Minimum-Hot-Spot Query Trees
for Wireless Sensor Networks

Georgios Chatzimilioudis
Dept. of Computer Science
University of California
Riverside
Riverside, 92507 CA USA
gchatzim@cs.ucr.edu

Demetrios
Zeinalipour-Yazti
Dept. of Computer Science
University of Cyprus
1678 Nicosia, Cyprus
dzeina@cs.ucy.ac.cy

Dimitrios Gunopulos
Dept. of Informatics and
Telecommunications
University of Athens
15784 Ilisia, Greece
dg@di.uoa.gr

ABSTRACT

We propose a distributed algorithm to construct a balanced communication tree that serves in gathering data from the network nodes to a sink. Our algorithm constructs a near-optimally balanced communication tree with minimum overhead. The balancing of the node degrees results in the minimization of packet collisions during query execution, that would otherwise require numerous retransmissions and reduce the lifetime of the network. We compare our simple distributed algorithm against previous work and a centralized solution and show that for most network layouts it outperforms competition and achieves tree balance very close to the centralized algorithm. It also has the smallest energy overhead possible to construct the tree, increasing the lifetime of the network even more.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

Keywords

Query Tree, Communication Cost, Wireless Sensor Network, Collision, Optimization

1. INTRODUCTION

Wireless sensor networks (WSNs) are a multi-purpose tool for a wide range of tasks. Nodes do not need physical connection in order to communicate with each other; they use radio transceivers. Also, they do not need to be attached to any static energy source; they have batteries on board. The price to pay is the energy needed for communication and on-board batteries have limited energy supplies. Further, the wireless channel in WSNs is unreliable with many dropped packets. Collisions between data packets is the main cause of dropped packets, when multiple sensor nodes attempt to access a shared wireless channel. Every dropped packet needs to be retransmitted adding to the energy consumption.
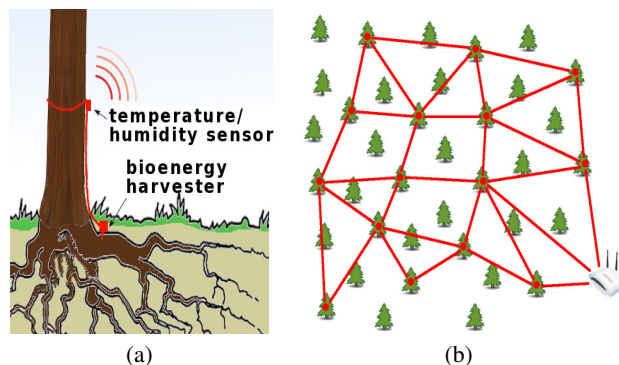
Figure 1: Application that greatly benefits from an energy-efficient query routing tree. (a) Energy harvesting for battery-less nodes for the (b) Voltree monitoring application [24]

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 [29, 17, 18, 30, 27].

In data acquisition systems, data from every node needs to be collected at a sink node. An example is shown in Figure 1(b). Existing approaches mainly use a *Query Routing Tree* (denoted as T), which provides each sensor with a path over which query answers can be transmitted to the querying node. Energy-efficient query routing trees are needed in a plethora of systems such as People-centric Sensing [3, 19], structural monitoring [13], urban monitoring [19] and environmental monitoring [24, 22, 31] among others.

For example the *Voltree* environmental monitoring sensor networks [24] (Figure 1) use a bioenergy harvester that converts living plant metabolic energy to electricity providing small but steady amounts of energy to a sensor device. Such applications deploy Query Routing Tree structures as a primitive mechanism for percolating query results to nodes that query the network. As another example imagine cyclists that journey through the main streets of a city. Each cyclist is equipped with a mobile device (e.g., Android, I-Phone or Maemo), that has the ability to interact with its integrated sensors. 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). A Query Routing Tree will provide a necessary overlay structure for collect-

ing and aggregating data (e.g., identify routes with low CO_2 levels in the city). For ease of exposition, in this paper we only focus on static sensor networks as opposed to mobile sensor networks.
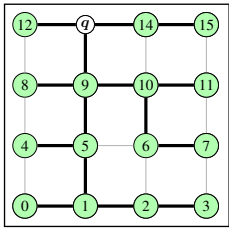
Figure 2: 4x4 grid network. A First-Heard-From tree.
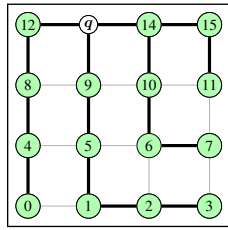
Figure 3: A *COPT* solution. $COV_{COPT} = 2.43$

Query routing trees are usually constructed in an ad-hoc manner and therefore there is no guarantee that the query workload will be distributed equally among all sensors. Various data acquisition frameworks (e.g., [17, 18, 27, 9, 16, 32, 15]) have been proposed to optimize query execution. Some by minimizing the data sent or the time that the transceiver is operating, and some by distributing the higher workload of the nodes around the sink node. All the previous techniques are orthogonal to our solution.

