Optimized Query Routing Trees for Wireless Sensor Networks

Panayiotis Andreou [a], Demetrios Zeinalipour-Yazti [a,*], Andreas Pamboris [b], Panos K. Chrysanthis [c], George Samaras [a]

[a] *Department of Computer Science, University of Cyprus, CY-1678, Nicosia, Cyprus*

[b] *Department of Computer Sc. and Engr., University of California - San Diego, San Diego, CA 92093, USA*

[c] *Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA*

Abstract

In order to process continuous queries over *Wireless Sensor Networks* (WSNs), sensors are typically organized in a *Query Routing Tree* (denoted as T) that provides each sensor with a path over which query results can be transmitted to the querying node. We found that current methods deployed in predominant data acquisition systems construct T in a sub-optimal manner which leads to significant waste of energy. In particular, since T is constructed in an ad-hoc manner there is no guarantee that a given query workload will be distributed equally among all sensors. That leads to data collisions which represent a major source of energy waste. Additionally, current methods only provide a topological-based method, rather than a query-based method, to define the interval during which a sensing device should enable its transceiver in order to collect the query results from its children. We found that this imposes an order of magnitude increase in energy consumption.

In this paper we present *MicroPulse*$^+$, a novel framework for minimizing the consumption of energy during data acquisition in WSNs. *MicroPulse*$^+$ continuously optimizes the operation of T by eliminating data transmission and data reception inefficiencies using a collection of in-network algorithms. In particular, MicroPulse$^+$ introduces: i) the *Workload-Aware Routing Tree (WART)* algorithm, which is established on profiling recent data acquisition activity and on identifying the bottlenecks using an in-network execution of the critical path method; and ii) the *Energy-driven Tree Construction (ETC)* algorithm, which balances the workload among nodes and minimizes data collisions. We show through micro-benchmarks on the CC2420 radio chip and trace-driven experimentation with real datasets from Intel Research and UC-Berkeley that *MicroPulse*$^+$ provides significant energy reductions under a variety of conditions thus prolonging the longevity of a wireless sensor network.

Key words: Query Routing Trees, Sensor Networks, Critical Path Method.

1 Introduction

Recent advances in embedded computing have made it feasible to produce small scale wireless sensor devices that can be utilized for the development of environmental monitoring systems under diverse conditions. Sensor devices are tiny computers, often as small as a coin or a credit card, that feature a low frequency processor that reduces power consumption, an on-chip flash memory for local storage, a wireless radio for communication, on-chip sensors, and an energy source such as AA batteries or solar panels [33]. Large-scale deployments of *Wireless Sensor Networks (WSNs)* have already emerged in environmental and habitant monitoring [51,33], structural monitoring [27] and urban monitoring [38]. Due to the limited energy source, WSN applications have to be founded on the premise of energy-conscious algorithms.

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 [62,35,34,64,59]. The energy consumption for transmitting 1 bit of data using the MICA mote [11] is approximately equivalent to processing 1000 CPU instructions [34]. In order to cope with this energy challenge sensing devices are forced to *power-down*[1] their radio transceiver (transmitter-receiver) between consecutive data acquisition rounds (i.e., *epochs*).

It has been shown that sensors operating at a 2% duty cycle can achieve lifetimes of 6-months using two AA batteries [35]. 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 [42,11] to the operating system layer [23], the programming language [16], the network layer [66] and the data management layer (e.g., storage [64,36], compression [14,46], query processing [34,44,62,24,59,35,43,32,65] and prediction [18]). 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 is established

* *Contact author: dzeina@cs.ucy.ac.cy, tel: +357-22-892755, fax: +357-22-892701*
Email addresses: panic@cs.ucy.ac.cy (Panayiotis Andreou), dzeina@cs.ucy.ac.cy (Demetrios Zeinalipour-Yazti), apambori@cs.ucsd.edu (Andreas Pamboris), panos@cs.pitt.edu (Panos K. Chrysanthis), cssamara@cs.ucy.ac.cy (George Samaras).

[1] The notion of powering-down the transceiver can be interpreted in this work either in a literal manner (i.e., the transceiver is completely powered-down) or in a metaphoric manner (i.e., the transceiver is configured in low-power listening mode [37] instead).

on the premise of *Query Routing Trees (denoted as T)*, which provide each sensor with a path over which query answers can be transmitted to a centralized querying node (i.e., *sink*). Our study reveals that predominant data acquisition frameworks [59,34,35] have overlooked the important parameter of constructing efficient query routing trees and that negatively impacts the energy efficiency of these systems. In particular, since T is constructed in an ad-hoc manner there are two major sources of inefficiencies:

- **Data Reception Inefficiencies:** 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* (as shown in Section 7.1 with a series of micro-benchmarks). Consequently, τ is an overestimate that leads to significant energy waste. For instance, a typical query with an epoch of 31 seconds over a three-tier network in TinyDB [35,34], will enforce each sensor to activate its transceiver for as much as 10 seconds while the required τ interval might only be a few milliseconds (as shown in Section 4.1).

- **Data Transmission Inefficiencies:** 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. For instance in Section 7.1, we show that the execution of a query over a node with 10 children will lead to a 48% loss rate of data packets, while executing the same query over a node with 100 children will lead to a 77% loss rate. These figures translate into an approximately threefold increase in energy demand due to inevitable re-transmissions of data packets. Consequently, unbalanced trees can severely degrade the network health and efficiency.

Contributions: In this paper we present *MicroPulse*$^+$, a novel framework for minimizing both the aforementioned sources of inefficiencies. The main intuition behind our framework is to continuously optimize the operation and structure of T by utilizing a collection of in-network processing algorithms. In particular, MicroPulse$^+$ introduces:

- The **Workload-Aware Routing Tree (WART)** algorithm, which minimizes data reception inefficiencies by profiling recent data acquisition activity and by identifying the bottlenecks using an in-network execution of the critical path method. In particular, it generates a time-synchronized topology in which WSDs know exactly when and for how long they should enable their transceiver; and
- The **Energy-driven Tree Construction (ETC)** algorithm, which min-

imizes data transmission inefficiencies by balancing the workload among participating nodes. In particular, it generates a near-balanced tree topology in which data collisions are minimized, thus WSDs have the capability to power down their transmitter much earlier.

Energy-efficient query routing trees are useful in a plethora of stationary sensor network systems. Below we show their applicability in the context of a Bio-Harvesting Sensor Network [55]. Additionally, we explain how such structures can be adapted in order to become the foundation of future applications in People-centric Sensing [7,6] scenarios.

Example 1 - Voltree Climate Sensor Network: Recently, Voltree Power [55] has engineered a bio-energy harvesting technology that allows sensor devices to recharge themselves by collecting the energy that is naturally produced by living trees or other large plants. This alternative, minimizes the cost of replacing batteries frequently, especially in large-scale deployments. 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 [55]. 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 constructing optimized trees is consequently of major importance in this work.

