A Network-aware Framework for Energy-efficient Data Acquisition in Wireless Sensor Networks

Panayiotis G. Andreou[a], Demetrios Zeinalipour-Yazti[a,*], George S. Samaras[a], Panos K. Chrysanthis[b]

[a]*Department of Computer Science, University of Cyprus, Nicosia, Cyprus.*
[b]*Department of Computer Science, University of Pittsburgh, USA.*

Abstract

Wireless Sensor Networks enable users to monitor the physical world at an extremely high fidelity. In order to collect the data generated by these tiny-scale devices, the data management community has proposed the utilization of declarative data-acquisition frameworks. While these frameworks have facilitated the energy-efficient retrieval of data from the physical environment, they were agnostic of the underlying network topology and also did not support advanced query processing semantics. In this paper we present KSpot$^+$, a distributed network-aware framework that optimizes network efficiency by combining three components: i) the *tree balancing module*, which balances the workload of each sensor node by constructing efficient network topologies; ii) the *workload balancing module*, which minimizes data reception inefficiencies by synchronizing the sensor network activity intervals; and iii) the *query processing module*, which supports advanced query processing semantics. In order to validate the efficiency of our approach, we have developed a prototype implementation of KSpot$^+$ in nesC and JAVA. In our experimental evaluation, we thoroughly assess the performance of KSpot$^+$ using real datasets and show that KSpot$^+$ provides significant energy reductions under a variety of conditions, thus significantly prolonging the longevity of a WSN.

Keywords: Wireless sensor networks, Top-k query processing, Network-awareness.

*D. Zeinalipour-Yazti, Email: dzeina@cs.ucy.ac.cy, Tel/Fax: +357-22-892755/01

Email addresses: panic@cs.ucy.ac.cy (Panayiotis G. Andreou), dzeina@cs.ucy.ac.cy (Demetrios Zeinalipour-Yazti), cssamara@cs.ucy.ac.cy (George S. Samaras), panos@cs.pitt.edu (Panos K. Chrysanthis)

1. Introduction

Technological advances in embedded systems, sensor components and low power wireless communication units have made it feasible to produce small-scale wireless sensor devices that can be utilized for ad hoc monitoring infrastructures. Large-scale deployments of *wireless sensor networks (WSNs)* have already emerged in environmental and habitat monitoring [35, 28], structural monitoring [16] and urban monitoring [27, 5]. To simplify deployment, these systems often employ middleware frameworks that allow users to disseminate queries and collect runtime measurements of sensor data. Since, sensor devices feature a limited energy budget (typically powered using 2xAA batteries [21, 35, 28]), one of the key design goals of any middleware system is power efficiency [34].

This study shows that although predominant data acquisition middleware frameworks [21, 40, 41, 32, 14, 8, 13, 23, 37, 19, 7] have succeeded in decreasing the overall energy consumption of the network by introducing power-aware in-network processing algorithms, they have overlooked the important parameter of the underlying network topology. In particular, most of the approaches establish query dissemination and data acquisition on the premise of *Query Routing Trees (QRTs)* constructed in an ad hoc manner [21, 40, 31] where each sensor selects as its parent the first node from which a query was received. Although, this approach generates an effective routing scheme between sensor nodes, it may prove highly inefficient as it does not provide any performance guarantees (e.g., for the workload incurred on each sensor node.)

More specifically, ad hoc QRTs present two major sources of inefficiencies: i) *data transmission inefficiencies:* QRTs do not provide any guarantee that the query workload will be distributed equally among all sensors. This leads to collisions during data transmissions that represent a major source of energy waste. Consequently, unbalanced trees can severely degrade the network health and efficiency; and ii) *data reception inefficiencies:* QRTs do not define the *waking window* (τ) of a sensing device (i.e., the continuous interval during which a sensor node has to enable its transceiver, collect and aggregate the results from its children, and then forward these results to its own parent.) Consequently, in many cases it is an over-estimate that leads to significant energy waste.

Addressing the aforementioned inefficiencies enables the generation of energy-efficient network topologies that rapidly decrease data reception and transmission inefficiencies. However, additional energy savings can be achieved by closely investigating the query execution process that takes place after generating an efficient topology. Current data acquisition middleware frameworks systems [21, 40,

14, 8, 13, 23, 33, 37] focus on producing a complete result set for a query. Conversely, a number of studies [10**?**] 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 [10] 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.

In this paper we present KSpot$^+$, a data-centric distributed middleware framework that advances the state-of-the-art by incorporating *network-awareness* to the data acquisition process. To accomplish this, KSpot$^+$ combines three basic components:

1. the *workload balancing module*, whose objective is to decrease data reception inefficiencies by automatically tuning the waking window, locally at each sensor without any *a priori* knowledge or user intervention;
2. the *tree balancing module*, whose objective is to decrease data transmission inefficiencies by transforming the initial QRT into a more balanced tree in a distributed manner; and
3. the *query processing module*, whose objective is to decrease the number and size of packets transmitted to the network by facilitating advanced query semantics (e.g., `Top-`k, `Group-By` queries.)

KSpot$^+$ is an open-source middleware framework[1] for WSNs that can be utilized in numerous application domains including environmental monitoring [35, 28], big ephemeral events [39, 24], structural monitoring [16], urban monitoring [27, 5], military/security applications [22, 29, 12], health monitoring [18, 26], etc. We now describe how KSpot$^+$ has been deployed in the context of two projects:

Environmental Monitoring and Emergency Management: Dynamic monitoring of forests and rivers as well as emergency management require the existence of large sensor network deployments that can provide realtime results. They involve hundreds of sensors and actuators deployed to cover thousands of square kilometers of forest areas thus producing huge amounts of data that require ample time to process. An important factor in sustaining such large sensor network deployments is the cost of maintenance associated with battery replacement. There are solutions today that can rapidly reduce this maintenance cost by utilizing alternative

[1] http://kspot.cs.ucy.ac.cy/

Figure 1: KSpot$^+$ is currently deployed as part of the forest fire monitoring system of the FireWatch project at the Lythrodontas forest in Nicosia, Cyprus.

means for energy replenishment including solar panels, bio-harvesting [38], etc. However, in order for these solutions to succeed effectively, the ratio of energy replenishment over energy consumption must be encouraging. KSpot$^+$ decreases energy consumption by minimizing both the size and number of packets, which increases the network's lifespan and reduces maintenance costs.

KSpot$^+$ is currently deployed as part of the forest fire monitoring system of the FireWatch project[2] (see Figure 1), sponsored by the Cyprus Research Promotion foundation and supervised by the Cyprus Department of Forests. The current network deployed at Lythrodontas forest, which is one of the high risk forest areas in Cyprus, consists of 20 MemSic IRIS nodes and is scheduled to gradually grow each year.

Big Ephemeral Events: International events (e.g., FIFA World Cup, World Expo) usually attract millions of participants during a very limited period of time.

[2]The FireWatch project, http://firewatch.cs.ucy.ac.cy/

Figure 2: KSpot$^+$'s prototype implementation deployment during the event "Researcher's Evening" at the Cyprus International Fair in 2009. (Left) Sensor nodes were placed over the pavilions using helium balloons. (Right) Sink station was connected with a laptop computer that projected the pavilions with the highest noise level.

The deployment of smart sensor networks (i.e., sensors, actuators, RFID) in buildings can contribute to improve the visitor's experience by providing the means to easily interact with its surroundings. For example, during these ephemeral big events, affluence-measuring sensors (e.g., sound, proximity) can form logical groups in order to build a compound resource that provides a real-time map of visitors' arrivals at the different pavilions and places and propose visitors an ideal tour, so as to maximize their experience and satisfaction. Additionally, if a crisis situation happens, these compound resources can also help to localize people to rescue. Furthermore, these deployments can be utilized in conjunction with smartphone networks in order to generate opportunistic social networks that form spontaneously according to relationships, which are explicit (e.g., friendship) and/or implicit (e.g., location, energy.)

In this context, we have deployed the KSpot$^+$ framework prototype implementation during the event "Researcher's Evening" at the Cyprus International Fair in 2009. Figure 2 shows two pictures of our deployment. Our objective was to create an acoustic map of the pavilions participating in the exhibition and direct the visitors towards the most popular ones (i.e., the most noisy). This was accomplished by forming logical clusters of the sensor nodes at each pavilion and then measuring the average sound level using the microphone sensor. KSpot$^+$ successfully monitored the pavilions by periodically visualizing the most popular locations (i.e., `Top-3` highest ranked logical groups) every 4s. Additionally, in order to demonstrate the interoperability of the KSpot$^+$ middleware, all acquired

results were also recorded in a local database. Noteworthy was that at the end, the organizing committee of the event requested the data trace for further analysis.

This work is found upon our previous work in [4], where we have presented the outline of our network-aware architecture. In this paper, we present a thorough description of the KSpot$^+$ framework, its features as well as a description of its basic components. KSpot$^+$ features a highly modular design that allows components to function individually or in cooperation according to the requirements of the application, thus allowing application designers to easily integrate new features into the design as well as to experiment under different settings. Additionally, KSpot$^+$ provides an extended SQL query syntax and mechanisms for logical clustering of sensor nodes through attribute-based naming. Finally, through our experiments we have shown that KSpot$^+$ is resilient in the presence of failures and scales linearly with the number of sensors in the network.

