ± 22 J
TINA	48 ± 1.57 J	183 ± 6 J	289 ± 15 J
INT	34 ± 1.01 J	170 ± 7 J	187 ± 08 J
MINT	19 ± 0.56 J	115 ± 4 J	139 ± 06 J

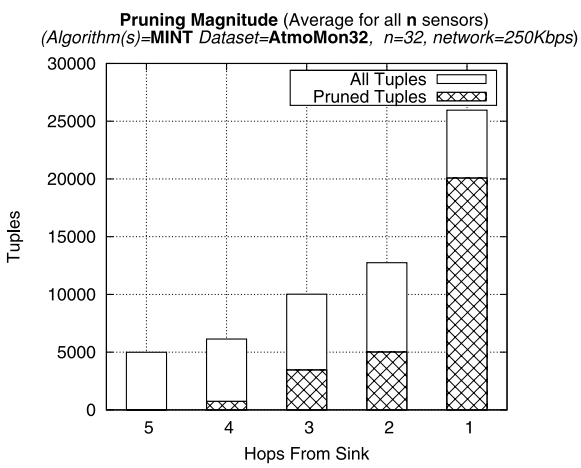

Fig. 15 Pruning Magnitude of MINT Views on the AtmoMon32 dataset

cantly the TAG and TINA algorithms. This means that the pruning mechanism of INT Views, significantly decreases the packet sizes thus minimizing energy consumption associated with transmission.

The results for all experiments are summarized in Table 3.

6.2 Experimental series 2: pruning magnitude

We next study the pruning magnitude of the k-Covered Bound-Set V_i' using the AtmoMon32 dataset. In Fig. 15 we plot with a white box the average number of tuples at each level of the topology (for all 1000 time instances). We also plot with a dashed box the aggregate number of tuples eliminated by Algorithm 2.

We observe that the closer we move towards the sink, the pruning power of our framework increases exponentially. This is attributed to the fact that the cardinality of V_i can increase in the worst case exponentially as well (i.e., each sensor reports a different room number). In particular, we observe that the pruning at level five to one ranges from 0% (where only leaf nodes exist), to 39% in level two and 77% in level one. It is important to highlight the fact that such a pruning presents a reduction of more than 20,000 tuples at level one alone.

A final remark is that these results apply to both MINT and INT, as these two algorithms only differ in how V_i' is maintained and not on the final content of the in-network view.

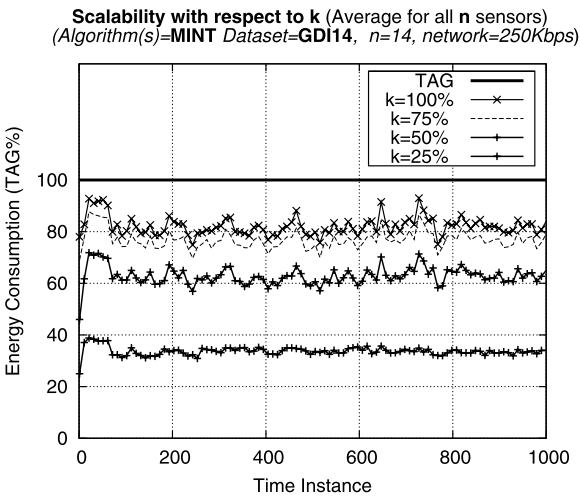

Fig. 16 Scalability with respect to k: In the worst case scenario ($k = 100\%$), the MINT algorithm maintains a competitive advantage over TAG

6.3 Experimental series 3: scalability with respect to k

In the third experimental series, we evaluate the efficiency of the MINT algorithm with respect to the parameter k. More specifically, we increase the parameter k while maintaining the same network topology. We expect that by increasing the k parameter, packet sizes will also increase as less packets will be pruned from V_i. We utilize the GDI14 dataset for this experiment and measure the average energy consumption for all sensor nodes. However, we mention that similar observations also do hold for the rest datasets.

Figure 16 shows the result of our experiment. For the lowest value of k ($k = 25\%$ of the answer set) the overall energy consumption is 66% less than TAG (19 ± 0.5 J). We observe that as the value of the k parameter increases, the performance gain is decreased. Particularly, for $k = 50\%$ and $k = 75\%$ the energy performance ratio compared to TAG reaches 36.5% (36 ± 1.6 J) and 23,4% (44 ± 1.5 J) respectively. This is expected as the number of results transmitted from each sensor node is correlated with the k parameter (i.e., higher values of k decrease the number of tuples eliminated from V_i). When the k parameter reaches 100% (i.e., all sensor nodes transmit all of their results), then the MINT Views algorithm behaves identically to the TINA algorithm. More specifically, since no pruning occurs on the sensors, the MINT Views algorithm only exploits temporal coherence exactly like the TINA algorithm. However, like TINA, MINT still maintains a competitive advantage of 18% (47 ± 1.59 J) decreased energy consumption over TAG (57 ± 2.52 J).