Example 2 - People-Centric Sensing: People-centric sensing [7,6], 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, illustrates a futuristic people-centric sensing scenario where cyclists journey through the main streets of a city. Each cyclist is equipped with

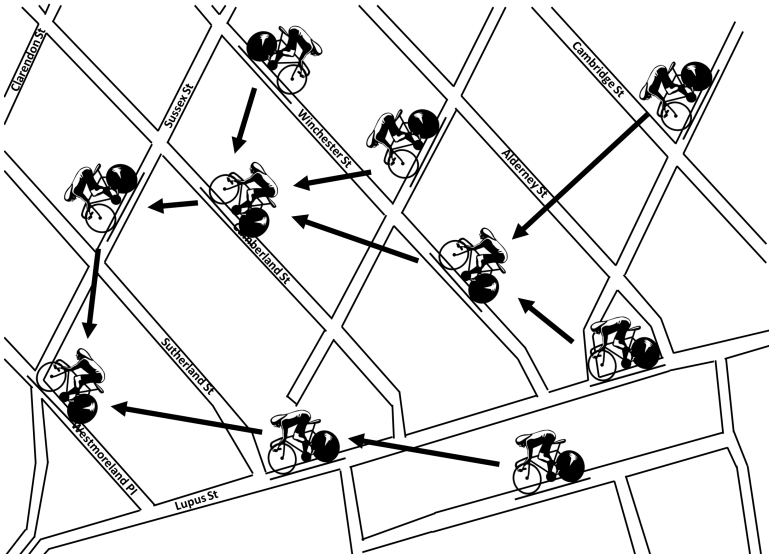

Fig. 1. **People-centric Sensing Example:** Cyclists collect data through their sensor equipped mobile devices (e.g., CO_2 level) during their ride. A given cyclist can query its neighborhood by constructing an ad-hoc query spanning tree.

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 query routing trees presented in this work. 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 paper.

This paper builds on our previous work in [63,4], in which we presented the preliminary design of the MicroPulse framework that minimizes the energy consumption by tuning the waking windows locally at each sensor. In this paper, we introduce several new improvements and extensions that are summarized as follows:

- We present a new tree-balancing construction algorithm, coined ETC, which investigates the effect of balanced routing trees in wireless sensor networks. This algorithm constructs near-balanced tree topologies minimizing in that

way the inherent collisions among neighboring nodes in unbalanced tree topologies.

- We combine the ETC algorithm with our earlier work in MicroPulse and create a uniform framework coined MicroPulse$^+$ that takes a holistic view on optimizing query routing trees in Sensor Networks. In particular, MicroPulse$^+$ optimizes both data reception deficiencies and data transmission deficiencies.
- We introduce an elaborate experimental study and solid experimental evidence for the motivation and efficiency of our propositions using a variety of real traces, querysets and a series of real micro-benchmarks on the CC2420 radio transceiver [52]. Finally, we perform an extensive experimental assessment of ETC both in isolation and in integration with the MicroPulse$^+$ framework.
- For completeness, we provide an overview of related work that has been proposed at different layers of the communication stack (i.e., physical layer, network layer, transport layer, etc.) We also qualitatively explain the differences and similarities of these techniques compared to the MicroPulse$^+$ framework.

Overall, this paper makes the following contributions to the state-of-the-art:

- We formulate the problem of adapting the waking window τ of a sensing device in order to conserve energy that can be used to prolong the longevity of the network and hence the quality of results. We solve the waking window problem by proposing the WART algorithms.
- We formulate the problem of constructing and maintaining an energy efficient query routing tree in a wireless sensor network. We solve this problem by introducing the ETC algorithm.
- We experimentally validate the efficiency of our propositions with an extensive experimental study that utilizes real sensor readings and a series of microbenchmarks. Our results are useful to any type of multi-hop network that relies on Query Routing Trees for data acquisition.

Roadmap: Section 2 formalizes our system model. Section 3 presents an overview of MicroPulse$^+$ as well as a description of its main components. Sections 4 and 5 thoroughly describe the WART and ETC algorithms that comprise the MicroPulse$^+$ framework. Section 6 presents our experimental methodology whereas Section 7 the results of our evaluation. Section 8 overviews the related work of the paper while Section 9 concludes this paper.

2 System Model

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

A user specifies a continuous query Q to be evaluated once during the interval of an *epoch* (denoted as e), which is the time interval after which each s_i $(i \leq n)$ will re-compute Q. For simplicity let us adopt a declarative SQL-like syntax (similar to [35,59]) to express the ideas presented in this paper in brevity. For instance, the following query declares that each sensing device should recursively collect the node identifier and the temperature from its children every 31 seconds and communicate the results to the sink.

```
SELECT nodeid, temp
FROM sensors
EPOCH DURATION 31 seconds
```

Note that our model also supports continuous aggregate queries. For instance, the following query declares that each sensing device should aggregate the average light measurement for each room from its children every 31 seconds and communicate the results to the sink.

```
SELECT roomid, AVG(light)
FROM sensors
GROUP BY roomid
EPOCH DURATION 31 seconds
```

Essentially, our framework supports any type of query as long as the query produces a continuous result which is percolated to the sink. Based on these continuous transmissions, we profile the workload of each sensor and compute the critical path.

Table 1
Definition of Symbols Utilized in the Paper

Symbol	Definition
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 Q
T=(S, E)	Query Routing Tree (S=vertices, E=edges)
d	Depth of the routing tree 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 Q
$children(s_i)$	Children List of sensor s_i
$APL(s_i)$	Alternate Parent List of sensor s_i
β	Balanced Branching Factor of network S

A user submits Q at some centralized querying node (denoted as s_0, or sink node) prior deployment and the system then initiates the execution of Q by disseminating it to the n sensors. In particular, the sink sends Q to one sensor s_1. Subsequently, s_1 recursively forwards Q to all of its neighbors until all n sensors have received the given query. Without loss of generality, we adopt the *First Heard From (*FHF*)* mechanism which is utilized in a variety of data acquisition frameworks such as [35,59,65,62] and where each sensor s_i selects as its parent the first node from which Q was received. This creates an acyclic subset of the communication graph G (i.e., a spanning tree) which is denoted as $T = (S, E')$, where $E' \subset E$. Each s_i also maintains a *Child Node List* (denoted as $children(s_i)$), which is trivially constructed during the creation of T (i.e., using an acknowledgment from each child to its parent). In more recent frameworks, like GANC [43] and Multi-Criteria Routing [32], T can be constructed based on query semantics, power consumption, remaining energy and others. In more unstable topologies a node can maintain several parents [10] in order to achieve fault tolerance but this might impose some limitations on the type of supported queries. 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. Such a list will be utilized by the ETC algorithm we propose in this paper but could also be utilized to select alternate parents in cases of failures.

3 The MicroPulse$^+$ Framework

In this section we provide an overview of the MicroPulse$^+$ framework. In particular, we will introduce the *Workload-Aware query Routing Tree (WART)* algorithm and the *Energy-driven Tree Construction (ETC)* algorithm. We shall start with a detailed analysis that reveals the motivation behind our propositions and then provide an outline for each individual algorithm.

We have already defined that the continuous interval during which a sensing device s_i $(i \leq n)$ enables its transceiver, collects and aggregates the results from its children, and then forwards them all together to its own parent is defined as the *waking window* τ. It is important to mention that the exact value of τ is query-specific and can not be determined accurately using current techniques. For instance, s_i $(i \leq n)$ 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.

Therefore, we aim 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-off:

- **Early-off Transceiver:** Shall s_i $(i \leq n)$ 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 Q; and
- **Late-off Transceiver:** Shall s_i $(i \leq n)$ 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.

The WART algorithm presented in this paper utilizes a novel algorithm for the dynamic adaptation of the τ values and is established on profiling recent data acquisition activity and on identifying the bottlenecks using an in-network execution of the Critical Path Method.

