KSPOT$^+$: A NETWORK-AWARE FRAMEWORK FOR ENERGY-EFFICIENT DATA ACQUISITION IN WIRELESS SENSOR NETWORKS

Panayiotis G. Andreou

University of Cyprus, 2011

Wireless Sensor Networks (WSNs) are composed of resource-constrained tiny-scale devices that 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 that provide similar functionality to traditional distributed query processing systems. 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 dissertation we present KSpot$^+$, a novel distributed network-aware framework for query-based data acquisition in WSNs that optimizes network efficiency by combining three novel components: i) the *Tree Balancing Module*, which balances the workload incurred on each sensor node during a query by constructing efficient network topologies; ii) the *Workload Balancing Module*, which minimizes data reception inefficiencies by synchronizing the network activity intervals of each sensor node; and iii) the *Query Processing Module*, which provides advanced query processing semantics (e.g., top-k, group-by) by employing a novel ranking mechanism that yields only the k-highest ranked answers, thus further minimizing energy consumption. KSpot$^+$ features a highly modular design in which each module can operate both in isolation and in combination with other modules, scales linearly with the addition of new sensor nodes in the network and is resilient to node and communication failures.

The KSpot$^+$ framework can be utilized in numerous application domains that require query-based data acquisition including environmental monitoring, big ephemeral events, structural monitoring, urban monitoring, security applications and health monitoring. Applications that benefit the most from the utilization of the KSpot+ framework are those that require continuous monitoring of the most important events of the network.

In order to validate the efficiency of our approach, we have created a prototype implementation of the KSpot$^+$ framework in nesC and JAVA. In our experimental evaluation, we thoroughly assess the performance of KSpot$^+$ using the real prototype system we developed and datasets from the University of California - Berkeley, the University of Washington and Intel Research Berkeley. We show that KSpot$^+$ provides significant energy reductions under a variety of conditions, thus significantly prolonging the longevity of a WSN compared to predominant approaches.

Panayiotis G. Andreou – University of Cyprus, 2011

KSPOT$^+$: A NETWORK-AWARE FRAMEWORK FOR ENERGY-EFFICIENT DATA ACQUISITION IN WIRELESS SENSOR NETWORKS

Panayiotis G. Andreou

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Cyprus

Recommended for Acceptance

by the Department of Computer Science

June, 2011

APPROVAL PAGE

Doctor of Philosophy Dissertation

KSPOT$^+$: A NETWORK-AWARE FRAMEWORK FOR ENERGY-EFFICIENT DATA ACQUISITION IN WIRELESS SENSOR NETWORKS

Presented by

Panayiotis G. Andreou

Research Supervisor _____
Prof. George S. Samaras

Research Co-Supervisor _____
Dr. Demetrios Zeinalipour-Yazti

Committee Member _____
Prof. Andreas Pitsillides

Committee Member _____
Prof. Constantinos Pattichis

Committee Member _____
Prof. Panos K. Chrysanthis

Committee Member _____
Prof. Evaggelia Pitoura

University of Cyprus

June, 2011

ACKNOWLEDGEMENTS

I am first of all deeply indebted to my advisor, Prof. George Samaras, for convincing me to pursue this Ph.D. and for guiding me through this difficult but exciting process. He has provided a tremendous amount of advice, I have space to only thank him for a subset. First, his never-ending support and guidance both during my research as well as at a personal level. It has ranged from a friendly "Hi, is everything ok?" to the ominous "Holidays do not apply to our kind of work, I expect this working yesterday!", without which I would have not accomplished so many things during the past few years. My favorite quote, "*Accomplishing the impossible means only that the boss will add it to your regular duties*" is rightfully his. Second, his insightful advice and ideas, discussing one work and suddenly realizing how it can progress 5 steps further. Third, his encouragement to participate in various national and international projects, I learned so much from a different perspective while in parallel provided the financial support that alleviated any burden to my family.

I am likewise massively indebted to my co-advisor and friend Dr. Demetris Zeinalipour. I have yet to understand how someone can accomplish so much and manage so many things, yet remaining "human"; I hope that part of his methodology has been imprinted on my way of thinking. He has shown me that new ideas can be put on paper as long as there is a deadline at hand. For every hour I put into my research, I think Demetris has devoted two. His insight, insistence on structure and simplicity as well as his tremendous patience with my writing have helped me to realize how technical writing can be utilized to frame scientific findings and publish new scientific knowledge to the community.

Special thanks to Prof. Andreas Pitsillides who has been my B.Sc. and M.Sc thesis advisor. He was the first one to believe in my abilities, I hope that I have met his expectations. I would like to thank him for constantly challenging my preconceptions and setting new standards for me to meet.

Many thanks to Prof. Panos Chrysanthis for his devoted energy and interest in my work. His insistence on continuous refinement and polishing has taught me that a good paper must also appear good. I would like to thank him for making me want to excel.

To my friend Polys Kourousides for being so eager to discuss all my math related enquiries. Without his expert knowledge, the implementation of the perimeter algorithm in the paper titled *"In-Network Data Acquisition and Replication in Mobile Sensor Networks"* would not have been possible.

To my friend and colleague Dr. Dimosthenis Georgiadis for his everlasting encouragement and selfless love. He has been a rock behind me these past five years, and before, reminding me that there is more in life than research and projects. In many respects, he has been like my personal regulator, ensuring that my excitement towards research did not consume my personal objectives. I am grateful to him for enriching my life, he is like a brother.

To my friend and office-mate Dr. Panagiotis Germanakos for his support and insightful brain-storming discussions. He has shown me that a research meeting does not have boundaries on location and time, extraordinary ideas can come from anywhere and anytime and from any singularly unexpected source. I have profited immensely from his suggestions, high standards and friendly company. My second favorite quote, "*Adaptation is the key to survival*" is rightfully his.

I have learned much and received helpful feedback from many others at the University of Cyprus, including Marios Belk, Christophoros Christoforou, Christoforos Panayiotou, Andreas Konstantinides and Josephine Antoniou.

To my friends, I would firstly like to inform them that “Yes, I have finally finished my Ph.D.!” in response to their infuriating question “Have you finished your Ph.D. yet?”. I would secondly like to thank them for believing in me and for forgiving me all those times going offline for long periods of time. It has been a great aid knowing that they are and always were there for me.

To my friend, Morfo Violari for her perpetual love and support to me. Thank you for enduring me on the good and the bad and most of all thank you for putting a lot of *Salsa* in my life.

Finally to my family, their continuous support and belief in my abilities has been a tremendous boost to me. Providing nature and nurture throughout my life, they taught me the importance of hard work and persistence. Without their love, care and encouragement, I would not have come this far.

To my mother,

without your love and understanding I would have never completed this colossal task,

without your belief that I can always succeed I would not have always succeeded,

without your encouragement I would have never been able to overcome any obstacles and difficulties, even the most hard and painful ones,

without your continuous reminder of how much proud you are of me I would not have struggled to make you more proud.

I dedicate this, my greatest achievement so far, to you.

CREDITS

Peer-reviewed Journal papers

1. *"Power Efficiency through Tuple Ranking in Wireless Sensor Network Monitoring"*, P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, Distributed and Parallel Databases Journal, Vol.29, No.1-2, pp.113-150, 2011.

2. *"In-Network Data Acquisition and Replication in Mobile Sensor Networks"*, P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, Distributed and Parallel Databases Journal, Vol.29, No.1-2, pp.87-112, 2011.

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

Peer-reviewed Conference papers

1. *"ETC: Energy-driven Tree Construction in Wireless Sensor Networks"*, P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, G. Samaras, 2nd International Workshop on Sensor Network Technologies for Information Explosion Era (SeNTIE'09), in conjunction with MDM'09, IEEE Press, May 18th - May 20th, Tapei, Taiwan, 2009.

2. *"FSort: External Sorting on Flash-based Sensor Devices"*, P. Andreou, O. Spanos, D. Zeinalipour-Yazti, G. Samaras, P. K. Chrysanthis, The 6th International Workshop on Data Management for Sensor Networks (DMSN 2009), August 24, Lyon, France, 2009

3. *"Perimeter-Based Data Replication and Aggregation in Mobile Sensor Networks"*, P. Andreou, D. Zeinalipour-Yazti, M. Andreou, P. K. Chrysanthis, G. Samaras, The 10th International Conference on Mobile Data Management (MDM 2009), IEEE Press, May 18th - May 20th, Tapei, Taiwan, 2009.

4. *"KSpot: Effectively Monitoring the K Most Important Events in a Wireless Sensor Network"*, P. Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, G. Samaras, The 25th International Conference on Data Engineering March (ICDE'09), Shanghai, China, May 29 - April 4, 2009.

5. *"Workload-aware Query Routing Trees in Wireless Sensor Networks"*, P. Andreou, D. Zeinalipour-Yazti, P. Chrysanthis, G. Samaras, The 9th International Conference on Mobile Data Management (MDM 2008), Beijing, China, April 27-30, 2008.

6. *"SenseSwarm: A Perimeter-based Data Acquisition Framework for Mobile Sensor Networks"*, D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis, G. Samaras, Intl. Workshop on Data Management for Sensor Networks (DMSN 2007 - VLDB 2007 Workshops), Vienna, Austria, 2007.

7. *"The MicroPulse Framework for Adaptive Waking Windows in Sensor Networks"*, D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis, G. Samaras, A. Pitsillides, IEEE First International Workshop on Data Intensive Sensor Networks (DISN 2007 - MDM 2007 Workshops), Mannheim, Germany, May 11, 2007.

8. *"MINT Views: Materialized In Network Top-k Views in Sensor Networks"*, D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis, G. Samaras, IEEE/ACM 8th International Conference on

Mobile Data Management (MDM 2007), Mannheim, Germany, May 7 - 11, ISBN 14244-1240-4, pp. 182-189, IEEE Press, 2007.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Table of Acronyms

Symbol	Definition
WSD	Wireless Sensor Device
WSN	Wireless Sensor Network
$\mathcal{Q}$	A Continuous Query
n	Number of Sensors
S	Set of Sensors $S = \{s_1, s_2, ..., s_n\}$
s_i	Sensor number i (s_0 denotes the sink).
m	Number of sensor recordings $\{a_1, a_2, ..., a_m\}$
e	Epoch duration (consecutive data acquisition round) of query $\mathcal{Q}$
$\mathcal{T} = (S, E)$	Query Routing Tree (S=vertices, E=edges)
d	Depth of the routing tree $\mathcal{T}$
w_i	Wake-up time of sensor s_i
τ_i	Waking window of sensor s_i
ψ	Total time needed to answer query $\mathcal{Q}$
d	Depth of the routing tree $\mathcal{T}$
$children(s_i)$	Children List of sensor s_i
$APL(s_i)$	Alternate Parent List of sensor s_i
β	Balanced Branching Factor of network S
k	Number of requested results
a_i	Attribute i
m	Number of Attributes at each sensor $\{a_1, a_2, ..., a_m\}$
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)

Chapter 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 (WSDs)* that can be utilized for the development of environmental monitoring systems. Similarly to embedded systems, WSDs are controlled by one or more main processing cores, typically either micro-controllers or digital signal processors, include components for data storage and enable both wired and wireless communication. One of the key differences of WSDs with embedded systems is that the former include one or more sensor components that allow real-time measurements of environmental and biological phenomena amongst others.

The ability of WSDs to communicate with each other in a wireless manner enables the formation of ad hoc *Wireless Sensor Networks (WSNs)* that can deployed in harsh environments and monitor physical phenomena under diverse conditions. Large-scale deployments of *Wireless Sensor Networks (WSNs)* have already emerged in environmental and habitant monitoring [121, 107], structural monitoring [70] and urban monitoring [93]. In these deployments, users are able to disseminate queries to the network that request runtime measurements of sensor data. One of the key goals in designing an energy-efficient WSN is to maximize the lifetime of the network thus

minimizing maintenance costs. This is not a trivial task since in the majority of cases, WSDs feature a limited energy budget as they are typically powered using AA or AAA batteries. In order to prolong the lifetime of the network, much research has been devoted to the design and implementation of energy-efficient hardware such as low-frequency processors [21, 57, 117], low-power communication units [136, 128, 119, 116, 36] and low-power sensor components [103]. Although these energy-efficient hardware designs can significantly decrease the energy overhead of a WSD, the efficient utilization of this hardware by energy-conscious applications/algorithms can further reduce the energy consumption the WSD and considerably prolong the overall lifetime of a WSN.

A decisive variable for prolonging the longevity of a WSN is to minimize the utilization of the wireless communication medium. It is well established that communicating over the radio in a WSN is the most energy demanding factor among all other functions, such as storage and processing [85, 86, 135]. For example, the energy consumption for transmitting 1 bit of data using the MICA mote is approximately equivalent to processing 1000 CPU instructions [86]. In order to cope with this energy challenge sensing devices are forced to power down their radio transceiver (transmitter-receiver) or utilize power-saving modes [103] between *epochs* (i.e., consecutive data acquisition rounds. More specifically, it has been shown that sensors operating at a 2% duty cycle can achieve lifetimes of 6-months using two AA batteries [86, 121, 107]. Supplementary approaches to cope with the energy challenge have been proposed at virtually all layers of the sensing device stack ranging from the hardware layer [103] to the operating system layer [61, 78, 17], the programming language [50], the network layer [145] and the data management layer (e.g., storage [142, 89, 13], compression [39, 113], query processing [85, 86, 135, 109, 110, 112, 81, 63] and prediction [52, 95]). A general theme in these supplementary approaches is to reduce the number of messages communicated between sensors prolonging in that way the lifetime of a WSN.

It is important to notice that the majority of existing approaches establish data acquisition on the premise of Query Routing Trees, denoted as $\mathcal{T}$ (a.k.a spanning trees or converge-cast trees), which provide each sensor with a path over which query answers can be transmitted to a centralized querying node (i.e., sink). Query Routing Trees are typically constructed in an ad hoc manner using the the First Heard From (FHF) mechanism, which is utilized in a number of data acquisition frameworks such as [85, 86, 135, 109, 110] and operates as follows. A user submits a query $\mathcal{Q}$ at the sink node and the system then initiates the execution of $\mathcal{Q}$ by disseminating it to the sensor network. In particular, the sink sends $\mathcal{Q}$ to sensor nodes within its communication range. Subsequently, sensor nodes receiving $\mathcal{Q}$ set the sink node as their parent nodes (i.e., all data will be forwarded to the parent node) and forward the query to all their neighbors. This procedure executes recursively until all nodes have assigned their parent nodes. Although, this simplistic procedure generates an effective routing scheme between sensor nodes, it may prove highly inefficient as it does not provide any guarantees on the workload incurred on each sensor node.

Our study revealed that predominant data acquisition frameworks [85, 135, 81, 112, 62, 60, 78, 17, 82, 47, 95, 118] have overlooked the important parameter of constructing efficient query routing trees and that negatively impacts the energy efficiency of these systems. In particular, since $\mathcal{T}$ is constructed in an ad-hoc manner there are two major sources of inefficiencies:

- **Data Reception Inefficiencies:** $\mathcal{T}$ structures 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). Note that τ is continuous because it would be very energy-demanding to suspend the transceiver more than once during the interval of an epoch. Consequently, τ is an over-estimate that leads to significant energy waste. For instance, a typical query with an

epoch of 31 seconds over a three-tier network in TinyDB [85], will enforce each sensor to activate its transceiver for listening for as much as 10 seconds while the required τ interval might only be a few milliseconds.

- **Data Transmission Inefficiencies:** $\mathcal{T}$ structures are constructed in an ad-hoc manner and therefore there is no guarantee that the query workload will be distributed equally among all sensors. That leads to data collisions during transmission which represent a major source of energy waste. Consequently, unbalanced trees can severely degrade the network health and efficiency.

Consequently, in designing our architecture, our **first objective** is to automatically tune τ, locally at each sensor without any a priori knowledge or user intervention. Note that in defining τ we are challenged with the following trade-offs: i) Early-off Transceiver: Shall a sensor device power-off the transceiver too early reduces energy consumption but also increases the number of tuples that are not delivered to the sink. Thus, the sink will generate an erroneous answer to a query, and ii) Late-off Transceiver: Shall a sensor device keep the transceiver active for too long decreases the number of tuples that are lost due to powering down the transceiver too early but also increases energy consumption. Thus, the network will consume more energy than necessary which is not desirable given the scarce energy budget of each sensor.

Although, addressing the first objective will significantly reduce 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 tree topology might be unbalanced. Therefore, our **second objective** is to transform the initial query routing tree $\mathcal{T}$ into a near-balanced tree $\mathcal{T}'$ in a distributed manner.

Addressing the first two objectives will rapidly decrease data reception and transmission inefficiencies that enable the generation of energy-efficient query routing trees. 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 frameworks systems [85, 135, 62, 60, 78, 17, 82, 47, 95, 118] focus on producing a complete result set for a query $\mathcal{Q}$. Conversely, a number of studies in data management systems [45, 24, 137, 143, 92, 15, 14, 20, 90] 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 [45] focuses on the subset of most relevant answers for two reasons: i) to minimize the energy cost 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.)

Top-k queries can be used in conjunction with materialized in-network views, in order to further minimize the cost of query execution. A view $\mathcal{V}$ in relational databases is a virtual table that contains the results from an arbitrary query $\mathcal{Q}$ which is evaluated every time $\mathcal{V}$ is referred to. In order to avoid the unnecessary and energy expensive re-execution of $\mathcal{Q}$ it is beneficial to store $\mathcal{V}$ on secondary storage. This introduces the notion of a materialized view. Materialized views have been studied in numerous seminal papers including [19, 30, 27, 74] and have a clear space versus time trade-off: A fully materialized view $\mathcal{V}$ requires more space but less time in evaluating $\mathcal{Q}$, whereas a partially materialized view $\mathcal{V}'$ requires less space but more time in evaluating $\mathcal{Q}$. Materialized views can potentially conserve energy as the application can avoid the expensive re-evaluation of the in-network query $\mathcal{Q}$.

Although a fully materialized view $\mathcal{V}$ maintains the complete results of a query $\mathcal{Q}$, the distributed nature of a sensor network environment, along with its distinct characteristics, imposes

some fundamental limitations to this model. Firstly, maintaining consistency between $\mathcal{V}$ and the underlying and distributed base relation $\mathcal{R}$ (defined by the sensor readings) is very expensive in terms of energy. Thus, we focus on maintaining a subset $\mathcal{V}' \subseteq \mathcal{V}$ that unveils only the k highest-ranked answers for some user defined k thus minimizing the related energy requirements. Secondly, $\mathcal{V}'$ 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.

As a result, our **third objective** is to decrease the energy cost of a query by introducing Top-k queries into the query execution process. Additionally, we further decrease both the number and size of communication packets by incorporating in-network views that reside on local storage.

The KSpot$^+$ architecture presented in this dissertation is composed of three novel components: i) the *Workload Balancing Module*, ii) the *Tree Balancing Module*, and iii) the *Query Processing Module*, which address the aforementioned objectives, respectively. The KSpot$^+$ framework can be utilized in numerous application domains that operate on stationary sensor networks including environmental monitoring [121, 107], big ephemeral events [133, 98], structural monitoring [70], urban monitoring [93], military and security applications [91, 35, 108], health monitoring [99, 84], etc. Additionally, the energy-efficient algorithms that lie in the foundations of each module can be utilized in a plethora of stationary sensor network systems. Below we show their applicability in the context of a Bio-Harvesting Sensor Network [125]. Furthermore, we explain how KSpot$^+$ can be adapted in order to become the foundation of future applications in People-centric Sensing [23, 22] scenarios.

Example 1 - Voltree Climate Sensor Network: Recently, Voltree Power [125] 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. Many Voltree devices form a wireless mesh network which is composed of many inexpensive sensor nodes that collect and report data on temperature, humidity, wind speed and direction. Data collected by the nodes are recursively transmitted from each node to its neighbors (i.e., forming a query routing tree) until these measurements reach a central base station that records the data for further analysis. Such networks have already been deployed by the United States Department of Agriculture (USDA) at five different sites [125]. These networks complement the USDA Forest Service's Remote Automated Weather Stations network. The Voltree Climate Sensor Network deploys Query Routing Tree structures much like its predecessor technology (Battery-powered Wireless Sensor Networks) and thus constructing energy-efficient trees is consequently of major importance.

Example 2 - People-Centric Sensing: People-centric sensing [23, 22], aims to support sensor-enabled applications that engage the general public through the use of their own personal mobile devices. The recent miniaturization and integration of sensors into popular consumer mobile devices (e.g., iPhone, HTC Touch Pro) has enabled a myriad of new sensor based applications for personal, social and public sensing. These applications can be utilized for increasing the sensing coverage of large public spaces and collect targeted information about their mobile device owners. The information can be then uploaded to a centralized database system or exchanged with neighboring mobile devices. What is really important is that these environments allow new levels of data sharing among commodity devices. Specifically, a particular device can request sensor data from available neighboring devices through the establishment of an adhoc link (e.g., through Bluetooth or Wi-Fi).

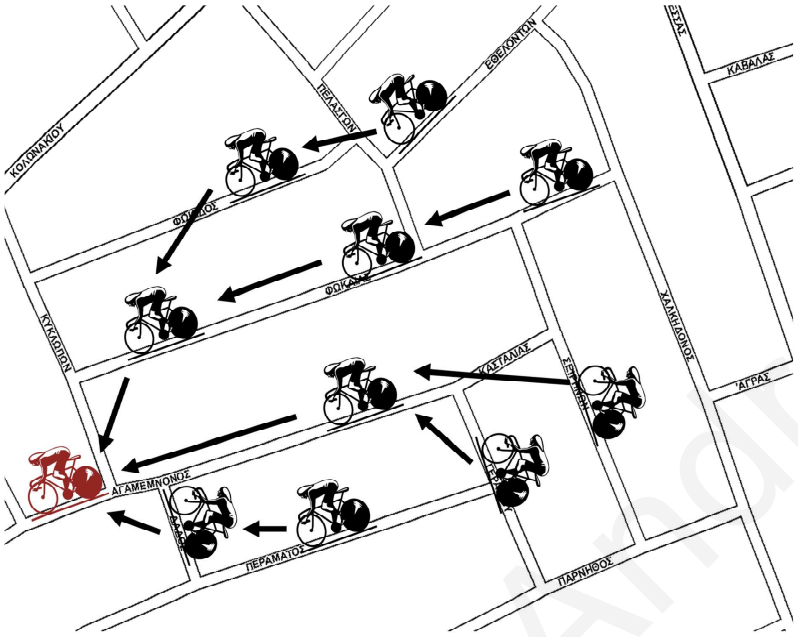

Figure 1-1: People-centric Sensing Example: Cyclists collect data through their sensor equipped mobile devices (e.g., CO_2 level) during their ride. A given cyclist can query its neighborhood by constructing an ad-hoc query spanning tree.

Figure 1-1, illustrates a futuristic people-centric sensing scenario where cyclists journey through the main streets of a city. Each cyclist is equipped with a mobile device that has the ability to interact with its integrated sensors during the ride. The measurements retrieved from these sensors can be used to quantify various aspects of the cyclic performance (e.g., current/average speed, heart rate, burned calories) as well as the environmental conditions (e.g. CO_2 level, car density) during the journey. The continuous sharing of these collected data can be utilized to create collaborative scenarios (e.g., identify routes with low CO_2 levels in the city).

A central component to realize such scenarios is the availability of some high-level communication structure, such as the energy-efficient query routing trees presented in this dissertation. Such structures can serve as a primitive mechanism for percolating query results to nodes that query the network. It must be noted that in People-centric sensing applications, the topology of

the network might change frequently. Consequently, it might be necessary to complement these structures with update mechanisms (e.g., reconstructing the query routing tree periodically either completely or incrementally), although a more detailed exploration of this aspect remains outside the scope of this dissertation.

1.1 KSpot$^+$: A Network-aware Framework for Energy-efficient WSNs

KSpot$^+$ is a data-centric framework that enables energy-efficient query processing on top of balanced and workload-optimized network topologies. It is composed of three basic loosely-coupled modules: i) the *Workload Balancing Module*, ii) the *Tree Balancing Module*, and iii) the *Query Processing Module* that specifically address the objectives mentioned in the previous section. It is important to note that these modules can operate both in isolation and in combination. This modular plug-and-play design allows application developers to experiment under different settings by enabling or disabling each module independently. Below, we briefly describe the key features for each module of the KSpot$^+$ framework:

A. The Workload Balancing Module (described in Chapter 4), investigates data reception/ transmission inefficiencies that occur from the unbalanced assignment of the query workload among sensor nodes. We show that in current data acquisition frameworks, the *waking window* τ of a sensing device is an over-estimate that leads to significant energy waste. To cope with these problems, the Workload Balancing Module employs the *Workload-aware Routing Tree (WART)* algorithm that identifies network bottlenecks by periodically profiling recent data acquisition and attempts to remove them by fine-tuning each sensor node's waking window through an in-network execution of the critical path method.

B. The Tree Balancing Module (described in Chapter 5), investigates the data reception/ transmission inefficiencies that occur from the construction of unbalanced query routing tree topologies. These structural inefficiencies lead to increased data collisions during transmission, which represent a major source of energy waste because they result in energy expensive retransmissions. We show that a centralized approach to cope with this problem requires a significant amount of resources and therefore is not suited for resource-constraint networks, such as WSNs. Conversely, the Tree Balancing Module employs the *Energy-driven Tree Construction Algorithm (ETC)*, which aims to reconstruct the initial query routing tree into a more balanced topology in a distributed and energy efficient manner. To accomplish this, ETC assumes a fixed workload on each sensor node and balances the network based on the optimal branching factor β, which takes into account network-based semantics that is the depth of the query routing tree and the total number of sensor nodes. In Chapter 2, we show that there are other approaches that incorporate query-based semantics and discuss how ETC can be extended to support them. A direct comparison with these approaches remains out of the scope of this dissertation.

C. The Query Processing Module (described in Chapter 6), extends the aptitude of traditional data-acquisition frameworks, like TinyDB and Cougar, by introducing the notion of Top-k queries into the query execution process. Top-k queries prune away tuples that will not participate in the final result thus minimizing the *size* of communication packets. This is facilitated by the *In-Network Views (INT)* algorithm of the Query Processing Module. Furthermore, in order to minimize the *number* of communication packets, the Query Processing Module employs the *Materialized In-Network Views (MINT)* algorithm, which extends the INT algorithm by materializing the results of the previous time instance and suppressing the results of the current time instance if they are identical.

In designing our architecture, we additionally focus on a number of desiderata besides Energy-efficiency (see Section 3.1) including fully Distributed and Autonomous Behavior of each sensor node, Modularity and Resilience in the presence of node and communication failures.

In the section below, we list the main contributions of this dissertation.

1.2 Contributions

The high-level contribution of this dissertation is to integrate network awareness and tuple ranking in data acquisition systems for WSNs. The proposed KSpot$^+$ accomplishes this with novel algorithms that offer contributions at both the network and query level, as well as prototype implementations and evaluations that demonstrate large energy savings in practice. The specific contributions of this dissertation are described below:

1.2.1 Novel Network-aware Framework

The overall contribution of this dissertation is to present, KSpot$^+$, a novel network-aware distributed framework for energy efficient data acquisition in WSNs. KSpot$^+$ seeks to overcome network-related inefficiencies and optimize the energy-efficiency of queries by combining 3 novel modules: i) the *Tree Balancing Module*, which balances the workload incurred on each sensor node by constructing efficient network topologies, ii) the *Workload Balancing Module*, which minimizes data reception inefficiencies by synchronizing the waking windows of each sensor node, and iii) the *Query Processing Module* that manages query execution and additionally employs a novel ranking mechanism that unveils only the k-highest ranked answers thus further minimizing energy consumption.

KSpot$^+$ supports a number of SQL-like queries including simple, filter, top-k and group-by queries as well as a number of distributive aggregates including duplicate-insensitive (e.g.,

`MAX, MIN`) and duplicate-sensitive (e.g., `SUM, COUNT, AVG`). KSpot$^+$ also supports holistic aggregates (e.g., `MEDIAN`) without performing in-network aggregation as it might compromise the correctness of the final result. Although the current version of KSpot$^+$ does not provide support for multi-query execution, Chapter 6 discusses extensions in this direction.

Additionally, KSpot$^+$ focuses on improving the Quality of Data (QoD) [110, 109] for all queries executed over the WSN. QoD can be measured in different dimensions [100] such as freshness, accuracy, completeness, relevancy, etc. In the KSpot$^+$ design, we focus on data accuracy and freshness as we assume that the underlying network layer provides retransmission mechanisms for coping with communication failures. In our experimental evaluation, we measure the number and size of these retransmissions in order to translate them into energy, which is the primary focus of our work. Furthermore, we adopt a number of additional network performance metrics such as the balancing error that quantifies the discrepancy between the initial query routing tree and the more balanced one generated by the KSpot$^+$ framework, and the network longevity defined as the time instance t where the average amount of energy in the network becomes equal to zero. Our network longevity metric, similarly to [123], 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 [109, 110]. This is because we are particularly interested in decreasing the overall energy consumption of the sensor network and not a single node.

Finally, KSpot$^+$ features a highly modular design in which each module can operate both in isolation and in combination with other modules, scales linearly with the addition of new sensor nodes in the network and is resilient to node and communication failures.

1.2.2 Novel Energy-Efficient Algorithms

The modules of the KSpot$^+$ framework presented in this dissertation are founded on energy conscious algorithms that enhance the existing literature. We highlight the main algorithmic contributions below:

- The *Workload-aware Routing Tree (WART) algorithm* [141, 11, 10] (described in Section 4.3) is a network-level algorithm, which profiles recent data acquisition activity within a WSN and discovers bottlenecks using an in-network execution of the critical path method. It then generates a time synchronized topology in which sensor devices identify exactly when and for how long they should have their transceivers powered on thus minimizing the energy spent on idle connections.

- The *Energy-driven Tree Construction (ETC) algorithm* [7, 10] (described in Section 5.3) is a network-level algorithm, which investigates the data reception/transmission inefficiencies that occur from the construction of unbalanced query routing tree topologies. It then attempts to alleviate this problem by reconstructing the initial query routing tree in a manner that minimizes the data transmission collisions between neighboring sensor devices.

- The *In-Network Top-k Views (INT) algorithm* [140, 12, 9] (described in Section 6.3) is a query-level algorithm, which uses a top-k pruning filter that focuses only on the k-highest ranked answers thus significantly minimizing the size of the packets transmitted by each sensor node. We also propose the *Materialized In-Network Top-k Views (MINT)* query-level algorithm (described in Section 6.3), which utilizes the temporal coherence between consecutively acquired sensor readings in order to further minimize the communication overhead (i.e., number of packets) of the INT algorithm.