6.4 Experimental series 4: cardinality of the GROUP-BY clause

In the fourth experimental series, we evaluate the efficiency of the MINT Views algorithm with respect to the cardinality of the GROUP-BY clause (i.e., the number of rooms that participate in the given query). More specifically, we manually set the number of rooms (R) to 2, 4 and 7 in the GDI14 dataset and uniformly distribute

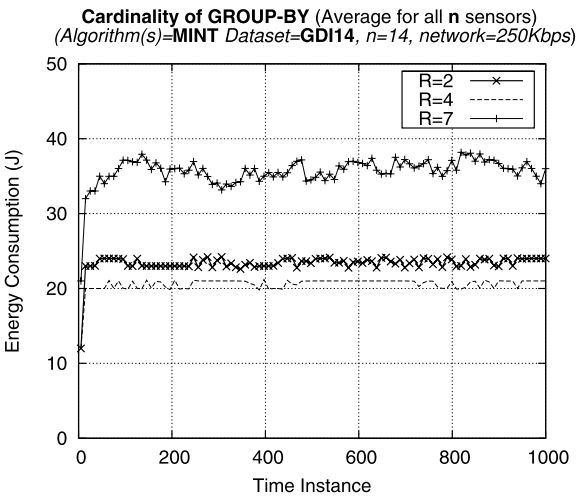

Fig. 17 Cardinality of the GROUP BY clause for 3 different room configurations ($R =$ Number of Rooms)

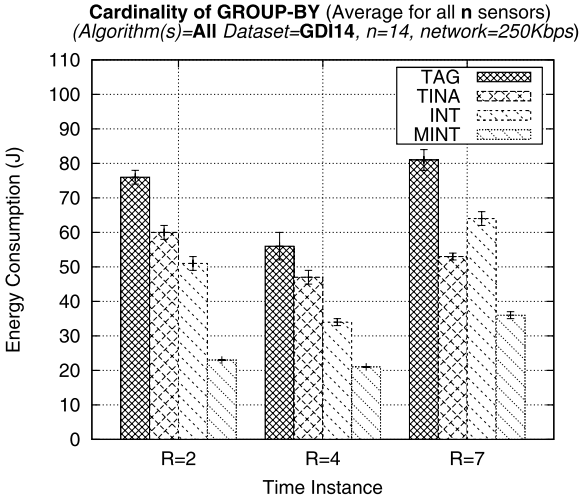

Fig. 18 Cardinality of the GROUP BY clause for all algorithms

the sensors in each room respectively. We measure the average energy consumption for all 14 sensor nodes. There are two important parameters that affect the cardinality attribute. Firstly, when R increases, so is the packet size, as the TopKRoom data structure allocates space to store R results. On the other hand, since a smaller number of sensors is distributed in each room, lower-level nodes can quickly calculate the exact result of a room thus the pruning magnitude is increased. Secondly, when R decreases the packet size also decreases for the same aforementioned reason. However, in this case the pruning magnitude rapidly decreases as only higher-level sensor nodes have a complete picture of the exact result for a room.

Figure 17 shows the first result of our experiment. We observe that the best energy performance occurs when the number of rooms R is equal to 4 (i.e., 21 ± 0.7 J). In the case of fewer rooms (i.e., $R = 2$) we observe that the energy consumption is slightly

increased although in this case the data payload of MINT becomes almost half the size. The reason for this increase, is the fact that the MINT pruning phase almost never prunes the V_i' structure on sensor nodes that have a hop count greater than 1 (i.e., the results have to reach nodes very close to the sink in order for a node to be able to eliminate tuples). On the other case, where $R = 7$, we observe a significant increase in energy consumption. This is because the data payload is now configured to store 7 tuples at each sensor which requires almost double overall transmission energy. However, in this case the pruning mechanism of MINT eliminates tuples at lower levels of the network topology and that is why the standard deviation of this experiment increases (i.e., $36 \pm 1{,}68$ J).

Figure 18 presents the results of all algorithms on the GDI14 dataset with different cardinalities. We have found that MINT always maintains an advantage against its competitors in all scenarios. In the case where $R = 7$, we observe that TINA presents better performance than INT. This is attributed to the fact that TINA suppresses many results from being transmitted to the network, due to its temporal coherence filter. On the other hand, MINT which employs both top-k pruning and the temporal coherence filter outperforms all algorithms.