The *Critical Path Method (CPM)* [20] 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 2 illustrates an example which will be utilized throughout the paper. 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 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 catch 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 2 have different weights. This is very typical for a sensor network as the link quality can vary across the network [51]. Another reason is that some sensors might have a different workload than other sensors. Note that our scheduling scheme is distributed which makes it

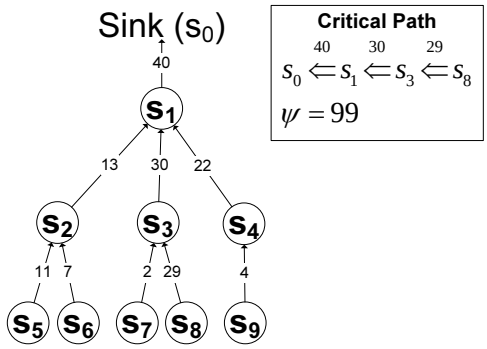

Fig. 2. Nine sensing devices (shown as vertices) and the respective workload between them (shown as edges) in order to answer some continuous query Q at the sink (s_0). The WART algorithm utilizes this information in order to locally adapt the waking window of each device using the *Critical Path Method*.

fundamentally different from centralized scheduling approaches like DTA [61] and TD-DES [8] that generate collision-free query plans at a centralized node. Additionally, our approach is also different from techniques such as [45] which segment the sensor network into sectors in order to minimize collisions during data acquisition.

Although the proposed approach 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. To facilitate our description, consider the example depicted on Figure 3 (left), which illustrates the initial ad-hoc query routing tree 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 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 3 (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.
- **Increased Data Transmission Collisions:** An unbalanced workload in-

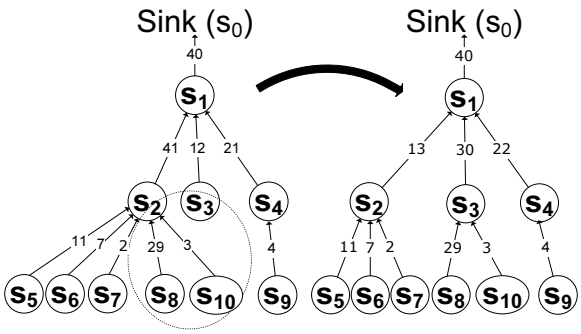

Fig. 3. **Left:** The initial ad-hoc query routing tree constructed using the *First Heard From method.* **Right:** The optimized workload-aware query routing tree constructed using the in-network ETC balancing algorithm.

creases data transmission collisions which represent a major source of energy waste in wireless communication. Our micro-benchmarks on the CC2420 radio transceiver, presented in Section 7.1, 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.

3.2 Outline of Operation

We shall now outline the operation of our framework in which the execution of the ETC algorithm is succeeded by the execution of the WART algorithm. In particular, the ETC algorithm is executed as a two step process that is summarized as follows:

(1) **Discovery Phase:** In this phase, the sink queries the network for the total number of sensors n and the maximum depth of the routing tree d. When variables n and d are received, the sink calculates a uniform *Optimal Branching Factor (β)* for each sensor.
(2) **Balancing Phase:** In this phase, the sink disseminates the β value back to the n nodes. Upon receiving β, each sensor conducts a number of local rearrangements to its local topology in order to create a near-balanced topology (a formal definition is provided in Section 5.1).

As soon as the ETC module completes the reconstruction of the query routing tree, our framework executes the WART algorithm which disseminates the continuous query Q to the network. WART then 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 3-phase process that is summarized below:

(1) **Construction Phase:** In this phase, the sink constructs a query routing tree and then queries the network for the total workload 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.

The next two sections will provide a more thorough description of the individual steps of our algorithms. Although the WART algorithm succeeds the operation of the ETC algorithm we shall present them in opposite order as the WART algorithm determines the most significant energy savings in the MicroPulse$^+$ framework.

4 The Workload-Aware Routing Tree (WART) Algorithm

In this section we describe the first algorithm of the MicroPulse$^+$ framework, coined WART. 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 transceiver). We start out with background work on waking window mechanisms in popular data acquisition systems such as TAG [34,35] and Cougar [59] and then describe the steps of our WART algorithm. For the subsequent sections let us assume that some arbitrary query Q has already been disseminated to the n sensors of the wireless sensor network.

4.1 Preliminaries and Background

In this subsection we will describe the waking window mechanism of the TAG and Cougar frameworks.

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 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.

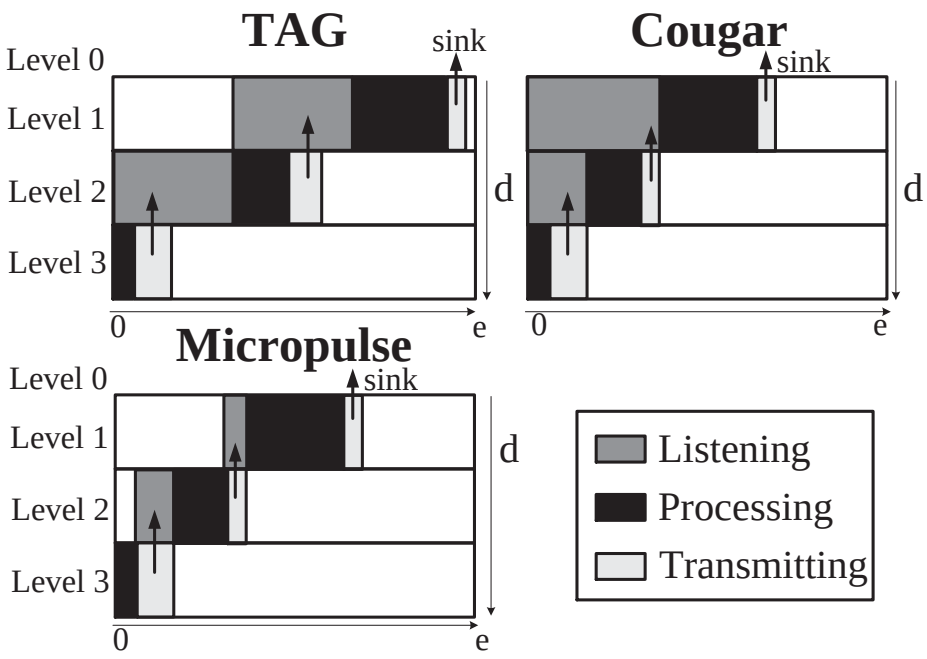

Fig. 4. The Waking (Listening) Window (τ) in TAG, Cougar and MicroPulse$^+$'s WART algorithm.

For instance, if the epoch is 31 seconds and we have a three-tier network (i.e., d=3) like Figure 4 (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 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 [34] but in reality it is too coarse. In the experimental Section 7, we found that this over-estimation is three orders of magnitudes 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 (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 [3]. 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 MicroPulse$^+$ and this approach are a) 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 b) 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.

In the following three subsections (4.2-4.4), we will detail the operation of the WART algorithms we propose in this paper. In particular, we will describe the construction phase, the dissemination phase and the adaptation phase.

4.2 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 [59]. 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 paper. 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 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 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 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., $s_{i,maxchild}^{in}$). 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., $\psi - s_j^{out}$). 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

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 $(i \leq n)$.

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 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.

4.4 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 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 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 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

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: $workload'_i = \psi_i - w_i$; ▷ 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** $(x < \delta)$ **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** $(x \leq \lambda_i)$ **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**

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.

5 Energy-driven Tree Construction (ETC) Algorithm

Even though the proposed WART algorithm can efficiently solve the waking window problem it does not optimize the query routing tree and that leads to increased collisions during data transmission. In this section we describe the second algorithm of the MicroPulse$^+$ framework coined ETC. Assuming an arbitrary query routing tree T_{input} constructed using the FHF approach, the objective of ETC is to transform T_{input} into a near-balanced tree T_{ETC} in a distributed manner. 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 but we choose to present them in opposite order as the WART algorithm determines the most significant energy savings in our framework.