Our Contributions More specifically, we make the following contributions:

- We present a detailed description of the KSpot$^+$ framework architecture including insight information on all its components and internal procedures.
- We experimentally validate the efficiency of KSpot$^+$ with an extensive experimental study that utilizes real sensor readings and real datasets from the Department of Atmospheric Sciences at the University of Washington, Intel Research Berkeley and University of California-Berkeley.
- We qualitatively explain the differences and similarities of existing WSN middleware frameworks compared to the KSpot$^+$ framework. To accomplish this, we provide a taxonomy of WSN middleware frameworks along four different dimensions: *power efficiency*, *topology optimization*, *workload optimization* and *top-k query support*.

The remainder of the paper is organized as follows: Section 2 presents the architecture of the KSpot$^+$ framework and Sections 3, 4 and 5 provide a description of its basic components. In Section 6 we present our experimental methodology and in Section 7 the results of our evaluation. Finally, Section 8 performs a qualitative comparison of related middleware system research works with KSpot$^+$ and Section 9 concludes our paper.

2. The KSpot$^+$ framework

KSpot$^+$ is a network-aware framework for WSNs built on top of a diverse set of energy-conscious algorithms. It inserts a profiling layer between the server

and the sensor network that discovers structural and workload inefficiencies and exploits them in order to generate balanced topologies that can be queried in an energy-efficient manner. It has three basic operations: i) to construct balanced network topologies; ii) to tune the waking windows of sensor nodes; and iii) to enable tuple ranking through `Top-`k queries.

In this section, we provide an overview of the KSpot$^+$ framework, its design principles and its basic components.

2.1. Design Goals

In order to build a practical system, we have taken into consideration the following desired properties:

- *Modularity*: Decomposing systems into a number of components that may be mixed and matched in a variety of configurations ensures a high degree of openness and usability. Our framework's architecture design consists of modular components that operate in an energy efficient manner both in isolation and in combination with each other as well as with other protocols.

- *Energy-Efficiency*: Battery-powered WSNs are expected to minimize maintenance cost by lasting for large periods of time without requiring battery replacements [20, 21, 2, 3, 35]. To accomplish this, any software that runs on a sensor device must be designed to operate in an energy-efficient manner. In the KSpot$^+$ framework, each module is founded on the premise of energy-conscious algorithms that minimize energy consumption and increase network longevity.

- *Distributed and Autonomous Behavior*: We focus on fully autonomous and decentralized behavior of KSpot$^+$ client-side components. More specifically, we minimize the maintenance of any global state or data structures at a centralized location and use only local knowledge. In the cases where global information is necessary for completing an operation it is acquired using specialized coordinator components. Note that this occurs only at the initialization of the network topology upon a balancing request or in case of node failures.

- *Scalability*: The network sizes of WSNs are expected to grow substantially in the next few years as the cost for manufacturing sensor devices continuously decreases [1]. Consequently, we consider scalability an extremely desirable property of our framework as it ensures that the performance of

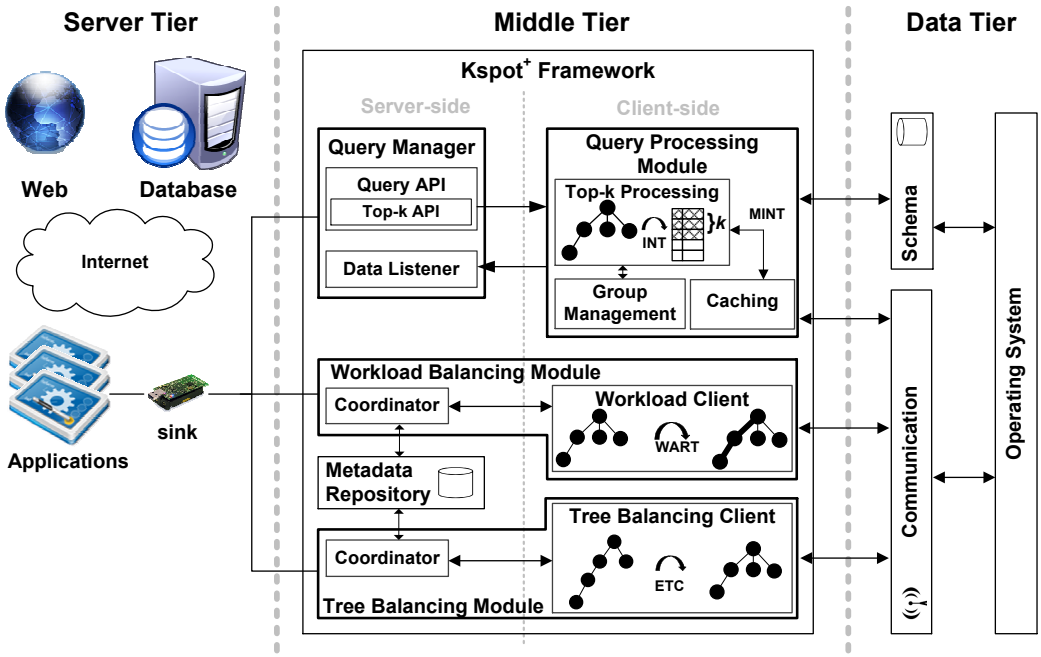

Figure 3: KSpot$^+$ framework architecture. The KSpot$^+$ client combines 3 components: the *tree balancing module*, which balances the sensor network topology, the *workload balancing module*, which balances the workload of each sensor node, and the *query processing module*, which handles query execution and facilitates Top-k query processing.

the system will maintain acceptable QoS standards regardless of the increasing network size. In our experiments, we show that the KSpot$^+$ framework is scalable by utilizing a number of datasets that vary from small-scale to large-scale sensor networks.

- *Failure Resilience*: WSNs are typically prone to imminent node failures triggered by temporary power-downs, malfunctions, environmental causes, etc. Maintaining resilience in such environments is vital for applications (e.g., fire detection/prediction) that require real-time results.

2.2. *KSpot$^+$ Framework Architecture Design*

The KSpot$^+$ framework lies between the server-tier and the data-tier as illustrated in Figure 3. Applications can post queries to the sensor network through the server-side query manager using the query API or request a balancing operation

(tree balancing and/or workload balancing request) through the respective server-side coordinator components. Queries are forwarded to the client-side query processing module, which in turn decides the best execution plan for the query and communicates with the schema layer in order to retrieve the actual data residing on local storage. As soon as the query results are ready, they are forwarded back to the application through the data server-side listener component. Applications can then share the data with online databases and web portals.

Balancing requests require global information, which is stored in the metadata repository. The Coordinator components recursively forward specialized messages to the sensor network requesting the local values. In the next step, these values are propagated in the opposite order until they reach the sink node. The sink node then calculates the critical path (ψ) and the optimal network branching factor (β) values and forwards them back to the coordinator components that proceed with balancing the network topology and each sensor node's workload locally.

We now describe in more detail the components of the KSpot$^+$ framework:

- The *workload balancing module:* (described in detail in Section 3), investigates data reception/transmission inefficiencies that occur from unbalanced assignment of the query workload amongst sensor nodes. It utilizes the *Workload-Aware Routing Tree* (WART) algorithm for the dynamic adaptation of the waking windows locally at each sensor node.

- The *tree balancing module:* (described in detail in Section 4), identifies structural inefficiencies in the initial QRT that occur from its ad hoc construction nature. It utilizes the *Energy-driven Tree Construction* (ETC) algorithm in order to remove these inefficiencies by reconstructing the tree in a balanced manner, which minimizes data collisions during communication.

- The *query manager:* is responsible for disseminating queries to the network and translating the network results into a tuple-format using the *data listener* component. It supports an SQL-like query syntax, which supports standard queries through the *query API*. Additionally, it extends the traditional SQL syntax of predominant data-centric middleware systems [21, 40, 32, 41] by introducing top-k query execution in the form of aggregates through the *top-k query API*. More details on the Query Syntax will be presented in Section 2.4.

- The *query processing module:* (described in detail in Section 5), is responsible for query execution as well as a number of services including group

management (see Section 2.5) and caching. It utilizes the INT algorithm for the execution of Top-k queries. Additionally, it incorporates a data caching mechanism that, in cooperation with INT, exploits temporal coherency between results of consecutive time instances (MINT.)

- The *data caching* component exploits the temporal coherency in order to suppress updates that do not change between consecutive time instances. At each epoch, the query results are stored in main memory before they are transmitted so that they can be compared with the results of the next epoch. We have chosen to store the results in main memory instead of flash storage because it increases the response time performance of the system.

- The *group management* (described in detail in Section 2.5) component is responsible for forming clusters of sensors by arranging them in logical groups. This is accomplished by attribute-based naming of the sensors based on specific query semantics.

We now present selected features of the KSpot$^+$ framework.