6.5 Experimental series 5: network lifetime

In the fifth experimental series we evaluate an extremely important parameter of sensor networks deployments, i.e., Network Lifetime. We define network lifetime as the average amount of energy in the network. In particular, let the following summation denote the amount of energy that is available at time instance t in a network of n sensors:

$$Energy(t) = \sum_{i=1}^{n} available\_energy(s_i, t)/n$$

where $available\_energy(s_i, t)$ denotes the energy that is available at sensor s_i ($i \leq n$) at time instance t. We define the *network lifetime*, similarly to [54], as the time instance t' at which $Energy(t') = 0$. Note that this applies only to the case where sensors operate using batteries. Double batteries (AA) used in many current sensor designs (including the TelosB sensor) operate at 3 V voltage and supply a current of 2,500 mAh (milliAmper per hour). Assuming similarly to [52], that only 2,200 mAh is available, we can calculate that 2xAA batteries offer 23,760 J (2,200 mAh $\times$ 60 min $\times$ 60 s $\times$ 3 V). We start with this initial energy and subtract at each epoch and for each sensor the energy required for processing the top-k query. We terminate this iteration when the termination condition is satisfied.

Figure 19 illustrates the average energy status of the sensor network, at each epoch, during the execution of a query using the GDI14 dataset. We notice that the available energy of sensors under TAG is consumed far faster than the MINT Views algorithm, leading to a lifetime of just 5,793 epochs (i.e., 193 minutes). TINA ranks third by offering 6,949 epochs (i.e., 231 minutes) and INT second with 9,768 epochs (i.e., 325 minutes). Finally, MINT consumes its available energy budget far later at epoch 43,824 (i.e., 565 minutes), and this is translated into a $\approx$292% increase of the network lifetime.

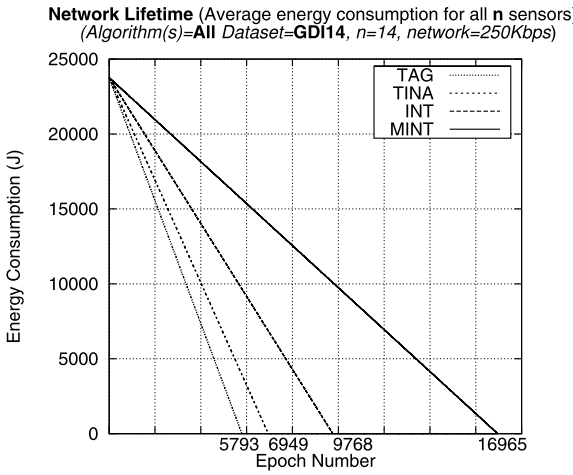

Fig. 19 Network lifetime for the TAG, TINA, INT and MINT Views algorithms presented in this paper

6.6 Experimental series 6: CC2420 receiver microbenchmark

During the execution of a continuous query on a multi-hop topology scenario, each sensor device will periodically (i.e., at each epoch) receive the results from its child sensors and transmit its results to its parent as previously described in Section 3. This process can be performed in two ways: (i) each sensor device can operate with its transceiver in continuous `STXON/SRXON` operation, or (ii) each sensor may choose to turn off its transceiver as soon as it obtains the results of its child sensors and power it on again at the beginning of the next epoch. We argue that the latter case imposes an additional overhead with regards to energy consumption on the sensor device and should be avoided.

In order to verify our argument, we have conducted one micro-benchmark on the CC2420 radio chip [53] (both attached to the TelosB [17] sensor and in TOSSIM [37]) and justify why data reception inefficiencies have to be optimized in current data acquisition systems. Specifically, we show why a sensor node should not change the state of its transceiver continuously as this increases the overall energy consumption on the sensor node.

In our experiment we transfer 1000 16-byte packets from a TelosB sensing device A to another TelosB sensing device B and measure the energy consumption of sensor A when this transfer is conducted in 1, 10, 100 and 1000 rounds respectively. In particular, we configure sensor B with an always-on transceiver and sensor A with a transceiver that changes its state from `on` (`STXON/SRXON`) to `off` (`SRFOFF`), 1 to 1000 times respectively. In order to measure the energy consumed by sensor A for the above function we utilized a multi-meter, to measure the circuit current, and we also measured the wall clock time until the given operations completed successfully.