1.2.3 Open-source KSpot$^+$ prototype implementation

We present the implementation of an open-source prototype implementation that demonstrates the full potential of the KSpot$^+$ framework. The prototype implementation is available at the KSpot+ project website [1] for users to download for free, redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation. The KSpot$^+$ project website additionally contains a number of related recourses including installation instructions, the version of the TinyOS operating system used for the development of KSpot$^+$, all the real datasets used in the experiments and a list of publications related to KSpot$^+$. The prototype demonstrates how the energy-efficient algorithms presented in this dissertation can be integrated into real systems. More specifically, the client-side components of each module of the KSpot$^+$ framework were implemented in nesC, the programming language of TinyOS [61] and can thus be deployed in a number of heterogeneous sensor devices (e.g., Mica, Mica2, MicaZ, Telos, iMote2, etc.). The server-side components of each module were implemented in JAVA because of its platform-independence. Additionally, we augment the description of each module with any considerations or problems that occurred during the implementation phase.

1.2.4 Experimental Evaluations Demonstrating Significant Energy Reductions

The KSpot$^+$ architecture and its three basic modules presented in this dissertation have been deployed and evaluated in actual testbeds. The evaluations reveal that each individual module significantly decreases the energy required for data acquisition. Additionally, we show the integration of KSpot$^+$'s modules can further decrease the energy consumption of a WSN setup. We summarize the major experimental contributions below:

[1] KSpot$^+$ download page, http://www.cs.ucy.ac.cy/ panic/kspot/

- **Workload Balancing Module:** In Section 7.3, we evaluate the WART algorithm of the Workload Balancing Module in isolation using three realistic datasets in order to simulate small-scale, medium-scale and large-scale WSNs. Additionally, we utilize three representative query sets that simulate variable workloads. We show that the Workload Balancing Module significantly decreases the energy consumption by one order of magnitude or more, compared to traditional approaches.

- **Tree Balancing Module:** In Section 7.4, we evaluate the ETC algorithm of the Tree Balancing Module using the same datasets and query-sets mentioned earlier. In addition to the energy consumption metric we also focus on the balancing error, a metric which allows us to quantitatively measure structural inefficiencies. We show that the distributed ETC algorithm offers significant energy savings while in parallel exhibits only a fraction of lower balancing accuracy than the centralized (i.e., optimal) approach.

- **Fusing the Workload and Tree Balancing Modules:** In Section 7.5, we fuse the Workload and Tree Balancing modules and evaluate them in comparison to the performance of the individual modules. We show that the fusion of the two modules decreases the energy consumption by two orders of magnitude compared to traditional approaches and one order of magnitude compared to the individual performance of each module.

- **Query Processing Module:** In Section 7.6, we illustrate the efficiency of the Query Processing Module, using an experimental methodology that utilizes various real and realistic traces. We focus on energy consumption, scalability as well as query semantic parameters. We show that the INT and MINT algorithms of the Query Processing Module decrease energy consumption compared to TinyDB, minimizes both the size and number of packets

transmitted over the network, scales linearly with and prolongs the longevity of a WSN deployment.

- **Overall evaluation of KSpot$^+$:** In Section 7.7, we assess the efficiency of the complete KSpot$^+$ framework. We show that the complete KSpot$^+$ framework can significantly decrease the overall energy consumption of the network.

1.3 Dissertation Outline

The rest of the dissertation is structured as follows: We start in Chapter 2 by presenting a classification of different data acquisition approaches for WSNs and perform a qualitative comparison of their features compared to KSpot$^+$. Related work in the areas of Power Conservation, View Management and Top-K Query Processing is also presented in this Chapter as these provide the foundations for each of the basic modules of the KSpot$^+$ framework. In Chapter 3, we present the KSpot$^+$ framework including its design considerations and specific application domains in which it can be utilized. Chapters 4, 5 and 6 thoroughly describe the basic modules of the KSpot$^+$ framework including their algorithmic foundations. In Chapter 7 we present an extensive experimental evaluation of the KSpot$^+$ framework using a proof of concept application that we developed. Finally, Chapter 8 presents our conclusions, remaining challenges and future directions.

Chapter 2

Related Work

In this Chapter we overview research efforts that relate to the KSpot$^+$ framework. We start by presenting a classification of different data-acquisition approaches and perform a qualitative comparison of their features compared to KSpot$^+$. Next, we provide a thorough study of research works that reside in the areas of i) Power Conservation, ii) View Management, and iii) Top-K Query Processing as these provide the foundations for each of the basic components of the KSpot$^+$ framework.

2.1 Middleware Approaches for WSNs

Traditional middleware frameworks such as the Common Object Request Broker Architecture (CORBA) [34], Java service oriented architecture (JINI) [66], Enterprise Java Beans (EJB) [96], Java Remote Method Invocation (RMI) [65] are considered heavyweight in terms of processor and memory requirements, which renders them highly inefficient for WSN deployments. In this section we present different middleware approaches tailored specifically for WSNs. Additionally we perform a qualitative comparison with the proposed KSpot$^+$ middleware framework. Finally, we summarize the results of our analysis at the end of this section in Table 2-1.

Data-centric Middleware Frameworks:

Cougar [135], considers the network as a virtual relational database and 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 [135], 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 [10] 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 [85], is one of the most popular data acquisition frameworks developed for TinyOS. Like Cougar, it is a data-centric middleware platform 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 query routing tree, 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 [10]. TinyDB employs Tiny Aggregation (TAG) [86], for energy efficient in-network aggregation of sensor results. KSpot$^+$ extends

the TinyDB in-network aggregation by enabling support for top-k queries that further minimize energy consumption by reducing the size and number of packets transmitted to the network.

Temporal coherency-aware in-Network Aggregation (TINA) [109] is another data-centric middleware framework developed at the University of Pittsburgh. TINA 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 minimizes both the size and number of transmitted packets. Additionally, it influences the construction of the query routing tree by incorporating query-based semantics using the Group-aware Network Configuration (GANC) [110] component. TINA achieves significant energy savings while maintaining specified quality of data. The temporal coherence tolerance settings are either defined by the user or dictated by the network in cases where the network cannot support the current tolerance level. 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) [112], is a middleware framework developed at the University of Delaware. SINA provides a set of programming abstractions that enable application designers to view the network as a collection of distributed objects. SINA consists of three basic components: i) Hierarchical Clustering, which forms a hierarchical network through clustering, ii) Attribute-based Naming, which assigns names to the sensor nodes according to their locally stored attributes and measurement capabilities, and iii) Location Awareness, which employs both absolute (i.e., GPS) and relative (i.e., RSSI indicators) localization techniques to determine the positions of the sensor nodes. As a result of the Attribute-based Naming component, the network can be viewed as a collection of datasheets where each sensor maintains a single datasheet locally that stores its measurements. This 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) [48, 49] is a middleware framework developed at the University of Manchester. SNEE 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 middlWare (DsWare) [81, 138], 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 on sensor-generated data. The filtering mechanism provides approximate instead of exact values in order to decrease communication overhead. Overall, DsWare seeks to exploit trade-offs between resource consumption and application quality in WSNs. This is accomplished by sacrificing accuracy for energy conservation. In KSpot$^+$, we minimize the overall energy consumption of the network without sacrificing the results of sensor nodes. Currently, DsWare is still in its infancy as only a fraction of the proposed components have been implemented and evaluated.

Application-centric Middleware Frameworks:

The Middleware Linking Applications and Networks (Milan) [60] is an application-driven middleware framework developed at the University of Rochester. Its high level application interface enables application designers to specify their QoS requirements inside the sensor network application code. These requirements are then translated by the middleware into protocol-specific commands that are able to reconfigure the network topology based on quality metrics (e.g., sensor remaining energy, channel bandwidth). 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, Milan unlike KSpot$^+$ does not consider the workload incurred on each sensor node, which may result in serious data reception inefficiencies.

The MidFusion [62] is a middleware architecture that aims to facilitate information fusion in sensor networks. MidFusion discovers and selects the best set of sensor nodes (w.r.t. energy efficiency) that can respond to an application request. This is accomplished by maintaining profiles (e.g., energy reserves) for each sensor node and is done in a transparent manner to the application. 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 [95] 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 [118] is another 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 to facilitate multi-hop communication. Although, both Aware [95] and Mires[118] 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.

Virtual Machine-based Middleware Frameworks:

Maté [78] is a virtual machine-based middleware developed at University of California-Berkeley. It acts as a TinOS byte interpreter component that resides on top of several system components including communication, sensorboard and storage. Maté's high level interface allows complex

programs to be written with minimal code, reducing in this way the energy for transmitting these programs to the sensor network. It is important to note that this high-level instruction set can transmit a message to the network using a single instruction. Application code is broken in capsules of 24 instructions, which allows a capsule to fit into a single TinyOS packet. Larger programs can be composed of multiple capsules. Updating capsules with newer versions is facilitated by augmenting each capsule with version meta-data. Maté was built by taking into account the system constraints of the Rene and Mica sensor devices and thus all of its sub-components must fit in 1KB of RAM and 16 KB of instruction memory. Maté can be ported to a number of sensor devices that run the TinyOS operating system including, Rene, Mica1, Mica2, MicaZ, TelosB. Unfortunately, the current implementation does not support other operating systems (e.g., Contiki, LiteOS).

MagnetOS [17] is a virtual machine-based middleware developed at Cornel University. It focuses on network-wide energy management, which can only be achieved by a distributed middleware approach. To accomplish this, MagnetOS provides a single system image of a unified Java virtual machine across heterogeneous sensor nodes of the WSN. Regular Java applications are transparently translated into distributed objects and ported to each sensor virtual machine. This enables applications to view the entire network as a single unified Java virtual machine. MagnetOS is comprised of two basic components: i) the server-side *static partitioning service* component, and ii) the client-side *dynamic runtime service* component. The *static partitioning service* is responsible for rewriting the byte-code of regular Java applications into objects or modules than can be ported to each distributed virtual machine (i.e., sensor node). Each object is further augmented with additional information in order to retain the original application's semantics regardless of the fact that the object will be distributed across many sensor nodes. The *dynamic runtime* provides services for object management including creation, invocation, placement, and migration, which

utilize a number of power aware algorithms. Additionally, it provides the means for application developers to explicitly adjust object placement and migration thus enabling experimentation under different settings. Both Mate [78] and MagnetOS [17] rely on a built-in ad hoc routing algorithm that may produce unbalanced and workload inefficient topologies.

Agent-based Middleware Frameworks:

Impala [82] is an agent-based middleware architecture that has been designed mainly to be a part of the ZebraNet [107, 83] sensor network but a prototype has also been developed for HP/Compaq iPAQ Pocket PC handhelds. It focuses on application modularity, adaptivity and reparability. To accomplish this, Impala's middleware layer utilizes three agent-based components: the Application Adapter, the Application Updater, and the Event Filter. The Application Updater facilitates Over-The-Air-Programming (OTAP) by receiving and installing application updates. However, it is important to note that only one application can be online at a time. The Application Adapter is responsible for dynamically adapting running applications in order to improve performance, energy-efficiency and robustness. On the one hand this ensures a high degree of energy-efficiency without user-intervention in a transparent manner. On the other hand, it prevents application designers to experiment under custom settings.

Agilla [47] is another mobile agent-based middleware specifically suited for applications coping with ephemeral events. The main component of the architecture is the Agilla Engine, which is responsible for the concurrent execution of all mobile agents residing on a sensor node. Additionally, it facilitates agent migration via the Agent Sender and Agent Receiver components. The network is partitioned into tuple-spaces where agents interact and coordinate. The Agilla platform has a low memory footprint and is well suited for resource-constrained networks such as WSNs. On the other hand, the coordination of mobile agents on a large-scale network can

Table 2-1: Classification and Comparison of Middleware Approaches for WSNs

Middleware Approach	Key Features	Energy-aware	Workload Opt.	Topology Opt.	Top-k Support	Approx. (A) /Exact (E)	Heterogeneity	Scalability	Mobility
Data-centric									
TinyDB [85]	SQL syntax, lifetime/event-based queries, In-network aggregation, Semantic routing trees	Y	Y	N	N	A/E	TinyOS	Y	N
Cougar [135]	SQL syntax, Virtual relational database, centralized optimizer	Y	Y	N	N	A/E	TinyOS	Y	N
TINA [109]	temporal coherence filters, group aware network configuration	Y	N	Y	N	A/E	TinyOS	Y	N
DsWare [81]	SQL syntax, real-time semantics, event-detection	Y	N	N	N	A	N	Y	N
SNEE [48]	rich, expressive language, scheduling of different workloads	Y	Y	N	N	A/E	N	Y	N
SINA [112]	Virtual spreadsheet database, Attribute-based naming, Hierarchical Clustering	Y	N	N	N	A	N	Y	N
KSpot$^+$	SQL syntax, in-network aggregation, top-k, topology balancing, workload balancing	Y	Y	Y	Y	A/E	TinyOS	Y	N
Application-driven									
Milan [60]	topology adaptation	Y	N	Y	N	A/E	N	N	N
MidFusion [62]	information fusion, sensor agents	Y	N	N	N	A/E	N	Y	N
Virtual Machine									
Maté [78]	bytecode interpreter, OTAP, code capsules	Y	N	N	N	A/E	TinyOS	Y	Y
MagnetOS [17]	Java VM, OTAP, Single System Image	Y	Y	N	N	A/E	Java	Y	Y
Publish-Subscribe									
Mires [118]	aggregation service, high-level interfaces	Y	N	N	N	A/E	TinyOS	N	N
AWARE [95]	sensor network & UAC coordination	Y	Y	N	N	A/E	TinyOS	N	N
Agent-based									
Impala [82]	adaptivity, reparability, OTAP, single executing application	Y	Y	N	N	A/E	Y	Y	N
Agilla [47]	self-adaptation, tuple-space abstraction, location addressing	Y	Y	N	N	A/E	TinyOS	N	Y

seriously hamper the overall performance of the network. Furthermore, unlike KSpot$^+$, Agilla does not consider topology inefficiencies, which may result in unavoidable hotspots during agent communication and coordination.

In conclusion, the majority of proposed middleware approaches employ mechanisms for reducing the overall energy consumption of the network thus increasing the longevity of a WSN as shown in Table 2-1. 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. Table 2-1, shows a feature comparison of all presented approaches. 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.

2.2 Power Conservation

In this section, we present a number of power conservation approaches that we have utilized as input in the design of the Tree and Workload Balancing modules of the KSpot$^+$ framework.

Power conservation is one of the key design features in all middleware approaches proposed for WSNs and numerous mechanisms have been proposed virtually at all layers of the traditional layered sensor communication stack. All these approaches attempt to decrease the energy consumption with two basic techniques: i) by turning the radio transceiver to low-power consumption mode during periods of inactivity, and ii) by improving the sensor node's operation (e.g., voltage scaling, employing multiple power levels). Most of these techniques are complementary to the techniques described in this dissertation while the rest come with their own trade-offs as we will show shortly.

In this section, we present an elaborate overview of techniques that decrease communication related power consumption in WSNs, using the ISO/OSI stack [73]. Such a categorization allows one to accurately capture the main focus and limitations of each presented technique. We shall also refer to cases of cross-layer optimizations individually. For the remainder of this section, we will present the universe of techniques in a bottom-up manner, starting from the physical layer and moving up to the application layer where KSpot$^+$ belongs to. We omit the Presentation and Session layers of the typical ISO/OSI stack as none of the presented techniques addresses these

layers specifically.

Physical Layer techniques:

This layer relates to the low-level sensor device hardware (circuitry, MCU, transceiver, etc) thus the opportunity for software-level power management is fairly limited. Yet, there are a few works [21, 57, 117] that look at individual and local power management optimizations.

Examples of these techniques are the *Dynamic Voltage Scaling (DVS)* [21] and *Embedded power supply for low power Digital Signal Processor (DSP)* [57] which are effective techniques for reducing the energy consumption of the CPU. The goal of these approaches is to adapt the processor's power supply and operating frequency to match any given computation load without degrading performance. *Dynamic Power Management (DPM)* [117] is another work that utilizes different power models to shut down various components (e.g., radio transceiver, CPU) when these are not required to operate. All of the above techniques, and generally any local power conservation mechanism at the physical layer, are supplementary to the KSpot$^+$ middleware framework.

MAC Layer techniques:

The Medium Access Control (MAC) layer facilitates the transfer of messages to and from the physical layer. Most of the protocols developed for the MAC layer deploy explicit mechanisms to avoid collisions when multiple sensor nodes attempt to access a shared channel. Most of the sensor network related works presented in this layer [119, 116, 136, 97] minimize energy consumption by minimizing collisions and overall usage of the shared access medium.

The *Coordinated Power Conservation algorithm (CPC)* [119] is an example of a MAC-layer power management protocol that coordinates the sleeping intervals of sensor nodes with the aid of a backbone. CPC starts out by selecting a set of backbone nodes as CPC servers. Next all CPC

clients that run on non-backbone nodes, request to turn the transceiver of the sensor node off when there is no communication activity in order to conserve power and extend network lifetime. CPC servers running on backbone nodes serve as coordinators to synchronize sleeping schedules of nodes within their coverage areas. The intuition of turning off the radio transceiver during periods of inactivity is very similar to the WART algorithm of the KSpot$^+$ Workload Balancing Module. However, CPC servers coordinate in a distributed manner without obtaining any global information from the base station, which does not provide a universal view of the system. Furthermore, the scheduling on WART is based on the query workload incurred on each sensor node while CRC misses the inclusion of such high-level semantics.

Power-aware Multi-Access Protocol with Signaling (PAMAS) [116], is another MAC-layer power management protocol that utilizes two independent radio channels in order to avoid overhearing among neighboring nodes. PAMAS does not attempt to reduce idle listening which is a major disadvantage, as nodes have their radio enabled during periods of inactivity reception. However, battery power is saved by intelligently turning-off sensor nodes that are not in active transmission. On the other hand, the popular *Sensor-MAC (S-MAC)* [136] protocol, utilizes a synchronization scheme that allows sensor nodes to realize periodic listening and sleeping during busy periods (i.e., when transmission from other nodes is detected). Furthermore, S-MAC consists of two additional components that handle: i) collision and overhearing avoidance by allowing sensor nodes receiving control packages not destined to them go to sleep, and ii) message passing by segmenting long messages into smaller ones and transmitting in a burst (i.e., RTS/CTS control messages are not used for each fragment). S-MAC has been further enhanced in [97] to minimize the end-to-end delay. Both PAMAS and S-MAC achieve high energy savings by allowing sensor nodes to sleep periodically. However, none of these approaches considers the underlying topology of the sensor network, intra-sensor relationships and high-level query semantics. In particular,

these techniques do not consider the workload of a continuous query, rather they assume a random variable workload. In the Tree Balancing Module of KSpot$^+$, we minimize collisions by constructing a near balanced routing tree through the ETC algorithm. Nevertheless, since the S-MAC protocol has been successfully integrated in TinyOS [136] as one of the primary MAC protocols, these techniques extend the power management capabilities of KSpot$^+$ inherently.

Sensornet Protocol (SP) [102], introduces a unified link level abstraction that is part of the sensor network architecture proposed in [36]. Specifically, SP provides shared neighbor management and message pool interfaces that allow network protocols to exchange messages efficiently and choose neighbors wisely without concentrating on link specifics. To accomplish this, these interfaces encapsulate the mechanisms of the particular link and physical layers that operate below SP. The authors show that various link-layer protocols can be expressed in terms of SP and subsequently mapped efficiently to various different link-level power management mechanisms.

Network Layer techniques:

This layer is responsible for delivering packets from a source node to a destination node through some routing mechanisms. In WSNs, routing is accomplished using multi-hop messages, thus many mechanisms in this layer attempt to discover optimal routing paths for energy efficient delivery of messages through intermediate hosts [53, 37, 132, 59].

The *Power-Aware Routing (PAR)* [53] technique proposes a routing policy that balances the overall power in the network by discovering routes that consume the least possible energy. Since in a non-uniform network, the majority of nodes do not consume power in an identical fashion, PAR favors nodes with generous power reserves. Another technique is the *Minimum Connected Dominating Sets (MCDS)* routing algorithm [37] which employs a virtual backbone that provides

shortest paths for routes as well as route updates in cases of node movements in order to minimize the overall energy required for routing multi-hop packets.

Both PAR and MCDS approaches assume an a priori established query routing tree. Any optimizations suggested by both approaches do not alter the state of the query routing tree. On the other hand, the KSpot$^+$ differs from these approaches as the Tree Balancing module reconstructs a near-balanced tree in order to minimize collisions prior to any further optimizations. Certain modules of PAR and MCDS (e.g., shortest path discovery) can be used in conjunction with KSpot$^+$ in order to achieve even more energy savings.

In *Modular Network Layer* [44] the authors decompose the network layer into smaller components that can be used by several protocols in parallel. This network layer operates on top of the popular Sensornet link-layer Protocol [102]. The intuition behind their approach is that the majority of network protocols have many commonalities. Encapsulating these commonalities and exposing them as service interfaces enables faster development of new protocols and run-time sharing of components. The authors evaluate their approach and find that Modular Network Layer can reduce both the memory and code of network protocols that run concurrently. Consequently, this work is supplementary to KSpot$^+$, as our protocol could have been implemented using this intermediate framework rather than in a standalone.

Transport Layer techniques:

The transport layer is responsible for the transfer of messages between two or more end systems using the network layer. One of the main objectives of the transport layer is the reliable and cost effective delivery of transferred messages between applications. The evolution of the techniques in this layer has been severely hampered by the fact that sensor networks feature node failures and collisions making reliable and cost effective communication often impossible.

One of the few works that addresses the above issues is the *TCP-Probing* [124] communication protocol, which introduces the concept of a probe cycle instead of standard TCP re-transmissions, congestion window and threshold adjustments. During probe cycles, data transmission is suspended and only probe segments are sent. The proposed scheme achieves high throughput performance whilst in parallel decreases the overall energy consumption for transmission. This is done without damaging the end-to-end characteristics of TCP. *Flush* [69] is another transport layer protocol for multi-hop wireless networks. Flush provides end-to-end reliability, reduces transfer time and adapts to time-varying network conditions. To accomplish these properties, Flush uses end-to-end acknowledgments, implicit snooping of control information and a rate-control algorithm that operates at each hop along a flow.

In contrast to the probe cycles of TCP-Probing and end-to-end acknowledgments of Flush, KSpot$^+$ uses the notion of a waking window during which a sensor may transmit a message repeatedly until it is successfully received by the recipient. The aforementioned techniques would introduce further delays as well as more energy waste since the sensors would have to exchange more messages in order to synchronize.

Application Layer techniques:

The main objective of this high level layer is to exploit the semantics of the network or application and low-level data in order to optimize the network structure among nodes and boost power management. Consequently, this layer has implicit interactions with lower levels of the communications stack (often referred to as cross-layer optimizations [4]). The techniques in this category can roughly be classified in the following categories: i) local techniques, in which low-level data semantics dictate the reaction of the application, and ii) cluster-based techniques, in which the reaction of the application is dictated by the cluster semantics (e.g., network proximity).

Application-Driven Power Management for Mobile Communication (ADPM) [72], is an example of an application-layer technique that enables the dynamic power configuration of the communication device. The goal of this work is to determine the appropriate tradeoff for battery lifetime versus response delay, while adjusting the sleep duration of the communication device. ADPM, just like the techniques in the physical layer, which adjust the power supply of the processor, is supplementary to our approach. Adaptive Energy-Conserving Routing (AdECoR) [131], is another application layer protocol that utilizes two algorithms for routing in resource constrained WSNs. The intuition behind this approach is that although switching-off the communication device may result in energy conservation it may also introduce delays in the network. AdECoR attempts to find a tradeoff between energy conservation and latency by utilizing application-level information. AdECoR differs from KSpot$^+$ as its application-level information does not include the high level query semantics used in KSpot$^+$. Furthermore, the concept of introducing delays in order to conserve power is not acceptable in KSpot$^+$ as we assume that queries have specific response time requirements that must be met. In [139], the authors propose a Data Transmission Algebra (DTA) that allows a centralized query optimizer to utilize lower layer communication protocols in scheduling sensor database queries. This generates query routing trees to maximize collision-free concurrent data transmissions. The Workload Balancing Module of the KSpot$^+$ framework focuses on minimizing the wake-up time of each sensor node and does not consider creating collision-free topologies. In KSpot$^+$ this is achieved by utilizing the Tree Balancing Module prior to the operation of the Workload Balancing Module. DTA can be utilized in conjunction with the Workload Balancing Module to minimize both the collisions and the wake-up time of the sensor nodes.

A query routing tree can be considered as a query evaluation plan. This observation motivated us to develop a Data Transmission Algebra (DTA) that allows a query optimizer to generate query routing trees to maximize collision-free concurrent data transmissions.

The second class of application layer techniques includes those techniques that use clustering mechanisms [132, 59, 29]. An example of these techniques is *Geographical Adaptive Fidelity (GAF) [132]*, which obtains location information using the Global Positioning System (GPS) in order to connect sensor nodes to a virtual grid (i.e., a semantic overlay based on geographical proximity). It then saves energy by keeping sensor nodes located in a particular grid area in sleeping state. The sleeping schedule uses a turn-based approach that aims to balance the load incurred on each sensor. *Energy-Efficient Communication Protocol for Wireless Micro-Sensor Networks (LEACH)* [59] is another cluster-based technique that minimizes overall energy consumption in WSNs by rotating the cluster head nodes in a random manner. This rotation allows the distribution of the energy load evenly among the sensor nodes in the network without draining the energy resources of an individual sensor node. One final cluster-based protocol is *SPAN* [29], which builds on the observation that when a region of a shared-channel has a sufficient density of nodes, only a small number of them needs to be present at any time to forward traffic for active connections. To accomplish this, SPAN utilizes a distributed, randomized algorithm that allows sensors to make local decisions as to when sleeping is appropriate.

GPS and SPAN, like KSpot$^+$ take advantage of global information to preserve energy. Both approaches switch-off some sensors based on some application-level parameters and force other sensors to seek alternate routing paths. However, switching-off some sensors means that they cannot participate in a given query and as a result, valuable results may be lost even for shorts period of time. LEACH differs from our approach since in KSpot$^+$ all nodes participate in a given query and none plays a separate role (e.g., cluster head) nor has more energy reserves than others.

The recent trend in wireless sensor networks is to interconnect existing sensor networks through dedicated web-based or geospatial-based information systems. Such systems operate over different operating systems, communication protocols and applications. To address the problem of communicating with such diverse sensor network systems, the works in [1, 94] have developed middleware systems that enable integration and management of many WSN sites. Additionally, multi-tier sensor systems like TENET [51], that aim to combine the low-power sensor devices we discuss in this dissertation with powerful 32-bit nodes (e.g., Stargates [35] or ordinary PCs), are another direction in sensor networks optimization. Yet, all these techniques are complementary to the approaches we outline in this dissertation as our techniques structure efficient and well-formed WSN deployments while middleware techniques utilize these as a building block.

2.3 View Management

In this section, we present a number of view management approaches that we have utilized as input in the design of the Query Processing module of the KSpot$^+$ middleware framework.

View Management has been an area of great contributions over the last decades [19, 30, 27, 74]. 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 [127, 126]. 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 [134, 87], 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 [40]. 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 [130] 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 [75] 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 [75] can be used in conjunction with the INT and MINT Views algorithms of the KSpot$^+$ Query Processing Module 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 [38]. 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 [76], 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 dissertation.

2.4 Top-k Query Processing

In this section, we present a number of top-k query processing approaches that we have utilized as input in the design of the Query Processing module of the KSpot$^+$ framework.

Top-k Query Processing has been studied in a variety of contexts including middleware systems [45, 46], web accessible databases [20, 90], stream processors [15], peer-to-peer systems [14] and other distributed systems [24, 144, 143]. It has been shown in numerous studies [45, 24, 20, 144], 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"*, retrieves 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 [45, 143, 24, 137] have been proposed for top-k query processing with the Threshold Algorithm (TA)[45] being the most predominant. In [24] the authors develop a three phase protocol (TPUT) which decreases the number of remote accesses in large networks. The TPAT algorithm [137] extends the TPUT algorithm by exploiting data distributions among nodes to improve pruning. In [143], 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 [92, 15, 143] that provide approximate answers. In [143], the authors propose the UB-K and UBLB-K algorithms that return upper/lower bounds instead of exact answers. In [15] 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 [92], 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 [115].

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* [109, 110]. 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 [15]. 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 [39].

In [129] 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 [14, 67], 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.

Other approaches range from efficient join processing algorithms in sensor networks [49, 31, 120, 68], to the underlying data management layer [28, 3, 41, 142] and network optimization [88, 32, 18, 11], among others. Yet, these approaches focus either on a different system model or a different problem formulation, than the work we present in the Query Processing Module of the KSpot$^+$ framework.

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.

Chapter 3

The KSpot$^+$ Framework

KSpot$^+$ is a novel 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.

KSpot$^+$'s modular design allows application designers to easily integrate new features into the design as well as experiment under different settings. KSpot$^+$'s modules can function individually or in cooperation according to the requirements of the application. Additionally, they can operate in conjunction with a variety of routing and query protocols (see Chapter 2).

In this Chapter, we provide an overview of the KSpot$^+$ architecture, its design principles and some information on its prototype implementation.

3.1 Design Goals

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

- **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 for our architecture. 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 [86, 85, 9, 121, 10]. To accomplish this, any software (i.e., OS, middleware, application) 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 [5]. Consequently, we consider scalability an extremely desirable property of our framework as it ensures that the performance of 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.

- **Resilience in the presence of Failures**: WSNs are typically prone to imminent node failures triggered from 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. In our experiments, we show that the algorithms that operate in each of the KSpot$^+$ modules have proven to work both resiliently and efficiently in the presence of failures.