5.1 Preliminaries and Background

In this section we will provide an overview of balanced trees in order to better frame the problem the ETC algorithm seeks to solve. As we have already mentioned in the motivation of this work, balanced trees have the following desirable properties: i) they decrease collisions during data transmission, and 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 1 - Balanced Tree ($T_{balanced}$): *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: $T_1 = (V, E) = (\{A, B, C, D\}, \{(B, A), (C, A), (D, B)\})$, where the pairs in the E set represent the edges of the binary tree. By visualizing T_1, we observe that the subtrees of A differ by at most one (i.e., $|height(B) - height(C)|$=0) and that the subtrees of B differ again by at most one (i.e., $|height(D) - height(NULL)|$=1). Thus, we can characterize T_1 as a balanced tree.

Notice that V has several balanced tree representations of the same height (e.g., the directed tree $T_2 = (\{A, B, C, D\}, \{(B, A), (C, A), (D, C)\})$). Similarly, V has also many balanced tree representations of different heights (e.g., the directed tree 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 T_{input} as this makes the construction process more efficient.

In order to derive a balanced tree ($T_{balanced}$) in a centralized manner we could utilize the respective balancing algorithms of AVL Trees, B-Trees and Red-Black Trees. 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* ($T_{near_balanced}$), defined as follows:

Definition 2 - Near-Balanced Tree ($T_{near_balanced}$): *A tree in which every internal node attempts to obtain a less or equal number of children to the optimal branching factor β.*

The objective of $T_{near_balanced}$ is to yield a structure similar to $T_{balanced}$ 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 $T_{near_balanced}$ and the optimal $T_{balanced}$. We will additionally show in Section 7.3.1 that constructing $T_{near_balanced}$ with the ETC algorithm yields an error of 11% on average for the topologies utilized in this paper.

5.2 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 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 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 T_{input} and the alternative parent list of each sensor. The alternative parent list will be useful in defining a set of parent reassignments that can lead to T_{CETC}.
(2) The sink s_0 conducts an exhaustive search of all possible T_{CETC} trees

and estimates their balancing error w.r.t. to the optimal $T_{balanced}$ tree. It finally chooses the one with the least cost using the $Balancing_Error$ formula presented in 7.3.1.

(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 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 T_{input} the objective of ETC is to transform T_{input} into a near-balanced tree T_{ETC} in a distributed manner. The ETC algorithm consists of a discovery and distributed balancing step which are described next.

5.3.1 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 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).

5.3.2 ETC Phase 2: Balancing

The second phase of the ETC algorithm involves the top-down reorganization of the query routing tree 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 the CETC algorithm in which s_i will order parent re-assignments for its children. The presented algorithm 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 the CETC algorithm 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 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 3. In particular, Figure 3 (left) displays $n = 10$ sensors arranged in an ad-hoc topology T_{input} with a depth $d = 2$. In order to transform 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

Algorithm 3 : **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 s_i should select (denoted as $newparent(s_i)$).

Output: A Near-Balanced Query Routing Tree T_{CETC}.

Execute these steps beginning at s_0 (top-down):

```
procedure Balance_Tree(s_i; children(s_i); ∀_{s_j ∈ children(s_i)} APL(s_j); )
    (β, newParent)=receive();                    ▷ Acquire info from s_i's parent.

    ▷ Step 1: Connect to new parent if needed
    while (newParent != NULL) do
        if (!connect(newParent)) then ▷ Cannot become a child of newParent.
            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).
        end if
    end while

    ▷ Step 2: Adjust the parent of the children nodes.
    if (|children(s_i)| <= β) then          ▷ Skip s_i as no change is necessary.
        for j = 1 to |children(s_i)| do
            send(β, NULL, s_j);             ▷ Send β and no newParent to child.
        end for
    else▷ Ask |children(s_i)| − β nodes to change their parent.
        while (|children(s_i)| > β) do
            s_j = getNext(children(s_i));
            if (|APL(s_j)| > 1) then
                newParent=AlternParent(APL(s_j), s_i);
                send(β, newParent, s_j);                    ▷ Send to s_j.
                children(s_i) = children(s_i) - s_j         ▷ Remove from children.
            else
                send(β, NULL, s_j);                         ▷ Report No change.
            end if
        end while
    end if
end procedure
```

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 par-

ent. 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 T_{ETC} is presented in Figure 3 (right).

6 Experimental Evaluation Methodology

In this section we describe our experimental methodology which involves both a set of real micro-benchmarks on the CC2420 radio chip [52], utilized on MICAz, TelosB and IMote2 sensing devices, and a set of trace-driven simulations with real datasets from Intel Research Berkeley and UC-Berkeley. The experimental evaluation described in this section focuses on three parameters: i) the **Energy Construction Cost**, for creating the WART and ETC structures proposed in this paper, ii) the **Energy Maintenance Cost**, for maintaining the two structures between consecutive epochs and iii) the **Balancing Error**, for the construction of the near-balanced tree with the ETC algorithm. We shall next describe the sensing device used in the experiments, the respective datasets and query workloads.

6.1 Sensing Device

We use the energy model of Crossbow's TelosB [11,42] research sensor device to validate our ideas (see Figure 5 (left)). 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 5.1μA 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.

6.2 Experimental Testbed

We have implemented MicroPulse$^+$ in nesC[16], the programming language of TinyOS[23]. TinyOS is an open-source operating system designed for wireless

embedded sensor nodes. It was initially developed at UC-Berkeley and has been deployed successfully on a wide range of sensor devices (e.g., Mica, Telos, IMote2, RISE[5] mote, etc.). TinyOS uses a component-based architecture that enables programmers to wire together in on-demand basis the minimum required components. This minimizes the final code size and energy consumption as sensor nodes are extremely power and memory limited. nesC [16] is the programming language of TinyOS and it realizes its structuring concepts and its execution model.

To compare our MicroPulse$^+$ framework with TAG and Cougar, we have implemented stripped-down editions of these protocols according to the description provided in Section 4.1. 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.

We utilize the TOSSIM [31] environment to conduct realistic simulations of our code when required. TOSSIM [31] 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 [47], a popular power modeling extension to 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 [47,64], to be more than 90% accurate. In particular, the authors in [47] 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.

In addition to our base implementation, we have also implemented a graphical user interface 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. 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.

6.3 Datasets

We utilize the following three realistic datasets in our trace-driven experiments in order simulate regular-scale, medium-scale and large-scale wireless sensor

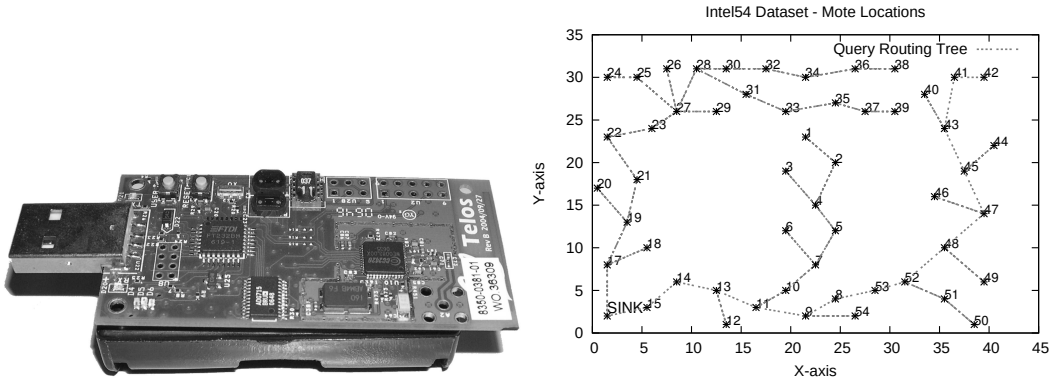

Fig. 5. **Left:** Crossbow's TelosB Mote (TPR2420). Our micro-benchmarks and trace-driven experiments utilize the energy model of the TelosB sensor device and the CC2420 radio transceiver. **Right:** The location of the 54 sensors in the Intel54 dataset and an ad-hoc query tree constructed using the FHF approach.