Figure 20 shows the result of the first micro-benchmark. We observe that by changing the transceiver status 1000 times consumes 195 μJ while conducting the same operation one time requires only 128 μJ. Although in both cases we transfer precisely the same amount of data, in the former case we spent 65% more energy. This increase occurs even though the CC2420 transceiver has very quick start-up times compared

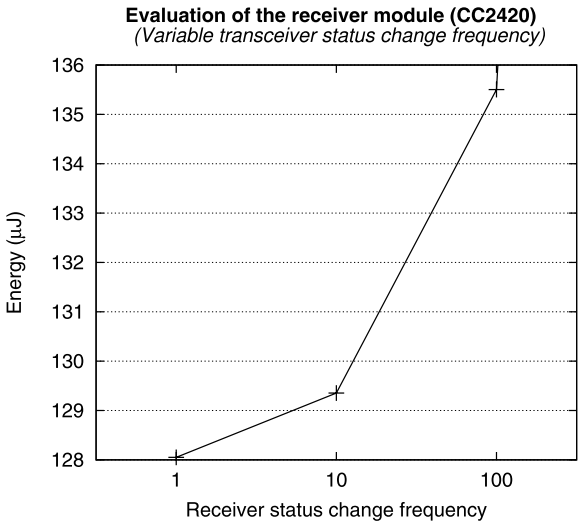

Fig. 20 Micro-benchmark using the CC2420 communication module. Changing the transceiver status from on to off many times significantly increases energy consumption

to other transceivers. Notice that during the startup of the RF module, a voltage regulator and crystal oscillator have to be started as well as become stable [53]. Thus, it is quite inefficient to change the transceiver state (from on to off and vice-versa) in a frequent manner.

Due to inherent limitations of TinyDB, all algorithms presented in the previous experimental sections were set to continuous transceiver on operation. Consequently, all experiments featured an increased energy consumption, but in reality this energy cost could have been reduced significantly had we optimized this parameter with techniques like [2, 64].

7 Related work

In this section we will overview related research efforts that relate to the foundations of the KSpot Architecture, i.e., (i) View management and (ii) Top-K query processing.

View management has been an area of great contributions over the last decades [7, 11, 13, 34]. Materialized Views, in particular, have been extremely important in OLAP and Data Warehousing, where users are required to get quick answers to their aggregate queries over extremely large datasets. Most of the proposed solutions assume powerful and complex centralized or distributed DBMSes. Materialized views have also been extremely important in mobile databases because they provided the means to support disconnected operations [56, 57]. Similarly to mobile databases, we focus on wireless (sensor) devices with limited energy, CPU and memory resources. Additionally, our work is fundamentally different from Temporal View Management [41, 61], as our queries are not historic.

The notion of views in the context of sensor networks, has appeared in three recent works. The first one proposes a new abstraction, coined *Model-based Views* which

provides users with a unified view of data that hides away the irregularities of sensor data [20]. These views are implemented outside the sensor network. Thus, their scope and objective is supplementary to our approach, in which we utilize in-network views to optimize the acquisition of data from sensing devices. The second work [59] is similar to our approach but it uses in-network views to support ad-hoc queries in a data-centric environment as opposed to continuous and top-k queries in our approach. Finally, in [35] the authors present two cluster-based techniques for materializing aggregated results in a sensor network. The proposed MINV framework replicates aggregated results on some or all sensor nodes inside a cluster and then uses these results as materialized in-network views in order to speed-up the execution of spatial aggregation queries. The proposed cluster-based techniques in [35] can be used in conjunction with the INT and MINT Views algorithms of the KSpot framework in order to further speed-up query execution as well as to improve the overall fault tolerance of the system since with MINV, local sensor results are available to other sensor nodes.

The problem of materialized views that are generated by top-k queries in a centralized DBMS scenario was recently addressed in [18]. In particular, the authors study the problem of answering a top-k query from a set of N materialized top-k answers. These answers refer to different top-k queries which are neither distributed nor organized in a hierarchy, as this is the case in our setting. Finally in [36], the authors propose to exploit fully materialized views in sensor networks in order to speedup the execution of multiple queries. However these views are complete, rather than top-k, therefore their setting is closer to the TINA framework rather than the solutions proposed in this paper.

Top-k query processing has been studied in a variety of contexts including middleware systems [24, 25], web accessible databases [8, 43], stream processors [5], peer-to-peer systems [4] and other distributed systems [9, 66, 67]. It has been shown in numerous studies [8, 9, 24, 67], that top-k query processing is meaningful only if the predicate k refers to a small subset of the complete answer set (usually up-to 5%). For larger values of k, the query optimizer can choose to retrieve the complete answer set. For instance the query "*Find the* $k = 5$ *rooms with the highest average temperature*," returns a subset of the complete answer set in order to minimize a cost metric that is associated with the retrieval of the complete answer set. This cost is usually measured in terms of disk accesses or network transmissions, depending on where the data physically resides.