3.2 KSpot$^+$ Framework Architecture Design

The KSpot$^+$ framework lies between the server tier and the data tier as illustrated in Figure 3-1. It follows the client-intercept-server [101] model and it is composed of server-side and client-side components. 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

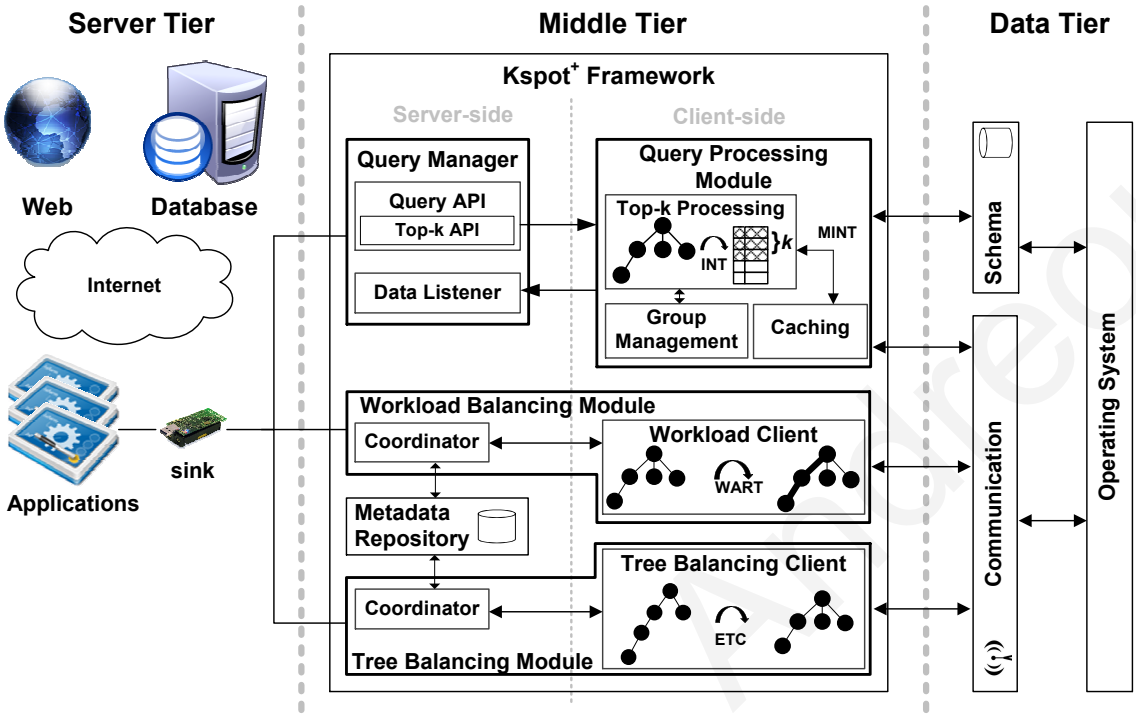

Figure 3-1: KSpot$^+$ framework architecture. The KSpot$^+$ client combines 3 novel components: the *Tree Balancing Module*, which balances the sensor network topology, the *Workload Balancing Module*, which balances the workload of each sensor node, and the *Query Processing Module*, which handles query execution and facilitates Top-k query processing.

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 meta-data 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, described in Chapter 4) and the optimal network branching factor (β, described in Chapter 5) 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 Chapter 4), investigates data reception/transmission inefficiencies that occur from unbalanced assignment of the query workload amongst sensor nodes. It utilizes the WART algorithm for the dynamic adaptation of the waking windows of each sensor node. The server-side Workload Balancing Coordinator starts by profiling recent data acquisition activity and then identifies the bottlenecks of the network through an in-network execution of the Critical Path Method. The acquired global critical path value (ψ) is stored in the *Meta-data Repository*. Immediately afterwards, the coordinator transmits the ψ value to the network recursively fine-tuning each sensor node's waking window τ locally through the Workload Client.

  In particular, the Workload Balancing process consists of three phases: i) the *Construction Phase*, where the sink node constructs a new query routing tree or utilizes an established one and then queries the network for the total critical path value ψ; ii) the *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) the *Adaptation Phase*, where each sensor node adapts its waking window according to new workload variations.

  A more elaborate description of the Workload Balancing Module along with its algorithmic foundations will be thoroughly presented in Chapter 4.

- The **Tree Balancing Module** (described in detail in Chapter 5), identifies structural inefficiencies in the initial query routing tree 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.

In particular, the Tree Balancing process consists of two phases: i) the *Discovery Phase*, where the sink node calculates the optimal branching factor β of the initial ad hoc network topology (β is stored in the *Meta-data Repository*); and ii) the *Balancing Phase*, where the sink disseminates β to the network and sensor nodes recursively conduct a number of local rearrangements to their parent assignment. This results in a more balanced network topology.

A more elaborate description of the Tree Balancing Module along with its algorithmic foundations will be thoroughly presented in Chapter 5.

- 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 [85, 135, 112, 81] 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 3.3.

- The **Query Processing Module** (described in detail in Chapter 6), is responsible for query execution as well as a number of services including group management and caching. Regular SQL queries are executed using the built-in query mechanism while standard Top-k queries are executed using the INT algorithm. In the case of Top-k queries that involve logical groups, these are coordinated through the group management component (see Section 3.4). The Query Processing Module also utilizes a data caching mechanism that in cooperation with the INT mechanism exploits temporal coherency between results of consecutive time instances (MINT).

In particular, the Query Processing process consists of three phases: i) the *Query Dissemination Phase*, where the sink node propagates a query to the network; ii) the *Processing Phase*, where 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; and iii) *Data Acquisition Phase*, where each sensor node recursively transmits its results to the network until they reach the sink node.

A more elaborate description of the Query Processing Module along with its algorithmic foundations will be thoroughly presented in Chapter 6.

- 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 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** 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 and application requirements. More details are presented in Section 3.4.

The KSpot$^+$ framework is composed of loosely-coupled modules that communicate in a message-passing manner. As illustrated in Figure 3-1, 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.

KSpot$^+$ modular design allows application designers to easily integrate new features into the design as well as experiment under different settings. KSpot$^+$'s modules can function individually or in cooperation[1] according to the requirements of the application. Additionally, they can operate in conjunction with a variety of routing and query protocols (see Chapter 2).

The KSpot$^+$ modules can be roughly classified by their awareness on the routing topology (topology-aware), the scheduling of the waking windows (schedule-aware) and the executed query (query-aware). The Tree Balancing Module is topology-aware but schedule and query-agnostic. The Workload Balancing Module is schedule-aware but topology and query-agnostic and the Query Processing Module is query-aware but topology and schedule-agnostic. Consequently, each module is complimentary to each other, an argument that we experimentally quantify in our Experimental Series 4 and 6. In Experimental Series 4 we present the combination of the Tree and Workload Balancing Modules (i.e., topology and schedule-aware but query agnostic) and show how these complimentary modules benefit from their cooperation. Finally, in Experimental Series 6 we present the combination of all modules (i.e., topology, schedule and query-aware), which demonstrate the additive effect of all modules on increasing the sensor network longevity.

[1]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 assignment between sensor nodes.

In the following sections, we present the services provided by the KSpot$^+$ framework. A more elaborate description of the Workload Balancing, Tree Balancing and Query Processing modules along with their algorithmic foundations will be thoroughly presented in Chapters 4, 5 and 6 respectively.

3.3 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$^+$ middleware architecture 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 mentioned 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 mentioned in the GROUP BY statement may additionally refer to a logical group assignment, as described in the next Section. 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.,

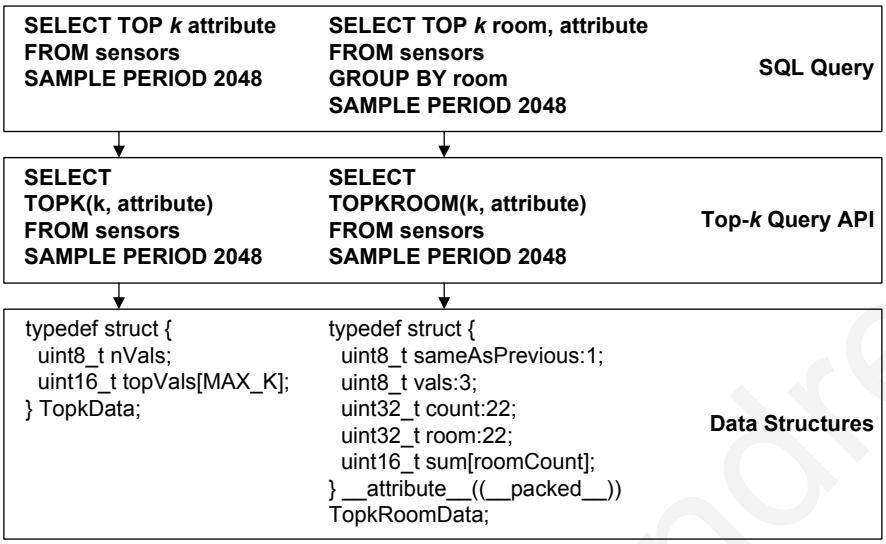

Figure 3-2: The KSpot$^+$ Top-k query syntax and their respective client-side data structures.

`MEDIAN`), thus all tuples have to be transmitted to the sink before the query can be executed. The benefits of the KSpot$^+$ framework can be seen when executing single-relation queries with distributive aggregate functions. 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.

Note that 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. 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. The two forms of Top-k queries supported by the KSpot$^+$ Top-k Query API along with their transformations from query text to data structures are illustrated in Figure 3-2.

3.4 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` 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 of 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, beacons [104]), which may not be always available, and also increases the overall message complexity. In KSpot$^+$, group management is accomplished through attribute-based naming with the following commands:

i. The server issues a *create logical group* request using the `createGroup(String groupid, int NodeID, int value)` statement. This is transformed into a group request and injected to the network. Note that the `String groupid` is transformed by a hash function into an integer value (`int groupid`) and the mapping is stored in a server-side mapping table. This is done in order to minimize the size of the command packet.

ii. As soon as the sensor node with matching ID receives the group-request it executes the `setGroup(int groupid, int value)` command and stores it in a dedicated table in main memory.

iii. when a sensor receives a `Group-By` query which involves logical group attributes, it aggregates the results based on the group attribute, which is obtained by the `getGroup(int groupid)` statement.

The Group Management API also supports the server-side `deleteGroup(String groupid, int NodeID)` command and the client-side `deleteGroup(int groupid)` for deleting a logical group.

3.5 Target Application Domains

The KSpot$^+$ framework can be utilized in numerous application domains including environmental monitoring [121, 107], big ephemeral events [133, 98], structural monitoring [70], urban monitoring [93], military and security applications [91, 35, 108], health monitoring [99, 84], etc. Below, we describe two representative application domains that KSpot$^+$'s features can be very beneficial.

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. However, in the case of a crisis, even delays of the order of milliseconds may not be acceptable as fire officers and associated authorities need to dynamically collaborate in realtime according to the evolution of the crisis in order to timely react upon detected fire and flooding risks. KSpot$^+$ alleviates this problem by focusing only on the k-most important events. This minimizes the time required to forward the

results to the querying node (i.e., decreased size of packets because of in-network pruning) as well as the time required to process the results (i.e., only k results are returned to the sink station).

Another 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 means for energy replenishment including solar panels, bio-harvesting [125], 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.

Big Ephemeral Events

International events (e.g., FIFA World Cup, World Expo) usually attract millions of participants during a very limited period of time. 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, ownership) and/or implicit (e.g., location, energy, capabilities).

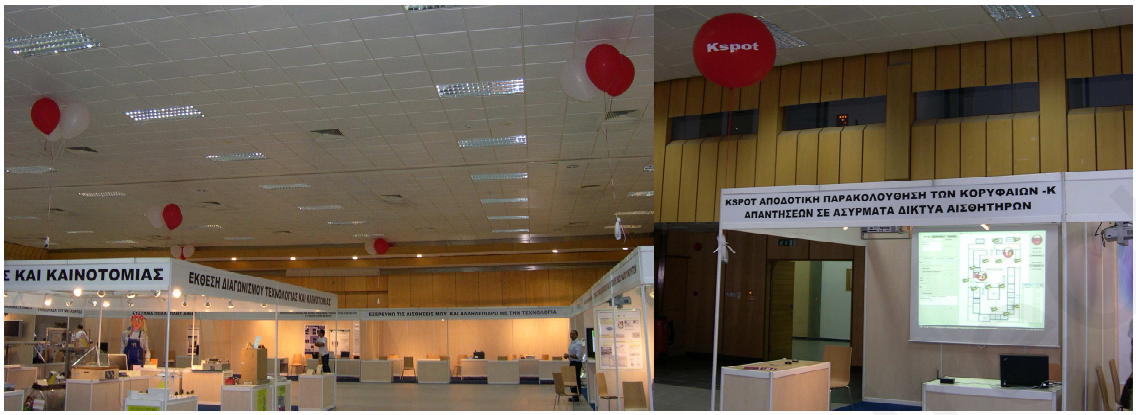

Figure 3-3: KSpot^{+} prototype implementation deployment during the event "Researcherś Evening" at the Cyprus International Fair in 2009. Sensor nodes were placed over the pavilions using helium balloons (left) and measured sound level. Sink station (right) was connected with a laptop computer that projected the pavilions with the highest noise level.

In this context, we have deployed a preliminary version of the KSpot^{+} framework prototype implementation during the event "Researcher's Evening" at the Cyprus International Fair in 2009[2] . Figure 3-3 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 4 seconds. 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.

[2]http://crpf.metacanvas.com/EL/int_cooperation/night/index.html

3.6 Proof of Concept Application

In order to assess the practicality and usability of the proposed KSpot$^+$ framework, we have developed a prototype implementation that demonstrates the full potential of the KSpot$^+$ framework. The components of the prototype are implemented in JAVA (server-side) and in nesC (client-side). We have selected nesC (TinyOS/TinyDB) for the implementation of the client-side components 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 [42] or LiteOS [25].

We have created a server-side desktop application, named the KSpot$^+$ GUI, in order to demonstrate the usability of our middleware as well as the applicability of our approach. The KSpot$^+$ GUI is used for: i) configuring the number of sensors and assigning them to logical groups, ii) execute Top-k and standards queries, and iii) for displaying the query results in a manner that highlights the ranking properties of the executed query (in the case of Top-k queries). In particular, the KSpot$^+$ GUI consists of three panels (see Figure 3-4):

i. The *Configuration Panel* (Figure 3-4, 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 in the same logical group. Additionally, the user can assign values to the $|\gamma|$ descriptors mentioned in Section 6.5.2. 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.

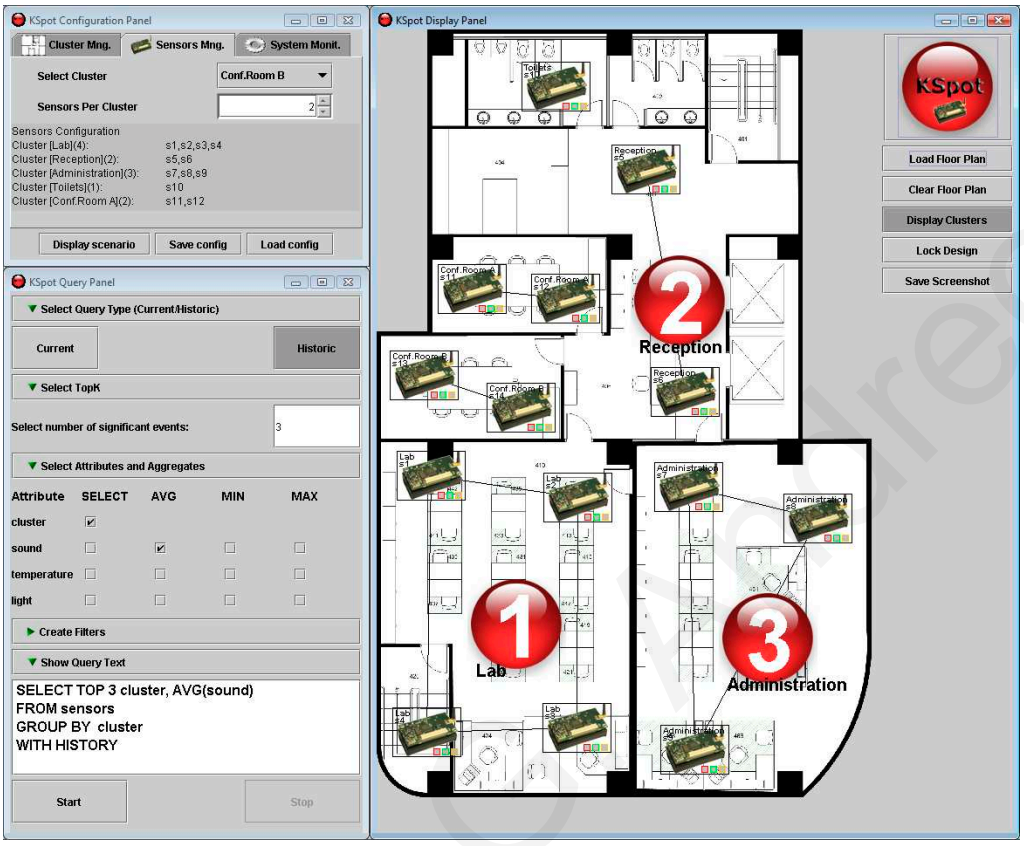

Figure 3-4: KSpot$^+$'s Graphical User Interface (GUI) allows users to administer the execution of standard and Top-k Queries through an intuitive and declarative graphical user interface. The above scenario conducts a Top-3 query over a 14-node sensor network organized in 6 logical clusters. The Display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

ii. The *Query Panel* (Figure 3-4, 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 Query API otherwise.

iii. The *Display Panel* (Figure 3-4, right), which allows a user to load an image representation of the scenario map. Subsequently, the user can drag-and-drop the 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.

Chapter 4

Workload Balancing Module

In this Chapter we present the Workload Balancing Module of the KSpot$^+$ framework. 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. This is accomplished by continuously profiling recent data acquisition activity and then identifying the bottlenecks of the network through an in-network execution of the Critical Path Method.

We start by presenting the motivation behind the proposed WART algotihm that lies in the foundations of the Workload Balancing Module followed by our system model and the basic terminology that will be utilized in the subsequent sections. Next, we present the WART algorithmic framework accompanied by a discussion of our considerations and specific implementation details.

4.1 Motivation and Preliminaries

We start by defining the *waking window* τ as the continuous interval during which a sensing device s_i enables its transceiver, collects and aggregates the results from its children, and then forwards them all together to its own parent. Note that τ is continuous because it would be very energy-demanding to suspend the transceiver more than once during the interval of an epoch (as shown in Section 7.2 with a series of micro-benchmarks) It is important to mention that the exact value of τ is query-specific and cannot be determined accurately using current techniques. For instance, s_i does not know in advance how many tuples it will receive from its children. Choosing the correct value for τ is a challenging task as any wrong estimate might disrupt the synchrony of the query routing tree.

We found that that predominant data acquisition frameworks [135, 86, 85, 81, 112] have not taken under consideration the optimization of the waking window and this degrades significantly the energy efficiency of these systems. Consequently, the objective of the WART algorithm is to generate a time synchronized topology in which sensing devices know exactly their waking window (i.e., they know when and for how long they should enable their transceivers).

To facilitate our description we briefly describe the waking window mechanisms of two popular data acquisition systems, TAG [86, 85] and Cougar [135] and the motivation behind our approach. For simplicity, let us assume that some arbitrary query $\mathcal{Q}$ has already been disseminated to the n sensors of the wireless sensor network.

Tiny aggregation (TAG): In this approach, the epoch e is divided into d fixed-length time *intervals* $\{e_1, e_2, \ldots, e_d\}$, where d is the depth of the routing tree rooted at the sink that conceptually interconnects the n sensors. The core idea of this framework is summarized as follows: *"when*

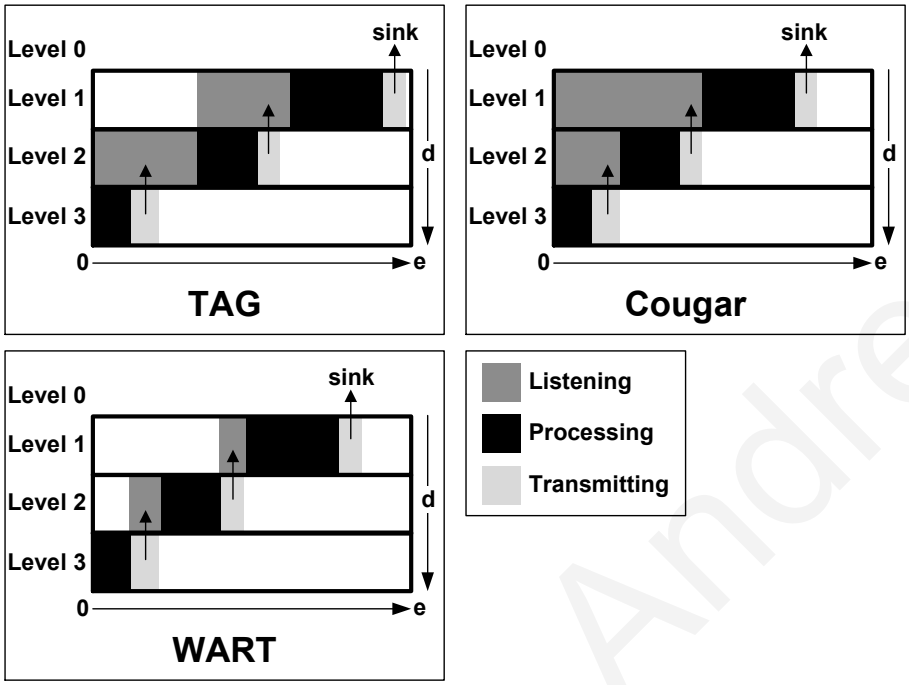

Figure 4-1: The Waking (Listening) Window (τ) in TAG, Cougar and WART algorithms.

nodes at level i+1 transmit then nodes at level i listen". More formally, a sensor s_i enables its transceiver at time instance $w_i = \lfloor e/d \rfloor * (d - depth(s_i))$ and keeps the transceiver active for $\tau_i = \lfloor e/d \rfloor$ time instances. Note that $\sum_{i=d}^{0}(e_i)$, where e_i defines the epoch at level i, provides a lower-bound on e, thus the answer will always arrive at the sink before the end of the epoch. Setting e as a prime number ensures the following inequality $\sum_{i=d}^{0}(e_i) < e$, which is desirable given that the answer has to reach the sink at time instance e.

For instance, if the epoch is 31 seconds and we have a three-tier network (i.e., d=3) like the one presented in Figure 4-1 (top, left), then the epoch is sliced into three segments $\{10,10,10\}$. During interval [0..10), nodes at level 3 will transmit while nodes at level 2 will listen; during interval [10..20) level 2 nodes transmit while level 1 nodes listen; and finally during [20..30), level 1 nodes transmit and the sink (level 0) listens. Thus, the answer will be ready prior the completion of time instance 31 which is the end of the epoch.

The parent wake-up window τ is clearly an over-estimation (in the above example 10 it is seconds!) of the actual time that is required to transmit between the children and a parent. The rationale behind this over-estimation is to offset the limitations in the quality of the clock synchronization algorithms [86] but in reality it is too coarse. In the experimental Section 7.3, we found that this over-estimation is three orders of magnitude larger than necessary. Additionally, it is not clear how τ is set 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).

Cougar: In this approach, each sensor maintains a *child waiting list* that specifies the children for each node. Such a list can be constructed by having each child explicitly acknowledging its parent during the query dissemination phase. Having the list of children enables a sensor to power down its transceiver as soon as all children have answered. This yields a set of non-uniform waking windows $\{\tau_1, \tau_2, \ldots\}$ as opposed to TAG where we have a single τ, which is uniform for all sensors (i.e., $\lfloor e/d \rfloor$). The main drawback of Cougar is that a parent node has to keep its transceiver active from the beginning of the epoch until all children have answered. In particular, it holds that $\tau_i > \tau_j$ if $depth(v_i) < depth(v_j)$. In order to cope with children sensors that may not respond, Cougar deploys a timeout h. To understand the drawback of Cougar consider Figure 4-1 (top, right), where level 2 and level 1 nodes have activated their transceivers at time instance zero and wait for the leaf nodes to respond. If a failure at some arbitrary node x occurs (e.g., at level 3) then each node on the path $x \rightarrow \ldots \rightarrow s_0$ has to keep its radio active for h additional seconds.

A recent paper that proposes a scheduling algorithm for wireless sensor networks has been presented in [6]. The authors define a probabilistic model that allows the evaluation of the packet

loss probability that results from the reduced radio activity. Based on the probabilistic model, the algorithm chooses the radio activity intervals that achieve optimal probability of successful packet delivery using three different strategies. The key differences between WART and this approach are: i) the proposed approach assumes that only one channel can be active at a given time whereas in our case all sensors that participate in a continuous query are active, and ii) the scheduling of the wake-up times is based on a probabilistic model whereas in our model the scheduling is based on profiling recent activity and determining the workload of each sensor. While this approach might be beneficial in cases of snapshot queries, our approach is focused on continuous queries.

The objective of the WART algorithm is to automatically tune τ, locally at each sensor without any a priori knowledge or user intervention. However, note that in defining τ we are challenged with the following trade-offs:

- **Early-off Transceiver:** Shall s_i power-off the transceiver too early reduces energy consumption but also increases the number of tuples that are not delivered to the *sink*. Thus, the sink will generate an erroneous answer to the query $\mathcal{Q}$; and

- **Late-off Transceiver:** Shall s_i keep the transceiver active for too long decreases the number of tuples that are lost due to powering down the transceiver too early but also increases energy consumption. Thus, the network will consume more energy than necessary which is not desirable given the scarce energy budget of each sensor.

WART dynamically adapts the waking window τ values by profiling recent data acquisition activity and identifying the bottlenecks of the network. This is accomplished using an in-network execution of the Critical Path Method.

The *Critical Path Method (CPM)* [54] is a graph-theoretic algorithm for scheduling project activities. It is widely used in project planning (construction, product development, plant maintenance, software development and research projects). The core idea of CPM is to associate each project milestone with a vertex v and then define the dependencies between the given vertices using *activities*. For instance, the activity $v_i \Leftarrow v_j$ denotes that the completion of v_i depends on the completion of v_j. Each activity is associated with a weight (denoted as $\overset{weight}{\Leftarrow}$) which quantifies the amount of time that is required to complete v_i assuming that v_j is completed. The critical path allows us to define the minimum time, or otherwise the maximum path, that is required to complete a project (i.e., milestone v_0). Any delay in the activities of the critical path will cause a delay for the whole project. In order to adapt the discussion to a sensor network context assume that each sensor s_i is represented by a CPM vertex. More formally, we map each s_i to the elements of the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ using a 1:1 mapping function $f : s_i \rightarrow v_i$, $i \leq n$. Also, let the descendent-ancestor relations of the sensor network be denoted as edges in this graph.

Figure 4-2 illustrates an example which will be utilized throughout the Chapter. The weights on the edges of the figure define the workload of each respective node (as the required time to propagate the query results between the respective pairs). It is easy to see that the total time to answer the query at the sink in the given network is at least ψ=99, since the critical path is $s_0 \overset{40}{\Leftarrow} s_1 \overset{30}{\Leftarrow} s_3 \overset{29}{\Leftarrow} s_8$. Having this information at hand, enables the scheduling of transmission between sensors. In particular, consider s_0 that operates solely in reception mode. Given that the maximum workload it expects from its only child s_1 is 40, s_0 only needs to enable its transceiver in the interval [59..99]. Similarly, s_1 which operates in both transmission and reception modes, needs to enable its transceiver for listening during the interval [29..59] to accommodate the most

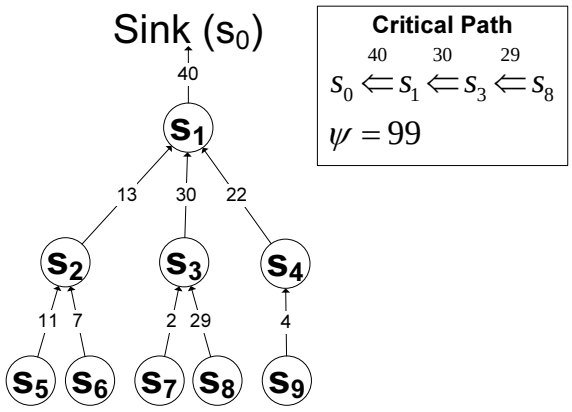

Figure 4-2: Nine sensing devices (shown as vertices) and the respective workload between them (shown as edges) in order to answer some continuous query $\mathcal{Q}$ at the sink (s_0). The WART algorithm utilizes this information in order to locally adapt the waking window of each device using the *Critical Path Method*.

demanding child s_3 with workload 30. Additionally, it needs to enable its transceiver for transmitting to its own parent during the interval [59..99]. Consequently, s_1 needs to keep its transceiver enabled during the interval [29..99]. A similar intuition also applies to other nodes.

Note that although the listening interval for each sensor s_i ($i \leq n$) will be scheduled by our approach, each sensor also keeps track of which children s_j ($j \leq n$) have already responded. When all children s_j ($j \leq n$) have reported their results to their parent then the parent node s_i ($i \leq n$) can immediately turn off its receiver as it does not expect any additional results from the s_js ($j \leq n$). Yet, it is obligated to wait for the right listening interval of its parent (i.e., $parent(s_i)$) before proceeding with the transmission of its own result. Finally, we would like to point out that a sensor s_i ($i \leq n$) might delay the transmission of its results for a number of reasons (e.g., sensor malfunction). In these cases, s_i will enable its transceiver only if it can anticipate the listening interval of its parent node (as the parent of s_i ($i \leq n$) will only be available during the given time interval).

Finally, note that the critical path allows a sensor s_j ($j \leq n$) to identify the interval during which its parent s_i ($i \leq n$) is expected to enable its own transceiver for reception. This is very

useful because in the subsequent epochs and under a different workload than the one utilized to compute its current τ interval, s_j can identify with local knowledge if it can still deliver the new workload without notifying s_i to adjust its τ interval.

It should be noted that the edges in Figure 4-2 have different weights. This is very typical for a sensor network as the link quality can vary across the network [121]. Another reason is that some sensors might have a different workload than other sensors. Note that our scheduling scheme is distributed, which makes it fundamentally different from centralized scheduling approaches like DTA [139] and TD-DES [26] that generate collision-free query plans at the centralized node. Additionally, our approach is also different from techniques such as [111], which segment the sensor network into sectors in order to minimize collisions during data acquisition.

4.2 System Model and Definitions

In this section we will formalize our system model and the basic terminology that will be utilized in the subsequent sections. The main symbols and their respective definitions are summarized in Table 4-1.

Let S denote a set of n sensing devices $\{s_1, s_2, ..., s_n\}$. Assume that s_i ($i \leq n$) is able to acquire m physical attributes $\{a_1, a_2, ..., a_m\}$ from its environment at every discrete time instance t. This generates at each t and for each s_i ($i \leq n$) one tuple of the form $\{t, a_1, a_2, ..., a_m\}$. This scenario conceptually yields an $n \times m$ matrix of readings R:=$(s_{ij})_{n \times m}$ for each timestamp. This matrix is *horizontally fragmented* across the n sensing devices (i.e., row i contains the readings of sensor s_i and $R = \cup_{i \in n} R_i$). Now let $G = (S, E)$ denote the network graph that represents the implicit network edges E of the sensors in S. The edges in E are implicit, because there is no explicit connection between adjacent nodes, but nodes are considered neighbors if they are

Table 4-1: Definition of Symbols for Chapter 4

Symbol	Definition
$\mathcal{Q}$	A Continuous Query
n	Number of Sensors $S = \{s_1, s_2, ..., s_n\}$
s_i	Sensor number i (s_0 denotes the sink).
m	Number of sensor recordings $\{a_1, a_2, ..., a_m\}$
e	Epoch duration of query $\mathcal{Q}$
$\mathcal{T} = (S, E)$	Query Routing Tree (S=vertices, E=edges)
d	Depth of the routing tree $\mathcal{T}$
w_i	Wake-up time of sensor s_i
τ_i	Waking window of sensor s_i
ψ	Total time needed to answer query $\mathcal{Q}$

within communication range (i.e., a fundamental assumption underlying the operation of a radio network).