networks.

i. **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 [25] 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 5 (right)). The depth of the initial query routing tree constructed with the FHF approach is 14.

ii. **Great Duck Island (GDI140)**: This is a medium-scale realistic dataset from the habitat monitoring project deployed in 2002 on the Great Duck Island which is 15km off the coast of Maine [51], USA. We utilize readings from the 14 sensors that had the largest amount of local readings in order to synthetically derive a sensor network composed of 140 nodes that follows the same distribution with the initial dataset. The GDI140 dataset includes readings such as: light, temperature, thermopile, thermistor, humidity and voltage. The average depth of the initial query routing tree constructed with the FHF approach is 24.

iii. **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 need of efficient query routing trees originates from the fact that many applications require the acquisition of data from large-scale environments (e.g., Wireless Sensor Networks, VANETs, People-centric Sensing, etc.) On the other hand, small-scale networks will possibly not require any specialized communication structures as these nodes might be only a few hops away from the sink. Consequently, we focus our experimental evaluation on these larger-scale network (i.e., 54 nodes, 140 nodes and 540 nodes).

6.4 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* [59,34] (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.

6.5 Communication Protocol

Our communication protocol is based on the ubiquitous for sensor networks IEEE standard 802.15.4 (the basis for the ZigBee [66] 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 [56], this scheme is agnostic of the data semantics exhibited at the higher levels of the communication stack. In this paper we exploit these higher level semantics in order to yield better collision handling.

Our data frames are structured as following [31]: Each message is associated with a 5 *Byte* TinyOS header [30]. 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.

7 Experimental Evaluation Results

In order to assess the efficiency of the algorithms presented in this paper we have conducted four experimental series. In the first series we have conducted two micro-benchmarks on the CC2420 radio transceiver in order to quantify the transmission and reception inefficiencies in a real setting. In the second series we have compared the energy consumption of the WART algorithm to the respective algorithms deployed in the Cougar and TAG frameworks under a variety of query workloads and topologies. In the third series we have studied the balancing error and the energy consumption of the ETC algorithm and in the fourth series we have evaluated the efficiency of the overall MicroPulse$^+$ framework focusing on energy consumption and system lifetime.

7.1 **Experimental Series 1:** *Micro-benchmarks*

In the first experimental series we have conduct two micro-benchmarks on the CC2420 radio chip [52] (both attached to the TelosB [11] sensor and in TOSSIM [31]) to justify why data reception and data transmission inefficiencies have to be optimized in current data acquisition systems. 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 6 (left) 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

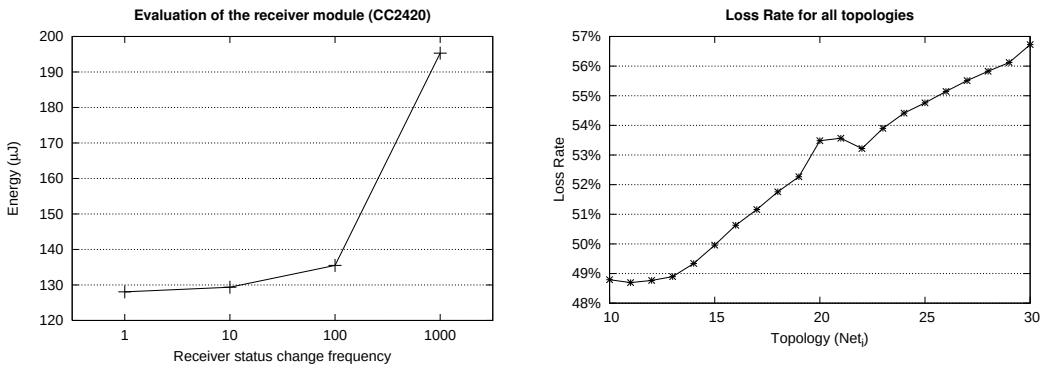

Fig. 6. Micro-benchmarks using the CC2420 communication module. **Left:** Changing the transceiver status from on to off many times significantly increases energy consumption. **Right:** Increasing the number of children per node x also increases collisions during data transmission to node x.

has very quick start-up times compared to other transceivers. Notice that during the startup of the RF module a voltage regulator and crystal oscillator have to be started as well and become stable [52]. 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 presented in this paper 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've 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 6 (right) 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 MicroPulse$^+$ algorithm presented in this paper 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.

7.2 **Experimental Series 2:** *Evaluation of the WART algorithm*

In the second experimental series we assess the efficiency of the WART algorithm in isolation from the ETC algorithm, presented in Section 7.3, in order to highlight the distinct properties of WART (in Section 7.4 we shall also present them in conjunction). Additionally, we will measure the energy of the radio transceiver independently from the rest of the components (flash, data acquisition board, etc.) in order to more accurately capture the differences between the presented algorithms.

7.2.1 Energy Consumption of WART

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 these were described in Section 6.

Energy Consumption for Single-Tuple Answers: Figure 7 (top-left) shows the energy consumption for the Intel54 dataset using the single-tuple query ST. We observe that TAG requires 11,227±2mJ, which is two orders of magnitudes more energy than the energy required by WART (i.e., only 53±35mJ). This is attributed to the fact that the transceiver of a sensor in