Distributed Top-k query processing algorithms can be classified according to the approach in which the data are fragmented over the network, that is vertically or horizontally. In vertically fragmented datasets, each sub-relation contains a subset of columns (attributes) of the original relation R. An example of a query that can be executed on a vertically partitioned dataset is "*Find the timestamp on which we had the highest temperature across all sensors*". Various algorithms [9, 24, 62, 66] have been proposed for top-k query processing with the Threshold Algorithm (TA)[24] being the most predominant. In [9] the authors develop a three phase protocol (TPUT) which decreases the number of remote accesses in large networks. The TPAT algorithm [62] extends the TPUT algorithm by exploiting data distributions among nodes

to improve pruning. In [66], the authors propose the Threshold Join Algorithm which operates on a multi-hop network (in contrast with TA,TPUT and TPAT) and further reduces communication by exploiting in-network aggregation.

While the aforementioned algorithms provide exact results for top-k queries there is a number of works [5, 44, 66] that provide approximate answers. In [66], the authors propose the UB-K and UBLB-K algorithms that return upper/lower bounds instead of exact answers. In [5] the authors use a centralized coordinator node which distributes filters to each source node so as to ensure that local top-k result correlate with the global top-k answer. In [44], the authors propose the KLEE algorithm which extends the TPUT algorithm by providing approximate answers. The idea is to provide an adaptive framework which allows trading-off efficiency against result quality and bandwidth saving against the number of communication messages. A sampling-based approach to optimize Top-k queries in sensor networks is also the core topic in [50].

In horizontally fragmented datasets, each sub-relation contains a subset of tuples (rows) of the original relation R. An example of a query that can be executed on a vertically partitioned dataset is "*Find the two rooms with the highest average temperature*".

A method for continually providing approximate answers in a hierarchical sensor network scenario by exploiting temporal coherency was addressed in *TINA* [47, 48]. The basic idea behind TINA is to send a reading from a sensor only if the reading differs from the last recorded reading by more than a stated tolerance ϵ. The problem of continually providing approximate top-k answers in a client-server setting was studied in [5]. The problem is tackled by installing arithmetic constraints at each node which define the current Top-k scores at any point. This work was later extended to a hierarchical sensor network environment in [19].

In [58] the authors propose a range caching algorithm for continuous top-k processing. This approach utilizes $k + 1$ individual filters that are selectively adapted rather than a hierarchical in-network pruning mechanism. In [4, 32], the problem of identifying the Top-k objects from relations which are horizontally fragmented over peers in a P2P environment is studied. The proposed solution depends on each peer having knowledge of the total score of each object that it manipulates. This is not possible for vertically partitioned relations, as this requires access to all relations in their entirety, which constitutes their approach inapplicable in our context.

Finally, most of the above horizontal approaches assume a star (or single-hop) communication topology, in which all nodes are directly accessible by the querying entity. On the other hand, our work has focused on the challenges of a hierarchical (or multi-hop) topology. In all cases the results are approximate and continuous over a single attribute, thus operate over individual attributes (columns), while our approach is exact and operates horizontally covering all tuple attributes.

8 Conclusions and future work

In this paper, we present an innovative framework for efficiently monitoring WSNs, to achieve both efficiency and query result quality. Our framework, coined *KSpot*,

utilizes a novel top-k query processing algorithm we developed, in conjunction with the concept of in-network views, in order to minimize the cost of query execution. In particular, we formulate the problem of constructing a hierarchy of recursively defined top-k views. We then describe the MINT Views algorithm that identifies the K highest-ranked answers efficiently. We also present a stateless, non-materialized version of MINT, coined *INT* (In-Network Top-k) Views that is appropriate for sensing device with limited memory.

To illustrate the efficiency of our framework, we have implemented a real system in nesC, which combines the traditional advantages of declarative acquisition frameworks, like TinyDB, with the ideas presented in this work. Extensive real-world testing and experimentation with traces from UC-Berkeley, University of Washington and Intel Research Berkeley, show that KSpot presents an up to 66% of energy savings compared to TinyDB, minimizes both the size and number of packets transmitted onto the network (up to 77%), prolonging in that way the longevity and health of a WSN deployment.

In the future we plan to incorporate an automated transceiver operation module that will automatically tune the waking window of each sensor device using application layer semantics [2, 64]. Additionally, we plan to investigate the applicability of these ideas over Mobile Sensor Networks and Networks of Smartphone Devices.

Acknowledgements We would like to thank Joe Polastre (University of California–Berkeley) for the Great Duck Island data trace and Panayiota Gianni (University of Cyprus) for assisting with the experimental evaluation. This work was supported in part the second author's Startup Grant, funded by the University of Cyprus between 2010–2011, the Open University of Cyprus under project SenseView, the US National Science Foundation under projects S-CITI (#ANI-0325353) and AQSIOS (#IIS-0534531), the European Union under the projects IPAC (#224395) and CONET (#224053), and the project FireWatch (#0609-BIE/09), sponsored by the Cyprus Research Promotion foundation.