2.3. Modular Design

The KSpot$^+$ framework is composed of loosely-coupled modules that communicate with message-passing. In Figure 3, the server-side components communicate with their client-side *accomplices* using different communication messages (different arrows departing from each server-side component.) The reason we have not opted for a unified communication mechanism is that this generates a tightly-coupled system, which would have compromised the modularity of the system as tightly coupled systems tend to exhibit a number of disadvantages including: i) *Decreased Reusability*, because dependent modules must be bundled together in order to be reused or tested; ii) *Increased Deployment Effort*, because module bundles will require more time to test and deploy; and iii) *Increased Maintenance*, as updates on one module may require re-testing of the whole bundle. Nevertheless, under a unified communication setting, we could have performed additional packet-level optimizations that could have decreased the energy requirements for transmission/reception.

The modular design of the KSpot$^+$ architecture allows application designers to easily integrate new features into the design as well as experiment under different settings. Furthermore, KSpot$^+$'s modules can function individually or in

cooperation[3] according to the requirements of the application.

2.4. Query Syntax

The KSpot$^+$ framework supports an SQL-like query syntax, which supports standard queries through the *query API* and `Top-`k queries through the `Top-`k *API*. In particular, the KSpot$^+$ framework utilizes 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* parameter in the `SELECT` statement refers to all measurements that can be acquired from the sensorboard as well as variables stored locally at each sensor node. The *attribute* parameter in the `GROUP BY` statement may additionally refer to a logical group assignment (see Section 2.5.) The *aggregate* parameter refers to all duplicate-insensitive aggregates supported. Roughly, these aggregates can be distinguished in: i) *distributive aggregates*, where records can be aggregated in-network without compromising correctness (e.g., duplicate-insensitive (`MAX, MIN`), duplicate-sensitive (`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. The benefits of the KSpot$^+$ framework are more evident in the case of single-relation queries with distributive aggregate functions. In contrast with other frameworks, we optimize queries with *multi-tuple* answers. Such answers can 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. Furthermore, 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.

2.5. Logical Group Management

The *group management* component realizes clustering of the sensor nodes by arranging them into logical groups. This is necessary in the case of `Group-By`

[3]When operating in cooperation, the operation of the Tree Balancing Module logically precedes the operation of the Workload Balancing Module, as the former reconstructs the network topology which may result in different workload assignments.

queries, where grouping may be achieved not only on predefined attributes (e.g., `nodeid`) but also on context-based attributes (e.g., building name, room number). To facilitate our description, consider an indoor deployment of four sensor nodes s_{1-4} in a building with two offices, A and B, such that s_{1-2} is located in office A and s_{3-4} in B. In order to inform each sensor node on its actual location (e.g., longitude, latitude) and then derive its logical location (i.e., office A or B), we could have utilized absolute localization techniques (e.g., Global Positioning System (GPS)) or relative localization techniques (e.g., RSSI indicators) and then perform the logical mapping on the server. However, this requires specialized hardware (e.g., GPS receiver), which may not be always available and also increases the overall message complexity. To overcome this, the KSpot$^+$ API supports commands for creating and deleting logical groups that are injected to the network and processed locally at the sensor nodes.

2.6. Proof of Concept Application

In order to assess the practicality and usability of the proposed KSpot$^+$ framework, we have developed a proof of concept application (KSpot$^+$ POCA) that demonstrates the full potential of KSpot$^+$ (see Figure 4). KSpot$^+$ POCA components are implemented in JAVA (server-side) and in nesC (client-side). We have selected nesC for the implementation of the client-side components for practical reasons as it provides a kernel of declarative data acquisition functionalities (i.e., SQL query syntax). However, we could have similarly applied our ideas on other sensor network operating systems (e.g., LiteOS [6]).

3. Workload Balancing Module

The Workload Balancing Module investigates data reception/transmission inefficiencies that occur from unbalanced assignment of the query workload amongst sensor nodes. It utilizes the Workload-aware Routing Tree (WART) algorithm for the dynamic adaptation of the waking windows of each sensor node. In particular, the Workload Balancing process consists of three phases: i) *construction phase*: where the sink node constructs a new QRT or utilizes an established one and then queries the network for the total critical path value ψ; ii) *dissemination phase*: where the sink node disseminates the critical path value ψ to the network and each sensor node tunes its waking window accordingly; and iii) *adaptation phase*: where each sensor node adapts its waking window according to new workload variations.

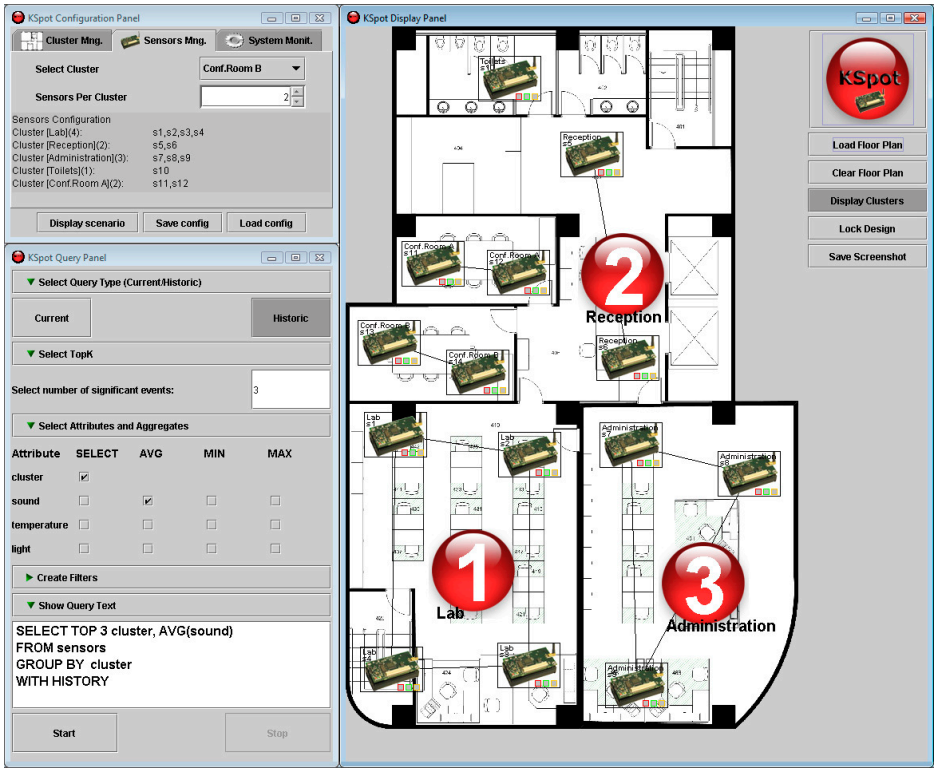

Figure 4: KSpot$^+$'s Graphical User Interface (GUI) allows users to administer the execution of standard and Top-k Queries through an intuitive and declarative graphical user interface. The above scenario conducts a Top-3 query over a 14-node sensor network organized in 6 logical clusters. The Display Panel (on the right) illustrates the three KSpot$^+$-Bullets for the three highest-ranked sensor clusters.

Construction Phase: The first phase of the WART algorithm starts out by having each node select one node as its parent. This results in a *waiting list* similar to Cougar [40]. To accomplish this task, the parent is notified through an explicit acknowledgment or becomes aware of the child's decision by snooping the radio.

In the next step, each sensor profiles the activity of the incoming and outgoing links and propagates this information to the sink. In particular, each sensor s_i executes one round of data acquisition by maintaining one counter for its parent connection (s_i^{out}) and one counter per child s_j connection ($s_{i,j}^{in}$.) These counters account for the *workload* between the respective sensors (i.e., the time required to propagate the query results between them) and are utilized to identify the critical path cost in the subsequent epochs. Note that these counters account for more

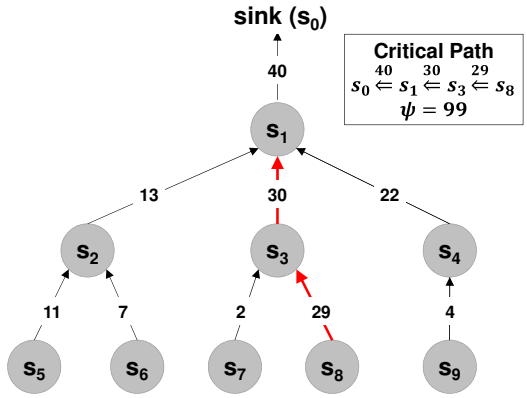

Figure 5: Nine sensing devices and the respective workload between them (shown as edges.) The WART algorithm utilizes this information in order to locally adapt the waking window of each device using the *Critical Path Method*.

time than what is required had we assumed a collision-free MAC channel. By projecting the time costs obtained for each edge to a virtual spanning tree creates a distributed QRT similar to the one depicted in Figure 5.