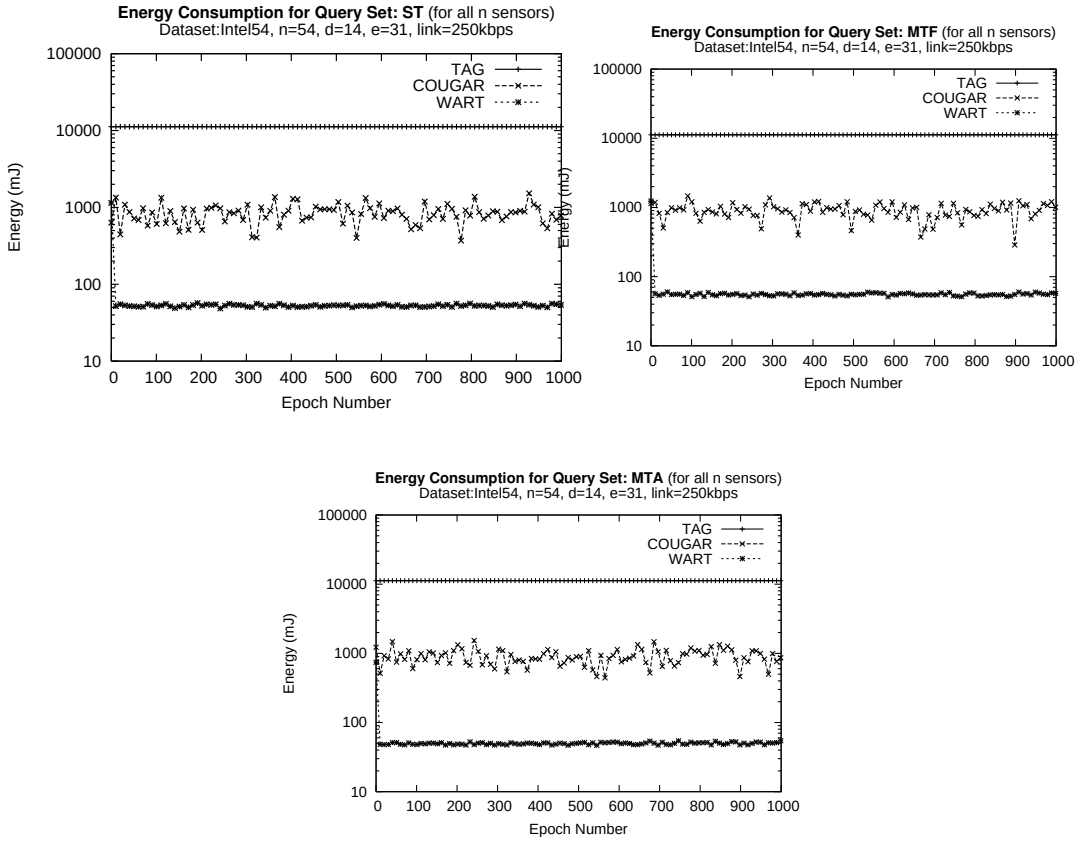

Fig. 7. **Energy consumption for Single-Tuple (top-left) and Multi-Tuple (top-right and bottom) answers**. The plots indicate the individual results for the TAG, Cougar and WART data acquisition algorithms. In all figures we observe that WART is at least one order of magnitude more efficient than its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

TAG is enabled for ≈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 ≈$146ms$ on average. Enabling the transceiver for over two seconds in TAG is clearly the driving force behind its inefficiency. Figure 7 (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 (top-left) also shows that the Cougar algorithm requires on average 882±250mJ, which is one order of magnitude more than the energy required by WART. The disadvantage of 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 (top-right and bottom) and also summarize these results in Table 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 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 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 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).

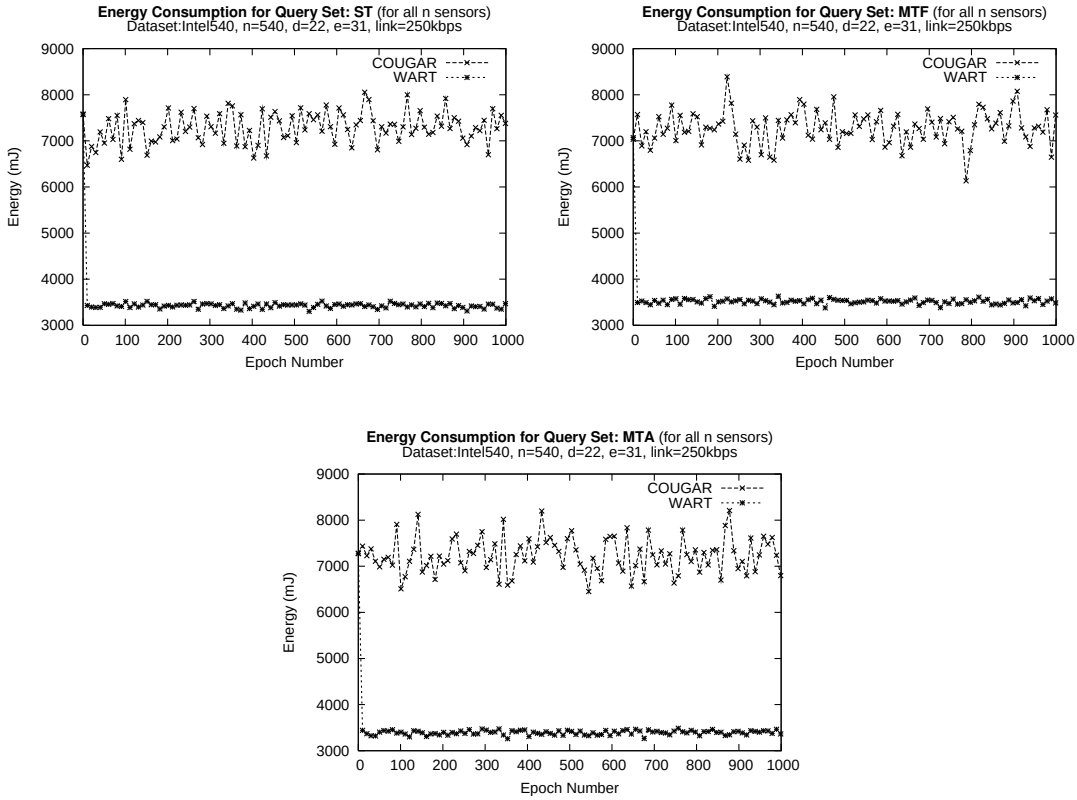

Fig. 8. **Energy Consumption in a Large-Scale Sensor Network (Intel540)**. The plots indicate the individual results for the ST, MTF and MTA queries using the Cougar and WART algorithms (we omit the TAG curve in this plot due to its inefficiency (i.e., $189,707mJ$)).

7.2.2 Probing WART in a Large-Scale Network

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 2 (third row).

The plots in Figure 8 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

Table 2
Energy Consumption results for experimental series 2: 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

than Cougar under these limitations, thus WART can have many practical applications in large-scale environments.

7.2.3 WART 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 4.4. 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 9 (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 6). 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 9 (bottom) shows the adaptation algorithm

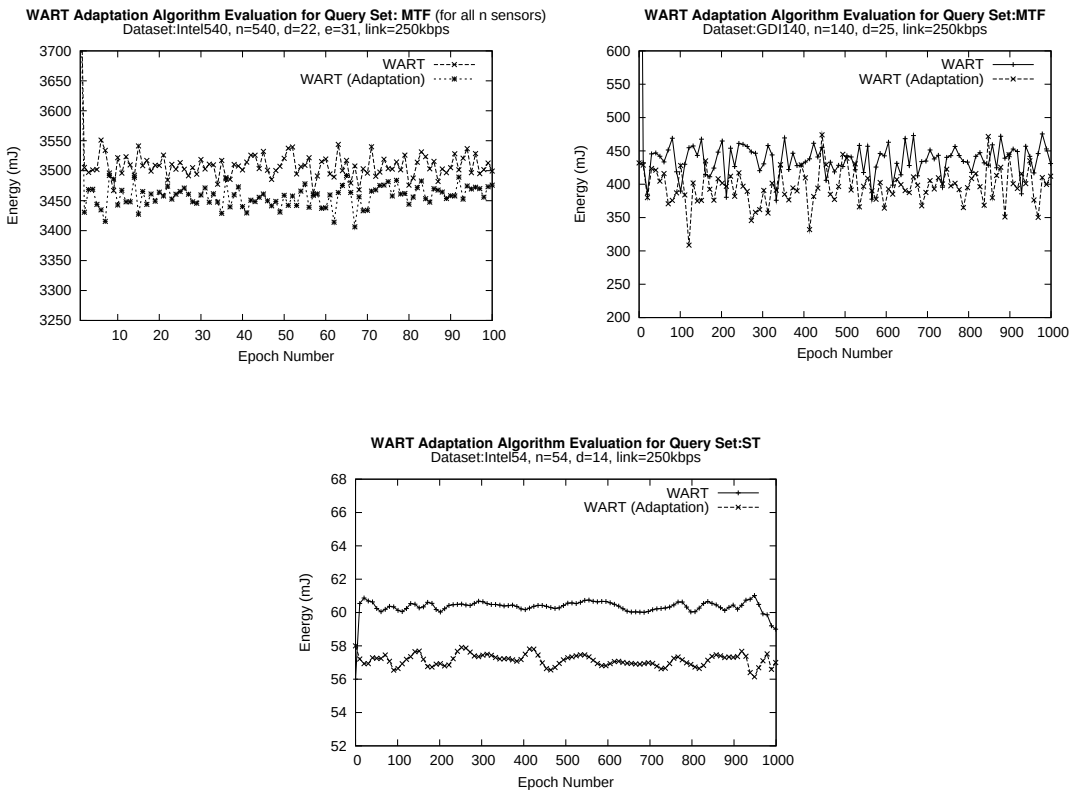

Fig. 9. WART's Adaptation algorithm evaluation for the Intel540 dataset (top-left), GDI140 dataset (top-right) and the Intel54 dataset (bottom).