References

1. Agrawal, D., Ganesan, D., Sitaraman, R.K., Diao, Y., Singh, S.: Lazy-adaptive tree: an optimized index structure for flash devices. Proc. VLDB Endow. **2**(1), 361–372 (2009)
2. Andreou, P., Zeinalipour-Yazti, D., Chrysanthis, P.K., Samaras, G.: Workload-aware query routing trees in wireless sensor networks. In: Proceedings of the 9th International Conference on Mobile Data Management (MDM'08), Beijing, China, April 27–30, pp. 189–196 (2008)
3. Andreou, P., Zeinalipour-Yazti, D., Vassiliadou, M., Chrysanthis, P.K., Samaras, G.: KSpot: effectively monitoring the k most important events in a wireless sensor network. In: Proceedings of the 25th International Conference on Data Engineering (ICDE'09), Shanghai, China, May 29–April 4, pp. 1503–1506 (2009)
4. Balke, W.-T., Nejdl, W., Siberski, W., Thaden, U.: Progressive distributed top-k retrieval in peer-to-peer networks. In: Proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, April 5–8, pp. 174–185 (2005)
5. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, California, USA, June 9–12, pp. 28–39 (2003)
6. Benenson, Z., Bestehorn, M., Buchmann, E., Freiling, F.C., Jawurek, M.: Query dissemination with predictable reachability and energy usage in sensor networks. In: Proceedings of the 7th International Conference on Ad-hoc, Mobile and Wireless Networks (ADHOC-NOW'08), Sophia-Antipolis, France, September 10–12, pp. 279–292 (2008)
7. Blakeley, J., Larson, P.A., Tompa, F.W.: Efficiently updating materialized views. In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data (SIGMOD'86), Washington, D.C., USA, May 28–30, pp. 61–71 (1986)

8. Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web accessible databases. In: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), San Jose, California, USA, February 26–March 1, pp. 369–382 (2002)
9. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC'04), St. John's, Newfoundland, Canada, July 25–28, pp. 206–215 (2004)
10. Cao, Q., Abdelzaher, T., Stankovic, J., He, T.: The LiteOS operating system: towards unix-like abstractions for wireless sensor networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN'08), St. Louis, Missouri, USA, April 22–24, pp. 233–244 (2008)
11. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: Proceedings of the 11th International Conference on Data Engineering (ICDE'95), Taipei, Taiwan, March 6–10, pp. 190–200 (1995)
12. Chaves, L.W.F., Buchmann, E., Hueske, F., Bohm, K.: Towards materialized view selection for distributed databases. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09), Saint Petersburg, Russia, March 23–26, pp. 1088–1099 (2009)
13. Colby, L.S., Griffin, T., Libkin, L., Mumick, I.S., Trickey, H.: Algorithms for deferred view maintenance. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96), Montreal, Quebec, Canada, June 4–6, pp. 469–480 (1996)
14. Coman, A., Nascimento, M.A.: A distributed algorithm for joins in sensor networks. In: Proceedings of the 19th International Conference on Scientific and Statistical Database (SSDBM '07), Banff, Canada, July 9–11, p. 27 (2007)
15. Coman, A., Sander, J., Nascimento, M.A.: Adaptive processing of historical spatial range queries in peer-to-peer sensor networks. Distrib. Parallel Databases **22**(3), 133–163 (2007)
16. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: Proceedings of the 20th International Conference on Data Engineering (ICDE'04), Boston, MA, USA, March 30–April 2, pp. 449–460 (2004)
17. Crossbow Technology Inc.: http://www.xbow.com/ (2010)
18. Das, G., Gunopulos, D., Koudas, N., Tsirogiannis, D.: Answering top-k queries using views. In: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06), Seoul, Korea, September 12–15, pp. 451–462 (2006)
19. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing historical information in sensor networks. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04), Paris, France, June 13–18, pp. 527–538 (2004)
20. Deshpande, A., Madden, S.R.: MauveDB: supporting model-based user views in database systems. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD'06), Chicago, Illinois, USA, June 26–29, pp. 73–84 (2006)
21. Diao, Y., Ganesan, D., Mathur, G., Shenoy, P.: Rethinking data management for storagecentric sensor networks. In: Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR'07), Asilomar, California, USA, January 7–10, pp. 22–31 (2007)
22. Dunkels, A., Gronvall, B., Voigt, T.: Contiki—a lightweight and flexible operating system for tiny networked sensors. In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04), Tampa, Florida, USA, November 16–18, pp. 455–462 (2004)
23. Earth Climate and Weather: University of Washington. http://www-k12.atmos.washington.edu/k12/grayskies/ (2010)
24. Fagin, R.: Combining fuzzy information from multiple systems. J. Comput. Syst. Sci. **58**(1), 83–99 (1999)
25. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'01), Santa Barbara, California, USA, May 21–23, pp. 102–113 (2001)
26. Galpin, I., Brenninkmeijer, C.Y.A., Jabeen, F., Fernandes, A.A.A., Paton, N.W.: Comprehensive optimization of declarative sensor network queries. In: Proceedings of the 21st International Conference on Scientific and Statistical Database Management (SSDBM'09), New Orleans, Louisiana, USA, June 2–4, pp. 339–360 (2009)
27. Gay, D., Levis, P., Von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesC language: a holistic approach to networked embedded systems. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03), San Diego, California, USA, June 9–11, pp. 1–11 (2003)