The final step is to percolate these local edge costs to the sink by recursively executing the following in-network function f at each sensor s_i:

$$f(s_i) = \begin{cases} 0 & \text{if } s_i \text{ is a leaf,} \\ max_{\forall j \in children(s_i)}(f(s_j) + s^{in}_{i,j}) & \text{otherwise.} \end{cases}$$

The critical path cost is then $f(s_0)$ (denoted for brevity as ψ.) Using our working example of Figure 5, we will end up with the following values : $f(s_{5 \leq i \leq 9}) = 0$, $f(s_4) = 4$, $f(s_3) = 29$, $f(s_2) = 11$, $f(s_1) = 59$ and $\psi = f(s_0) = 99$.

Dissemination Phase: In this phase, the critical path cost ψ is propagated top-down, from the sink to the leaf sensors, with a message complexity of $O(n)$. Each sensor s_i locally defines three parameters using ψ that enable it to derive: i) the time instance during which it should wake up (i.e., w_i); ii) the interval during which it should listen for readings and to transmit results (i.e., τ_i); and iii) the workload increase tolerance of the parent of s_i (i.e., λ_i) which signifies when the synchrony of the QRT might be disrupted.

In the first step, a query is aborted when the critical path is larger than the epoch, which signifies an error in the user query. In the second step the wake up time instance w_i is calculated, such that s_i has enough time to collect the tuples from all its children s_j. In practice, this is defined by the child of s_i with the largest

workload (i.e., $s^{in}_{i,maxchild}$.) The second step also defines the waking window of τ_i, which is the complete window during which s_i will enable its transceiver. In the third step, the children of s_i are notified with the adjusted critical path cost (i.e., $\psi - s^{out}_j$.) Furthermore, s_i also notifies its children s_j with the workload increase tolerance of s_i (i.e., λ_i) and a flag which signifies whether these nodes belong to the critical path. Thus, s_j can intelligently schedule its transmissions in cases of local workload deviations.

Adaptation Phase: The Adaptation Phase adapts the WART QRT in cases of workload changes. In the first step, the workload indicators of the current epoch and the previous epoch are calculated. If the workload has changed by more than a user-defined threshold, we consider this change as significant and proceed with the adaptation of the routing tree otherwise, the procedure aborts. A significant deviation has to request the re-construction of the routing tree using the construction and dissemination phases. For instance, if the workload of s_3 changes from 30 time instances to 35 time instances (see Figure 5) then this will trigger the re-construction of the WART QRT and this change should be propagated to all nodes in the network. Although this case is possible, our experimental study in [3] has shown that it is not frequent. Finally, the algorithm handles the more common case where the change does not occur on the critical path. In such a case, if the workload is *decreased* by x then a sensor locally delays its wake up variable by x (i.e., to $w_i + x$.) For instance, if the workload of s_2 drops from 13 to 11 (thus, $x = 2$), then $w^{new}_2 = w_2 + x = 46 + 2 = 48$. Similar adjustments are performed in the case where the workload is *increased*. However, when the change affects the critical path (e.g., s_2's workload increased from 13 to 32 thus, $x = 19$ that is larger than $\lambda_2 = 17$), this yields the re-construction of the tree as such an increase might potentially create a new critical path.

Critical path reconstruction rrequency: One important question that arises is how often to expect changes to the critical path as this might severely degrade the longevity of the network. In [3], we have observed that queries yielding approximately the same amount of results (e.g., single-tuple queries or multi-tuple queries with fixed size) benefit the most from WART's optimization phase. This is expected as the critical path value is only calculated at the start of the execution and continues to be valid until a node or communication failure becomes present. In the event of a node failure, the results of the path rooted at the failed node are not transmitted and therefore the critical path is not affected as the parent node of the failed node will wait for the results for the same amount of time as it would if the node was active. However, in the case of a communication failure, which results in the retransmission of the results by the failed sensor node, the efficiency

of the network may be affected especially if the failed node lies on the critical path. This can also lead to data loss if the node misses the waking window of its parent node. In the case of event-based queries or queries with multi-tuple results of arbitrary size (e.g., filter queries) the critical path reconstruction frequency is increased. This happens because the workload incurred on each sensor node may change rapidly between subsequent epochs. However, even in these cases, the Workload Balancing Module still manages to conserve energy as can be seen by the results of [3].

4. Tree Balancing Module

Although the Workload Balancing Module significantly reduces the energy consumption of the sensors by scheduling communication activities based on the workload, it still does not take into account the fact that the QRT topology might be unbalanced. The Tree Balancing Module identifies structural inefficiencies in the initial QRT that occur from its ad hoc construction nature. It utilizes the *Energy-driven Tree Construction* (ETC) algorithm in order to remove these inefficiencies by reconstructing the tree in a balanced manner, which minimizes data collisions during communication. The ETC algorithm consists of a discovery and distributed balancing step which are described next.

Discovery Phase: The first phase of the ETC algorithm starts out by having each node select one node as its parent. During this phase, each node also records its local depth (i.e., $depth(s_i)$) from the sink. Notice that $depth(s_i)$ can be determined based on a *hops* parameter that is included inside the tree construction request message. A node s_i also maintains a child node list ($children$) and an alternate parent list (APL.) The APL list is constructed locally at each sensor by *snooping* (i.e., monitoring the radio channel while other nodes transmit and recording neighboring nodes) and comes at no extra cost. This list is utilized by the ETC algorithm for parent reassignment during the reconstruction of the QRT, but it can also be used for selecting alternate parents in cases of failures. The sink then queries the network for the total number of sensors n and the maximum depth of the routing tree d. Such a query can be completed with a message complexity of $O(n)$. When variables n and d are received, the sink calculates, the *optimal branching factor* ($\beta = \sqrt[d]{n}$.)

Balancing phase: The Balancing Phase of the ETC algorithm involves the top-down reorganization of the QRT such that this tree becomes near-balanced. In particular, the sink disseminates the β value to the n nodes using the reverse acquisition tree. When a node s_i receives the β value from its parent s_p it initiates

parent re-assignments for its children. The Balancing Phase is divided into two steps: i) s_i's connection to its newly assigned parent *newParent*, and ii) the transmission of parent reassignment messages to children nodes, in which the given nodes are instructed to change their parent. When such a message has arrived, s_i obtains the β value and the identifier of its *newParent*. If *newParent* has a specific node identifier then s_i will attempt to connect to that given node. Notice that if *newParent* cannot accommodate the connect request from s_i then the procedure has to be repeated until completion or until the alternative parents are exhausted.

Note that we have chosen to do parent reassignments at s_i, rather than at the individual child s_j, because s_i can more efficiently eliminate duplicate parent assignments (i.e., two arbitrary children of s_i will both not choose *newParent*.) If the number of children is less than β then the procedure halts. In the contrary case, we have to eliminate $|children(s_i)| - \beta$ children from s_i. Thus, we iterate through the child list of s_i and attempt to identify a child s_j that has at least one alternate parent. If an alternative parent can not be determined for node s_j then it is not meaningful to request a change of s_i's parent.

Extending the optimal branching factor: ETC assumes that all sensors feature the same workload and that the workload of a parent sensor is directly proportional to the number of its child nodes. The rationale behind this assumption is that the majority of queries typically incur the same workload (i.e., the same number of tuples) on each sensor node. However, there are queries (e.g., filter queries, event-based queries) that may impose significantly different workloads on each sensor node. In order to tackle this problem, we could have easily extended the definition of the optimal branching factor [3][4] to take into account the workload of each sensor node rather than the global number of sensor nodes (n) and the depth (d) of the QRT. One way to accomplish this would be to first execute the WART algorithm of the Workload Balancing Module (described in Section 3), which discovers the workload incurred on each sensor node by profiling recent data acquisition activity and then to execute the ETC algorithm in order to create a more workload-balanced topology.

Balancing based on network vs. query semantics: Although, we have shown in our experiments [3] that balancing based on network semantics (i.e., d, n) offers significant energy savings, there are occasions where it may present conflicts with the optimizations proposed by the Query Processing Module (presented in

[4]the optimal branching factor takes into account network semantics (e.g., number of child nodes, depth of the query routing tree, workload of each sensor node.)

Section 5) where optimization is achieved by taking into account query-based semantics. A work that incorporates query-based semantics in the network optimization phase is presented in [30] where the authors configure the network in order to benefit the execution of `Group-By` queries using the *Group-Aware Network Configuration* (GANC) framework. However, one drawback that may arise is that this approach can limit the efficiency of other types of queries that could have benefited from the network-based semantics optimization. Since, in the KSpot$^+$ framework, each module can be enabled or disabled according to the requirements of the application, we could have easily substituted the Tree Balancing Module with GANC in order to support query-based semantics in the network optimization phase. Recall from Section 2.3 that this is also one of the reasons we have not opted for a unified communication scheme as it would decrease the modularity of our framework.

5. Query Processing Module

The Query Processing Module is responsible for query execution as well as a number of services including group management and caching. The procedure starts by propagating a query $\mathcal{Q}$ to the network. Next, each sensor node acquires its local sensor readings, merges them with all values acquired from its child nodes and process them using the INT/MINT algorithms. Finally, each sensor node recursively transmits its results until they reach the sink node. The INT/MINT Views algorithms consist of three phases: i) *the creation phase*, executed during the first acquisition of readings from the distributed sensors. This phase results in n distributed views V_i ($i \leq n$); ii) *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 iii) *the update phase*, executed once per epoch, during which s_i updates its parent node with V_i'.