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 **Experimental Series 3:** *Evaluation of the ETC algorithm*

In the third experimental series we assess the efficiency of the MicroPulse$^+$ ETC algorithm. We start out by assessing the construction quality of the ETC algorithm and then proceed with an in-depth evaluation of our algorithm.

7.3.1 Measuring the Balancing Error

Our first objective is to measure the quality of the tree, with regard to 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.

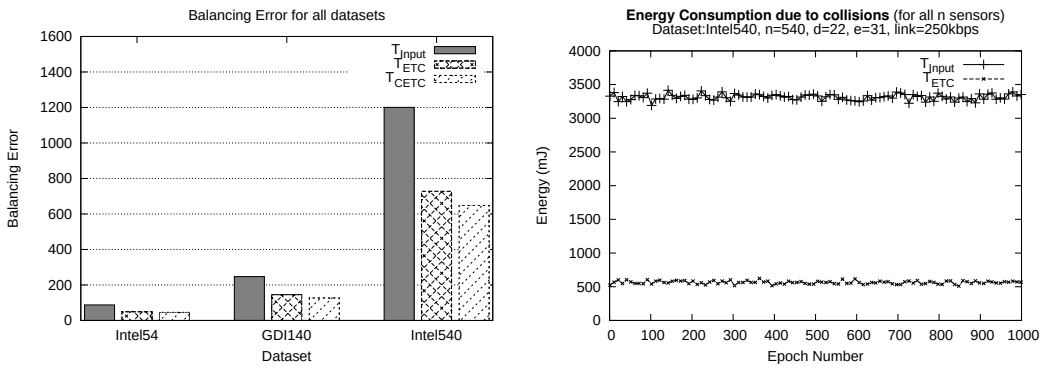

Fig. 10. **Left:** Measuring the Balancing Error of the FHF (T_{input}), CETC (T_{CETC}) and ETC (T_{ETC}) algorithms; **Right:** Energy Consumption due to re-transmissions in an unbalanced topology (T_{input}) and in a near-balanced topology (T_{ETC})

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 10 (left), 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 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.

7.3.2 Energy Consumption of ETC

In order to translate the effects of the Balancing_Error into an energy cost, we conduct another experiment using the Intel540 dataset. 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 10 (right) displays the energy consumption of the two structures. We observe that the energy required for re-transmissions using T_{input} is 3,314±50mJ. On the other hand, T_{ETC} requires only 566±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.

7.4 **Experimental Series 4:** *Evaluation of the MicroPulse$^+$ Framework*

In the fourth experimental series we assess the efficiency of the complete MicroPulse$^+$ framework, which deploys the ETC algorithm to balance the query routing tree and then utilizes the WART algorithm to optimize the waking windows of the sensor nodes. In particular, we conduct two experiments focusing on energy consumption and system lifetime.

7.4.1 Energy Consumption of MicroPulse$^+$

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 11 (left) illustrates the energy savings of using the ETC algorithm in conjunction with WART. While WART requires on average 3,510±126mJ, MicroPulse$^+$ uses only 749±269mJ which translates in an additional 78% decrease in energy consumption on average. In particular, we have observed a threefold improvement of MicroPulse$^+$ compared to the execution of the WART algorithm in isolation. Additionally, we have noticed that the near-

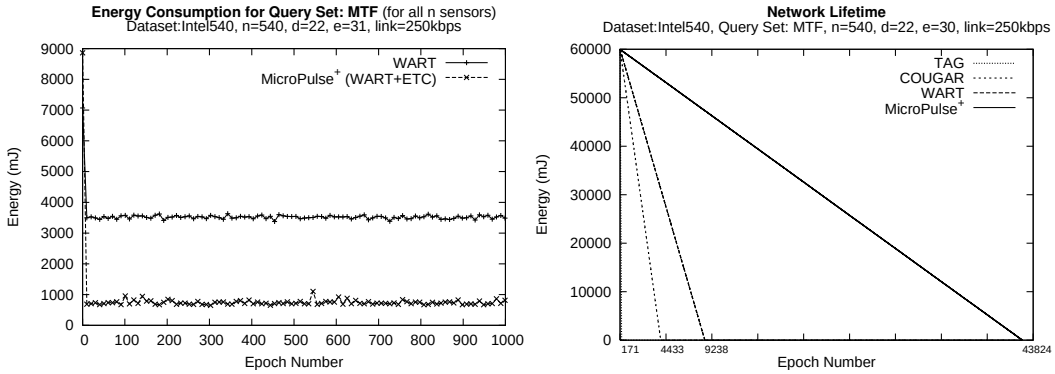

Fig. 11. **Left:** Energy Consumption comparison of MicroPulse$^+$ against WART. The plot indicates that the ETC technique provides a three-fold improvement to the savings incurred by the WART algorithm. **Right:** Network Lifetime for the various algorithms presented in this paper.

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 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.

7.4.2 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 [53], as the time instance t' at which $Energy(t') = 0$. Note that this applies only to the case where sensors operate using batteries. Double batteries (AA) used in many current sensor designs (including the TelosB sensor) operate at 3V voltage and supply a current of 2,500 mAh (milliAmpere per hour). Assuming similar to [51], 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 11 (right) 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 MicroPulse$^+$ framework, 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, MicroPulse$^+$ reaches its limit far later at epoch 43,824 (i.e., 365 hours) and this can be translated into a ≈78% increase of the network lifetime.

7.4.3 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 a MicroPulse$^+$ framework. The above discussion shows that it is relatively easy to extend the MicroPulse$^+$ framework into a multi-query execution environment although a detailed investigation of this parameter is outside the scope of this paper.

8 Related Work

Power conservation 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 disabling/hibernating the radio transceiver 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 paper while the rest come with their own tradeoffs 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 widely adopted ISO/OSI communication stack [29]. 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 MicroPulse$^+$ 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 [21,49] that look at individual and local power management optimizations.

Examples of these techniques are the *Dynamic Voltage Scaling (DVS)* and *Embedded power supply for low power Digital Signal Processor (DSP)* [21] 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)* [49] 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 MicroPulse$^+$ framework we presented in this paper.

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 [50,48,60,40] minimize energy consumption by minimizing collisions and overall usage of the shared access medium.

The *Coordinated Power Conservation algorithm (CPC)* [50] 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 MicroPulse$^+$ framework. However, CPC servers coordinate in a distributed manner without obtaining any global information from the base station. That does not provide CPC server with a universal view of the system. Furthermore, the scheduling on WART is based on the query workload incurred on each sensor node while in CRC misses the inclusion of such high-level semantics.

Power-aware Multi-Access Protocol with Signaling (PAMAS) [48], 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)* [60], 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 [40] to minimize the end-to-end delay. Both PAMAS and S-MAC achieve high energy savings by allowing sensor 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. Recall that in MicroPulse$^+$, 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 [60] as one of the primary MAC protocols, these techniques extend the power management capabilities of MicroPulse$^+$ inherently.