A user submits $\mathcal{Q}$ at some centralized querying node (denoted as s_0, or sink node) prior deployment and the system then initiates the execution of $\mathcal{Q}$ by disseminating it to the n sensors. In particular, the sink sends $\mathcal{Q}$ to one sensor s_1. Subsequently, s_1 recursively forwards $\mathcal{Q}$ 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 middleware frameworks such as [85, 135, 81, 112] and where each sensor s_i selects as its parent the first node from which $\mathcal{Q}$ was received. This creates an acyclic subset of the communication graph G (i.e., a spanning tree) which is denoted as $\mathcal{T} = (S, E')$, 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 $\mathcal{T}$ (i.e., using an acknowledgment from each child to its parent). In more recent frameworks, like GANC [110] and Multi-Criteria Routing [80], $\mathcal{T}$ can be constructed based on query semantics, power consumption, remaining energy and others. In more unstable topologies a node can maintain several parents [33] in order to achieve fault tolerance but this might impose some limitations on the type of supported queries.

4.3 Algorithmics

In this section we describe the underlying algorithms of the Workload Balancing Module.

4.3.1 The WART Algorithm

At the foundation of Workload Balancing Module lies the WART algorithm that determines the period during which each s_i should wake up and the precise duration of this wake-up. In particular, WART is executed as a three-phase process that is summarized below:

1. **Construction Phase:** In this phase, the sink constructs a query routing tree (or uses an already established one) and then queries the network for the critical path value ψ.

2. **Dissemination Phase:** The sink disseminates ψ to the network and each sensor tunes its waking window accordingly.

3. **Adaptation Phase:** This phase is executed either periodically or when a topology change occurs. With this step each sensor adapts its waking window τ according to the new workload.

4.3.2 WART Phase 1: Construction

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 [135]. 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 towards the sink. In particular, each sensor s_i executes one round of data acquisition by maintaining one counter for its parent connection (denoted as s_i^{out}) and one

counter per child connection (denoted as $s_{i,j}^{in}$), where j denotes the identifier of the child. These counters measure the *workload* between the respective sensors (as the required time to propagate the query results between the respective pairs) and will be utilized to identify the critical path cost in the subsequent epochs. Note that these counters account for more time than what is required had we assumed a collision-free MAC channel. Additionally, it is important to mention that we could have deployed a more complex structure rather than the counters s_i^{out} and $s_{i,j}^{in}$, that would allow a sensor to obtain a better statistical indicator of the link activity, but these ideas are outside the scope of this dissertation. By projecting the time costs obtained for each edge to a virtual spanning tree creates a distributed *Query Routing Tree* similar to the one depicted in Figure 4-2.

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:

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

The critical path cost is then $f(s_0)$ (denoted for brevity as ψ). Using our working example of Figure 4-2, 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 $\psi = f(s_0) = 99$.

4.3.3 WART Phase 2: Dissemination

In this phase each sensor s_i ($i \leq n$) locally defines three parameters using the critical path cost ψ_i. These parameters enable s_i 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 query routing tree might be disrupted.

Algorithm 1 presents the main steps of this procedure which propagates ψ_i top-down, from the sink to the leaf sensors, with a message complexity of $O(n)$. The first step aborts the case where the critical path is larger than the epoch (which signifies an error in the user query). The second step calculates the wake up time instance w_i, such that s_i has enough time to collect the tuples from all its children s_j ($\forall j \in children(s_i)$). 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., $\psi - s^{out}_j$). Concurrently with step three, 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.

To facilitate our presentation we will now simulate the execution of Algorithm 1 on the example of Figure 4-2. To simplify the discussion, assume that the costs a, b and c (which account for *processing*, *the inaccurate clock* and the *collisions at the MAC layer*) are all equal to *zero*. Additionally, assume that the critical path cost is small enough to fit within the epoch (i.e., $\psi << e$). In particular, with $\psi = 99$ we get the following quadruples $(s_i, w_i, \tau_i, \lambda_i)$ at each sensor: $\{ (s_0, 59, [59..99), 0)$, $(s_1, 29, [29..99), 0)$, $(s_2, 46, [35..59), 17)$, $(s_3, 29, [0..59), 0)$, $(s_4, 37, [33..63), 8)$, $(s_5, 35, [35..46), 0)$, $(s_6, 39, [39..46), 4)$, $(s_7, 27, [27..29), 27)$, $(s_8, 0, [0..29), 0)$, $(s_9, 33, [33..37), 0)$ $\}$

To understand the benefits of the workload increase tolerance parameter λ_i, consider the scenario where node s_7 increases its workload by 15 time instances. Since $\lambda_7 = 29 - 2 = 27$, s_7 knows that the transceiver of its parent s_3 is enabled for 27 additional time instances, thus s_7 can start delivering the workload earlier (i.e., $w_7 = 12$ instead of $w_7 = 27$) succeeding in completing the transmission on-time.

Algorithm 1 : **WART Dissemination Phase**

Input: n sensing devices $\{s_1, s_2, \ldots, s_n\}$ and the sink s_0, the Critical Path cost ψ, the epoch e.
Output: A set of n waking windows τ_i ($i \leq n$), wake-up time instances w_i ($i \leq n$) and workload increase tolerance thresholds λ_i ($i \leq n$)
Execute these steps beginning from s_0 (top-down) and assuming that $\psi_0 = \psi$:

1. If $\psi_i > e$ then abort *"The Critical Path is larger than the Epoch"*.
2. For each child s_j of s_i ($\forall s_j \in children(s_i)$), find the maximum $s^{in}_{i,j}$. The child with the maximum $s^{in}_{i,j}$ is denoted as $s^{in}_{i,maxchild}$. The wake time w_i is calculated as follows:

$$w_i = \psi_i - s^{in}_{i,maxchild} - a - b - c, \tag{1}$$

   where a, b and c are three variables which offset the costs of *processing*, *the inaccurate clock* and *collisions at the MAC layer*, respectively.

   The waking window of s_i is the interval:

$$\tau_i = [w_i..(w_i + s^{in}_{i,maxchild} + s^{out}_i)) \tag{2}$$

3. Disseminate the following information to each s_i's child s_j ($\forall j \in children(s_i)$):
   (a) The value ψ_i.
   Upon receiving ψ_i, each s_j computes its own ψ_j as follows:

$$\psi_j = \psi_i - s^{out}_j \tag{3}$$

   (b) The value $s^{in}_{i,maxchild}$.
   Upon receiving $s^{in}_{i,maxchild}$, each s_j utilizes this value to define the *workload increase tolerance* (λ_j) of s_i as perceived by s_j, as follows:

$$\lambda_j = s^{in}_{i,maxchild} - s^{out}_j \tag{4}$$

4. Repeat steps 2-5, recursively until all sensors in the network have set w_i, τ_i and λ_i respectively ($i \leq n$).

4.3.4 WART Phase 3: Adaptation

In this section we describe an efficient distributed algorithm for adapting the WART query routing tree in cases of workload changes.

First notice that the naive approach to cope with workload changes is to re-construct the WART tree in every epoch. The message cost of such an approach is analyzed as follows: the WART construction phase has a message complexity of $O(1)$ as it can be executed in parallel with the acquisition of data tuples from sensors (i.e., the critical path cost can be piggybacked with data tuples). The dissemination phase on the other hand, has a message complexity of $O(n)$ as it

Algorithm 2 : **WART adaptation phase**

Input: A sensor s_i, the critical path value ψ_i, the wake-up time w_i, the waking window τ_i, a flag which indicates if s_i lies on the critical path, an error threshold δ.
Output: An updated set of w_i, τ_i and λ_i values.

```
1:  procedure Adapt(s_i)
2:      ▷ Step 1: Calculate Workload Indicators
3:      workload'_i = ψ_i − w_i;                          ▷ Workload of previous epoch
4:      for j = 1 to children(s_i) do
5:          add(tuples(s_j), workload_i);                 ▷ Build new workload
6:      end for
7:      add(tuples(s_i), workload_i);                     ▷ Append local tuples
8:      x = |workload_i − workload'_i|                    ▷ Workload Deviation
9:      if (x < δ) then
10:         signal(finished);                             ▷ Negligible Workload Change
11:     end if
12:     ▷ Step 2: Important Workload Change on the CP
13:     if (cp_i) then
14:         send("Critical Path Re-construction", s_j);
15:         signal(finished);
16:     end if
17:     ▷ Step 3: Important Work Change NOT on the CP
18:     if (workload_i decreased by x) then
19:         w_i = w_i + x; ▷ Adjust local wakeup time
20:     else▷ Workload was Increased by x
21:         if (x ≤ λ_i) then                             ▷ x is less than the available slack
22:             w_i = w_i − x; ▷ Adjust local wakeup time
23:         else
24:             send("Request Critical Path Re-construction", s_j);
25:         end if
26:     end if
27:     signal(finished);
28: end procedure
```

requires the dissemination of the critical path cost to all n nodes in the network. The algorithm we propose in this section can circumvent the $O(n)$ cost incurred by the dissemination phase in every epoch by deploying a set of rules we describe in the next algorithm.

Algorithm 2, presents the WART adaptation algorithm which proceeds in three steps. The first step of the algorithm (lines 2-11) calculates the workload indicators of the current epoch (i.e., $workload_i$) and the previous epoch (i.e., $workload'_i$). If the workload has changed by more than a user defined user threshold δ in line 9, we consider this change as significant and proceed with the adaptation of the routing tree in line 12. Otherwise, we disregard this deviation and abort the algorithm. Assuming a significant deviation, step 2 in line 12 handles the case where the change

occurs on the critical path. In such a case, s_i 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 4-2) then this will trigger the re-construction of the WART routing tree and this change should be propagated to all nodes in the network. Although this case is possible, our experimental study in Section 7.3 has shown that it is not frequent.

Finally, step 3 of Algorithm 2 (lines 17-26) 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 (line 18) 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 $w_2^{new} = w_2 + x = 46 + 2 = 48$. Similarly if the workload is *increased* by x (line 20) then there are two cases: i) the increase is less or equal to the slack λ_i, and ii) the increase is greater than the slack λ_i. For the first case (i), consider a workload increase at s_2 from 13 to 18 (thus, $x = 5$ that is smaller than $\lambda_2 = 17$). This yields the following adaptation of the wake up time $w_2^{new} = w_i - x = 46 - 5 = 41$. For the second case (ii), consider a workload increase at s_2 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.

4.4 Discussion

Critical Path Reconstruction Frequency: In the event of a change in the critical path, the Workload Balancing Module needs to re-calculate the critical path value and disseminate it to all n nodes in the network thus has a message complexity of $O(n)$. One important question that arises is how often to expect changes to the critical path. This might severely degrade the longevity of the network. In the experiments presented later in Section 7.3.1, 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 the Workload Balancing Module optimization phase. This is

expected as the critical path value is only calculated at the start of the execution and continues to be valid (i.e., not requiring reconstruction) 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 sensor node in which the failure occurred, 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 in question 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 Section 7.3.1.

Chapter 5

Tree Balancing Module

Although the Workload Balancing Modules 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 tree topology might be unbalanced.

In this Chapter we present the Tree Balancing Module of the KSpot$^+$ framework. The Tree Balancing Module identifies structural inefficiencies in the initial query routing tree 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.

We start by presenting the motivation behind the ETC algorithm that lies in the foundations of the Tree Balancing Module followed by our system model and the basic terminology that will be utilized in the subsequent sections. Next, we present the ETC algorithmic framework accompanied by a discussion of our considerations and specific implementation details.

5.1 Motivation and Preliminaries

Even though the WART algorithm presented in Chapter 4 can efficiently resolve the waking window problem, it does not optimize the query routing tree and that leads to increased collisions during data transmission. Assuming an arbitrary query routing tree $\mathcal{T}_{input}$ constructed using the FHF approach, the objective of ETC is to transform $\mathcal{T}_{input}$ into a near-balanced tree $\mathcal{T}_{ETC}$ in a distributed manner.

To facilitate our description, consider the example depicted on Figure 5-1 (left), which illustrates the initial ad-hoc query routing tree $\mathcal{T}$ created on top of a 10-node sensor network with the First-Heard-From approach. In the example we observe that node s_2 is inflicted with a high workload (i.e., 5 child nodes) while other nodes at the same level (i.e., s_3 and s_4), only have zero and one child nodes respectively. Notice that both s_8 and s_{10} are within communication range from s_3 (i.e., the dotted circle), thus these nodes could have chosen the latter one as their parent. Unfortunately, the FHF [85, 135, 109] approach is not able to take these semantics into account as it conducts the child-to-parent assignment in a network-agnostic manner. Additionally, unbalanced topologies pose some important energy consumption challenges which are summarized as follows:

- **Decreased Lifetime and Coverage:** Since the majority of the energy capacity is spent on transmitting and receiving data, the available energy of sensors with a high workload will be depleted more rapidly than the others. For example, in Figure 5-1 (left) sensor s_2's energy will be depleted 93/12=7.75 faster than s_3, that is ($(\sum_{i=0}^{children(s_2)}(s_i, s_2) + (s_2, s_1))$ / $(\sum_{i=0}^{children(s_3)}(s_i, s_3) + (s_3, s_1))$, and 3.72 times faster than s_4 (i.e., 93/25). In addition, if s_2's energy is depleted and no alternate parents are available for sensors s_{5-7} then the coverage of the network will be reduced dramatically.

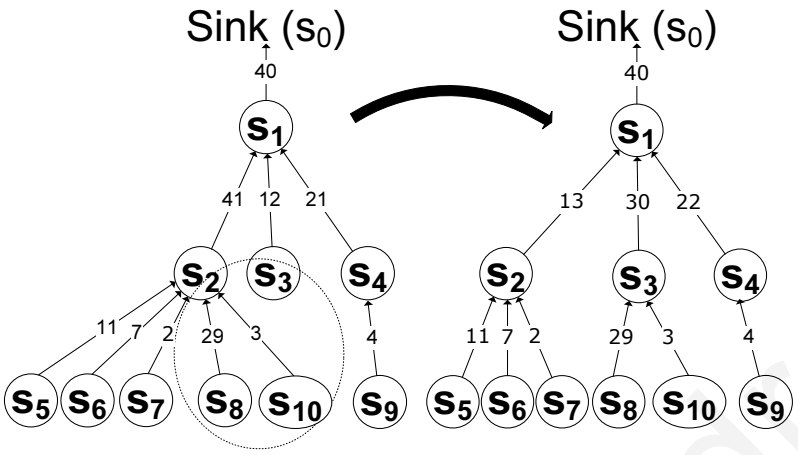

Figure 5-1: (Left) The initial ad-hoc query routing tree constructed using the *First Heard From method.* (Right) The optimized workload-aware query routing tree constructed using the in-network ETC balancing algorithm.

- **Increased Data Transmission Collisions:** An unbalanced workload increases data transmission collisions which represent a major source of energy waste in wireless communication. Our micro-benchmarks on the CC2420 radio transceiver (see Section 7.2, unveil that crowded parent hubs like s_2 might yield loss rates of up to 80%, thus inflicting many re-transmissions to successfully complete the data transfer task between nodes.

We start out with some definitions on balanced trees and then present the ETC algorithm both in a centralized setup and a distributed setup. Notice that the ETC algorithm logically precedes the operation of the WART algorithm (i.e., if T is reconstructed then WART must be re-executed).

5.2 System Model and Definitions

In this section we will formalize our system model and the basic terminology that will be utilized in the subsequent sections. The main symbols and their respective definitions are summarized in Table 5-1.

Table 5-1: Definition of Symbols for Chapter 5

Symbol	Definition
n	Number of Sensors $S = \{s_1, s_2, ..., s_n\}$
s_i	Sensor number i (s_0 denotes the sink).
$\mathcal{T} = (S, E)$	Query Routing Tree (S=vertices, E=edges)
d	Depth of the routing tree $\mathcal{T}$
$children(s_i)$	Children List of sensor s_i
$APL(s_i)$	Alternate Parent List of sensor s_i
β	Balanced Branching Factor of network S

We utilize the same system model as presented in Chapter 4 with some additions in order to accommodate the requirements of the ETC algorithm. Once more, we adopt the *First Heard From (*FHF*)* mechanism for the construction of the ad hoc query routing tree $\mathcal{T}$. Besides the *Child Node List*, we additionally supplement each sensor s_i with an *Alternate Parents List* (denoted as $APL(s_i)$). 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 $\mathcal{T}$, but it can also be used for selecting alternate parents in cases of failures.

As we have already mentioned in the motivation of this Chapter, balanced trees have the following desirable properties: i) they decrease collisions during data transmission, ii) they decrease query response times, and iii) they increase system lifetime and coverage. Balanced trees can improve the asymptotic complexity of `insert`, `delete` and `lookup` operations in trees from $O(n)$ time to $O(log_b n)$ time, where b is the branching factor of the tree and n the size of the tree. We shall next provide some formal definitions to be utilized in our description:

Definition 5.1 - Balanced Tree ($\mathcal{T}_{balanced}$): *A tree where the heights of the children of each internal node differ at most by one.*

The above definition specifies that no leaf is much farther away from the root than any other leaf node. For ease of exposition consider the following directed tree: $\mathcal{T}_1 = (V, E)$ = $(\{A, B, C, D\}$, $\{(B, A), (C, A), (D, B)\})$, where the pairs in the E set represent the edges of the binary tree. By visualizing $\mathcal{T}_1$, we observe that the subtrees of rooted at node A differ by at most one (i.e., $|height(B) - height(C)|$=0) and that the subtrees rooted at B differ again by at most one (i.e., $|height(D) - height(NULL)|$=1). Thus, we can characterize $\mathcal{T}_1$ as a balanced tree.

Notice that V has several balanced tree representations of the same height (e.g., the directed tree $\mathcal{T}_2 = (\{A, B, C, D\}, \{(B, A), (C, A), (D, C)\})$). Similarly, V has also many balanced tree representations of different heights (e.g., the directed tree $\mathcal{T}_3 = (\{A, B, C, D\}, \{(B, A), (C, A), (D, A)\})$ which has a height of one rather than two). Finally, in a balanced tree every node has approximately β children, where β is equal to $\sqrt[d]{n}$ (the depth of every balanced tree is $d = \log_\beta n$, thus $\beta^d = n$ and $\beta = \sqrt[d]{n}$). The ETC algorithm presented in this section focuses on the subset of balanced trees which have the same height to $\mathcal{T}_{input}$ as this makes the construction process more efficient.

In order to derive a balanced tree ($\mathcal{T}_{balanced}$) in a centralized manner we could utilize the respective balancing algorithms of AVL Trees [2], B-Trees [105] and Red-Black Trees [55]. However, that would assume that all nodes are within communication range from each other which is not realistic. Thus, the ETC algorithm seeks to construct a *Near-Balanced Tree* ($\mathcal{T}_{near\_balanced}$), defined as follows:

Definition 5.2 - Near-Balanced Tree ($\mathcal{T}_{near\_balanced}$): *A tree in which every internal node attempts to obtain a number of child nodes as close to the optimal branching factor β.*

The objective of $\mathcal{T}_{near\_balanced}$ is to yield a structure similar to $\mathcal{T}_{balanced}$ without imposing an impossible network structure (i.e., nodes will never be enforced to connect to other nodes that are

not within their communication range). We shall later also define an error metric for measuring the discrepancy between the yielded $\mathcal{T}_{near\_balanced}$ and the optimal $T_{balanced}$. We will additionally show in Section 7.4.1 that constructing $\mathcal{T}_{near\_balanced}$ with the ETC algorithm yields an error of 11% on average for the topologies utilized in this paper.

5.3 Algorithmics

In this section we describe the underlying algorithms of the Tree Balancing Module.

5.3.1 The Centralized ETC (CETC) Algorithm

Let us first devise an algorithm for constructing a near-balanced query routing tree in a centralized manner. In particular, we will devise the *Centralized ETC (CETC)* algorithm, which obtains global knowledge before proceeding into the generation of the near-balanced tree (the tree will be denoted for clarity as $\mathcal{T}_{CETC}$). We show that such a centralized solution poses an extremely high complexity rendering it inefficient for wireless sensor networks. This necessitates the use of a lower complexity distributed approach. For this, we devise in the next section the distributed ETC algorithm that constructs a structurally similar tree to $\mathcal{T}_{CETC}$ in a distributed manner.

The CETC algorithm consists of three steps:

1. A sink (s_0) node executes an in-network query in order to acquire the initial input tree $\mathcal{T}_{input}$ and the alternative parent list of each sensor. The alternative parent list will be useful in defining a set of parent re-assignments that can lead to $\mathcal{T}_{CETC}$.

2. The sink s_0 conducts an exhaustive search of all possible $\mathcal{T}_{CETC}$ trees and estimates their balancing error w.r.t. to the optimal $\mathcal{T}_{balanced}$ tree. It finally chooses the one with the least cost using the $Balancing\_Error$ formula presented in 7.4.1.

Algorithm 3 : **Centralized ETC (CETC) Algorithm**

Input: An Ad-hoc Query Routing Tree $\mathcal{T}_{input}$.
Output: A Near-Balanced Query Routing Tree $\mathcal{T}_{CETC}$.

1. Execute the following query at the sink node:
   `SELECT sensorid, parentid, alternate_parents, height FROM SENSORS`.
   Notice that *height* is evaluated using the following in-network function with an O(1) cost and *height(s_0)* defines the height (d) of $\mathcal{T}_{CETC}$:

$$height(s_i) = \begin{cases} 0 & \text{if } s_i \text{ is a leaf,} \\ max_{\forall j \in children(s_i)}(height(s_j)+1) & \text{otherwise.} \end{cases}$$

2. Utilize two boolean matrices PM:=$(parent_{ij})_{|V|\times|V|}$ and APM:=$(alternate\_parent_{ij})_{|V|\times|V|}$ to represent the child-to-parent and child-to-alternate_parent relationships acquired in step 1. $PM_{ij} = 1$ denotes that node j is a parent of node i (a similar idea applies to the APM matrix).
3. Using APM and PM, find an assignment of parents such that the following are satisfied:
   a. $height(\mathcal{T}_{CETC}) == height(\mathcal{T}_{input})$.
   b. Each child s_j ($j \leq n$) has exactly one parent s_i ($i \leq n$).
   c. Each child s_j ($j \leq n$) attempts to obtain less or equal children to β.
   d. The discrepancy between an optimal solution $\mathcal{T}_{balanced}$ and a feasible solution $\mathcal{T}_{CETC}$ is the minimum possible. To quantify this discrepancy, define the *Balancing_Error* metric as follows:

$$Balancing\_Error(\mathcal{T}_{CETC}) := \sum_{i=0}^{n} |\beta - \sum_{j=0}^{n} PM_{ij}|$$

4. Find an assignment that satisfied the constraints 3a-3d by executing the A* graph search algorithm [58, 106] in order to find the $\mathcal{T}_{CETC}$ with the minimum Balancing_Error. In particular, search through the solution space by computing the following heuristic in each step:

$$Heuristic := Steps(\mathcal{T}_{Input}, \mathcal{T}_{CETC}) + Balancing\_Error(\mathcal{T}_{CETC})$$

   The A* algorithm gives priority to the routes that have the smallest value in the heuristic (i.e., are most likely to lead towards the goal). The algorithm terminates when then heuristic changes by less than δ between k consecutive steps, where δ, k are user-defined thresholds.
5. Disseminate $\mathcal{T}_{CETC}$ to the network so that $\mathcal{T}_{Input}$ is adjusted accordingly.

3. The sink s_0 disseminates the identified tree back to the n nodes so that these can make the required adjustments.

It is easy to see that the first step of the CETC algorithm has a message complexity of $O(n)$ (i.e., each node will transmit exactly one message) but each message has a size of $O(n^2)$ (i.e., in a fully connected graph each node will have n-1 alternate parents). The second step is conducted on the sink node and requires in the worst case to explore the complete solution space which has a size

of $O(n^2!)$. Note that the CETC algorithm is a computationally intensive algorithm and therefore the second step of the algorithm might end up delivering a solution which does not match the initial acquired state of the network that was acquired in step 1 (as the network state might have changed). Finally, the algorithm needs to propagate the solution back to the n nodes and that has again a message complexity of $O(n)$ with each message being $O(n^2)$.

5.3.2 The Distributed ETC Algorithm

The ETC algorithm presented in this section overcomes the problems of the Centralized ETC algorithm by conducting the calculation of the optimized routing tree in a distributed manner. In particular, given an arbitrary query routing tree $\mathcal{T}_{input}$ the objective of ETC is to transform $\mathcal{T}_{input}$ into a near-balanced tree $\mathcal{T}_{ETC}$ in a distributed manner. The ETC algorithm consists of a discovery and distributed balancing step which are described next.

5.3.3 ETC Phase 1: Discovery

The first phase of the ETC algorithm starts out by having each node select one node as its parent using the FHF approach. 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. In particular, the hops parameter is initialized to zero and is incremented each time the tree construction request is forwarded to the children nodes of some node. A node s_i also maintains a child node list ($children$) and an alternate parent list (APL) according to the description we provided in Section 5.2.

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, similar to the CETC algorithm, the Optimal Branching Factor (β).

5.3.4 ETC Phase 2: Balancing

The second phase of the ETC algorithm involves the top-down reorganization of the query routing tree $\mathcal{T}_{input}$ 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 the execution of Algorithm 4 in which s_i will order parent re-assignments for its children. Algorithm 4 is divided into two main steps: i) lines 3-8: s_i's connection to its newly assigned parent *newParent*, and ii) lines 9-25: the transmission of parent reassignment messages to children nodes, in which the given nodes are instructed to change their parent.

In line 2 of Algorithm 4 each node s_i ($\forall s_i \in S - s_0$) waits in blocking mode until an incoming message interrupts the $receive()$ command. When such a message has arrived, s_i obtains the β value and the identifier of its *newParent*. The next objective (line 4) is to identify whether *newParent* is equal to NULL, in which case s_i does not need to change its own parent (i.e., we proceed to line 9). On the contrary, if *newParent* has a specific node identifier then s_i will attempt to connect to that given node (lines 4-8). 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.

In line 9 we proceed to the second step of the algorithm in which s_i's children might be instructed to change their parent node. We choose to do such a reassignment 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*). In line 10 we skip s_i if the number

Algorithm 4 : **ETC balancing algorithm**

Input: A node s_i; The children-list of s_i (denoted as $children(s_i)$); The alternate parent list for each child of s_i (denoted as $APL(s_j)$, where $s_j \in children(s_i)$); The Optimal Branching Factor β; The new parent that s_i should select (denoted as $newparent(s_i)$).
Output: A Near-Balanced Query Routing Tree $\mathcal{T}_{CETC}$.
Execute these steps beginning at s_0 (top-down):

```
1:  procedure Balance_Tree(s_i; children(s_i); ∀_{s_j∈children(s_i)} APL(s_j); )
2:      (β, newParent)=receive();                          ▷ Acquire info from s_i's parent.
3:      ▷ Step 1: Connect to new parent if needed
4:      while (newParent != NULL) do
5:          if (!connect(newParent)) then                  ▷ Cannot become a child of newParent.
6:              newParent = getNewParent(parent(s_i)) ▷ Involves 1 round-trip. Parent returns NULL if
                    no new Parent is available (in which case s_i stays with its current parent).
7:          end if
8:      end while
9:      ▷ Step 2: Adjust the parent of the children nodes.
10:     if (|children(s_i)| <= β) then                     ▷ Skip s_i as no change is necessary.
11:         for j = 1 to |children(s_i)| do
12:             send(β, NULL, s_j);                        ▷ Send β and no newParent to child.
13:         end for
14:     else▷ Ask |children(s_i)| − β nodes to change their parent.
15:         while (|children(s_i)| > β) do
16:             s_j = getNext(children(s_i));
17:             if (|APL(s_j)| > 1) then
18:                 newParent=AlternParent(APL(s_j), s_i);
19:                 send(β, newParent, s_j);               ▷ Send to s_j.
20:                 children(s_i) = children(s_i) - s_j     ▷ Remove from children.
21:             else
22:                 send(β, NULL, s_j);                    ▷ Report No change.
23:             end if
24:         end while
25:     end if
26: end procedure
```

of children is less than β. In the contrary case (line 14), we have to eliminate $|children(s_i)| - \beta$ children from s_i. Thus, we iterate through the child list of s_i (line 16) and attempt to identify a child s_j that has at least one alternate parent (line 17). If an alternative parent can not be determined for node s_j then it is obviously not meaningful to request a change of s_i's parent (line 22).

Let us now simulate the execution of the ETC algorithm using the illustration of Figure 5-1. In particular, Figure 5-1 (left) displays $n = 10$ sensors arranged in an ad-hoc topology $\mathcal{T}_{input}$ with a depth $d = 2$. In order to transform $\mathcal{T}_{input}$ into a near-balanced topology each node has to

obtain approximately $\beta = 3.16$ children (i.e., $\sqrt[2]{10}$). To simplify our discussion, but w.l.o.g., let us assume that the only sensors with multiple entries in their alternate parent list (APL) are s_8 and s_{10}. In particular, assume that we have the following values: $APL(s_8)$=$\{s_3\}$ and $APL(s_{10})$=$\{s_3\}$.

The ETC algorithm is initiated at the sink node s_0. Since s_0 has less than $\beta = 3.16$ children it transmits β and `newParent=NULL` to its only child s_1. Similarly, s_1 transmits β and `newParent=NULL` to its children s_2, s_3 and s_4. Let us now consider s_2 which receives the above parameters in line 2 of Algorithm 4. Since `newParent =NULL`, s_2 does not need to change its parent (lines 3-8). It has to however instruct some of its children to change their parents as `|children(`s_2`)|>`β. Thus, it processes its children nodes in sequential order, starting at s_5 and ending at s_{10}, instructing some of them to change their parent. In particular, s_{5-8} are instructed to retain their initial parent while s_8 and s_{10} are instructed to change their parent to s_3 (i.e., they receive the messages `send(`$3.16, s_3,\ s_8$`)` and `send(`$3.16, s_3, s_{10}$`)` respectively. In our example s_3 can accommodate s_8's and s_{10}'s request as `|children (`s_3`)|=0`. Under different conditions however, satisfying such requests might not be possible. Thus, each node might request from its parent another alternative parent (i.e., lines 5-7). The updated near-balanced tree $\mathcal{T}_{ETC}$ is presented in Figure 5-1 (right).