Creation phase: In the first step of this phase, each sensor retains the tuples that satisfy some query $\mathcal{Q}$ (e.g., temp>60). We only project the attributes related to $\mathcal{Q}$ prior to storing the result in the in-memory buffer V_i. The next step of the algorithm merges the tuples that arrive from the children of s_i into V_i. This yields an *In-Network View* similar to Figure 6. 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 $\mathcal{Q}$. 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

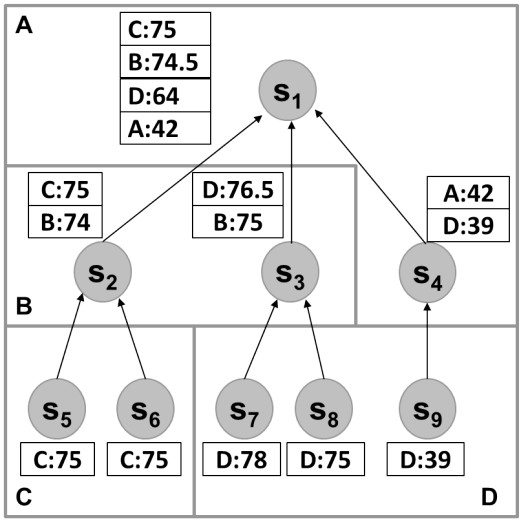

Figure 6: Sensor network deployment of 9 sensors assigned in four rooms $\{A, B, C, D\}$ measuring temperature. A recursively defined in-network view (V) maintaining the local average temperature for each room.

we seek to optimize this process through the proposition of the pruning phase.

Pruning phase: The Pruning Phase constructs a hierarchy of views, where ancestor nodes in the routing hierarchy maintain a superset view of their descendants. Consider a query $\mathcal{Q}$ which returns the k rooms with the highest average temperature. If s_i could locally define the k-highest answers to $\mathcal{Q}$ (at s_0), then s_i could use this information to prune 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 Figure 7 (left), where we illustrate the V_i for a given sensor. Prior to the execution of $\mathcal{Q}$, assume that we established that γ_1=*"maximum possible temperature value"*=120 and γ_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

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

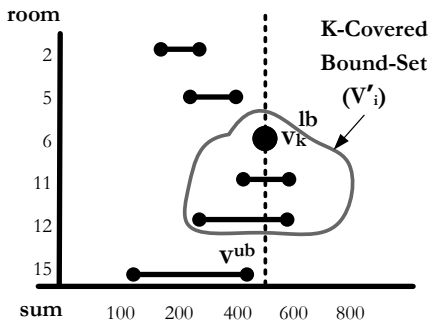

Figure 7: (Left) A Materialized In-Network View V_i of sensor s_i. and the lower-bound (lb) and upper-bound (ub) ranges utilized for generating the k-covered bound set V_i'. (Right) The (lb,ub) ranges for the various returned tuples at some arbitrary node and the k-covered bound set V_i'. We only propagate a tuple u to the parent of s_i, if $u \in V_i'$.

range of possible values for the sum attribute at the sink. This enables 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 kth 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 7 (right), visually depicts this idea.

Update phase: In the previous step, we transformed V_i into a pruned subset V_i'. The Update Phase incrementally and recursively updates V_i'. Let T' denote the V_i' taken at the last execution of $\mathcal{Q}$. 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* [31].) 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-execution of the Pruning Phase.

In particular, any tuple update x with an upper bound (denoted as x^{ub}) less than the v_k^{lb} can be *ignored*. In the opposite case, we add the tuple x to the set of candidates V_i'. 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 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' and transmit the complete V_i' to its parent. 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., $|V_i|$.)

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 these mechanisms are extremely interesting in the context of WSNs, as they allow us to trade accuracy versus energy consumption, in KSpot$^+$ 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 or a Flash-based buffer. For instance, a typical MICA mote with a 2KB SRAM might need to exploit the 512KB on-chip flash memory, while Intel's iMote might easily store these results in the 64KB SRAM. There is a growing trend for more available local storage in sensor devices [25] and therefore local buffering of results is not a threat to our model.

Impact of the k parameter: Similarly to traditional DQP systems, the value of k is typically user-defined and is closely related to the application requirements. Selecting an extremely low value for k (e.g., in a `Top-1` query), might cause the INT/MINT algorithms to omit important results that are vital to the application. For example, in a forest fire monitoring system where alerts are caused by high temperature values, there may be two regions that present identical temperature readings. In our current implementation, the Query Processing module will randomly return one of the extreme values if a `Top-1` query has been injected to the network. However, our approach can be easily adapted to not suppress identical readings by updating only a minor fraction (2 lines) of the implementation code. Selecting an extremely large value for k (e.g., in a `Top-90%` query), might cause the pruning of the INT/MINT algorithms to rapidly decrease as the in-network pruning filters will not be able to omit tuples from the k-covered bound-set.

6. Experimental Methodology

In this section, we describe our experimental methodology which involves a set of trace-driven simulations with real datasets from Intel Research Berkeley and UC-Berkeley. We shall next describe the sensing device and testbed parameters used in our experiments.

Datasets: We utilize the following real datasets in our trace-driven experiments in order simulate wireless sensor networks of various 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 15km off the coast of Maine [35], 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 [9]. More specifically, each of the 32 sensors maintains the average temperature and windspeed on an hourly basis for 208 days between June 2003 and June 2004 (i.e., 4990 time instances.)

iii. Intel Research Berkeley (Intel54): This is a real dataset that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley [15] between February 28th and April 5th, 2004. The sensors were equipped with weather board and collected time-stamped topology information along with humidity, temperature, light and voltage values once every 31 seconds. The dataset includes 2.3 million readings collected from these sensors. We use readings from the 54 sensors that had the largest amount of local readings.

iv. FireWatch (FW12): This is a real dataset from the FireWatch monitoring system deployed at the Lythrodontas forest, which is one of the high risk forest areas in Cyprus, in 2012. We utilize readings from 12 out of 20 sensors as 8 sensors were utilized solely for ensuring multipath routing in case of destroyed nodes. The FW12 dataset includes readings such as: light, temperature, humidity, wind, rainfall (%) and voltage.

Sensing device: We use the energy model of Crossbow's TelosB [25] research sensor device to validate our ideas. TelosB is an ultra-low power wireless sensor equipped with an 8 MHz MSP430 core, 1MB of external flash storage, and a 250kbps Chipcon (now Texas Instruments) CC2420 RF Transceiver that consumes 23mA in receive mode (Rx), 19.5mA in transmit mode (Tx), $7.8mA$ in active mode (MCU active) with the radio off and $5.1\mu A$ in sleep mode. Our performance measure is *Energy*, in *Joules*, that is required at each discrete time instance to resolve the query.

Failures: We utilize a failure rate of 20% in our trace-driven experiments in order to simulate failures. Consequently, certain nodes do not participate (i.e., communication or node failure) in a given epoch. In the cases where node failures affect the critical path this is automatically translated into a chain of delayed

waking windows that force the re-execution of the WART algorithm (workload balancing module). The required energy is measured in the experiments.

Evaluation Parameters: We emphasize in energy consumption and network longevity in our experiments. We refer the reader to [2, 3] for additional evaluation parameters such as k, group cardinality, tuple pruning and balancing efficiency that were used in the evaluation of each individual module.

Our simulation experiments were performed on a Lenovo Thinkpad T61p PC with an Intel Core 2 Duo CPU running at 2.4GHz and 4.0 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 five times and present the average energy consumption for each type of plot.

7. Experimental Results

Our experimental section focuses on two aspects. In the first experiment, we study the effect of incorporating network-awareness into the data acquisition process. To accomplish this, we compare two modes of the KSpot$^+$ framework: i) KSpot$^+$ with only the Query Processing Module enabled, *KSpot$^+$ (MINT)*[5]; and ii) KSpot$^+$ incorporating network-awareness (i.e., all KSpot$^+$ modules are enabled, *KSpot$^+$ (ETC+WART+MINT)*). Notice that the latter, firstly utilizes the ETC algorithm to balance the QRT, then utilizes the WART algorithm to optimize the waking windows of the sensor nodes and finally executes a `Top-`k query using the MINT algorithm. The energy overhead related to the tree construction process and workload balancing scheme are taken into account in the total energy consumption of *KSpot$^+$ (ETC+WART+MINT)*.

We have selected the MINT algorithm for both versions as it presents the higher energy savings in our framework.

Secondly, we evaluate the performance of the full KSpot$^+$ framework in comparison with two predominant data acquisition frameworks TinyDB (TAG) and TINA.

Energy consumption: In the first experiment, we evaluate the energy consumption of `KSpot`$^+$ `(MINT)` and `KSpot`$^+$ `(ETC+WART+MINT)`. We execute a continuous `Top-`k query, on the GDI14, AtmoMon32, Intel54 and FW12 datasets and measure the energy consumption for each dataset separately.