Sensornet Protocol (SP) [41], introduces a unified link level abstraction that is part of the sensor network architecture proposed in [12]. 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 [19,13,58,22].

The *Power-Aware Routing (PAR)* [19] 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 [13] 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 MicroPulse$^+$ differs from these approaches as the ETC algorithm 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 MicroPulse$^+$ in order to achieve even more energy savings.

In *Modular Network Layer* [15] 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 [41]. 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 MicroPulse$^+$, 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* [54] 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* [26] 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, MicroPulse$^+$ 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 [2]). 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) [28], 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) [57], 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 MicroPulse$^+$ as its application-level information does not include the high level query semantics used in MicroPulse$^+$. Furthermore, the concept of introducing delays in order to conserve power is not acceptable in MicroPulse$^+$ as we assume that queries have specific response time requirements that must be met.

The second class of application layer techniques includes those techniques that use clustering mechanisms [58,22,9]. An example of these techniques is *Geographical Adaptive Fidelity (GAF) [58]*, 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) [22]* 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 th energy resources of an individual sensor node. One final cluster-based protocol is *SPAN [9]*, 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 MicroPulse$^+$ take advantage of global information to preserve energy like MicroPulse$^+$. 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 MicroPulse$^+$ 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, these works [1,39] have developed middleware systems that enable integration and management of many WSN sites. Additionally, multi-tier sensor systems like TENET [17], that aim to combine the low-power sensor devices we discuss in this work with powerful 32-bit nodes (e.g., Stargates or ordinary PCs), are another direction in sensor networks optimization. Yet, all these techniques are complementary to the approaches we have outlined in this paper as our techniques structure efficient and well- formed WSN deployments while middleware techniques utilize these as a building block.

9 Conclusions and Future Work

In this paper we present *MicroPulse*$^+$, a novel framework for minimizing the consumption of energy during data acquisition in WSNs. MicroPulse$^+$ introduces two novel concepts: i) the *Workload-Aware Routing Tree (WART)* algorithm, which is established on profiling recent data acquisition activity and on identifying the bottlenecks using an in-network execution of the critical path method; and ii) the *Energy-driven Tree Construction (ETC)* algorithm, which balances the workload among nodes and minimizes data collisions. Our experimentation with micro-benchmarks on the CC2420 radio chip and trace-driven experimentation with real datasets from Intel Research Berkeley and UC-Berkeley show that the *MicroPulse*$^+$ algorithms provide orders of magnitudes energy reductions under a variety of conditions prolonging the longevity of a sensor network.

In the future we plan to study how these ideas can be incorporated into existing data acquisition frameworks such as TinyDB and deploy them in a real environment. Integrating our ideas in a fully functional declarative acquisition system poses many additional challenges that we aim to tackle in the future (e.g., low-level protocol integration, efficient internal structures, deployment issues, etc.). We believe that by integrating these ideas in a fully functional system will enable us to study system-level effects and thus better demonstrate the efficacy of our proposed approaches.

Acknowledgments

We would like to thank Joe Polastre (UC Berkeley) for the Great Duck Island data trace. This work was supported in part by the Open University of Cyprus under the project SenseView, the US National Science Foundation under projects S-CITI (#ANI-0325353) and AQSIOS (#IIS-0534531) and the European Union under the project mPower (#034707).

References

[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, 2007, pp. 198-205.

[2] 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, 2006, pp. 1102-1107.

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

[4] P. Andreou, D. Zeinalipour-Yazti, P.K. Chrysanthis, G. Samaras, "Workload-aware Optimization of Query Routing Trees in Wireless Sensor Networks", In Proceedings of the Nineth International Conference on Mobile Data Management, Beijing, China, April 27-30, 2008, pp. 189-196.

[5] 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.

[6] 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, Article No.18, 2006.

[7] 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, 2008, pp.12-21.

[8] 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, 2003, pp. 1-8.

[9] 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, 2001, pp. 85-96.

[10] J. Considine, F. Li, G. Kollios, J. Byers, "Approximate aggregation techniques for sensor databases", In Proceedings of the 20th International Conference on Data Engineering, Boston, MA, USA, 2004, p. 449.

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

[12] 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, 2005, pp. 24.

[13] 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, 1997, pp. 376-380.

[14] 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, Paris, France, 2004, pp. 449.

[15] 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, 2006, pp. 249-262.

[16] 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 Conference on Programming Language Design and Implementation, San Diego, CA, USA, 2003, pp. 1-11.

[17] 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. SenSys 2006: 153-166.

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

[19] 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, 1999, pp. 380-383.

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

[21] 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, 1997, pp. 425-435.

[22] 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, 2000, pp. 3005-3014.

[23] 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, 2000, pp. 93-104.

[24] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks", In Proceedings of the The Annual International Conference on Mobile Computing and Networking, Boston, MA, USA, August 6-11, 2000, pp. 56-67.

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

[26] 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, 2007, pp. 351-365.

[27] 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, 2007, pp. 254-263.

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

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

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

[31] 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, Los Angeles, California, USA, 2003, pp. 126-137.

[32] 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, Vol. 1, No. 4, 2005, pp. 313-326.

[33] 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, 2004, pp. 256-269.

[34] 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 Fifth Symposium on Operating Systems Design and Implementation, Boston, MA, 2002, pp. 131-146.

[35] 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, San Diego, CA, USA, June 9-12, 2003, pp. 491-502.

[36] 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, 2006, pp. 425-426.

[37] D. Moss, J. Hui, K. Klues, "Low Power Listening, Core Working Group, TEP 105

[38] 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.

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

[40] 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, 2009, pp. 322-343.

[41] 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, 2005, pp. 76-89.

[42] J. Polastre, R. Szewczyk, D.E. Culler, "TELOS: Enabling Ultra-low Power Wireless Research", In Fourth International Symposium on Information Processing in Sensor Networks, Los Angeles, CA, USA, April 25-27, 2005, pp. 364-369.

[43] 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, Vol 13, No. 4, December, 2004, pp. 384-403.

[44] 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, San Diego, CA, USA, September 19, 2003, pp.69-76.

[45] 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, 2005, pp. 53-60.

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

[47] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications", In ACM SenSys, 2004, pp. 188-200.

[48] 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, 1998, pp. 5-26.

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

[50] 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, 2002, pp. 3330-3335.

[51] 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, Baltimore, MD, USA, November 3-5, 2004, pp. 214-226.

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

[53] H. Thomas, S. Yi, H.D. Sherali, "Rate allocation in Wireless Sensor Networks with Network Lifetime Requirement", In *The ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Tokyo, Japan, 24-26, 2004, pp. 67-77.

[54] 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, 2000, pp. 12.

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

[56] 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, 2001, pp. 221-235.

[57] 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.

[58] 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, 2001, pp. 70-84.

[59] Y. Yao, J.E. Gehrke, "Query Processing in Sensor Networks", In Proceedings of the First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, January, 2003, pp. 5-8.

[60] 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, 2004, pp. 493-506.

[61] 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, 2004, pp. 668-672.

[62] 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 International Conference on Mobile Data Management, Mannheim, Germany, May 7 - 11, 2007, pp. 182-189.

[63] 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 International Conference on Mobile Data Management, International Workshop on Data Intensive Sensor Networks, Mannheim, Germany, May 11, 2007, pp. 351-355.

[64] 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 conference on USENIX Conference on File and Storage Technologies, San Francisco, CA, USA, December 13-16, 2005, pp. 31-44.

[65] 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, Trondheim, Norway, August 29, 2005, pp. 61-66.

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