28. Gu, Y., Lo, A., Niemegeers, I.: A survey of indoor positioning systems for wireless personal networks. IEEE Commun. Surv. Tutor. **11**(1), 13–32 (2009)
29. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. ACM SIGPLAN Not. **35**(11), 93–104 (2000)
30. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM'00), Boston, Massachusetts, USA, August 6–11, pp. 56–67 (2000)
31. Intel Lab Data: http://db.csail.mit.edu/labdata/labdata.html (2010)
32. Kalnis, P., Ng, W.-S., Ooi, B.-C., Tan, K.-L.: Answering similarity queries in peer-to-peer networks. In: Proceedings of the 13th International World Wide Web Conference (WWW'04), New York City, NY, USA, May 19–21, pp. 482–483 (2004)
33. Klan, D., Hose, K., Sattler, K.-U.: Developing and deploying sensor network applications with AnduIN. In: Proceedings of the 6th Workshop on Data Management for Sensor Networks (DMSN'09), Lyon, France, August 24, No. 11 (2009)
34. Larson, P.-A., Yang, H.Z.: Computing queries from derived relations. In: Proceedings of the 11th International Conference on Very Large Data Bases (VLDB'85), Stockholm, Sweden, August 21–23, pp. 259–269 (1985)
35. Lee, C.K., Zheng, B., Lee, W.-C., Winter, J.: Materialized in-network view for spatial aggregation queries in wireless sensor network. ISPRS J. Photogramm. Remote Sens. **62**(5), 382402 (2007)
36. Lee, K.C.K., Lee, W.-C., Zheng, B., Winter, J.: Processing multiple aggregation queries in geo-sensor networks. In: Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA'06), Singapore, April 12–15, pp. 20–34 (2006)
37. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03), Los Angeles, California, USA, November 5–7, pp. 126–137 (2003)
38. Li, Q., Beaver, J., Amer, A., Chrysanthis, P.K., Labrinidis, A.: Multi-criteria routing in wireless sensor-based pervasive environments. J. Pervasive Comput. Commun. (JPCC'05) **1**(4), 313–326 (2005)
39. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TAG: a tiny aggregation service for ad-hoc sensor networks. ACM SIGoPS Oper. Syst. Rev. **36**(SI), 131–146 (2002). Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)
40. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, California, USA, June 9–12, pp. 491–502 (2003)
41. Maiocchi, R., Pernici, B.: Temporal data management systems: a comparative view. IEEE Trans. Knowl. Data Eng. (TKDE'91) **3**(4), 504–524 (1991)
42. Malhotra, B., Nascimento, M.A., Nikolaidis, I.: Better tree—better fruits: using dominating set trees for MAX queries. In: Proceedings of the 5th Workshop on Data Management for Sensor Networks (DMSN'08), Auckland, New Zealand, August 24, pp. 1–7 (2008)
43. Marian, A., Gravano, L., Bruno, N.: Evaluating top-k queries over web-accessible databases. ACM Trans. Database Syst. (TODS'04) **29**(2), 319–362 (2004)
44. Michel, S., Triantafillou, P., Weikum, G.: KLEE: a framework for distributed top-k query algorithms. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), Trondheim, Norway, August 30–September 2, pp. 637–648 (2005)
45. Polastre, J., Szewczyk, R., Culler, D.E.: TELOS: enabling ultra-low power wireless research. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05), Los Angeles, California, USA, April 25–27, pp. 364–369 (2005)
46. Sadler, C., Zhang, P., Martonosi, M., Lyon, S.: Hardware design experiences in zebraNet. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, Maryland, USA, November 3–5, pp. 227–238 (2004)
47. Sharaf, M.A., Beaver, J., Labrinidis, A., Chrysanthis, P.K.: TiNA: a scheme for temporal coherency-aware in-network aggregation. In: Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe'03), San Diego, California, USA, September 19, pp. 69–76 (2003)
48. Sharaf, M.A., Beaver, J., Labrinidis, A., Chrysanthis, P.K.: Balancing energy efficiency and quality of aggregate data in sensor networks. Int. J. Very Large Data Bases (VLDBJ'04) **13**(4), 384–403 (2004)