[5]We refer the reader to [2, 3] for the evaluation of each component of KSpot$^+$ in isolation

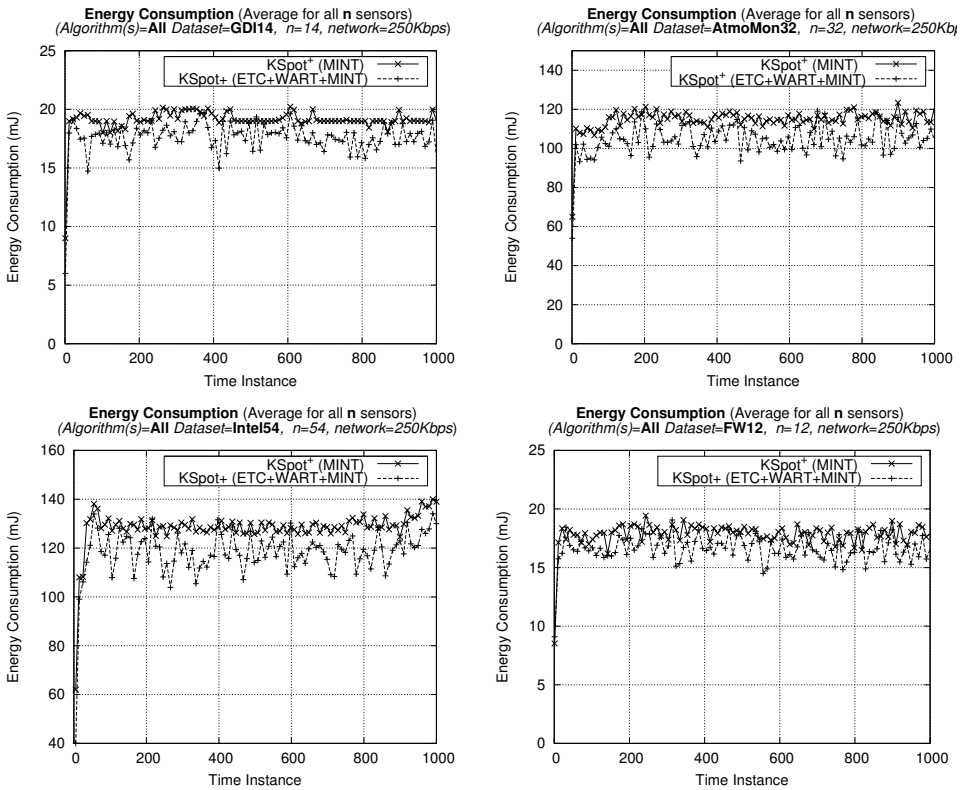

Figure 8: Energy consumption for KSpot$^+$ (MINT) and KSpot$^+$ (ETC+WART+MINT) using the TelosB energy model.

In Figure 8 (top-left), we plot the results using the GDI14 dataset. We observe that the KSpot$^+$ framework using only the MINT algorithm consumes on average 19$\pm$1J. On the other hand, when the KSpot$^+$ framework operates with all modules, we observe a decrease $\approx$6% on average energy 18$\pm$1J. However, we also observe that the standard deviation has negatively increased which proves that there are fluctuations in energy consumption caused by the workload and tree balancing modules. This is expected as under our experimental setting both node and communication failures occur that trigger the reconstruction and adaptation phases of the tree and workload balancing modules respectively. This results in additional packets to be transmitted to the network.

The same observations apply also for the FW12, AtmoMon32 and Intel54 datasets, with the complete KSpot$^+$ framework maintaining a competitive advantage over `KSpot`$^+$ `(MINT)`. In particular, we observe that the complete KSpot$^+$

Table 1: Average energy consumption for the TAG, TINA, and KSpot$^+$ framework under different datasets.

Algor. \ Dataset	GDI14	AtmoMon32	Intel54	FW12
TAG	57±3J	234±2J	523±22J	53±1J
TINA	48±2J	183±6J	289±15J	43±1J
INT	34±1J	170±7J	187±08J	31±1J
KSpot$^+$ MINT	19±1J	115±4J	139±06J	17±1J
KSpot$^+$ ETC+WART+MINT	18±1J	105±8J	118±10J	16±1J

framework consumes 16±1J in the FW12 dataset, 105±8J in the AtmoMon32 dataset, and 118±10J in the Intel54 dataset, which translates in 8%, 9% and 15% decrease in energy consumption respectively. The results for all experiments are summarized in Table 1.

In conclusion, the complete KSpot$^+$ framework demonstrates large energy gains when operating both with isolated components or full-fledged. It is important to note that the query processing module demonstrates much larger gains (in the order of Joules) compared to the other two modules (in the order of milli-Joules) of the KSpot$^+$ framework. This shows that in-network pruning combined with exploiting temporal coherence can be of higher benefit in cases where applications require monitoring of the k most important events.

Network lifetime: In the second experiment, we evaluate the network lifetime. We define network lifetime, similar to [36, 2, 3], as the time instance t' at which $Energy(t') = 0$. This definition, adopts a universal perspective of the sensor network (i.e., measures the energy depletion across the whole spectrum of participating sensors) as opposed to existential energy depletion metrics (i.e., measure when the energy is depleted on a single node) utilized in other works [31, 30]. This is because we are particularly interested in decreasing the overall energy consumption of the sensor network and not a single node. 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 3V voltage and supply a current of 2500 mAh (milliAmpere per hour.) Assuming similar to [35], that only 2200mAh is available and that all current is used for communication, we can calculate that AA batteries offer $23,760J$ ($2200mAh \times 60min \times 60s \times 3V$.) We terminate this iteration when the termination condition is satisfied.

Figure 9 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

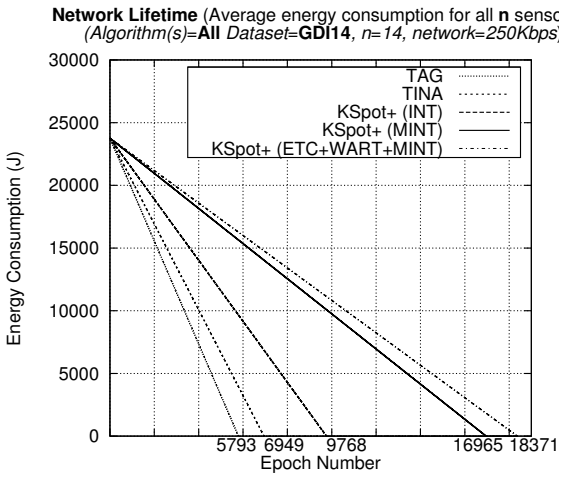

Figure 9: Network lifetime for all algorithms.

the available energy of sensors under TAG is consumed faster than all algorithms, leading to a lifetime of just 5793 epochs (i.e., 193 minutes). TINA ranks fourth by offering 6949 epochs (i.e., 231 minutes). The KSpot$^+$ framework with only the INT algorithm enabled ranks third with 9,768 epochs (i.e., 325 minutes). Next, the KSpot$^+$ framework with only the MINT algorithm consumes its available energy budget far later at epoch 16,965 (i.e., 565 minutes). Finally, the full KSpot$^+$ framework, which includes all modules enabled, ranks first at epoch 18,371 (i.e., 612 minutes), and this is translated into a $\approx$317% increase of the network lifetime compared to TAG.

8. Related Work

Traditional middleware frameworks such as the Common Object Request Broker Architecture (CORBA), Java service oriented architecture (JINI) are considered heavyweight in terms of processor and memory requirements, which renders them highly inefficient for WSN deployments. In this section, we present middleware frameworks tailored specifically for WSNs that perform data acquisition operations and are thus closely related to the proposed KSpot$^+$ framework. More specifically, we classify these middleware frameworks according to their approach into three categories: Data-centric, Application-centric and Publish subscribe. We then perform a qualitative comparison between KSpot$^+$ and the presented mid-

dleware approaches across 4 different dimensions: Energy Awareness, Workload Optimization, Topology Optimization and Top-k support in order to highlight the benefits of utilizing the KSpot$^+$ framework. The results of our analysis are summarized in Table 2.

Data-centric Middleware Frameworks: are middleware frameworks closely related to KSpot$^+$ that view the network as a virtual relational database and inject query messages, which are then processed locally at each sensor node.

Cougar [40], is one of the first data-centric approaches for wireless sensor networks. Each sensor node acts as a database that stores the node's measurements locally and the network acts as a distributed database. In Cougar, queries as well as management operations are translated to query messages, which are then injected to the network. Similar to KSpot$^+$, Cougar [40], employs a centralized optimizer, which maintains status information about the network in order to coordinate sensor nodes in an energy-efficient manner. However, in Cougar, this centralized approach requires a massive amount of messages to be transmitted back and forth to the sink station thus increasing energy consumption. Furthermore, in [3] we have shown that node and communication failures severely hamper the efficiency of this coordination scheme as they cause sensor nodes, especially the ones in higher levels, to stay in reception mode longer than required.