5.4 Discussion

Effect of sensor locations and topology properties on network performance: The performance of the wireless sensor network is affected by the topology design and particularly by the physical locations of the sensor nodes, the network structure (i.e., flat or hierarchical) as well as the sensor reception/transmission power. These topology design issues (commonly known as topology control) usually affect the energy efficiency, coverage, connectivity and fault tolerance of the network. Dense topologies, for example, where the majority of sensor nodes are deployed near the

sink node (i.e., low depth query routing trees), benefit from the low energy consumption because of the short communication range of each sensor node, but may be severely hampered by data transmission collisions, low coverage, etc. Additionally, sensor nodes closer to the sink node will with high probability become hotspots (i.e., critical nodes with high relay traffic load) resulting in premature depletion of their energy reserves. In contrast, sparse WSN deployments favor the coverage performance metric by deploying sensor nodes at large distances (i.e., high depth query routing trees), reduce data transmission collisions and decrease hotspots. However, enforcing sensor nodes to communicate over large distances increases the individual power consumption of each sensor node. Additionally, for the aforementioned reasons, sparse WSN deployments are vulnerable to network disconnections and partitions.

In order to minimize the data transmission collisions, the ETC algorithm attempts to balance the number of child nodes of each inner node according to the optimal branching factor value while in parallel maintaining the same depth as the original query routing tree. This happens as by increasing the depth of the query routing tree results in: i) an increase of time and energy overhead for propagating a query to the network as the same query packets will travel more hops to arrive at sensor nodes that reside in larger depths; and ii) an increase of time and energy overhead for recursively forwarding the results of sensor nodes residing in larger depths to the sink node. Conversely, increasing the depth also results in the assignment of less child nodes for each inner node, which decreases data transmission collisions. Depending on the application requirements, ETC can be easily extended to construct topologies with greater or smaller height by removing the condition dictating that the depth must remain unchanged. However, when a change of depth occurs then the optimal branching factor becomes obsolete and the reconstruction phase must be reinitiated. Additionally, this procedure may run for infinite loops unless a termination condition

(e.g., maximum number of runs) is introduced. This will result in a significant increase of energy and time, which will have a negative effect on the longevity of the network.

Extending the Optimal Branching Factor: ETC assumes that all sensor nodes feature the same workload and that the workload of a parent sensor node 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 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 query routing tree. One way to accomplish this would be to first execute the WART algorithm of the Workload Balancing Module (described in Chapter 4), 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: The ETC algorithm presented in this Chapter, optimizes the network efficiency by generating a more balanced topology that minimizes data transmission collisions. To accomplish this, ETC balances the network using the optimal branching factor that takes into account network semantics (e.g., number of child nodes, depth of the query routing tree, workload of each sensor node). Although in our experiments we show that these optimizations offer significant energy savings (see section 7.4) there are occasions where they may present conflicts with the optimizations proposed by the Query Processing Module (presented in the next Chapter) 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 [110] where the authors configure the network in order to benefit the execution of Group-by queries using the *Group-Aware Network Configuration* (GANC) framework. 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 Chapter 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. In conclusion, taking into account the Group-By query-based semantics can introduce additional optimizations for Group-By queries during their execution by the Query Processing Module. 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. Therefore, depending on the application requirements, designers can incorporate new modules and/or disable the existing ones in order to further minimize the overall energy consumption of the network.

Chapter 6

Query Processing Module

In this Chapter we present the Query Processing Module of the KSpot$^+$ framework. The Query Processing Module is responsible for query execution as well as a number of services including group management and caching. Regular SQL queries are executed using the built-in query mechanism while standard Top-k queries are executed using the INT algorithm. In the case of Top-k queries that involve logical groups, these are coordinated through the group management component. The Query Processing Module also utilizes a data caching mechanism that in cooperation with the INT mechanism exploits temporal coherency between results of consecutive time instances (MINT).

In particular, the Query Processing process consists of three phases: i) *Query Dissemination Phase*, where the sink node propagates a query to the network; ii) *Processing Phase*, where 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; and iii) *Data Acquisition Phase*, where each sensor node recursively transmits its results to the network until they reach the sink node.

We start by presenting the motivation behind the proposed INT/MINT algotihms that lie in the foundations of the Query Processing Module followed by our system model and the basic terminology that will be utilized in the subsequent sections. Next, we present the INT/MINT algorithmic framework accompanied by a discussion of our considerations and specific implementation details.

6.1 Motivation and Preliminaries

At the foundation of Query Processing Module lies MINT Views, a novel algorithm that minimizes messaging and thus energy consumption in the execution of continuous monitoring queries. Like other works [85, 135, 86, 81, 112], 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 Figure 6-1, where we illustrate a deployment of 9 sensors in a 4-room building. We are interested in answering $Query1$ 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
```

With the TinyDB-based [86, 85] 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)* (Figure 6-1 on the right). V materializes the result of Q

[1]For clarity in Figure 6-1, we only depict the average (i.e., sum/count).

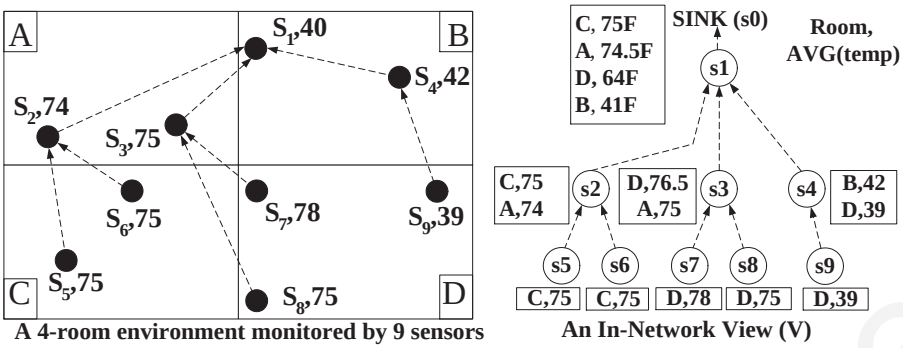

Figure 6-1: The left figure illustrates a sensor network scenario that consists of 9 sensors $\{s_1, ..., s_9\}$ deployed in four rooms $\{A, B, C, D\}$. The label next to each sensor denotes the identifier of the node and the local temperature reading. The figure on the right presents a recursively defined In-Network View (V) to $Query1$. The label next to each node indicate the local averages for each room.

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 $Query1$ 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^{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 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 ($\gamma = \{\gamma_1, \gamma_2, ...\}$), 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 6.5.2). Without loss of generality, in the rest of our discussion we will utilize the following instances: γ_1 =“*Maximum possible temperature value”* and γ_2 =“*Number of sensors in each room”*. For instance, the temperature sensor on the TelosB Weather Board [121] might only record values between -40F to 250F and the barometric pressure module can only measure pressure in the range 300mb to 1100mb.

6.2 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$^+$ Query Processing Module. We will then outline the motivation behind the phases of these algorithms. The main symbols and their respective definitions are summarized in Table 6-1.

Table 6-1: Definition of Symbols for Chapter 6

Symbol	Definition
Q	A Query
k	Number of requested results
s_i	Sensor number i (s_0 denotes the sink).
n	Number of Sensors $\{s_1, s_2, ..., s_n\}$
m	Number of Attributes at each sensor $\{a_1, a_2, ..., a_m\}$
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 utilize the same system model as presented in Chapter 5 with some additions in order to accommodate the requirements of the INT/MINT algorithms. Once more, we adopt the *First Heard From (*FHF*)* mechanism for the construction of the ad hoc query routing tree $\mathcal{T}$. Assume that s_i ($i \leq n$) is able to acquire m physical attributes $A = \{a_1, a_2, ..., a_m\}$ from its environment at every discrete time instance t. This generates tuples of the form $\{t, a_1, a_2, ..., a_m\}$ 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 $R = \cup_{i \in n} R_i$.

6.3 Algorithmics

In this section we describe the underlying algorithms of the Query Processing Module. As already mentioned in Section 6.5.3, the KSpot$^+$ Query Processing Module 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 6.1). 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.

6.3.1 The INT/MINT Algorithms

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 [61] after the query has been disseminated to the n sensors. In the following sections we thoroughly describe each phase of the MINT Views algorithm.

6.3.2 MINT Phase 1: Creation

The first phase of the algorithm is a recursive execution of Algorithm 5 at all sensors in a given network. Recall that a sensor generates an $(m+1)$-tuple of the form $v = \{t, a_1, a_2, ..., a_m\}$ 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 Crossbow Weather board which was utilized in the Great Duck Island study [121] supplements the motes with 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 (line 4-13). This yields an in-network view similar to Figure 6-1 (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.

Algorithm 5 : **Construct MINT/INT View**

Input: A distributed sensor s_i ($\forall s_i \in S$) that generates m attributes $\{a_1, a_2, ..., a_m\}$, a query Q, an empty buffer $V_i = \{\}$

Output: A set of n distributed views $V = \{V_1, V_2, ..., V_n\}$.

```
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 j^th 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 l^th 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
```

6.3.3 MINT Phase 2: Pruning

Algorithm 5 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

```
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 $Query2$ (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

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

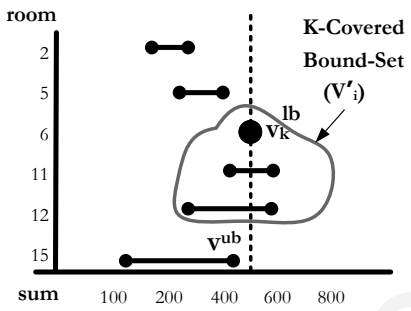

Figure 6-2: The left table illustrates the V_i of a given node during the execution of query $Query2$. The right figure illustrates the intuition of the pruning algorithm. In particular, we plot the (lb,ub) ranges for the various returned tuples at some arbitrary node. We then generate a k-covered bound set V_i' using Algorithm 6. We only propagate a tuple u to the parent of s_i, if $u \in V_i'$.

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 6-2 (left), where we illustrate the V_i for a given sensor. Prior to the execution of $Query2$ 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 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 6-2

(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 6.1: 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 k^{th} highest lower bound[2] .

Algorithm 6 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 Figure 6-2, 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

[2]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 6 : **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 $\gamma = \{\gamma_1, \gamma_2, ...\}$, 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**

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.*

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 5.1, a contradiction □

Algorithm 7 : **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 6
9:              send(V_i', parent(s_i)); // Complete V_i' update
10:         end if
11:     end if
12:     T' = V_i';
13: end procedure
```

6.3.4 MINT Phase 3: Update

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 5 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 6.

Algorithm 7 presents the MINT Update Algorithm and Figure 6-3 illustrates the respective steps of the algorithm. In particular, line 3 of Algorithm 7 shows that any tuple update x with

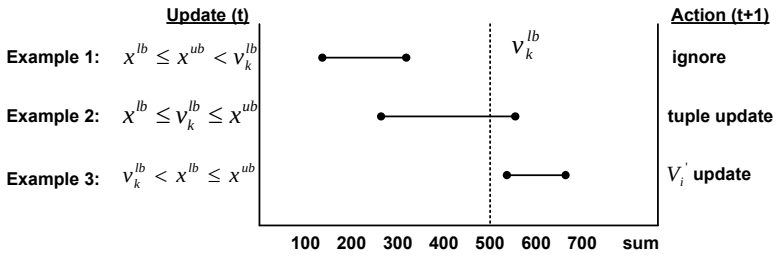

Figure 6-3: The figure illustrates how different tuples will be handled during the update phase.

an upper bound (denoted as x^{ub}) less than the v_k^{lb} can be *ignored* (also see respective example 1 of Figure 6-3). In the opposite case, we add the tuple x to the set of candidates V_i' (line 4 of Algorithm 7 and example 2 of Figure 6-3).

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' using Algorithm 6 and transmit the complete V_i' to its parent (also see respective example 3 of Figure 6-3). 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 6 for the first time.

6.4 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 7 (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 6. 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 dissertation 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 2KB SRAM might need to exploit the 512KB on-chip flash memory, while Intel's i-mote might easily store these results in the 64KB SRAM. There is a growing trend for more available local storage in sensor devices [142] 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. However, this approach is prone to human errors and may result in two types of inefficiencies:

- **Extremely small values of k**: 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 INT/MINT algorithm's implementation code.

- **Extremely large values of** k: 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. However, as we show in the experiments in Section 7.6.3, even in these extreme scenarios the temporal coherence filter of the MINT algorithm still manages to reduce the energy consumption by 18% for the tested queries.

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.

6.5 Implementation Details

In this section we present specific implementation details regarding the components presented in the previous sections.

6.5.1 INT/MINT Internal Operations

An abstract representation of the internal mechanisms of the INT and MINT algorithms that operate inside the TopkRoomM module is illustrated in Figure 6-4 (left). Similar to all aggregate

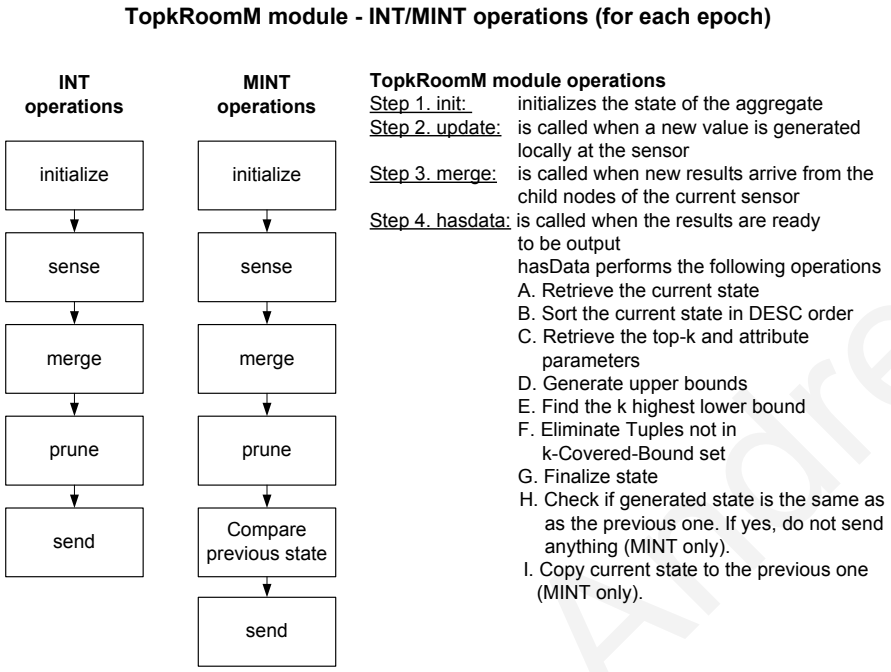

Figure 6-4: Internal operations of the INT and MINT Views algorithms.

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 Figure 6-4 (right). The pruning procedure of the underlying INT and MINT algorithms that operate inside the TopkM and TopkRoomM modules are thoroughly described in Section 6.3.

6.5.2 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 over-estimates. While our experimental evaluation in Section 7.6 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 (10x4 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.

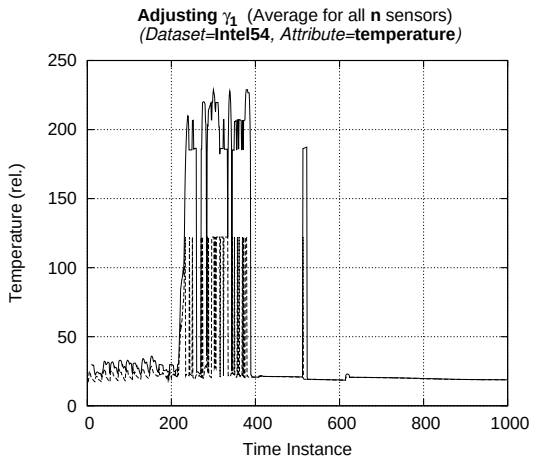

Figure 6-5: Dynamic adaptation of the γ_1 descriptor using a sliding window prediction mechanism.

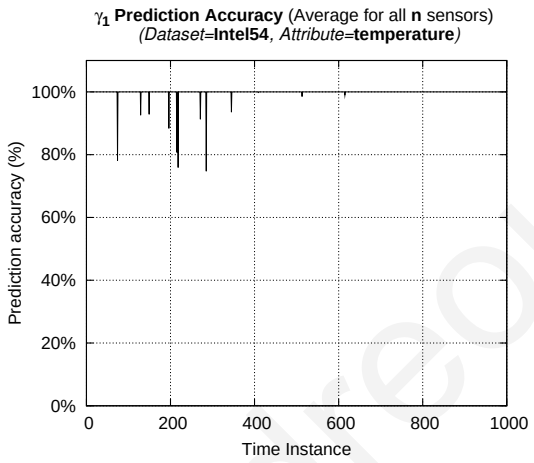

Figure 6-6: Prediction accuracy (γ_1) of the sliding window prediction mechanism.

We can configure the γ_1 descriptor using the following formula: γ_1=$MAX_{i=0}^{|CB|}(val_i \in \text{CB})$ + $2\times\sigma_{i=0}^{N}(val_i \in \text{CB})$, where σ is the standard deviation.

Figure 6-5 shows how γ_1 is dynamically adapted through 1000 timestamps using the Intel54 dataset presented in Section 7.1. We observe that γ_1 is tightly bounding the real recorded value, i.e., it is approximately $\approx$ 4% higher than the recorded value. Additionally, Figure 6-6 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 [56]. 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 dissertation.

6.5.3 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 client system. The KSpot$^+$ data structures presented in Figure 3-2 will be thoroughly described in Section 7.1.

Chapter 7

Experimental Evaluation

In this chapter we assess the efficiency of the KSpot$^+$ framework. We start by describing 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 University of California-Berkeley. Next, we present two micro-benchmarks on the CC2420 radio transceiver [122] in order to quantify its reception and transmission inefficiencies.

We conduct six experimental series that demonstrate the benefits of the KSpot$^+$ framework. In Experimental series 1 we conduct two micro-benchmarks on the CC2420 radio chip to justify why data reception and data transmission inefficiencies have to be optimized in current data acquisition systems. Experimental series 2 and 3 evaluate the performance of the Workload Balancing module and Tree Balancing module respectively. Experimental series 4 evaluates the combination of the aforementioned modules. Experimental series 5 evaluates the Query Processing module in isolation and finally Experimental series 6 evaluates the overall KSpot$^+$ framework.

In our experiments, we focus on a number of parameters including: i) Energy Consumption Cost; ii) Balancing Error; iii) Pruning Magnitude; iv) Scalability with respect to k; v) Cardinality of the GROUP-BY queries; and vi) Network Lifetime.

7.1 Experimental Methodology

In this section we describe our experimental methodology which involves both a set of real micro-benchmarks on the CC2420 radio chip [122], utilized on MICAz, TelosB and IMote2 sensing devices, and a set of trace-driven simulations with real datasets from the Department of Atmospheric Sciences at the University of Washington, Intel Research Berkeley and University of California-Berkeley. The experimental evaluation focuses on a number of parameters including Energy Consumption, Scalability and Network Lifetime.

7.1.1 Experimental Testbed

In order to evaluate the KSpot$^+$ algorithms presented in this dissertation with their competitors, we have implemented all algorithms in TinyOS.

TinyOS is an open-source operating system designed for wireless embedded sensor nodes. It was initially developed at University of California-Berkeley and has been successfully deployed 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 an on-demand basis. This minimizes the final code size and energy consumption as sensor nodes have extremely limited power and memory. nesC [50] is the programming language of TinyOS and it realizes its structuring concepts as well as its execution model.

We utilize the TOSSIM [79] environment to conduct realistic trace-driven simulations of our code with a variety of input datasets described next. TOSSIM [79] 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 [114], a popular power modeling extension of TOSSIM. As TelosB is not part of the PowerTOSSIM module, we

had to extend PowerTOSSIM by incorporating a new energy model for TelosB. PowerTOSSIM has been shown [114, 142], to be more than 90% accurate. In particular, the authors in [114] 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 like TinyDB and Surge that were shown to have an error of 9.5% on average. Consequently, the accuracy will remain at the same high levels with our integrated TelosB power model.

To compare the WART and ETC algorithms of the KSpot$^+$ framework with TAG and Cougar, we have implemented stripped-down editions of these protocols according to the descriptions provided in Section 4. We did not choose to use the TAG implementation (integrated within TinyDB) as there was no practical way to separate its implementation from the rest system due to low-level implementation details. Additionally, Cougar never emerged to an open source implementation stack.

To compare the INT and MINT Views algorithms we have implemented, or ported, all algorithms 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 [109], INT and MINT Views, were implemented from scratch in nesC [50]. Note that in the case of TINA, we utilize its baseline version, which does not include the Group-Aware Network Configuration (GANC) optimizer, so as to focus on the network optimizations for all algorithms.

In addition to our base implementation, we have also implemented a graphical user interface (see Section 3.6) that allows us to visualize the connectivity of query routing trees by displaying sensor nodes in circles and the connections to their parents using straight lines.

```
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
```

Figure 7-1: Sample execution scenario for the MINT Views algorithm on the GDI dataset.

```
…
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
…
```

Figure 7-2: Trace from the PowerTOSSIM log file.

Figure 7-1 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., 1000s (-t=1000)) and collect power statistics in the `.trace` file. An example of the PowerTOSSIM trace file is depicted in Figure 7-2. 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 2.4GHz 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 20-250MB, 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. The processing of results for the GDI140 and Intel540 datasets in many cases required 2-3 days of simulation and several hours of processing. Our simulation statistics are depicted in Figure 7-3.

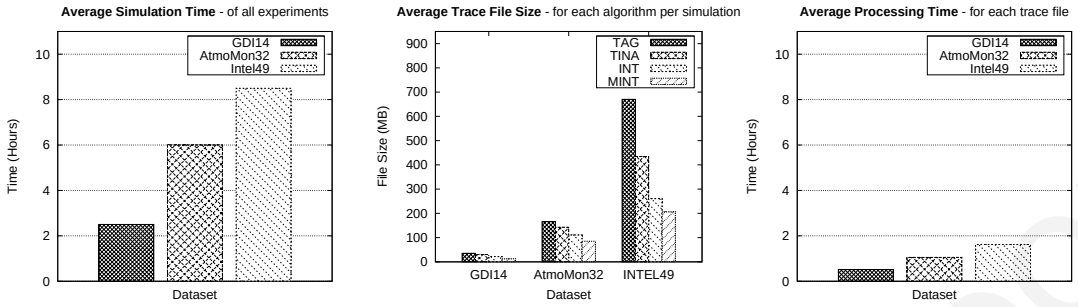

Figure 7-3: Simulation Statistics: Average Simulation Time required for each experiment (Left); Average File Size required for storing the power statistics for each algorithm (Middle). Average Processing Time for each power trace file (Right).

7.1.2 Sensing Device

We use the energy model of Crossbow's TelosB [35, 103] research sensor device to validate our ideas (see Figure 7-4). 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. We utilize a failure rate of 20% in our trace-driven experiments in order to simulate failures. In particular, a sensor has a probability of 0.2 to not participate in a given epoch. 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.).

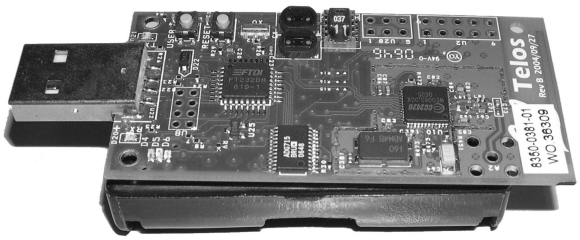

Figure 7-4: Crossbow's TelosB Mote (TPR2420). Our micro-benchmark and trace-driven experiments utilize the energy model of the TelosB sensor device and the CC2420 radio transceiver.

```
<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
…
```

Figure 7-5: Trace from the Lossy-Builder output.

7.1.3 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 Figure 7-5 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 Figure 7-5 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.

7.1.4 Communication Protocol

Our communication protocol is based on the ubiquitous for sensor networks IEEE standard 802.15.4 (the basis for the ZigBee [145] 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 0.25-0.5s before retransmission. Although collisions might be handled at a certain degree by the MAC layer [128], this scheme is agnostic of the data semantics exhibited at the higher levels of the communication stack. In this thesis we exploit these higher level semantics in order to yield better collision handling.

Our data frames are structured as following [79]: Each message is associated with a 5 *Byte* TinyOS header [77]. This is augmented with an additional 6B application layer header that includes: (i) the sensor identifier (1B), (ii) the message size (4B) and the depth of a cell from the querying node (1B). In each message we allocate 2B for environmental readings (e.g., temperature, humidity, etc.), 4B for aggregate values (`max, min` and `sum`) and 8B for timestamps. ZigBee's MAC layer dictates a maximum data payload of 104 bytes thus we segment our data packets whenever this is required.

The TinyDB message frames associated with the KSpot algorithms are structured as follows [79]: 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.

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 x 3-bits = 21), iv) *room (22 bits)*: This attribute records the room id of each room, and v) *sum (16 bits x 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.

7.1.5 Datasets

We utilize the following real and realistic datasets in our trace-driven experiments in order simulate small-scale, medium-scale and large-scale wireless sensor networks.

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 [121], 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 [43]. More specifically, each of the 32 sensors

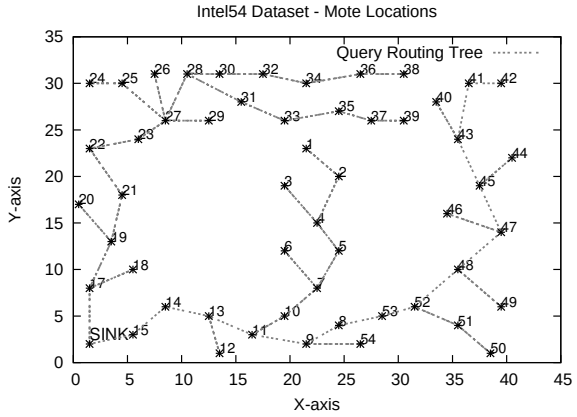

Figure 7-6: The location of the 54 sensors in the Intel54 dataset and an ad-hoc query tree constructed using the FHF approach.

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 (Intel54)**: This is a real dataset that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley [64] 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 54 sensors that had the largest amount of local readings since some of them had many missing values. More specifically, we utilize the real coordinates of the 54 sensors (see Figure 7-6). The depth of the initial query routing tree constructed with the FHF approach is 14.

iv. **Great Duck Island (GDI140)**: In order to evaluate our approach on a medium-scale sensor network we synthetically derive a sensor network composed of 140 nodes that follows the same distribution with the GDI14 dataset. The average depth of the initial query routing tree constructed with the FHF approach is 24.

v. **Intel Research Berkeley (Intel540)**: In order to evaluate our approach on a large-scale sensor network we synthetically derive a 540-node network based on the Intel54 dataset. The distribution of the dataset follows again the same distribution with the Intel54 dataset. The average depth of the initial query routing tree constructed with the FHF approach is 22.

7.1.6 Query Sets

We utilize three representative queries from two predominant classes of queries in wireless sensor networks.

The first class of such queries is *aggregate selection queries* [135, 86] (i.e., `SELECT agg() FROM sensors`). Roughly, these queries 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. The separation between the above cases is important as each individual case defines a different workload per edge (i.e., distributive aggregates have a *fixed workload* of one tuple per edge while holistic aggregates a *variable workload*).

The second class of representative queries is *non-aggregate selection queries* (e.g., `SELECT moteid FROM sensors`). Assuming a static topology such queries generate a *fixed workload* per edge, unless we apply a predicate on the query (e.g., `temperature > X`) and generate in this manner a *variable workload* per edge.

In our experiments we utilize the following query-sets which encapsulate all the above cases:

- *Single-Tuple queries (***ST***)*: where a sensor transmits exactly one tuple per epoch. Distributive aggregates belong to this category. We utilize the following representative query in our study:

```
SELECT moteid, temperature
FROM sensors
WHERE temperature=MAX(temperature)
EPOCH DURATION 31 seconds
```

- *Multi-Tuple queries with Fixed size (***MTF***)*: where a sensor transmits a set on f tuples per epoch, where f is a constant. Holistic aggregates and non-aggregate selection queries with a fixed workload belong to this category. We utilize the following representative query in our study:

```
SELECT moteid, temperature
FROM sensors
EPOCH DURATION 31 seconds
```

- *Multi-Tuple results with Arbitrary size (***MTA***)*: where a sensor transmits a set of f' tuples per epoch, where f' is a variable that might change across different epochs. Non-aggregate selection queries with a variable workload belong to this category. We utilize the following representative query in our study:

```
SELECT moteid, temperature
FROM sensors
WHERE temperature>39
EPOCH DURATION 31 seconds
```

Each query features an epoch duration which specifies the amount of time that sensors have to wait before re-computing the continuous query. Additionally, for the Cougar and WART algorithms, we set the child waiting timer h to 200ms. If the timer for a sensor s_i runs out then s_i will

not wait for any more results from its children. Such a timer is deployed to avoid situations where nodes have to wait for children nodes for an unspecified amount of time.

7.2 Experimental Series 1: Microbenchmarks

In the first experimental series we have conducted two micro-benchmarks on the CC2420 radio chip [122] (both attached to the TelosB [35] sensor and in TOSSIM [79]) to justify why data reception and data transmission inefficiencies have to be optimized in current data acquisition frameworks. For the first type of inefficiency we show why a sensing device should not change the state of its transceiver more than once during the interval of an epoch. That supports our argument that query results have to be communicated between sensors at a specific time instance rather than at several time instances. For the second type of inefficiency we justify why a sensor network should minimize the number of hub nodes (i.e., nodes with several children) as these increase collisions during data transmission and thus also increase energy consumption.

In the first micro-benchmark 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 7-7 shows the result of the first micro-benchmark. We observe that by changing the transceiver status 1000 times consumes $195\mu J$ while conducting the same operation one time requires only $128\mu J$. 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 to other transceivers. Notice that during the startup of

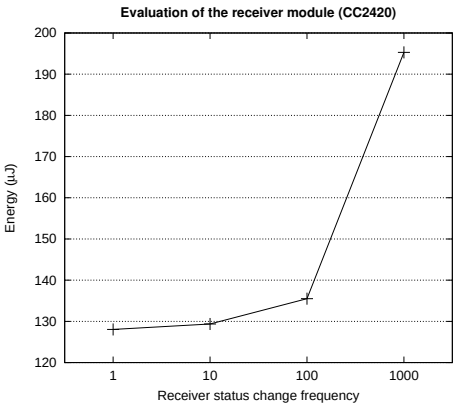

Figure 7-7: Micro-benchmarks using the CC2420 communication module: Changing the transceiver status from on to off many times significantly increases energy consumption.

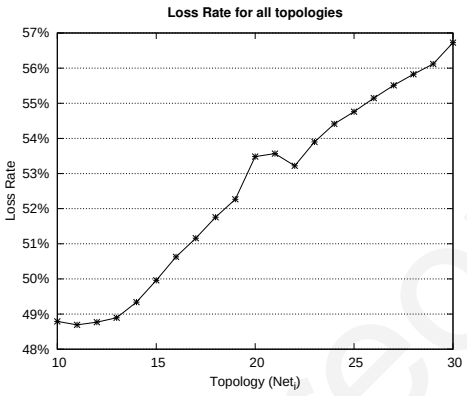

Figure 7-8: Micro-benchmarks using the CC2420 communication module: Increasing the number of children per node x also increases collisions during data transmission to node x.

the RF module a voltage regulator and crystal oscillator have to be started as well and become stable [122]. Thus, it is quite inefficient to change the transceiver state (from on to off and vice-versa) more than once during the interval of an epoch. The WART algorithm of the Workload Balancing Module presented in Section 4 assigns a specific time interval to each child node during which query results have to be transmitted to a parent node, thus the transceiver is enabled only once.

In the second micro-benchmark we justify why a sensor network should minimize the number of hub nodes (i.e., nodes with a large in-degree). For this purpose we construct 20 star topologies Net_i ($10 \geq i \geq 30$) with each of which features i children nodes, and evaluate the loss rate when all children nodes attempt to transmit data packets to a given sink node. In particular, each node attempts to transmit a 16-byte packet to a given sink node for 60 seconds (that accounts to approximately 250 messages in our setting). We utilized the TOSSIM environment along with its LossyBuilder module that created "lossy" radio models for each topology. The lossy model we have created (for each of the topologies) places the sensors at various distances from the sink node and generates a Gaussian packet loss probability distribution for each distance. TOSSIM then

generates packet loss rates for each sensor-sink pair by sampling these distributions and translates this into independent bit error rates.

For each topology Net_i ($10 \geq i \geq 30$) we measure: i) the *Total Packets Sent* from all sensors to s_0 (denoted as P_i^T), and ii) the *Total Packets Received* from s_0 (denoted as P_i^0). We next evaluate each topology's loss rate by using the formula:

$$LossRate(Net_i) = 1 - (\frac{P_i^0}{P_i^T}) \quad (5)$$

Figure 7-8 illustrates the loss rate for the 20 presented topologies. We can observe an almost linear increase in the loss rate for topologies with more than 10 children nodes. For a setup of 30 children nodes we observe a loss rate of over 56%. We tried to scale the experiments to 100 children nodes and observed that the loss rate peaked at 77%. But even for smaller-scale cases, many data packets do not reach their designated destination in the first attempt and need to be re-transmitted (the energy cost of this deficiency will be documented in the subsequent experiments). It should be noted these findings are highly correlated with the lossy model generated by the TOSSIM's LossyBuilder component. More pessimistic lossy models would have generated even higher loss rates. However, investigating the results of our experiments indicates that nodes closer to the sink node manage to transmit more messages successfully and that is why the loss rates may appear somewhat optimistic. The Tree Balancing Module of KSpot$^+$ framework presented in Section 5, distributes the children of overloaded nodes to neighboring nodes and assigns different wake-up times decreasing in that way data transmission collisions and energy consumption.

7.3 Experimental Series 2: Evaluation of the Workload Balancing Module

In this experimental series, we assess the efficiency of the Workload Balancing Module presented in Chapter 4. More specifically, we have conducted four experiments that evaluate the efficiency of the underlying WART algorithm under variable workloads and network sizes. In the first experiment we compare the energy consumption of the WART algorithm to the respective algorithms deployed in the Cougar and TinyDB frameworks using real datasets under a variety of query workloads and topologies. In the second experiment we demonstrate the scalability of the WART algorithm on large scale networks using the realistic datasets presented in Section 7.1. Next, in the third experiment we evaluate the the adaptation phase of the WART algorithm. Finally, in the fourth experiment we show that the utilization of the WART algorithm can significantly increase the longevity of a wireless sensor network.

More specifically, this experimental evaluation focuses on two parameters:

1. the **Energy Consumption Cost**, for the WART algorithm presented in Chapter 4 under variable workloads (single-tuple results, multi-tuple results with fixed size and multi-tuple results with arbitrary size), and

2. the **Network Lifetime**, of the WART algorithm compared to the TAG and Cougar Algorithms used in the TinyDB and Cougar middleware frameworks respectively.

Table 7-1 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

7.3.1 Energy Consumption

We study the energy consumption of the WART, Cougar and TAG algorithms for the different combinations of query sets (ST, MTF and MTA) to datasets (Intel54, GDI140 and Intel540) as

Table 7-1: Configuration parameters for all experimental series of Section 7.3.

Section	Objective	Datasets	Workload	Algorithms
7.3.1	Energy Consumption Medium-scale	Intel54	ST,MTA MTF	TAG,Cougar, WART
7.3.2	Energy Consumption Large-scale	GDI140, Intel540	ST,MTA MTF	TAG,Cougar, WART
7.3.3	Energy Consumption	Intel54,GDI140, Intel540	ST,MTA MTF	WART WART-adaptation
7.3.4	Network Lifetime	Intel540	MTF	TAG,Cougar, WART

these were described in Section 7.1.

Energy Consumption for Single-Tuple Answers: Figure 7-9 (top-left) shows the energy consumption for the Intel54 dataset using the single-tuple query ST. We observe that TAG requires 11,227$\pm$2mJ, which is two orders of magnitudes more energy than the energy required by WART (i.e., only 53$\pm$35mJ). This is attributed to the fact that the transceiver of a sensor in TAG is enabled for $\approx$2.14 seconds in each epoch (i.e., $\lfloor e/d \rfloor = 31$ (epoch duration)/14 (tree depth)), while in WART it is only enabled for $\approx 146ms$ on average. Enabling the transceiver for over two seconds in TAG is clearly the driving force behind its inefficiency. Figure 7-9 (top-left) also shows that the WART energy curve quickly drops to the mean value of $53mJ$ within the first epoch (i.e., the sudden drop at the beginning of the curve). Notice that WART runs very much like Cougar during the first epoch but our algorithm then intelligently exploits the waking window cost to preserve energy.

Figure 7-9 (top-left) also shows that the Cougar algorithm requires on average 882$\pm$250mJ, which is one order of magnitude more than the energy required by WART. The disadvantage of

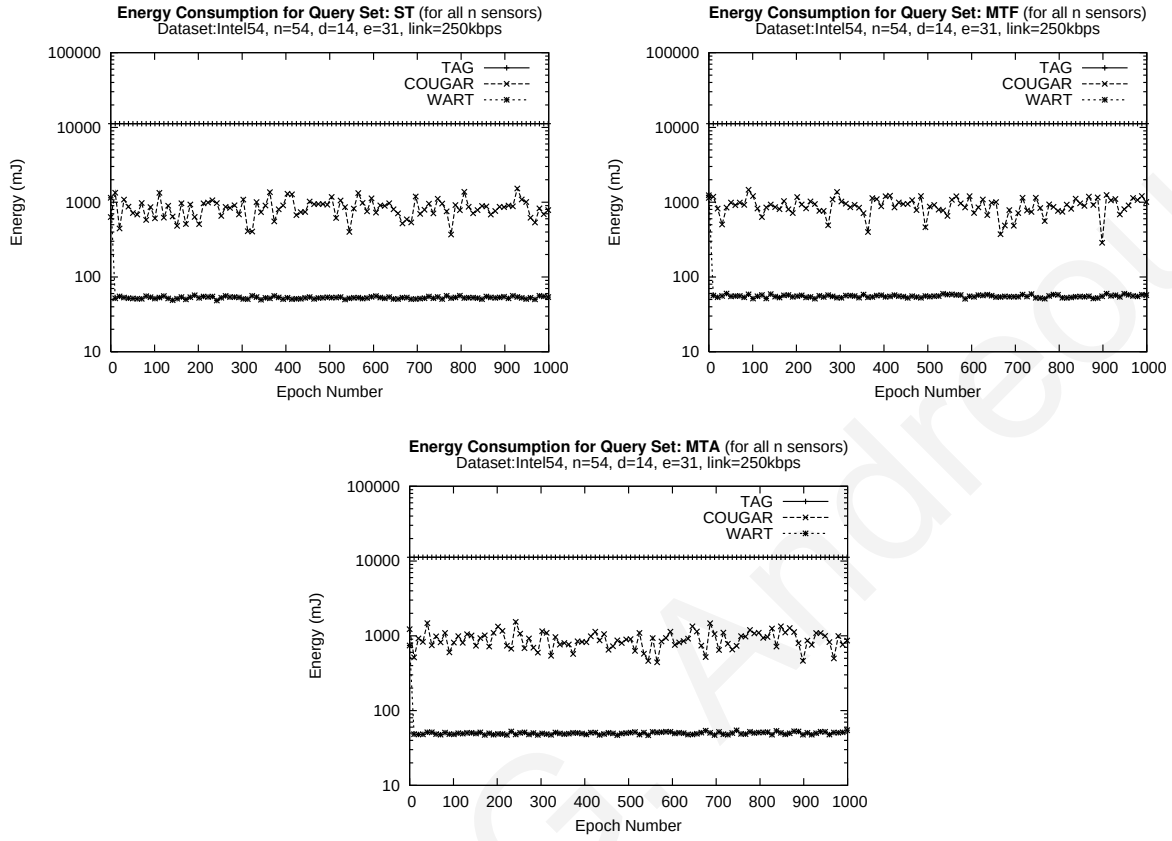

Figure 7-9: Energy consumption for Single-Tuple (top-left) and Multi-Tuple (top-right and bottom) answers. The plots indicate the individual results for the TAG, Cougar and WART data acquisition algorithms. In all figures we observe that WART is at least one order of magnitude more efficient than its competitors.

the Cougar algorithm originates from the fact that the parents keep their transceivers enabled until all the children have answered or until the local timer h has expired (in cases of failures). Thus, any failure is automatically translated into a chain of delayed waking windows all of which consume more energy than necessary. One final observation regarding the Cougar algorithm is that it features a large standard deviation (i.e., $250mJ$), which signifies that certain nodes consume more energy than others. This is attributed to the fact that the cost of failures in Cougar is proportional to the depth of the node that caused the failure. In particular, failures at a large depth (i.e., closer

to the leaf nodes) will generate a larger chain of waking windows, thus will be more energy demanding than failures that occur at a small depth (i.e., closer to the sink).

Energy Consumption for Multi-Tuple Answers: We shall next measure the energy cost of the queries with multi-tuple answers (i.e., MTF and MTA) again over the Intel54 dataset and present our results in Figure 7-9 (top-right and bottom) and also summarize these results in Table 7-2 (first row). From the figures and the table we can draw the following conclusions: i) the WART algorithm has the same compelling benefits compared to TAG and Cougar, although the incurred workload for the three queries is very different; and ii) the MTA query consumes on average less energy than the ST query for all algorithms (see Table 7-2 (first row)). This is attributed to the fact that MTA is associated with a predicate that limits the cardinality of sensor answers below one in certain cases, while the ST query yields exactly one answer per sensor. On the contrary, the MTF query has an increased energy consumption compared to the ST query (i.e., between 1-10mJ) as it generates multiple tuples at each node. To explain this, first notice that it is relatively inexpensive to pack a small number of additional tuples into a message, given that the transmission cost is dominated by the packet header and not by the payload. As the cardinality of an MTF query is bounded above by the number of sensors n (see query MTF), in practice this yields only a small increase in the number of messages. Thus, the additional energy consumption of the MTF query over the ST query is very small.

By evaluating the same algorithms over the medium-size GDI140 dataset, presented in Table 7-2 (second row), we observe that WART continues to maintain a competitive advantage over the other two algorithms. Another observation is that the TAG algorithm has a slightly better performance compared to the previous experiment but its performance is still two orders of magnitudes worse than WART. In particular, we noticed that the TAG-to-WART performance ratio is

Table 7-2: Energy Consumption results for experimental series 7.3.1: Evaluation of the WART, Cougar and TAG algorithms under different queries and datasets.

Dataset	Query / Algor.	**ST**	**MTF**	**MTA**
Intel54	**TAG**	11,227±02mJ	11,228±02mJ	11,225±01mJ
	Cougar	882±250mJ	893±239mJ	877±239mJ
	WART	53±35mJ	56±37mJ	50±21mJ
GDI140	**TAG**	58,380±24mJ	58,380±25mJ	58,374±26mJ
	Cougar	1,435±176mJ	1,443±181mJ	1,432±176mJ
	WART	429±39mJ	438±37mJ	425±34mJ
Intel540	**TAG**	189,691±53mJ	189,707±49mJ	189,670±51mJ
	Cougar	7,269±37mJ	7,317±37mJ	7,257±37mJ
	WART	3,431±14mJ	3,510±12mJ	3,398±13mJ

slightly decreased (i.e., 136%) compared to the respective performance ratio recorded with the Intel54 dataset which was 211%. Such a decrease is explained as follows: the depth of the query routing tree in GDI140 was 22 and thus each sensor has to maintain its radio open for ≈ 1.40 seconds in each epoch (i.e., $\lfloor e/d \rfloor = 31$ (epoch duration)/22 (tree depth)). On the contrary, the depth of the query routing tree in Intel54 was 14 and thus each sensor has to maintain its radio open for a larger window in each epoch (i.e., ≈ 2.14 seconds).

7.3.2 Large-Scale Network Study

In the third experiment of this series we evaluate the WART algorithm against the Cougar algorithm using the Intel540 dataset, which represents a large-scale wireless sensor network. We have omitted the presentation of the TAG algorithm as it has a very high energy cost (i.e., $189,707mJ$). To facilitate our presentation we also summarize the mean and standard deviation of our results in Table 7-2 (third row).

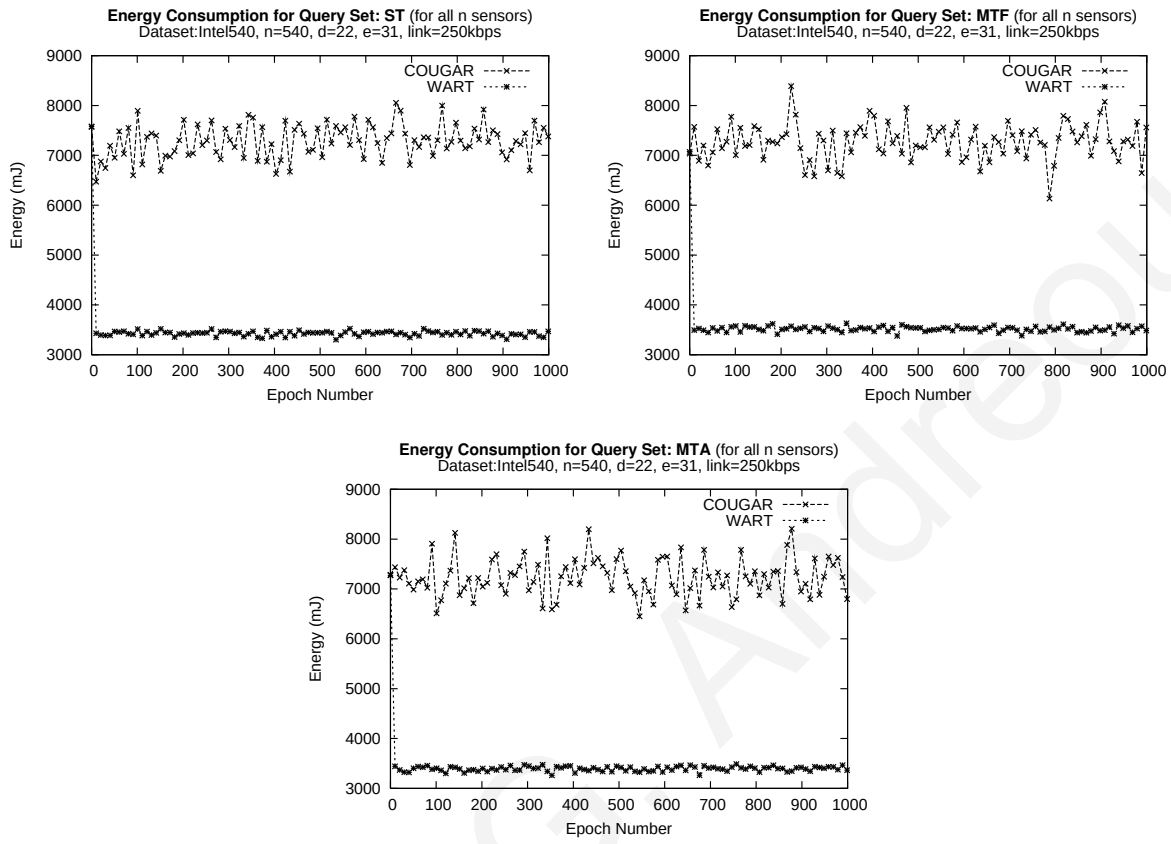

Figure 7-10: Energy Consumption in a Large-Scale Sensor Network (Intel540). The plots indicate the individual results for the ST, MTF and MTA queries using the Cougar and WART algorithms (we omit the TAG curve in this plot due to its inefficiency (i.e., $189,707mJ$)).

The plots in Figure 7-10 show that WART requires only 3,446mJ on average (i.e., the mean of the plots for all three queries) while Cougar requires as much as 7,281mJ for the acquisition of values from all 540 nodes. This shows that WART retains a significant competitive advantage over Cougar even for large-scale wireless sensor networks. For all queries we noticed that the WART-to-Cougar performance ratio is slightly increased (i.e., 47%) compared to the respective performance ratio noticed with the Intel54 dataset (which was only 6%). Such an increase was expected as larger networks have a higher probability of transient network conditions and arbitrary failures. The above characteristics are causes that lead to the disruption of the query routing tree synchrony. Nevertheless, the WART approach is still 53% more energy efficient than Cougar under these limitations, thus WART can have many practical applications in large-scale environments.

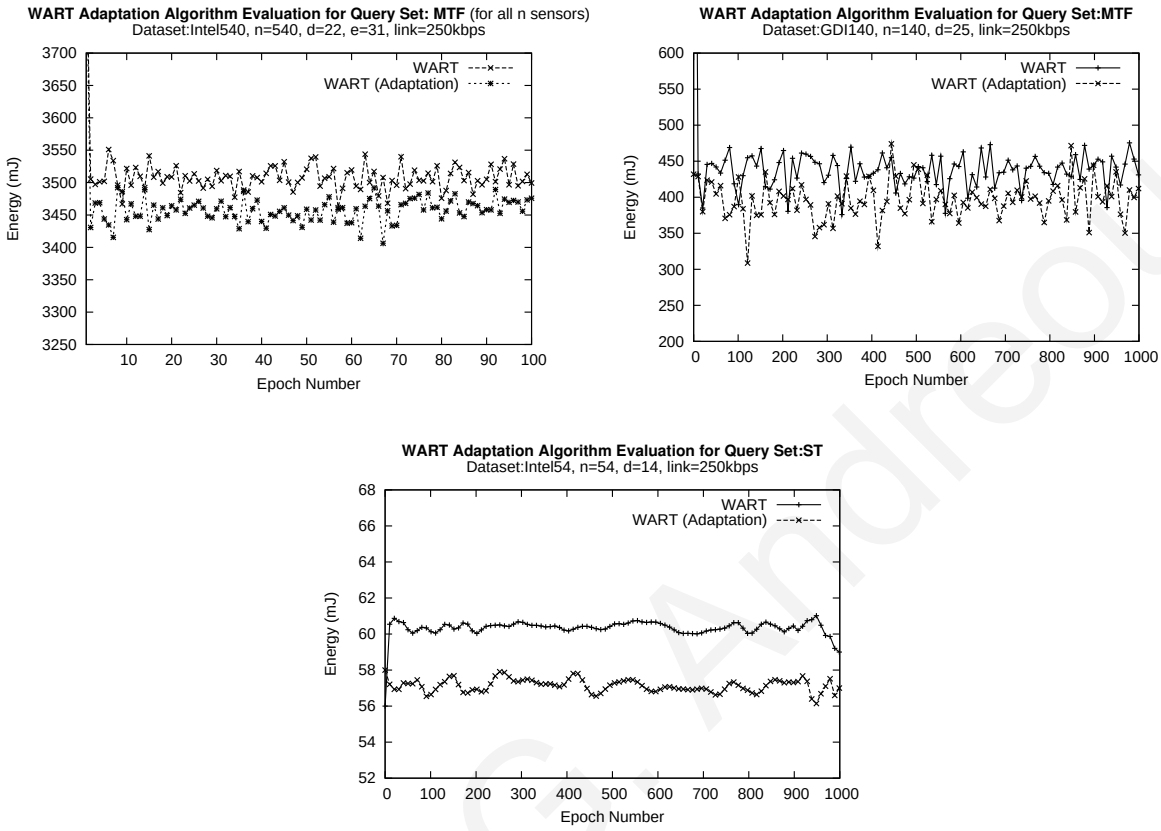

Figure 7-11: WART's Adaptation algorithm evaluation for the Intel540 dataset (top-left), GDI140 dataset (top-right) and the Intel54 dataset (bottom).

7.3.3 Adaptation Phase Evaluation

In the last experiment of this series we evaluate the WART adaptation algorithm. So far we have assumed that the critical path is re-constructed in every epoch during the execution of a query, thus introduced an additional cost of $O(n)$ messages. In the following experiments we aim to investigate the efficiency of the WART adaptation algorithm and verify the savings we claimed in Section 4.3.4. We compare WART with no adaptation against a version that employs the adaptation rules of Algorithm 2 during data acquisition. For this experimental series we utilize the Intel54, Intel540 and GDI140 datasets and present the results for the MTF query only as the other two queries expose a similar behavior.

Figure 7-11 (top-left, top-right) shows that the invocation of the adaptation rules in Algorithm 2 for the Intel540 and GDI140 large scale networks can yield additional energy savings of $60mJ$ and $36mJ$ respectively. Given that one packet in our setting was 128 bytes we can estimate that the transmission of such a packet requires $144\mu J$ (see Section 7.1). In the case of the Intel540 dataset, the quantity of 60mJ is approximately equivalent to 416 messages (i.e., $60mJ/144\mu J$) whereas in the case of the GDI140 dataset the quantity of 36mJ is approximately equivalent to 290 messages (i.e., $36mJ/144\mu J$). This result is consistent with our analysis were we expected $O(n)$ additional messages during the dissemination phase. Figure 7-11 (bottom) shows the adaptation algorithm on a small scale network (i.e., Intel54). The result indicates that even for such small-scale networks we might observe some energy savings but these are not very significant (i.e., only $2mJ$). This is attributed to the fact that the adaptation rules in small-scale networks are not invoked as frequently as workload deviations occur more rarely.

7.3.4 Network Lifetime

The final performance criterion we have considered is 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*, similar to [123], 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)

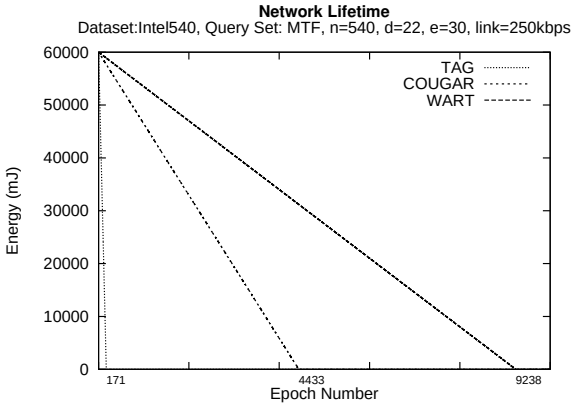

Figure 7-12: Network Lifetime evaluation of the WART, TAG and Cougar algorithms.

utilized in other works [109, 110]. 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 2,500 mAh (milliAmpere per hour). Assuming similar to [121], that only 2,200mAh is available and that all current is used for communication, we can calculate that AA batteries offer $23,760J$ ($2,200mAh \times 60min \times 60s \times 3V$). In order to speed up our experiments we start with an initial energy of $60,000mJ$ subtract at each epoch and for each sensor the energy required for communication. When terminate this iteration when the termination condition is satisfied.

Figure 7-12 illustrates the average energy status of the sensor network, at each epoch, during the execution of a query. We notice that the energy of sensors under TAG is consumed far faster than the WART algorithm, leading to a lifetime of just 171 epochs (i.e., 85 minutes). Cougar comes second by offering 4,433 epochs (i.e., 36 hours), Finally, WART comes third with 9,238 epochs (i.e., 77 hours) which can be translated to ≈208% increase in network lifetime.

7.4 Experimental Series 3: Evaluation of the Tree Balancing Module

In this experimental series, we assess the efficiency of the Tree Balancing Module presented in Chapter 5. More specifically, we have conducted two experiments that evaluate the efficiency of the underlying ETC algorithm under different network sizes.

In the first experiment we measure the quality of the near-balanced query routing tree that is generated by the ETC algorithm T_{ETC} against the original ad hoc query routing tree T_{input} and the one generated by the Centralized ETC algorithm (CETC) T_{CETC}. In the second experiment we compare the energy consumption of T_{input} with T_{ETC} with respect to data collisions.

More specifically, this experimental evaluation focuses on two parameters:

1. the **Balancing Error**, for measuring the balancing quality of the near-balanced tree generated by the ETC algorithm, and

2. the **Energy Consumption Cost**, with respect to data collisions of the original ad hoc query routing tree T_{input} and the one generated by the ETC algorithm (ETC) T_{ETC}.

Table 7-3 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

Table 7-3: Configuration parameters for all experimental series of Section 7.4.

Section	Objective	Datasets	Algorithms
7.4.1	Balancing Error	Intel54, GDI140, Intel540	Ad hoc, ETC CETC
7.4.2	Energy Consumption	Intel540	Ad hoc, ETC

7.4.1 Measuring the Balancing Error

Our first objective is to measure the quality of the tree, w.r.t the balancing factor, that is generated by the ETC algorithm. Thus, we measure the balancing error of the generated trees as this was presented in Section 5.3.1. Recall that the Balancing_Error of a query routing tree was defined as follows:

$$Balancing\_Error(T_{CETC}) := \sum_{i=0}^{n} |\beta - \sum_{j=0}^{n} PM_{ij}|$$

where $\beta = \sqrt[d]{n}$ and $PM_{ij} = 1$ denotes that node i is a parent of node j and $PM_{ij} = 0$ the opposite. Notice that this table is fragmented vertically in the case of the distributed ETC algorithm but can be obtained easily with a message complexity of O(n), where each message has a size of $O(n^2)$ in the worst case.

For this experiment we generated one query routing tree per dataset (i.e., Intel54, GDI140 and Intel540) using the three described algorithms: i) the First-Heard-From approach, which constructs an ad-hoc spanning tree T_{input} without any specific properties, ii) the CETC algorithm, which transforms T_{input} into the best possible near-balanced tree T_{CETC} in a centralized manner, and iii) the ETC algorithm, which transforms T_{input} into a near-balanced tree T_{ETC} in a distributed manner.

Figure 7-13, presents the results of our evaluation which demonstrates the following properties: i) All three approaches feature some balancing error, which indicates that in all cases it is not feasible to construct a fully balanced tree $T_{balanced}$. This is attributed to the inherent structure of the sensor network where certain nodes are not within communication radius from other nodes. ii) The second observation is that the FHF approach has the worst Balancing_Error, which is an indicator that FHF can rarely produce any proper balanced topology and that increases data

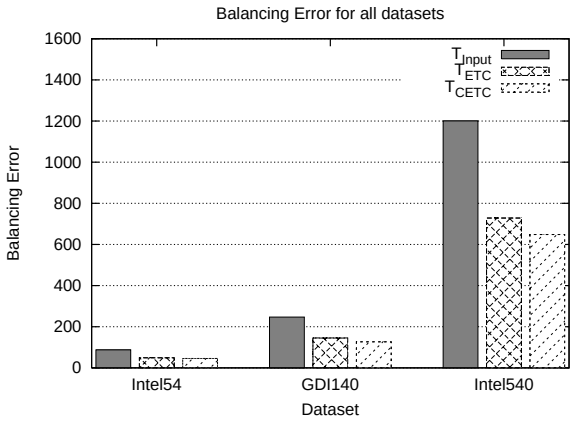

Figure 7-13: Measuring the Balancing Error of the FHF (T_{input}), CETC (T_{CETC}) and ETC (T_{ETC}) algorithms

transmission collisions and energy consumption (shown in next experiment). In particular, the balancing error of the FHF approach is on average 91% larger than the respective error for the CETC algorithm. iii) The third and most important observation is that the distributed ETC algorithm is only 11% less accurate than the centralized CETC algorithm. Therefore, even though the ETC algorithm does not feature any global knowledge, it is still able to create a near-balanced topology in a distributed manner.

7.4.2 Energy Consumption

In order to translate the effects of the Balancing_Error into an energy cost, we conduct another experiment using the Intel540 dataset. More specifically, we generate two query routing trees: a) T_{input}, constructed using the First-Heard-From approach, and b) T_{ETC} constructed using the ETC algorithm. We configure our testbed to only measure the energy required for re-transmissions due to collisions in order to accurately capture the additional cost of having an unbalanced topology.

Figure 7-14 displays the energy consumption of the two structures. We observe that the energy required for re-transmissions using T_{input} is 3,314$\pm$50mJ. On the other hand, T_{ETC} requires only

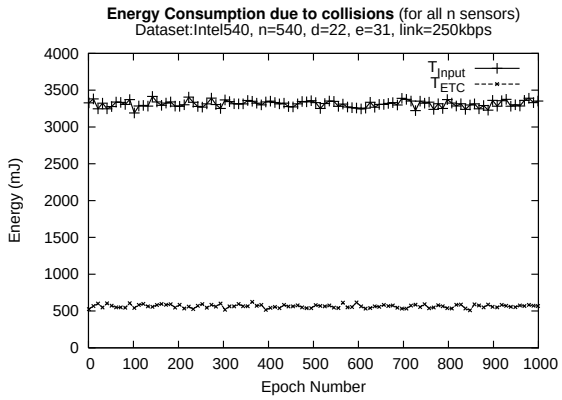

Figure 7-14: Energy Consumption due to re-transmissions in an unbalanced topology (T_{input}) and in a near-balanced topology (T_{ETC})

566$\pm$22mJ which translates to additional energy savings of 83%. The reason why T_{ETC} presents such great additional savings is due to the re-structuring of the query routing tree into a near balanced query routing tree which ensures that data transmissions collisions are decreased to a minimum.

7.5 Experimental Series 4: Fusing the Workload and Tree Balancing Modules

In experimental series 2 and 3 we studied the performance of the Workload and Tree Balancing modules in isolation in order to highlight their distinct properties. In this experimental series we study them in conjunction to demonstrate their combined efficiency. In particular, we fuse the Workload and Tree Balancing modules, labeled `ETC+WART`, as follows. The fused algorithm deploys firstly the ETC algorithm to balance the query routing tree and then utilizes the WART algorithm to optimize the waking windows of the sensor nodes.

More specifically, this experimental evaluation focuses on two parameters:

1. the **Energy Consumption Cost**, for the ETC+WART algorithm presented in Chapter 4 under variable workloads (single-tuple results, multi-tuple results with fixed size and multi-tuple results with arbitrary size), and
2. the **Network Lifetime**, of the ETC+WART algorithm compared to the WART, TAG and Cougar Algorithms.

Table 7-4 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

Table 7-4: Configuration parameters for all experimental series of Section 7.5.

Section	Objective	Datasets	Workload	Algorithms
7.5.1	Energy Consumption	Intel540	MTF	ETC+WART, WART
7.5.2	Network Lifetime	Intel540	MTF	TAG, Cougar, WART, ETC+WART

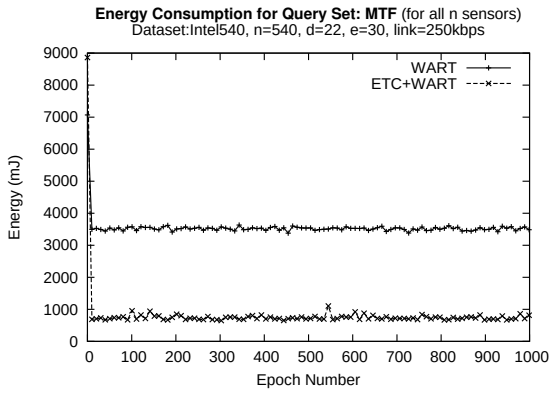

Figure 7-15: Energy Consumption comparison of ETC+WART against WART. The plot indicates that the ETC technique provides a threefold improvement to the savings incurred by the WART algorithm

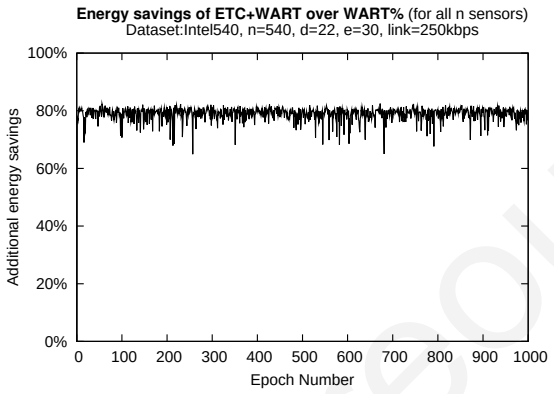

Figure 7-16: Energy savings of ETC+WART against WART (%)

7.5.1 Energy Consumption

In the first experiment of this series we measure the energy consumption of the integrated WART and ETC algorithms using the Intel540 dataset and the MTF query. We have observed similar results for the other combinations of query-to-datasets as well and omitted these results for brevity.

Figure 7-15 illustrates the energy savings of using the ETC algorithm in conjunction with WART. While WART requires on average 3,510$\pm$126mJ, ETC+WART uses only 749$\pm$269mJ which translates in an additional 78% decrease (see Figure 7-16) in energy consumption on average. In particular, we have observed a threefold improvement of ETC+WART compared to the execution of the WART algorithm in isolation. Additionally, we have noticed that the near-balanced routing tree generated by the ETC algorithm will not only reduce data collisions, and thus data re-transmissions, but will also have a positive effect on the WART scheduling algorithm.

The reason why the efficiency of the WART algorithm increases under no failures can be explained as follows: In a topology with limited failures the critical path cost is not re-computed

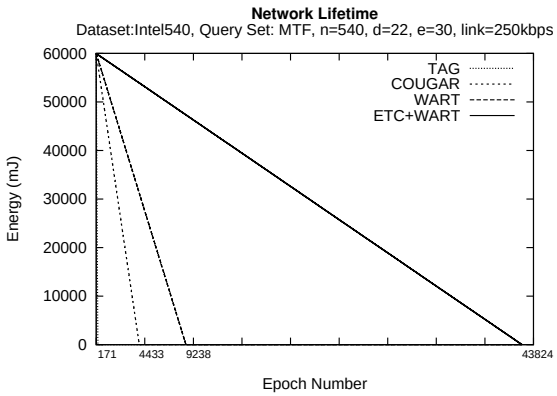

Figure 7-17: Network Lifetime for all algorithms.

very often. Thus, the communication overhead is minimized. Additionally, in a topology with a small number of failures we also have a smaller number of parent waiting for their children (i.e., a fewer number of expired h timers). Consequently, minimizing data transmission collisions automatically triggers a whole range of new characteristics which improve the overall quality of our framework.

7.5.2 Network Lifetime

The final performance criterion we have considered is network lifetime. We use the same definition for define network lifetime as the one presented in Section 7.3.4.