49. Shnayder, V., Hempstead, M., Chen, B., Werner-Allen, G., Welsh, M.: Simulating the power consumption of large-scale sensor network applications. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, MD, USA, November 3–5, pp. 188–200 (2004)
50. Silberstein, A., Braynard, R., Ellis, C., Munagala, K., Yang, J.: A sampling-based approach to optimizing top-k queries in sensor networks. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, Georgia, USA, April 3–8, p. 68 (2006)
51. Stern, M., Buchmann, E., Bohm, K.: Towards efficient processing of general-purpose joins in sensor networks. In: Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE'09), Shanghai, China, March 29–April 2, pp. 126–137 (2009)
52. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04), Baltimore, Maryland, USA, November 3–5, pp. 214–226. (2004)
53. Texas Instruments: CC2420, single-chip 2.4 GHz IEEE 802.15.4 compliant and ZigBee(TM) ready RF transceiver. Texas Instrument Document. http://www.ti.com/lit/gpn/cc2420 (2007)
54. Thomas, H., Yi, S., Sherali, H.D.: Rate allocation in wireless sensor networks with network lifetime requirement. In: Proceedings of the 5th ACM International Symposium on Mobile ad hoc Networking and Computing (MobiHoc'04), Tokyo, Japan, May 24–26, pp. 67–77 (2004)
55. Voltree Power Inc.: http://www.voltreepower.com/ (2010)
56. Weissman-Lauzac, S., Chrysanthis, P.K.: Personalizing information gathering for mobile database clients. In: Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'02), Madrid, Spain, March 11–14, pp. 49–56 (2002)
57. Weissman-Lauzac, S., Chrysanthis, P.K.: Utilizing versions of views within a mobile environment. In: Proceedings of the International Conference on Computing and Information (ICCI'98), Winnipeg, Manitoba, Canada, June 17–20, pp. 201–208 (1998)
58. Wu, M., Xu, J., Tang, X., Lee, W.-C.: Top-k monitoring in wireless sensor networks. IEEE Trans. Knowl. Data Eng. **19**(7), 962–976 (2007)
59. Xia, P., Chrysanthis, P.K., Labrinidis, A.: Similarity-aware query processing in sensor networks. In: Proceedings of the 14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'06), Island of Rhodes, Greece, April 25–26, p. 8 (2006)
60. Yao, Y., Gehrke, J.E.: The cougar approach to in-network query processing in sensor networks. ACM SIGMOD Rec. (SIGMOD'02) **31**(3), 9–18 (2002)
61. Yang, J., Widom, J.: Maintaining temporal views over non-temporal information sources for data warehousing. In: Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'98), Valencia, Spain, March 23–27, pp. 389–403 (1998)
62. Yu, H., Li, H., Wu, P., Agrawal, D., Abbadi, A.E.: Efficient processing of distributed top-k queries. In: Proceedings of the 16th International Conference on Database and Expert Systems (DEXA'05), Copenhagen, Denmark, August 22–26, pp. 65–74 (2005)
63. Zeinalipour-Yazti, D., Andreou, P., Chrysanthis, P.K., Samaras, G.: MINT views: materialized in network top-k views in sensor networks. In: Proceedings of the 8th International Conference on Mobile Data Management (MDM'07), Mannheim, Germany, May 7–11, pp. 182–189 (2007)
64. Zeinalipour-Yazti, D., Andreou, P., Chrysanthis, P.K., Samaras, G., Pitsillides, A.: The MicroPulse framework for adaptive waking windows in sensor networks. In: Proceedings of the 1st International Workshop on Data Intensive Sensor Networks (DISN'07), Mannheim, Germany, May 11, pp. 351–355 (2007)
65. Zeinalipour-Yazti, D., Lin, S., Kalogeraki, V., Gunopulos, D., Najjar, W.: MicroHash: an efficient index structure for flash-based sensor devices. In: Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05), San Francisco, California, USA, December 13–16, pp. 31–44 (2005)
66. Zeinalipour-Yazti, D., Lin, S., Gunopulos, D.: Distributed spatio-temporal similarity search. In: Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM'06), Arlington, VA, USA, November 6–11, pp. 14–23 (2006)
67. Zeinalipour-Yazti, D., Vagena, Z., Gunopulos, D., Kalogeraki, V., Tsotras, V., Vlachos, M., Koudas, N., Srivastava, D.: The threshold join algorithm for top-k queries in distributed sensor networks. In: Proceedings of the 2nd International Workshop on Data Management for Sensor Networks (DMSN'05), Trondheim, Norway, August 29, pp. 61–66 (2005)
68. ZigBee Alliance: ZigBee specification. ZigBee Document 053474r06, Version 1.0 (2004)