TinyDB [21], is one of the most popular data acquisition frameworks developed for TinyOS. Like Cougar, it is a data-centric middleware framework that supports SQL-syntax queries over the sensor network. Additionally, TinyDB supports a number of different query sets including historic, event-based and lifetime queries. TinyDB's power-aware optimizer employs a cost-based mechanism in order to choose the most energy-efficient query execution plan, which may involve prioritizing data delivery, adapting sampling rates and minimizing power consumption. This often enforces a uniform waking window for all sensor nodes depending on the depth of the QRT, which in the majority of cases it is clearly an overestimate. The rationale behind this over-estimation is to offset the limitations in the quality of the underlying clock synchronization algorithms of the operating system but in reality it is too coarse [2]. TinyDB employs Tiny Aggregation (TAG) [20], for energy efficient in-network aggregation of sensor results. KSpot$^+$ extends this in-network aggregation scheme by enabling support for advanced query semantics (e.g., top-k, group-by) that further minimize energy consumption by reducing the size and number of packets transmitted to the network.

Temporal coherency-aware in-Network Aggregation (TINA) [31], works on top of existing in-network aggregation like TAG and Cougar, and similarly to the Query Processing Module, introduces a temporal coherency filter that mini-

Table 2: Classification and comparison of middleware approaches for WSNs

Middleware approach	Key Features	Energy-aware	Workload opt.	Topology opt.	Top-k support
Data-centric					
TinyDB [21]	SQL syntax, lifetime/event-based queries, Semantic routing trees	Y	Y	N	N
Cougar [40]	SQL syntax, Virtual relational database, centralized optimizer	Y	Y	N	N
TINA [31]	temporal coherence filters, group aware network configuration	Y	N	Y	N
DsWare [41]	SQL syntax, real-time semantics, event-detection	Y	N	N	N
SNEE [11]	Rich and expressive language, workload scheduling	Y	Y	N	N
SINA [32]	Virtual spreadsheet db, Attribute-based naming, Hierar. Clust.	Y	N	N	N
KSpot$^+$	SQL syntax, in-network aggregation, advanced query semantics	Y	Y	Y	Y
Application-driven					
Milan [13]	topology adaptation	Y	N	Y	N
MidFusion [14]	information fusion, sensor agents	Y	N	N	N
Publish-Subscribe					
Mires [33]	aggregation service, high-level interfaces	Y	N	N	N
AWARE [23]	sensor network & UAV coordination	Y	Y	N	N

mizes both the size and number of transmitted packets. Additionally, it influences the construction of the QRT by incorporating query-based semantics using the Group-aware Network Configuration (GANC) [30] component. TINA achieves significant energy savings while maintaining specified quality of data. The MINT algorithm of the KSpot$^+$ Query Processing module utilizes a temporal coherence filter like TINA but also incorporates in-network pruning, which introduces additional energy savings.

Sensor Information Networking Architecture (SINA) [32], provides a set of programming abstractions that enable application designers to view the network as a collection of distributed objects. SINA enables application designers to easily query the network (either a single sensor or a group of them) using an extended SQL syntax (SQLT) that incorporates attribute-based naming in the filtering process. Additionally, the architecture provides a set of configuration and communication primitives that enable scalable and energy-efficient organization and query-processing. However, in achieving energy-efficient topologies, SINA may sacrifice the results of some sensors to avoid data collisions. This may result in the production of inaccurate results at the sink node thus it is not applicable for data sensitive applications.

The *Sensor NEtwork Engine* (SNEE) [11] employs a query optimizer that receives metadata information about the available resources (e.g., memory, energy), the WSN topology and also predictive cost models. These are then used for computing the worst-case upper-bounds for the output size and time taken for operations. SNEE combines a rich, expressive query language, named SNEEQL, which provides extensive support on the JOIN operators incorporating techniques found on classical DQP architectures. Unlike KSpot$^+$, the proposed query language does

not directly address Top-k queries although we assume that they can be incorporated as an aggregate function. Furthermore, SNEE supports workload balancing by scheduling different workloads to different sites in the network thus effectively reducing the energy. However, SNEE assumes that the underlying infrastructure employs an efficient protocol for self-organization of the topology thus neglecting to investigate the effects of an unbalanced topology. KSpot$^+$ addresses the latter with the aid of the Tree Balancing Module.

The *Data services middleWare* (DsWare) [41], provides data abstractions to applications in order to improve the performance of real-time execution and reduce the communication cost. It inserts a layer between the applications and the sensor network, which is composed of server and sensor-side components. Like KSpot$^+$, the server-side components store meta-data about the network and additionally handle all coordination activities and provide mechanisms for prediction. The sensor-side components manage the state of the sensor nodes and provide a filtering mechanism that provides approximate instead of exact values in order to decrease communication overhead. In KSpot$^+$, we minimize the overall energy consumption of the network without sacrificing the results of sensor nodes.

Application-centric Middleware Frameworks: The *Middleware Linking Applications and Networks* (Milan) [13] consists of a high level application interface that enables application designers to specify their QoS requirements inside the sensor network application code. Similarly to KSpot$^+$, Milan's architecture extends to the network protocol stack thus allowing the middleware to perform power control on the communication medium as well as topology changes according to heuristics. However, unlike KSpot$^+$, Milan does not consider the workload incurred on each sensor node, which may result in serious data reception inefficiencies.

The *MidFusion* [14] is a middleware architecture that aims to facilitate information fusion in sensor networks. MidFusion assumes that a routing strategy is provided by the operating system of the sensor network and that failures in the network can only occur due to communication interference. Therefore, unlike KSpot$^+$ it does not consider data transmission/reception inefficiencies that occur because of unbalanced routing structures or uneven workload distributed amongst sensor nodes. Additionally, MidFusion may omit sensor nodes from the data acquisition process because of the QoS requirements of the application, which may lead to inaccurate results.

Publish-subscribe Middleware Frameworks: The *Aware* [23] middleware platform provides components to enable the cooperation between fixed and mobile sensor nodes in addition to Unmanned Aerial Vehicles (UAVs.) It is based on the

publish/subscribe paradigm where the flow of information is coordinated through data channels. Each device publishes its capabilities (i.e., data channels) and attributes to a centralized registry where other devices can subscribe to and receive feeds. Aware supports packet-level optimizations that focus on content rather than address; the network acts as a global filtering mechanism, minimizing in this way the communication overheads.

Mires [33] is a publish/subscribe middleware system built on top of TinyOS. It encapsulates the low-level generic interfaces of the operating system and provides high-level services to the applications. In addition to the publish/subscribe layer, Mires incorporates a routing module that facilitates multi-hop communication. Although, both Aware and Mires support a number of packet-level optimizations that can greatly decrease the number of communication packets, additional energy savings can be achieved by optimizing the network topology.

In summary, the majority of presented middleware approaches employ mechanisms for reducing the overall energy consumption of the network thus increasing the longevity of a WSN as shown in Table 2. However, they neglect the important parameter of constructing an energy efficient topology and operate on top of the initial ad hoc query routing tree. Additionally, most approaches often assume a fixed workload distributed uniformly on all sensor nodes. Consequently, it is not clear how efficient they will operate under a variable workload, which occurs under the following circumstances: i) from a non-balanced topology, where some nodes have many children and thus require more time to collect the results from their dependents; and ii) from multi-tuple answers, which are generated because some nodes return more tuples than other nodes (e.g., because of the query predicate.) Furthermore, none of the approaches support top-k queries, which can significantly decrease the overall number and size of transmitted packets. Finally, few of the proposed middleware approaches have been implemented and tested in real environments. Like all presented approaches, the KSpot$^+$ middleware framework focuses on energy efficiency but additionally employs mechanisms that generate a more efficient topology as well as provide support for top-k queries.

9. Conclusions

Current data-centric frameworks for WSNs suffer from data reception/transmission inefficiencies because they operate on the presumption that the underlying network topology is efficient. This paper advocates an alternative framework design that looks upon the network characteristics as well as the intrinsic properties of the data dissemination/acquisition process. In this context, three novel tech-

niques were developed with opportunities of applications that go beyond the current problem settings (e.g., people-centric sensing [27, 5], smartphone networks). Through our experimental evaluation, we have shown that incorporating network-awareness can provide significant energy reductions and increase the longevity of the wireless sensor network.

The KSpot$^+$ framework presented in this paper assumes that the routing topology is stationary. However this is not the case in mobile environments such as Mobile Sensor Networks (MSNs) and Vehicular Ad hoc Networks (VANETs). In the future, we plan extend KSpot$^+$ to support such mobile environments. Furthermore, since the operation of MSNs is severely hampered by the fact that failures are omnipresent, fault-tolerance schemes become of prime importance. In these settings, data acquisition needs to be succeeded by efficient in-network storage (e.g., [17]), such that these events can later be retrieved by the user. We plan to extend KSpot$^+$ with fault tolerance mechanisms that will ensure the continuous operation of data acquisition even in these harsh environments.