Figure 7-17 illustrates the average energy status of the sensor network, at each epoch, during the execution of a query. We notice that the energy of sensors under TAG is consumed far faster than ETC+WART, leading to a lifetime of just 171 epochs (i.e., 85 minutes). Cougar comes second by offering 4,433 epochs (i.e., 36 hours) and WART third with 9,238 epochs (i.e., 77 hours). Finally, ETC+WART reaches its limit far later at epoch 43,824 (i.e., 365 hours) and this can be translated into a $\approx$474% increase of the network lifetime (over WART).

7.5.3 Multi-query Execution

In a real system, it will be necessary to execute several queries, possibly belonging to individual users, concurrently. In this subsection we discuss at an abstract level how this can be realized. First, notice that the WART algorithm, which minimizes data reception inefficiencies by profiling recent data acquisition activity, can maintain separate profiles for the individual queries running over a given query routing tree. Furthermore, the ETC algorithm, which generates a near-balanced tree topology that minimizes data collisions, is query-independent. In particular, an ETC tree reconfigures itself based on a balancing factor that is derived directly from the branching factor of a node in a query routing tree. Consequently, the same physical tree might apply to any query running over KSpot$^+$. The above discussion shows that it is relatively easy to extend the KSpot$^+$ framework into a multi-query execution environment although a detailed investigation of this parameter is outside the scope of this dissertation.

7.6 Experimental Series 5: Evaluation of the Query Processing Module

In order to assess the efficiency of the Query Processing Module we have conducted six experiments. In the first experiment 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 experiment, we study the pruning magnitude of the INT and MINT Views algorithms. In the third experiment, 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 experiment 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. Finally, in the fifth experiment, we show that the utilization of the Query Processing Module can significantly increase the longevity of a wireless sensor network.

More specifically, the experimental evaluation focuses on five parameters:

1. the **Energy Consumption Cost**, for the INT and MINT Views algorithms proposed in Chapter 6 compared to two other popular query processing algorithms namely, TAG and TINA,

2. the **Pruning Magnitude**, of the k-Covered Bound-Set V_i^i of the INT and MINT Views algorithms,

3. the **Scalability with respect to** k, were we evaluate the efficiency of the MINT Views algorithm with different values of the k parameter,

4. 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

5. the **Network Lifetime**, of all algorithms presented in this Section.

Table 7-5 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

Table 7-5: Configuration parameters for all experimental series of Section 7.6.

Section	**Objective**	**Dataset**	k	**Rooms (R)**
7.6.1	Energy Consumption	GDI14, AtmoMon32, Intel54	5%	4-7
7.6.2	Pruning Magnitude	AtmoMon32	5%	7
7.6.3	Scalability of k	GDI14	5%-100%	4
7.6.4	GROUP-BY cardinality	GDI14	5%	1-7
7.6.5	Network Lifetime	GDI14	5%	4

7.6.1 Energy Consumption

In the first experiment, 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.

In Figure 7-18 (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.

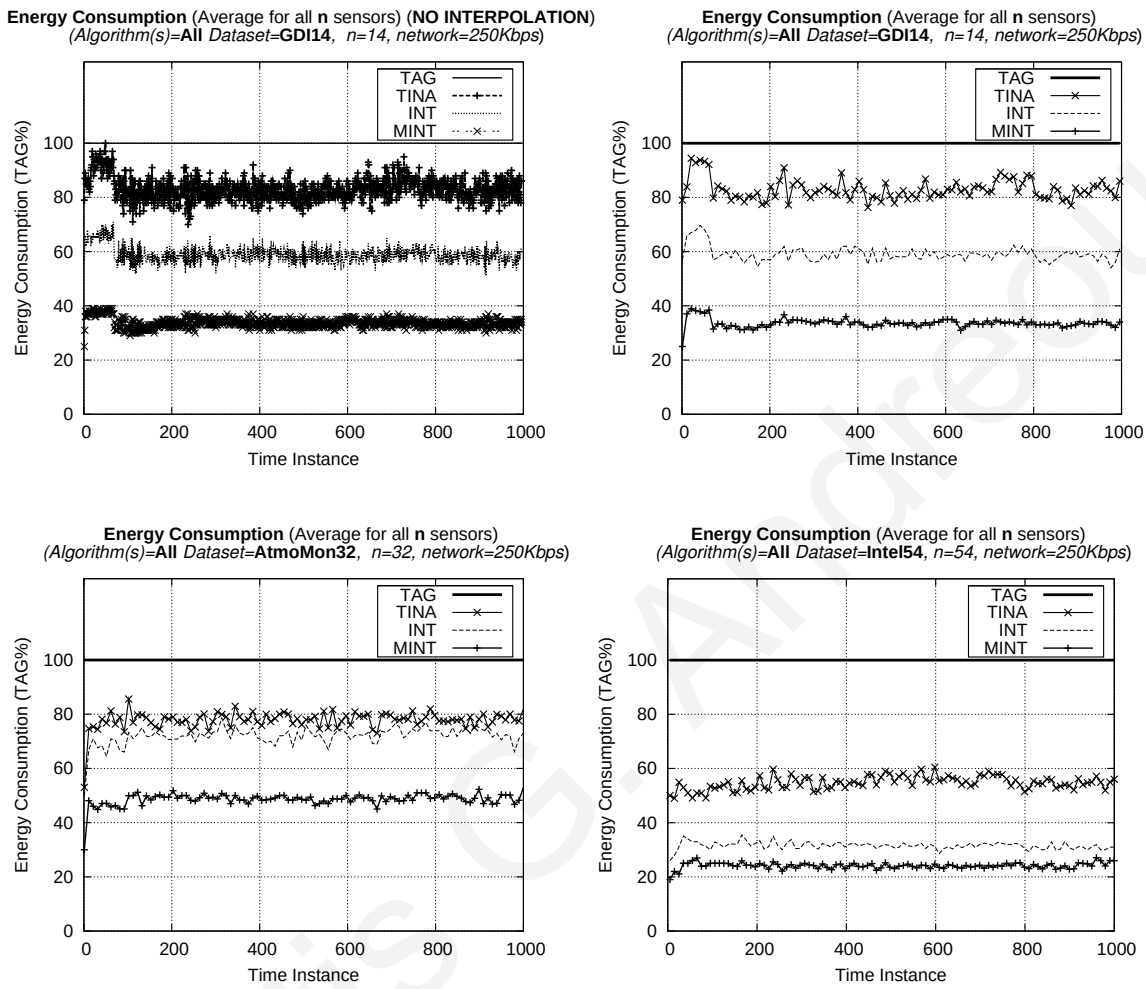

Figure 7-18: Energy Consumption for the TAG, TINA, INT View and MINT View algorithms using the TelosB sensor energy model.

In Figure 7-18 (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$\pm$2.52J average energy for all 14 nodes of the network. Recall that in TAG, a sensor always transmits all aggregated tuples to the sink. Although TINA still returns all answers to the sink, it takes the average energy consumption down to 48$\pm$1.57J. This validates that exploiting temporal coherence can be beneficial in most cases. The INT Views approach on the other hand, performs in-network pruning of the results which reduces the energy consumption to 34$\pm$1J (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$\pm$0.56J 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.

By repeating the same experiment on the AtmoMon32 dataset, we observe in Figure 7-18 (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$\pm$4J versus 234$\pm$2J). 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.

Table 7-6: Average Energy Consumption for all sensors in experimental series 7.6.1: Evaluation of the TAG, TINA, INT and MINT Views algorithms under different datasets.

Algor. \ Dataset	GDI14	AtmoMon32	Intel54
TAG	57±2.52J	234±2J	523±22J
TINA	48±1.57J	183±6J	289±15J
INT	34±1.01J	170±7J	187±08J
MINT	19±0.56J	115±4J	139±06J

In the final dataset, Intel54 (Figure 7-18 (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 Intel54 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 significantly 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 7-6.

7.6.2 Pruning Magnitude

We next study the pruning magnitude of the k-Covered Bound-Set V_i' using the AtmoMon32 dataset. In Figure 7-19 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,

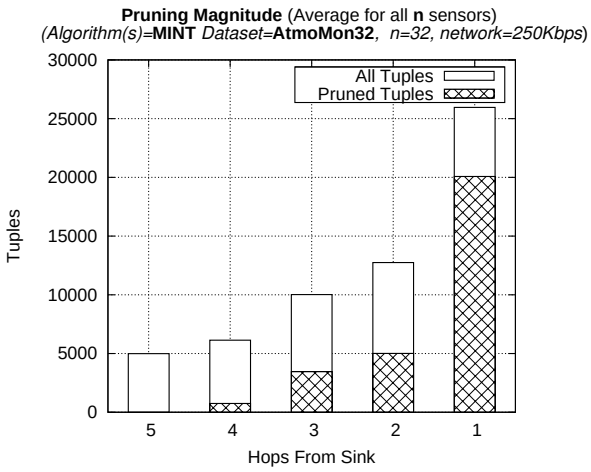

Figure 7-19: Pruning Magnitude of MINT Views on the AtmoMon32 dataset.

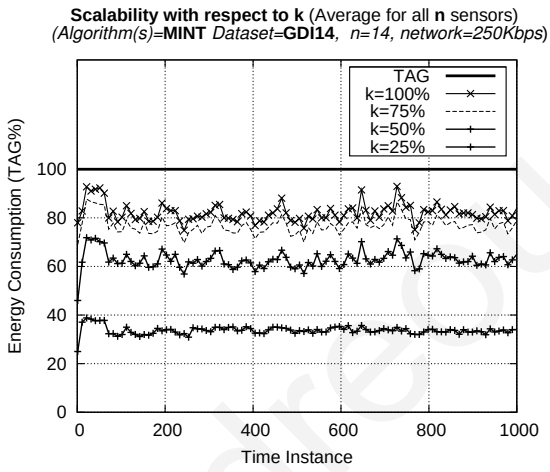

Figure 7-20: Scalability with respect to k: In the worst case scenario (k=100%), the MINT algorithm maintains a competitive advantage over TAG.

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.

7.6.3 Scalability with respect to k

In the third experiment, 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 7-20 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±0.5J). 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±1.6J) and 23,4% (44±1.5J) 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±1.59J) decreased energy consumption over TAG (57±2.52J).

7.6.4 Cardinality of the GROUP-BY Clause

In the fourth experiment, 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 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.

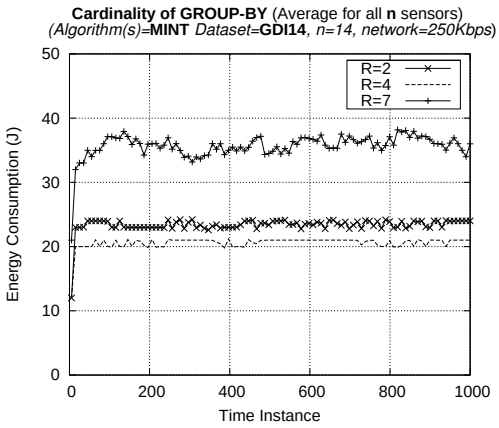

Figure 7-21: Cardinality of the GROUP BY clause for 3 different room configurations (R=Number of Rooms).

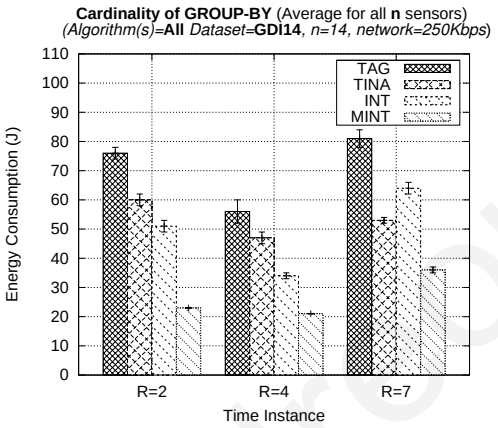

Figure 7-22: Cardinality of the GROUP BY clause for all algorithms.

Figure 7-21 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$\pm$0.7J). In the case of fewer rooms (i.e., R=2) we observe that the energy consumption is slightly increased although in this case the data payload of MINT becomes almost half the size. The reason for this increase, is the fact that the MINT pruning phase almost never prunes the V_i' structure on sensor nodes that have a hop count greater than 1 (i.e., the results have to reach nodes very close to the sink in order for a node to be able to eliminate tuples). On the other case, where R=7, we observe a significant increase in energy consumption. This is because the data payload is now configured to store 7 tuples at each sensor which requires almost double overall transmission energy. However, in this case the pruning mechanism of MINT eliminates tuples at lower levels of the network topology and that is why the standard deviation of this experiment increases (i.e., 36$\pm$1,68J).

Figure 7-22 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.

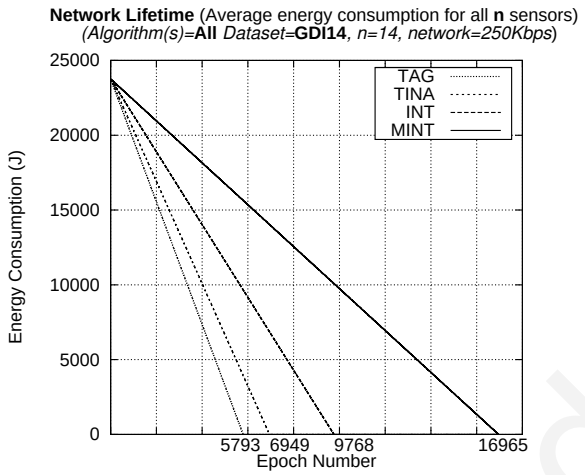

Figure 7-23: Network lifetime for the TAG, TINA, INT and MINT Views algorithms presented in this Section.

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.

7.6.5 Network Lifetime

The final performance criterion we have considered is network lifetime. We use the same definition for define network lifetime as the one presented in Section 7.3.4.

Figure 7-23 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 16,965 (i.e., 565 minutes), and this is translated into a $\approx$292% increase of the network lifetime.

7.7 Experimental Series 6: Evaluation of KSpot$^+$

In the final experimental series we evaluate the performance of the full KSpot$^+$ framework in comparison with the algorithms presented in this dissertation. In particular, we compare two modes of the KSpot$^+$ framework: i) KSpot$^+$ with only the Query Processing Module enabled, `KSpot+ (MINT)`, and ii) KSpot$^+$ with all modules enabled, `KSpot+ (ETC+WART+MINT)`. Notice that the latter, firstly utilizes the ETC algorithm to balance the query routing tree, then utilizes the WART algorithm to optimize the waking windows of the sensor nodes and finally executed a top-k query using the MINT algorithm. We have selected the MINT algorithm for both versions as it presents the higher energy savings in our framework.

More specifically, this experimental evaluation focuses on two parameters:

1. the **Energy Consumption Cost**, for the KSpot$^+$ (MINT) and KSpot$^+$ (ETC+WART+MINT) algorithms presented in Chapters 4, 5 and 6,

2. the **Network Lifetime**, of all algorithms presented in this dissertation

Table 7-7 summarizes the configuration parameters for all experiments mentioned in the subsequent sections.

Table 7-7: Configuration parameters for all experimental series of Section 7.7.

Section	Objective	Datasets	Algorithms
7.7.1	Energy Consumption	GDI14, AtmoMon32, Intel54	KSpot$^+$ (MINT) KSpot$^+$ (ETC+WART+MINT)
7.7.2	Network Lifetime	GDI14	All Algorithms

7.7.1 Energy Consumption

In the first experiment, we evaluate the energy consumption of KSpot$^+$ (MINT) and KSpot$^+$ (ETC+WART+MINT). We a top-k query Q, as mentioned in Section 7.1 on the GDI14, AtmoMon32 and Intel54 datasets and measure the energy consumption for each dataset separately.

In Figure 7-24 (top-left), we plot the results using the GDI14 dataset. As it was previously observed in Section 7.6.1 the KSpot$^+$ framework using only the MINT algorithm consumes on average 19$\pm$0.5J. On the other hand, when the KSpot$^+$ framework operates with all modules we observe a decrease $\approx$6% on average energy 18$\pm$1.19J. 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 AtmoMon32 and Intel54 datasets, with the complete KSpot$^+$ framework maintaining a competitive advantage over KSpot$^+$ (MINT). In particular, we observe that the complete KSpot$^+$ framework consumes 105$\pm$8J in the AtmoMon32 dataset and 118$\pm$10J in the Intel54 dataset, which translates in 9% and 15% decrease in energy consumption respectively.

In conclusion, the complete KSpot$^+$ framework demonstrates large energy gains when operating both with isolated components or full-fledged. It is important to note though, that the Query Processing Module demonstrates much larger gains (in the order of Joules) against the

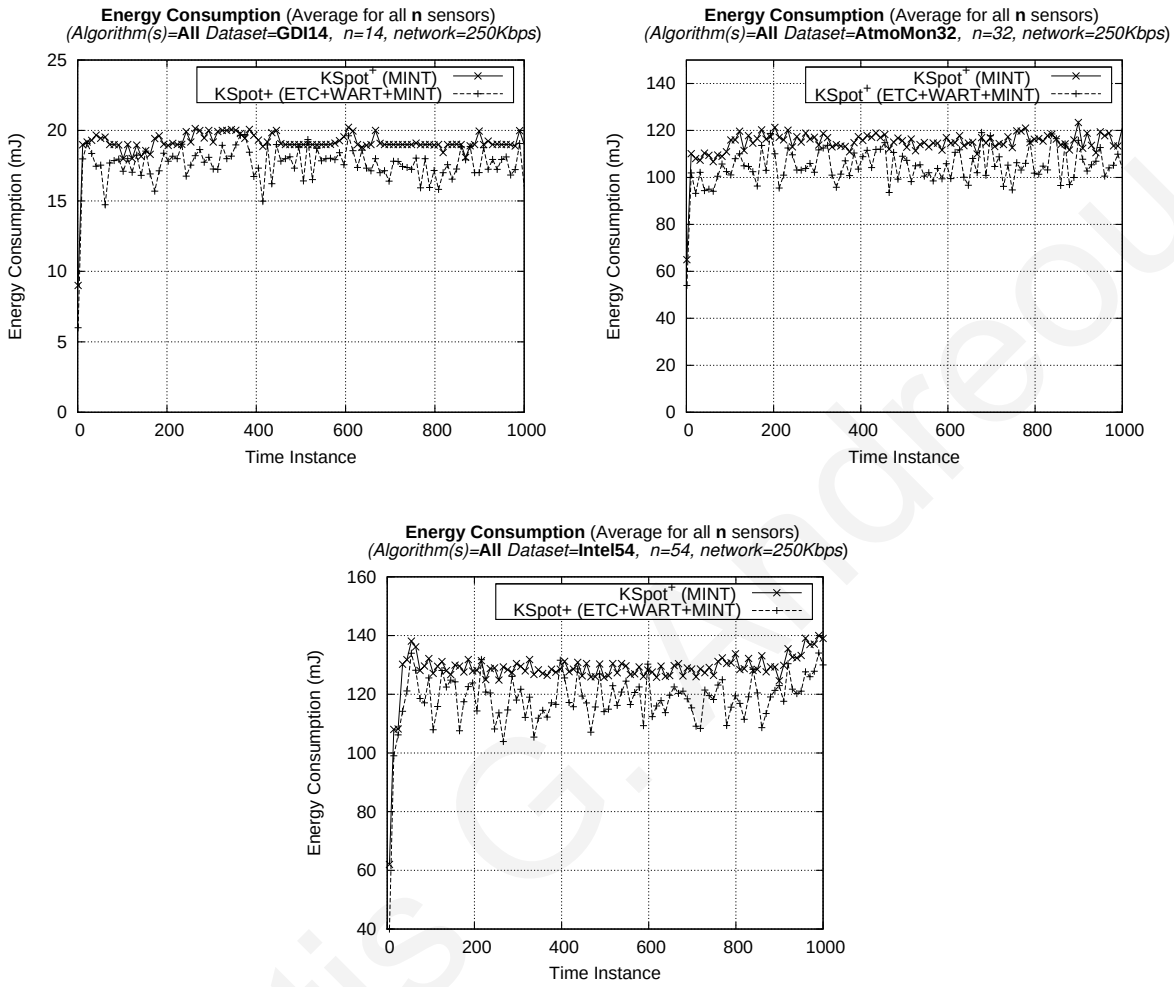

Figure 7-24: Energy Consumption for the KSpot$^+$ (MINT) and KSpot$^+$ (ETC+WART+MINT) using the TelosB sensor energy model.

other two modules (in the order of milliJoules) of the KSpot$^+$ framework. This shows that in-network pruning combined with exploiting temporal coherence can be of higher benefit in cases where application require monitoring of the k most important events in the sensor network.

The results for all experiments are summarized in Table 7-8. For ease of exposition, we have included the results for all algorithms as these were presented in Section 7.6.1.

Table 7-8: Average Energy Consumption for all sensors in experimental series 7.7.1: Evaluation of the TAG, TINA, INT and MINT Views algorithms under different datasets.

Algor. \ Dataset	GDI14	AtmoMon32	Intel54
TAG	57±2.52J	234±2J	523±22J
TINA	48±1.57J	183±6J	289±15J
INT	34±1.01J	170±7J	187±08J
KSpot$^+$ MINT	19±0.56J	115±4J	139±06J
KSpot$^+$ ETC+WART+MINT	18±1.19J	105±8J	118±10J

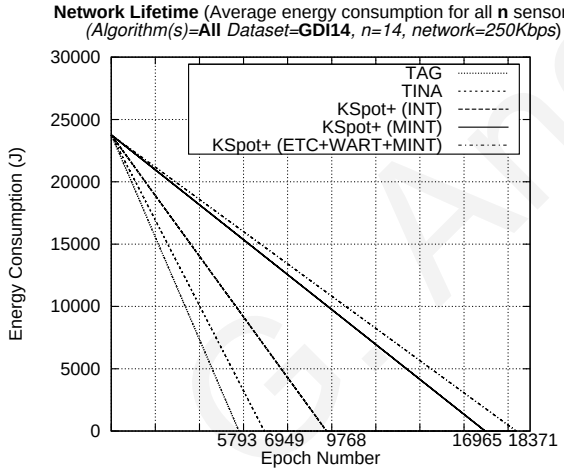

Figure 7-25: Network Lifetime for all algorithms.

7.7.2 Network Lifetime

The final performance criterion we have considered is network lifetime. We use the same definition for define network lifetime as the one presented in Section 7.3.4.

Figure 7-25 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 faster than all algorithms, leading to a lifetime of just 5,793 epochs (i.e., 193 minutes). TINA ranks fourth by offering 6,949 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 ≈317% increase of the network lifetime compared to TAG.

7.8 Experimental Evaluation Summary

The experimental evaluation of Sections 7.1 to 7.7 has demonstrated the effectiveness of incorporating network awareness during the execution of queries in Wireless Sensor Networks. In summary, the following conclusions are drawn:

- We have conducted experiments using real and realistic datasets of various sizes, ranging from small-scale (GDI14), to medium-scale (AtmoMon32, Intel54) and large-scale (GDI140, Intel540), which show that the performance of the KSpot$^+$ framework scales linearly with the addition of new sensor nodes to the network (Sections 7.3 to 7.7).

- In all the experimental series, we considered various query types including simple (e.g., selection, filter) and complex (e.g., top-k, group-by) that generated various workloads ranging from single-tuple to multi-tuple results. The KSpot$^+$ framework has demonstrated significant energy savings in all the aforementioned query types and workloads (Sections 7.3 to 7.7).

- The KSpot$^+$ framework has proven to be resilient in the case of node and communication failures (Sections 7.3 to 7.7).

- The combination of both network optimization and query optimization yields superior energy performance than focusing on a single objective. We have shown that when the modules of the KSpot$^+$ framework operate in combination, KSpot$^+$ achieves far greater energy savings and significantly increases the network longevity (Sections 7.5 and 7.7).

- Each module has been evaluated with a number of experiments in Sections 7.3, 7.4 and 7.6, and we have shown that each one demonstrates significant performance gains. Depending on the application requirements or user preferences, each module can be enabled or disabled accordingly providing their own performance contribution.

Chapter 8

Conclusions

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 the development and deployment of Wireless Sensor Networks for monitoring the environment at a high fidelity. Monitoring applications normally incorporates a query execution process that enables users to disseminate queries to the network and acquire the results. Query processing in WSNs is typically performed on the premise of a query routing tree, which provides each sensor with a path over which query answers are propagated to a centralized querying node. Our study revealed that predominant data acquisition frameworks suffer from serious data reception/transmission inefficiencies due to the unbalanced workload incurred on sensor nodes as well as the ineffective construction of this query routing tree. This leads to increased energy waste on each sensor as well as degrades the overall energy performance of the network.

This dissertation 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 techniques were developed with opportunities of applications that go beyond the current problem settings (e.g., smartphone networks [71], People-centric sensing [93]).

Through our experimental evaluation, we have shown that KSpot$^+$ can provide significant energy reductions and increase the longevity of the wireless sensor network.

We conclude by summarizing the benefits of our proposed KSpot$^+$ framework and mention some opportunities for future work.

8.1 KSpot$^+$: A Network-aware Framework for Energy-efficient WSNs

KSpot$^+$ presents a framework for optimizing the query execution and data acquisition processes by incorporating query semantics and the intrinsic properties of the wireless sensor network in order to improve energy-efficiency as follows:

- Structural inefficiencies are omni-present due to the ad hoc construction nature of the initial query routing tree. The KSpot$^+$ Tree Balancing Module minimizes these inefficiencies by reconstructing the query routing tree in a balanced manner, which reduces data collisions during communication.

- The unbalanced assignment of the query workload amongst sensor nodes incurred by the ad hoc construction of the query routing tree leads to data reception/transmission inefficiencies that severely degrade the energy performance of the network. The KSpot$^+$ Workload Balancing Module alleviates this problem by profiling recent data acquisition activity, identifying the bottlenecks of the network through an in-network execution of the Critical Path Method, and then dynamically adapts the waking windows of each sensor node.

- Efficiently monitoring the most important events is often more important than monitoring all the network. The KSpot$^+$ Query Processing Module specifically addresses this problem by incorporating top-k query execution in the current stack. This enables minimization of

both the size and number of packets that are transmitted through the network leading to massive energy savings.

- Organizing the sensor nodes into logical clusters, is essential in the cases where monitoring of specific areas is required. The current design incorporates logical grouping that enables sensor nodes to locally prune results by predicting their maximum values.

- Since Wireless Sensor Networks are typically prone to imminent node failures (triggered from temporary power-downs, malfunctions, environmental causes etc), maintaining is extremely vital for applications. In our experiments, we have shown that the KSpot$^+$ framework is resilient to such failures and therefore is well suited for the aforementioned scenarios.

This dissertation builds the above ideas into a practical data-centric framework for WSNs. Further, it integrates top-k execution within the current query processing stack, develops practical algorithms for balancing both the structure and workload of the wireless sensor network, provides prototype implementations of the proposed designs, and testbed evaluations that demonstrate significant energy reductions in comparison with traditional data acquisition frameworks. Additionally, its modular design ensures a high degree of openness and usability as well as combination with other protocols. Thus, we believe that this work makes a strong case for an alternative framework design tailored specifically for energy-efficient wireless sensor networks.

8.2 Lessons Learned

The most important conclusion drawn from this dissertation is that incorporating network and query semantics into the query dissemination and data acquisition process can significantly reduce the overall energy consumption of the network. However, the development of the prototype

implementation has provided us with valuable experience on various aspects concerning WSN deployments and additionally enabled us to deduct a number of conclusions that can aid future researches that will tackle similar topics. We present some of the major lessons learned below:

- **Simulation vs. Real Deployment:** During the early stages of our work, we developed a number of simulators for the evaluation of each of KSpot$^+$ modules independently. Although, the results we obtained from this approach were highly correlated with the results of the prototype implementation, the latter presented a more thorough assessment of all parameters that are involved during the operation of a WSN deployment (e.g., physical layer, storage layer, etc), which were not considered in the simulations.

- **Significant amount of time required for real experiments:** As presented in the experimental methodology in Section 7.1, the time required for experimenting with network sizes over 50 nodes was massive (i.e., several hours for the Intel54 dataset and even days for the GDI140 and Intel540 dataset).

- **Protocols are tightly coupled to the Operating System:** The protocols/algorithms presented in this dissertation were developed on top of the TinyOS operating system, which currently supports the largest amount of heterogeneous sensor nodes [61, 42, 25]. This involved learning the architecture of the operating system, a new programming language and in the case of deployment on different sensor nodes the specific component APIs for each sensor node (if they were different), which was very time consuming.

- **Temporal coherence of sensor results:** In order for a temporal coherence filter (e.g., the Query Processing Module's MINT temporal coherence filter) to work effectively, the sensor readings must follow some distribution (e.g., uniform distribution). This is typical for sensor readings like temperature and humidity where values do not change significantly during

subsequent time intervals. However, this is not the case for other sensor readings like sound-level, whose values may continuously fluctuate with high standard deviation. Therefore, application developers must carefully examine the required sensor readings and tune the parameters of KSpot$^+$ (e.g., gamma descriptors) accordingly.

8.3 Remaining Challenges and Future Work

The systems in this dissertation address the major challenges of energy-efficient data management in wireless sensor networks. Below, we enumerate a few challenging issues that we believe the KSpot$^+$ can be extended to support.

As it was discussed in Section 4.4, the Workload Balancing Module may suffer from frequent reconstructions of the critical path value and its energy-costly dissemination to all the sensor nodes of the network. This occurs when the application executes certain types of queries like event-based queries or queries with multi-tuple results of arbitrary size (e.g., filter queries), which cause the workload incurred on each sensor node to change rapidly between subsequent epochs. One way to alleviate this problem is to dynamically configure an additional parameter (like the a, b and c parameters that offset the costs of processing, inaccurate clock and collisions at the MAC layer) in order to increase the waking window of each sensor node. Finding a golden ratio w.r.t energy consumption between reconstructing the critical path value or extending the waking window of each sensor device is subject to further investigation.

In the Tree Balancing Module of the KSpot$^+$ framework we have assumed that the workload of a parent sensor node is directly proportional to the number of its child nodes. However, as it was discussed in Section 5.4, there are queries (e.g., filter queries, event-based queries) that may impose significantly different workloads on each sensor node. In the future, we plan to extend the definition of the optimal branching factor in order to take into account additional parameters

(e.g., the workload of each sensor node rather than the global number of sensor nodes (N) and the depth (d) of the query routing tree). Furthermore, in Chapter 2 we have studied research works that perform balancing based on query and not network semantics. A thorough study of how such techniques can be incorporated in the Tree Balancing Module and their benefits compared to the current implementation needs to be investigated. Finally, an important aspect that considerably affects the network performance is the actual deployment (i.e., placement) of the sensor nodes. Many research works have tackled this problem and particularly interesting are the ones that incorporate multiple objectives (e.g., energy efficiency, coverage, connectivity, fault tolerance, etc.) rather than the single objective of energy efficiency that is considered in this dissertation. Incorporating multi-objective optimization algorithms into KSpot$^+$ could significantly increase the QoS provided by the framework.

The Query Processing Module currently supports a number of different query types and aggregate functions as already described in Section 3.3. However, in the future, or depending on the application requirements, the need for new queries and aggregate functions may arise. Aggregate functions can be easily added into the current version of the implementation by following a procedure that includes: i) publishing the aggregate function to the catalog; ii) developing a nesC module that implements the aggregate functionality; and iii) updating the Parser implementation in order to translate the SQL text format in the appropriate query format. On the other hand, new types of queries and query operators will require major updates to the implementation, including the development of new Parser, Data Listener and Communication modules, both at the server as well as the client-side components of KSpot$^+$. Increasing the expandability of our framework is an important aspect that we will investigate in the future.

The algorithms presented within this dissertation are founded on the assumption that the routing topology is stationary (i.e., sensor nodes do not change locations) and this can only be affected

by node failures. However, recent advances in distributed robotics and low power embedded systems have enabled a new class of Wireless Sensor Networks that have the ability to move, named Mobile Sensor Networks (MSNs). MSNs can be used in new application domains like land, ocean and air exploration and monitoring, automobile applications and a wide range of other scenarios. The absence of a stationary network infrastructure in these settings, makes continuous data acquisition to some sink point a non-intuitive task as it results in a dynamic query routing tree that continuously adapts. How will data acquisition be established in such environments is still an open issue.

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 in-network storage, such that these events can later be retrieved by the user. We have recently began to study in [8] how we can extend the current framework with fault tolerance mechanisms that will ensure the continuous operation of data acquisition even in these harsh environments.

Another, important parameter that we have started to investigate in [13] is the energy cost associated with retrieving the locally stored sensor results in an efficient manner. While KSpot$^+$ assumes that the cost of retrieving any set of stored results from the flash-based storage is identical, we have shown in [13] that this is not the case. More specifically, we have developed an algorithm, which is an efficient external sorting algorithm tailored specifically for flash-based devices that we plan to incorporate into the KSpot$^+$ framework.