Acknowledgements

This work was supported in part by the Cyprus Research Promotion foundation under Project FireWatch (#0609-BIE/09), the Open University of Cyprus under the Project SenseView, the University of Cyprus under the project SmartLab and the second author's Startup Grant, the European Commission under Projects CONET(#FP7-224053), mPower (#034707) and MiraculousLife (#FP7-ICT-2013-10), and the US National Science Foundation under Projects S-CITI (#ANI-0325353), AQSIOS (#IIS-0534531) and Astroshelf (#OIA-1028162). Additionally, we would like to thank Joe Polastre (UC Berkeley) for the Great Duck Island data trace and Kostas Papageorgiou (Cyprus Department of Forests) for providing the FireWatch dataset.

References

[1] I.F. Akyildiz, R. Sivakumar, E. Ekici, J. Cavalcante de Oliveira, J. McNair, "NETWORKING 2007. Ad Hoc and sensor networks, Wireless Networks, Next Generation Internet", In Proceedings of the 6th International IFIP-TC6 Networking Conference, Atlanta, GA, USA, May 14-18, Vol.4479, ISBN 978-3-540-72605-0, 2007.

[2] P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, "Power Efficiency through Tuple Ranking in Wireless Sensor Network Monitoring",

Distributed and Parallel Databases Journal, Vol.29, No.1-2, pp.113-150, 2011.

[3] P. Andreou, D. Zeinalipour-Yazti, A. Pamboris, P.K. Chrysanthis, G. Samaras, "Optimized Query Routing Trees for Wireless Sensor Networks", In Information Systems Journal, Volume 36, Issue 2, pp.267-291, April, 2011.

[4] P. Andreou, D. Zeinalipour-Yazti, G. Samaras, P.K. Chrysanthis, "Towards a Network-aware Middleware for Wireless Sensor Networks", The 8th International Workshop on Data Management for Sensor Networks, Seattle, WA, USA, August 29, 2011.

[5] A.T. Campbell, S.B. Eisenman, N.D. Lane, E.Miluzzo, R.A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, G-S. Ahn, "The Rise of People-Centric Sensing", In IEEE Internet Computing: Mesh Networking, pp. 30-39, July/August, 2008.

[6] Q. Cao, T. Abdelzaher, J. Stankovic, T. He, "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, April 22-24, USA, pp.233-244, 2008.

[7] C-M. Chao, T-Y. Hsiao, "Design of structure-free and energy-balanced data aggregation in wireless sensor networks", Journal of Network and Computer Applications (JNCA'14), January, Vol.37, pp. 229-239, 2014.

[8] O. Diallo, J.J.P.C. Rodrigues, M. Sene, "Real-time data management on wireless sensor networks: A survey", Journal of Network and Computer Applications (JNCA'12), July, Vol.35, No.3, pp. 1013-1021, 2012.

[9] Earth Climate and Weather, University of Washington, http://www-k12.atmos.washington.edu/k12/grayskies/

[10] R. Fagin, "Combining Fuzzy Information from Multiple Systems", In Journal of Computer and System Sciences, Montreal, Canada, February, Vol.58, No.1, pp.83-99, 1999.

[11] I. Galpin, C.Y.A. Brenninkmeijer, F. Jabeen, A.A.A. Fernandes, N.W. Paton., "An Architecture for Query Optimization in Sensor Networks", In

Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE'08), Cancun, Mexico, April 7-12, pp.1439-1441, 2008.

[12] G. Gupta, M. Misra, K. Garg, "Energy and Trust Aware Mobile Agent Migration Protocol for Data Aggregation in Wireless Sensor Networks", Journal of Network and Computer Applications (JNCA'14), February, In Press, 2014.

[13] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, M.A. Perillo, "Middleware to support sensor network applications", In IEEE Network, Vol. 18, Is. 1, pp.6-14, Jan/Feb, 2004.

[14] A. Hitha, K. Mohan, S. Behrooz, "MidFusion: An adaptive middleware for information fusion in sensor network applications", In Information Fusion, Special Issue on Distributed Sensor Networks, Vol.9, Is.3, pp.332-343, July, 2008.

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

[16] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, M. Turon, "Health Monitoring of Civil Infrastructures using Wireless Sensor Networks", In Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07), Cambridge, MA, USA, April, ACM Press, pp.254-263, 2007.

[17] W-H Liao, H-C Yang, "A power-saving data storage scheme for wireless sensor networks", Journal of Network and Computer Applications (JNCA'12), July, Vol.35, No.2, pp. 818-825, 2012.

[18] K. Lorincz, B. Chen, G.W. Challen, A.R. Chowdhury, S. Patel, P. Bonato, M. Welsh, "Mercury: A Wearable Sensor Network Platform for High-fidelity Motion Analysis", Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09), November, Berkeley, CA, 2009.

[19] H.V. Luu, X. Tang, "An efficient algorithm for scheduling sensor data collection through multi-path routing structures", Journal of Network and Computer Applications (JNCA'14), February, Vol.38, pp. 150-162, 2014.

[20] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", Proceedings of the 5th

symposium on Operating Systems Design and Implementation (OSDI'02) , Vol.36, No. SI, pp.131-146, 2002.

[21] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks", Proceedings of the International Conference on Management Of Data (SIGMOD'03), San Diego, CA, USA, June 9-12, pp.491-502, 2003.

[22] MemSic Technology Inc., http://www.memsic.com/

[23] A. Ollero, M. Bernard, M.L. Civita, L. van Hoesel, P.J. Marron, J. Lepley, E. de Andres, "AWARE: Platform for Autonomous self-deploying and operation of Wireless sensor-actuator networks cooperating with unmanned AeRial vehiclEs", In Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR 2007), Rome, Italy, September 27-29, pp.1-6, 2007.

[24] J. Paradiso, M. Feldmeier, "Ultra-Low-Cost Wireless Motion Sensors for Musical Interaction with Very Large Groups" In Proceedings of the 2002 International Computer Music Conference, Gothenburg, Sweden, September 16-21, pp.83-87, 2002.

[25] J. Polastre, R. Szewczyk, D.E. Culler, "TELOS: Enabling Ultra-low Power Wireless Research", Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN'05), Los Angeles, CA, USA, April 25-27, pp. 364-369, 2005.

[26] M. Roantree, J. Shi, P. Cappellari, M.F. O'Connor, M. Whelah, N. Moyna, "Data Transformation and Query Management in Personal Health Sensor Networks", Journal of Network and Computer Applications (JNCA'12), July, Vol.35, No.4, pp. 384-403, 2012.

[27] I.Rose, M. Welsh, "Mapping the Urban Wireless Landscape with Argos", Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys10), Zurich, Switzerland, November 35, 2010.

[28] C. Sadler, P. Zhang, M. Martonosi, S. Lyon, "Hardware Design Experiences in ZebraNet", Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys'04), Baltimore, Maryland, USA, November 3-5, pp.227-238, 2004.

[29] SELEX Galileo Inc., http://www.selex-sas.com/

[30] M.A. Sharaf, J. Beaver, A. Labrinidis, P.K. Chrysanthis, "Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks", International Journal on Very Large Data Bases (VLDBJ'04), December, Vol.13, No.4, pp.384-403, 2004.

[31] M.A. Sharaf, J. Beaver, A. Labrinidis, P.K. Chrysanthis, "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, CA, USA, September 19, pp.69-76, 2003.

[32] C-C. Shen, C. Srisathapornphat C., C. Jaikaeo, "Sensor information networking architecture and applications", In IEEE Personal Communications, Vol.8, No.5, pp.52-59, August, 2001.

[33] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, J. Kelner "Mires: a publish/subscribe middleware for sensor networks", In Personal Ubiquitous Computing, Vol.10, No.1, December, 2005.

[34] I. Stojmenovic, "Handbook of Sensor Networks: Algorithms and Architectures", Wiley, ISBN: 978-0-471-68472-5, November, 2005.

[35] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, D. Culler, "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.

[36] H. Thomas, S. Yi, H.D. Sherali, "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.

[37] D. Virmani, T. Sharma, R. Sharma, "Adaptive Energy Aware Data Aggregation Tree for Wireless Sensor Networks", In International Journal of Hybrid Information Technology (IJHIT'13), Vol. 6, No. 1, January, 2013.

[38] Voltree Power Inc., http://www.voltreepower.com/

[39] D.B. Yang, H.H. Gonzalez-Ba, L.J. Guibas, "Counting People in Crowds with a Real-Time Network of Simple Image Sensors", In Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV'03), Nice, France, October 13-16, Vol.1, 2003.

[40] Y. Yao, J.E. Gehrke, "The cougar approach to in-network query processing in sensor networks", In ACM SIGMOD Record (SIGMOD'02), September, Vol.31, No.3, pp.9-18, 2002. Vol.12, No.3, pp.493-506, 2004.

[41] X. Yu, K. Niyogi, S. Mehrotra, N. Venkatasubramanian, "Adaptive Middleware for Distributed Sensor Environments", In IEEE Distributed Systems Online, Vol.4, No.5, May 2003. ibitemtja D. Zeinalipour-Yazti, S. Lin, D. Gunopulos, "Distributed Spatio-Temporal Similarity Search", In ACM international conference on Information and knowledge management (CIKM'06), Arlington, VA, USA, November 6-11, pp.14-23, 2006.