Finally, our work was focused solely on sensor networks. However, the ideas presented in this dissertation can be applied to other wireless devices such as smartphone devices. Communication between these devices so far has been achieved by Internet-enabled access points (e.g., WiFi, 3G). However, it has not yet been proven that this is the most energy-efficient manner to forward results

between devices. We plan to investigate how the energy-efficient algorithms presented in this work can be ported onto these devices in order to form energy-efficient, multi-hop communication protocols that can become the foundation of future energy-efficient applications (e.g., People-centric Sensing [23, 22]).

Bibliography

[1] K. Aberer, M. Hauswirth, A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks" In Proceedings of the 2007 International Conference on Mobile Data Management, Mannheim, Germany, May 7-11, pp.198-205, 2007.

[2] G.M. Adel'son-Vel'skii , E.M. Landis, "An Algorithm for the Organization of Information", In Dokl. Akad. Nauk SSSR 146 (1962), 263-266 (Russian). English translation in Soviet Math. Dokl. 3, pp.1259-1262, 1962.

[3] D. Agrawal, D. Ganesan, R.K. Sitaraman, Y. Diao, S. Singh, "Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices", In Proceedings of the Very Large Databases (VLDB) Endowment, Vol.2, No.1, pp.361-372, 2009.

[4] I.F. Akyildiz, M.C. Vuran, O.B. Akan, "A Cross-Layer Protocol for Wireless Sensor Networks", In Proceedings of the 40th Annual Conference on Information Sciences and Systems, Princeton, NJ, USA, March 22-24, pp.1102-1107, 2006.

[5] 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.

[6] G. Amati, A. Caruso, S. Chessa, "Application-driven, Energy-efficient Communication in Wireless Sensor Networks", In Computer Communications, Vol.32, No.5, pp.896-906, 2009.

[7] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, G. Samaras, "ETC: Energy-driven Tree Construction in Wireless Sensor Networks", In Proceedings of the 2nd International Workshop on Sensor Network Technologies for Information Explosion Era (SeNTIE'09), IEEE Press, Tapei, Taiwan, May 18-20, 2009.

[8] P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, "In-Network Data Acquisition and Replication in Mobile Sensor Networks", Distributed and Parallel Databases Journal, Vol.29, No.1-2, pp.87-112, 2011.

[9] 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.

[10] P. Andreou, D. Zeinalipour-Yazti, A. Pamboris, P.K. Chrysanthis, G. Samaras, "Optimized Query Routing Trees for Wireless Sensor Networks", Information Systems Journal, Vol.36, No.2, pp.267-291, April, 2011.

[11] P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, "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.

[12] P. Andreou, D. Zeinalipour-Yazti, M. Vassiliadou, P. K. Chrysanthis, G. Samaras, "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.

[13] P. Andreou, O. Spanos, D. Zeinalipour-Yazti, G. Samaras, P. K. Chrysanthis, "FSort: External Sorting on Flash-based Sensor Devices", The 6th International Workshop on Data Management for Sensor Networks (DMSN 2009), August 24, Lyon, France, 2009.

[14] W-T. Balke , W. Nejdl, W. Siberski, U. Thaden, "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.

[15] B. Babcock, C. Olston, "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.

[16] A. Banerjee, A. Mitra, W. Najjar, D. Zeinalipour-Yazti, V. Kalogeraki and D. Gunopulos, "RISE Co-S : High Performance Sensor Storage and Co-Processing Architecture", Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, Santa Clara, California, USA, 2005.

[17] R. Barr, J.C. Bicket, D.S. Dantas, B. Du, T.W.D. Kim, B. Zhou, E.G. Sirer, "On the Need for System-Level Support for Ad hoc and Sensor Networks", ACM Operating Systems Review, Vol.36, No.2, pp.1-5, April, 2002.

[18] Z. Benenson, M. Bestehorn, E. Buchmann, F.C. Freiling, M. Jawurek, "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.

[19] J. Blakeley, P.A. Larson, F.W. Tompa, "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.

[20] N. Bruno, L. Gravano, A. Marian, "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.

[21] T.D. Burd, R.W. Brodersen, "Design Issues for Dynamic Voltage Scaling", In Proceedings of the International Symposium on Low Power Electronics and Design, Rapallo, Italy, pp.9-14, 2000.

[22] A.T. Campbell, S.B. Eisenman, N.D. Lane, E. Miluzzo, R.A. Peterson, "People-centric urban sensing", In Proceedings of the Second annual international workshop on Wireless internet, Boston, Massachusetts, USA, Article No.18, 2006.

[23] 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 Vol.12, No.4, July-August, pp.12-21, 2008.

[24] P. Cao, Z. Wang, "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.

[25] 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.

[26] U. Cetintemel, A. Flinders, Y. Sun, "Power-efficient Data Dissemination in Wireless Sensor Networks", In Proceedings of the Third ACM International Workshop on Data engineering for Wireless and Mobile Access, San Diego, CA, USA, pp.1-8, 2003.

[27] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, "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.

[28] L.W.F. Chaves, E. Buchmann,F. Hueske, K. Bohm, "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.

[29] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, "Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in ad hoc Wireless Networks", In Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking, Rome, Italy, July 16-21, pp.85-96, 2001.

[30] L.S. Colby, T. Griffin, L. Libkin, I.S. Mumick, H. Trickey, "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.

[31] A. Coman, M.A. Nascimento, "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, pp.27, 2007.

[32] A. Coman, J. Sander, M.A. Nascimento, "Adaptive processing of historical spatial range queries in peer-to-peer sensor networks", In Distributed and Parallel Databases (DAPD'07), December, Vol.22, No.2(3), pp.133-163, 2007.

[33] J. Considine, F. Li, G. Kollios, J. Byers, "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.

[34] Common Object Request Broker Architecture (CORBA), http://www.corba.org/

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

[36] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, J. Zhao, "Towards a Sensor Network Architecture: Lowering the Waistline", In Proceedings of the 10th conference on Hot Topics in Operating Systems, Santa Fe, NM, Vol.10, June 12-15, pp.24, 2005.

[37] B. Das, V. Bharghavan, "Routing in ad-hoc Networks using Minimum Connected Dominating Sets", In IEEE International Conference on Communications, Montreal, Que., Canada, June 08-12, pp.376-380, 1997.

[38] G. Das, D. Gunopulos, N. Koudas, D. Tsirogiannis, "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.

[39] A. Deligiannakis, Y. Kotidis, N. Roussopoulos, "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.

[40] A. Deshpande, S.R. Madden, "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.

[41] Y. Diao, D. Ganesan, G. Mathur, P. Shenoy, "Rethinking Data Management for Storage-centric 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.

[42] A. Dunkels, B. Gronvall, T. Voigt, "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.

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

[44] C.T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, I. Stoica, "A Modular Network Layer for Sensornets", In Proceedings of the Seventh Symposium on Operating Systems Design and Implementation, Seattle, WA, USA, November 6-8, pp.249-262, 2006.

[45] 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.

[46] R. Fagin, A. Lotem, M. Naor, "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.

[47] C-L. Fok, G-C. Roman, C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks", ACM Transactions on Autonomous and Adaptive Systems, Vol.4, No.3, July, 2009.

[48] 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 2008 IEEE 24th International Conference on Data Engineering (ICDE'08), Cancun, Mexico, April 7-12, pp.1439-1441, 2008.

[49] I. Galpin, C.Y.A. Brenninkmeijer, F. Jabeen, A.A.A. Fernandes, N.W. Paton, "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.

[50] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, D. Culler, "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.

[51] O. Gnawali, K-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, E. Kohler, "The tenet architecture for tiered sensor networks", In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, USA, pp.153-166, 2006.

[52] S. Goel, T. Imielinski, "Prediction-based Monitoring in Sensor Networks: Taking Lessons from MPEG", In ACM SIGCOMM Computer Communication Review, Vol.31, No.4, pp.82-98, 2001.

[53] J. Gomez, A.T. Campbell, M. Naghshineh, C. Bisdikian, "Power-aware Routing in Wireless Packet Networks", In IEEE Mobile Multimedia Communications, San Diego, CA, USA, November 15-17, pp.380-383, 1999.

[54] J.L. Gross, J. Yellen, "Graph Theory & Its Application", Chapman & Hall/CRC Press, ISBN: 158488505X, 2005.

[55] L.J. Guibas, R. Sedgewick, "A Dichromatic Framework for Balanced Trees", In 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, October 16-18, pp.8-21, 1978.

[56] Y. Gu, A. Lo, I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Communications Surveys & Tutorials* Vol.11, No.1, pp.13-32, 2009.

[57] V. Gutnik, A.P. Chandrakasan, "Embedded Power Supply for Low-Power DSP", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.5, No.4, December, pp.425-435, 1997.

[58] P.E. Hart, N.J. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernetics, Vol.4, No.2, July, pp.100-107, 1968.

[59] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", In Proceedings of the 33rd Hawaii International Conference on System Sciences, Washington, DC, USA, Vol.8, pp.3005-3014,

2000.

[60] W.B. Heinzelman, A.L. Murphy, H.S. Carvalho, M.A. Perillo, "Middleware to support sensor network applications", IEEE Network, Vol.18, No.1, pp.6-14, Jan/Feb, 2004.

[61] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors", In Proceedings of the Nineth International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, pp.93-104, 2000.

[62] A. Hitha, K. Mohan, S. Behrooz, "MidFusion: An adaptive middleware for information fusion in sensor network applications", Information Fusion, Special Issue on Distributed Sensor Networks, Vol.9, No.3, pp.332-343, July, 2008.

[63] C. Intanagonwiwat, R. Govindan, D. Estrin, "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.

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

[65] Java Remote Method Invocation (RMI), http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[66] Java service oriented architecture (JINI), www.jini.org/

[67] P. Kalnis, W-S. Ng, B-C. Ooi, K-L. Tan, "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.

[68] D. Klan, K. Hose, K-U. Sattler, "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.

[69] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, I. Stoica, "Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks", In Proceedings of the Fifth ACM Conference on Embedded Networked Sensor Systems, Sydney, Australia, November 6-9, pp.351-365, 2007.

[70] 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 Sixth International Conference on Information Processing in Sensor Networks, Cambridge, MA, USA, April, ACM Press, pp.254-263, 2007.

[71] A. Konstantinidis, D. Zeinalipour-Yazti, P. Andreou, G. Samaras, "Multi-Objective Query Optimization in Smartphone Networks", In Proceedings of the 12th International Conference on Mobile Data Management (MDM'11), Lulea, Sweden, June 6-9, 2011.

[72] R. Kravets, P. Krishnan, "Application-driven Power Management for Mobile Communication", In Wireless Networks Journal, Vol.6, No.4, pp.263-277, 2000.

[73] J.F. Kurose, K.W. Ross, "Computer Networking - A Top Down Approach, 5th Edition", Addison Wesley, ISBN: 0-13-607967-9, March 31, 2009.

[74] P.-A. Larson, H.Z. Yang, "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.

[75] C.K. Lee, B. Zheng, W.-C. Lee and J. Winter, "Materialized In-Network View for Spatial Aggregation Queries in Wireless Sensor Network", ISPRS Journal of Photogrammetry and Remote Sensing, May, Vol.62, No.5, pp.382402, 2007.

[76] K.C.K. Lee, W.-C. Lee, B. Zheng, J. Winter, "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.

[77] P. Levis, "TinyOS Implementation Documentation", 2007.

[78] P. Levis, D. Culler, "Maté: a tiny virtual machine for sensor networks", SIGOPS Operating Systems Review, Vol.36, No.5, pp.85-95, October, 2002.

[79] P. Levis, N. Lee, M. Welsh, D. Culler, "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.

[80] Q. Li, J. Beaver, A. Amer, P.K. Chrysanthis, A. Labrinidis, "Multi-Criteria Routing in Wireless Sensor-Based Pervasive Environments", In Journal of Pervasive Computing and Communications (JPCC'05), Vol.1, No.4, pp.313-326, 2005.

[81] S. Li, S.H. Son, J.A. Stankovic, "Event detection services using data service middleware in distributed sensor networks", In Proceedings of the 2nd international conference on Information processing in sensor networks (IPSN'03), Palo Alto, CA, USA, pp.502-517, 2003.

[82] T. Liu, M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems", SIGPLAN Notices, Vol.38, No.10, pp.107-118, June, 2003.

[83] T. Liu, C. Sadler, P. Zhang, M. Martonosi, "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet", Proceedings of the 2nd international conference on Mobile systems, applications, and services, Boston, MA, USA, June 6-9, pp.256-269, 2004.

[84] 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.

[85] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, "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.

[86] S.R. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In Proceedings of the 5th symposium on Operating systems design and implementation (OSDI'02) , Vol.36, No. SI, pp.131-146, 2002.

[87] R. Maiocchi, B. Pernici, "Temporal Data Management Systems: A Comparative View", In IEEE Transactions on Knowledge and Data Engineering (TKDE'91), December, Vol.3, No.4, pp.504-524, 1991.

[88] B. Malhotra, M.A. Nascimento, I. Nikolaidis, "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), 5 Auckland, New Zealand, August 24, pp.1-7, 2008.

[89] A. Mani, M. Rajashekhar, P. Levis, "TINX: a Tiny Index Design for Flash Memory on Wireless Sensor Devices", In Proceedings of the 4th international conference on Embedded networked sensor system, Boulder, CO, USA, October 31-November 3, pp.425-426, 2006.

[90] A. Marian, L. Gravano, N. Bruno, "Evaluating Top-k Queries over Web-Accessible Databases", In ACM Transactions on Database Systems (TODS'04), June, Vol.29, No.2, pp.319-362, 2004.

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

[92] S. Michel, P. Triantafillou, G. Weikum, "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.

[93] R. Murty, A. Gosain, M. Tierney, A. Brody, A. Fahad, J. Bers, M. Welsh, "CitySense: A Vision for an Urban-Scale Wireless Networking Testbed", Harvard University Technical Report TR-13-07, September, 2007.

[94] S. Nath, J. Liu, F. Zhao, "SensorMap for Wide-Area Sensor Webs", In ACM Computer journal Vol.40, No.7, pp.90-93, 2007.

[95] 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.

[96] Oracle Enterprise JavaBeans Technology (EJB), http://www.oracle.com/technetwork/java/javaee/ejb/index.ht

[97] N.A. Pantazis, D.J. Vergados, D.D. Vergados, C. Douligeris, "Energy efficiency in wireless sensor networks using sleep mode TDMA scheduling", In Ad Hoc Networks, Vol.7, No.2, March, pp.322-343, 2009.

[98] 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.

[99] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growden, D. Standaert, M. Akay, J. Dy, M. Welsh, P. Bonato, "Monitoring Motor Fluctuations in Patients With Parkinson's Disease Using Wearable Sensors", IEEE Transactions on Information Technology in Biomedicine, November, Vol.13, No.6, 2009.

[100] L.L. Pipino, Y.W. Lee, R.Y. Wang, "Data quality assessment", In ACM Journal of Communications, Vol.45, No. 4, April, 2002.

[101] E. Pitoura and G. Samaras, "Data Management for Mobile Computing", Kluwer Academic Publishers, ISBN 0-7923-8053-3, 1998.

[102] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, I. Stoica, "A unifying link abstraction for wireless sensor networks", In Proceedings of the 3rd ACM conference on Embedded Networked Sensor Systems, San Diego, CA, USA, November 2-4, pp.76-89, 2005.

[103] J. Polastre, R. Szewczyk, D.E. Culler, "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.

[104] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, I. Stoica, "Geographic Routing without Location Information", In Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom'03), San Diego, CA, USA, September 14-19, pp.96-108, 2003.

[105] B. Rudolf, "Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms", In Acta Informatica Journal, Vol.1, No.4, December, pp.290-306, 1972.

[106] S. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach", second edition Prentice Hall, December 30, 2002.

[107] C. Sadler, P. Zhang, M. Martonosi, S. Lyon, "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.

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

[109] 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, california, USA, September 19, pp.69-76, 2003.

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

[111] D. Sharma, V.I. Zadorozhny, P.K. Chrysanthis, "Timely Data Delivery in Sensor Networks using Whirlpool", In Proceedings of the 2nd international workshop on Data management for sensor networks, Trondheim, Norway, pp.53-60, 2005.

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

[113] O. Shigiltchoff, P.K. Chrysanthis, E. Pitoura, "Adaptive Multiversion Data Broadcast Organizations", In the Information Systems Journal, Vol.29, No.6, September, pp.509-528, 2004.

[114] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh, "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.

[115] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, J. Yang, "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, pp.68, 2006.

[116] S. Singh, C.S. Raghavendra, "PAMAS-power Aware Multi-access Protocol with Signalling for Ad-hoc Networks", In ACM SIGCOMM Computer Communication Review, Vol.28, No.3, pp.5-26, 1998.

[117] A. Sinha, A. Chandrakasan, "Dynamic Power Management in Wireless Sensor Networks", In IEEE Design and Test of Computers, Vol.18, No.2, pp.62-74, 2001.

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

[119] C. Srisathapornphat, C.C. Shen, "Coordinated Power Conservation for Ad hoc Networks", In Proceedings of the IEEE International Conference on Communications (ICC 2002) , Vol.5, May, pp.3330-3335, 2002.

[120] M. Stern, E. Buchmann, K. Bohm, "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.

[121] 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.

[122] Texas Instruments, "CC2420, Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee(TM) Ready RF Transceiver", In *Texas Instrument Document* `http://www.ti.com/lit/gpn/cc2420`, 2007.

[123] 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.

[124] V. Tsaoussidis, H. Badr, "TCP-probing: towards an Error Control Schema with Energy and Throughput Performance Gains", In Proceedings of the International Conference on Network Protocols, Osaka, Japan, November, pp.12, 2000.

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

[126] S. Weissman-Lauzac, P.K. Chrysanthis, "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.

[127] S. Weissman-Lauzac, P.K. Chrysanthis, "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.

[128] A. Woo, D.E. Culler, "A Transmission Control Scheme for Media Access in Sensor Networks", In Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking, Rome, Italy, July 16-21, pp.221-235, 2001.

[129] M. Wu, J. Xu, X. Tang, W-C. Lee, "Top-k Monitoring in Wireless Sensor Networks", In IEEE Transactions on Knowledge and Data Engineering (TKDE'07), July, Vol.19, No.7, pp.1041-4347, 2007.

[130] P. Xia, P.K. Chrysanthis, A. Labrinidis, "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, pp.8, 2006.

[131] Y. Xu, J. Heidemann, D. Estrin, "Adaptive Energy-conserving Routing for Multihop Ad-hoc Networks", Technical Report TR-2000-527, USC/Information Sciences Institute, October, 2000.

[132] Y. Xu, J. Heidemann, D. Estrin, "Geography-informed Energy Conservation for Ad Hoc routing", In Proceedings of the Sevent Annual International Conference on Mobile Computing and Networking, Rome, Italy, pp.70-84, 2001.

[133] 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-October 16, Vol. 1, ISBN: 0-7695-1950-4, 2003.

[134] J. Yang, J. Widom, "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.

[135] 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.

[136] W. Ye, J. Heidemann, D. Estrin, "Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks", In IEEE/ACM Transactions on Networking (TON), Vol.12, No.3, pp.493-506, 2004.

[137] H. Yu, H. Li, P. Wu, D. Agrawal, A.E. Abbadi, "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.

[138] X. Yu, K. Niyogi, S. Mehrotra, N. Venkatasubramanian, "Adaptive Middleware for Distributed Sensor Environments", IEEE Distributed Systems Online, Vol.4, No.5, May, 2003.

[139] V. Zadorozhny, P.K. Chrysanthis, A. Labrinidis, "Algebraic Optimization of Data Delivery Patterns in Mobile Sensor Networks", In Proceedings of the Database and Expert Systems Applications, 15th International Workshop, Zaragoza, Spain, August 30-September

3, pp.668-672, 2004.

[140] D. Zeinalipour-Yazti, P. Andreou, P.K. Chrysanthis, G. Samaras, "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.

[141] D. Zeinalipour-Yazti, P. Andreou, P.K. Chrysanthis, G. Samaras, A. Pitsillides, "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.

[142] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, W. Najjar, "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.

[143] D. Zeinalipour-Yazti, S. Lin, D. Gunopulos, "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.

[144] D. Zeinalipour-Yazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, D. Srivastava, "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.

[145] ZigBee Alliance, "ZigBee specification", In *ZigBee Document 053474r06, Version 1.0*, 2004.

Appendix A

Our experimental evaluation utilizes the following real and realistic datasets in our trace-driven experiments in order simulate small-scale, medium-scale and large-scale wireless sensor networks.

Great Duck Island (GDI14) Dataset

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 [121], 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.

The GDI dataset has the following properties:

Table 8-1: GDI14 Dataset Information

Parameter	**Value**
Dataset Type	Real
Number of Sensors (N)	14
Number of Attributes (m)	14
Epoch duration (e)	30 seconds
Average Depth (d) of $\mathcal{T}$	3

The GDI14 dataset includes the following attributes/readings for each sensor node:

Table 8-2: GDI14 Dataset Attributes

Attribute	Description	Data Type
packet_time	transmitted packet date and time	Date/Time
nodeid	the node identifier	Integer
light_raw	light reading	Integer
temp_raw	temperature reading	Integer
thermopile_raw	thermopile reading	Integer
thermistor_raw	thermistor reading	Integer
humidity_raw	humidity reading	Integer
intersema_pressure_reading	intersema pressure reading	Integer
intersema_pressure_raw	intersema pressure raw reading	Integer
intersema_temp_reading	intersema temperature reading	Integer
intersema_temp_raw	intersema temperature raw reading	Integer
voltage_reading	voltage reading	Integer
seqno	sequence/epoch number	Integer
crc	cyclic redundancy check	Integer

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 [43]. 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).

The AtmoMon32 dataset has the following properties:

Table 8-3: AtmoMon32 Dataset Information

Parameter	Value
Dataset Type	Real
Number of Sensors (N)	32
Number of Attributes (m)	9
Epoch duration (e)	1 hour
Average Depth (d) of $\mathcal{T}$	5

The AtmoMon32 dataset includes the following attributes/readings for each sensor node:

Table 8-4: AtmoMon32 Dataset Attributes

Attribute	Description	Data Type
packet_time	transmitted packet date and time	Date/Time
nodeid	the node identifier	Integer
light_raw	light reading	Integer
temperature	temperature reading	Integer
air_speed	air speed reading	Integer
air_temp	air temperature reading	Integer
humidity_raw	humidity reading	Integer
pressure_raw	barometric pressure reading	Integer
cumulative_rain	cumulative rain reading	Integer

Intel Research Labs (Intel54) Dataset

This is a real dataset that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley [64] 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 54 sensors that had the largest amount of local readings since some of them had many missing values. More specifically, we utilize the real coordinates of the 54 sensors. The depth of the initial query routing tree constructed with the FHF approach is 14.

The Intel54 dataset has the following properties:

Table 8-5: Intel54 Dataset Information

Parameter	Value
Dataset Type	Real
Number of Sensors (N)	54
Number of Attributes (m)	8
Epoch duration (e)	31 seconds
Average Depth (d) of $\mathcal{T}$	14

The Intel54 dataset includes the following attributes/readings for each sensor node:

Table 8-6: Intel54 Dataset Attributes

Attribute	Description	Data Type
date	date of transmitted packet	Date/Time
time	time of transmitted packet	Text
epoch	epoch number	Integer
moteid	mote/node identifier	Integer
temperature	temperature reading	Integer
humidity	humidity reading	Integer
light	light reading	Integer
voltage	voltage reading	Integer

Great Duck Island (GDI140)

In order to evaluate our approach on a medium-scale sensor network we synthetically derive a sensor network composed of 140 nodes that follows the same distribution with the GDI14 dataset. The average depth of the initial query routing tree constructed with the FHF approach is 24.

The GDI140 dataset has the following properties:

Table 8-7: GDI140 Dataset Information

Parameter	Value
Dataset Type	Realistic
Number of Sensors (N)	140
Number of Attributes (m)	14
Epoch duration (e)	30 seconds
Average Depth (d) of $\mathcal{T}$	24

Intel Research Berkeley (Intel540)

In order to evaluate our approach on a large-scale sensor network we synthetically derive a 540-node network based on the Intel54 dataset. The distribution of the dataset follows again the

same distribution with the Intel54 dataset. The average depth of the initial query routing tree constructed with the FHF approach is 22.

The Intel540 dataset has the following properties:

Table 8-8: Intel540 Dataset Information

Parameter	**Value**
Dataset Type	Realistic
Number of Sensors (N)	540
Number of Attributes (m)	8
Epoch duration (e)	31 seconds
Average Depth (d) of $\mathcal{T}$	22