We reduce energy consumption by minimizing collisions during query execution. We balance the number of incoming streams for internal nodes of a query routing tree. In Figure 2 we show a naive first-heard-from query routing tree, which is the most common form of acquisitional query tree used. Nodes with many incoming streams are primary candidates for network hot-spots, where packet collisions cause numerous retransmissions. The more the retransmissions the greater the energy overhead for executing a query. The optimal tree can be seen in Figure 3.

It is shown that executing 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 30 children will lead to a 56% loss rate [1]. These figures translate into an approximately three-fold increase in energy demand due to inevitable re-transmissions of data packets. Consequently, unbalanced query routing trees can severely degrade the network health and efficiency.

In this paper we introduce a distributed algorithm, coined *Minimum HotSpot Query Routing Tree* (*MHS*), for constructing an efficient query routing tree in wireless sensor networks that minimizes collisions during query execution. *MHS* deploys a distributed tree balancing process, where the node degree is balanced for each tree depth. *In particular,* MHS *is based on letting nodes select parents sequentially while "snooping" the wireless channel and counting the degree of their candidate parent nodes. When the time comes to select a parent, the node simply selects the parent with the minimum degree.*

The *MHS* algorithm provides *balanced query routing trees* with the following desirable properties: i) they decrease collisions during data transmission, ii) they decrease query response time, and iii) they increase system lifetime and coverage. In addition, the cost for running *MHS* is the lowest possible. *MHS* performs very close to the optimal, especially in networks that resemble real world sensor network layouts. We validate these claims in our experimental evaluation section. In order to assess the efficiency of *MHS*, we compare it against the *ETC* [1] algorithm and against a centralized optimization algorithm *COPT* for balancing the query routing tree.

Our contributions can be summarized as follows:

- We define balance in a query routing tree taking into account the communication restrictions in a network. This allows us

to decompose the tree construction problem into a set of simpler subproblems (*Balanced Assignment Problem*).

- We propose a novel distributed algorithm (*MHS*) to approximate the *balanced query routing tree construction* solution.

- We prove *MHS* poses lower bound communication overhead.

- We show through simulations that *MHS* improves the balance of the resulting query routing tree significantly over the previously proposed algorithm and for real world networks even performs as good as the centralized algorithm.

We discuss related work in Section 2, present the system model and define the *balanced query routing tree* problem (Section 3). In Section 4 we analyze the problem, decompose it into simple subproblems and present our centralized algorithm. The description of our distributed algorithm follows in Section 5, the experimental evaluation in Section 6 and a short summary Section 7.

2. RELATED WORK

In this section we present some popular techniques for minimizing energy consumption in wireless sensor networks, starting from the Physical layer and moving up to the Application layer according to the ISO/OSI communication stack [14].

Various Medium Access Control (MAC) layer algorithms exist for minimizing collisions and wireless channel usage (e.g. [20, 21, 28]). However, none of these approaches considers the underlying topology of the sensor network, inter-sensor relationships and high-level query semantics. The MAC layer algorithms are orthogonal to our problem, where we try to construct a query routing tree that inherently has small chances of collisions in the first place.

Network layer mechanisms exist to discover optimal routing paths for energy efficient delivery of messages through intermediate hosts (e.g. [8, 25, 9]). Existing approaches make use of an a-priori established query routing tree. Our work constructs a balanced tree specific for a given query and is also able to exploit opportunistic in-network processing on the nodes.

Transport layer works like [12, 23], propose techniques to either minimize the on-time of the radio by synchronizing neighboring nodes for transmission and reception, or avoid dropped packets and collisions by using small probe messages and snooping. These proposals are orthogonal and can coexist with our algorithms.

In the Application layer various data acquisition frameworks (e.g. [17, 18, 27, 15]) that aim at minimizing the data sent or the operating time of the transceiver. Works focusing solely on optimizing the query routing tree structure are either centralized approximations [7] or assume that every sensor has the potential transmission power to reach the sink node directly. Algorithms proposed in [9, 16, 32] alternate the leader node that will gather all the data and finally forward the query answer to the base-station in order to avoid the hot-spots around the base-station.

A distributed algorithm to create a query routing tree without the assumption that every node can transmit directly to the sink, has been proposed in [26]. Their algorithm balances the data load to be transmitted from one tree level to the next. The goal is to balance the data received and relayed by each node in the network. The energy savings by this work are mostly theoretical since they do not deal with collisions occurring from many nodes trying to communicate with the same parent. As shown by Andreou et al. [1], the energy loss due to hot-spots in the tree can not be neglected.

Andreou et al. [1] is the only work minimizing collisions by balancing the query routing tree, to the best of our knowledge. They aim at solving the same problem with the same assumptions. Their

algorithm, called *ETC*, constructs an initial temporal unbalanced tree and then lets nodes communicate back and forth in order to reorganize and balance the tree. *ETC* adds significant communication overhead in order to balance the tree. A bigger downfall for *ETC* is that it is based on the global branching factor of the tree, which in many network topologies results in a badly balanced tree.

We propose a distributed algorithm (*MHS*) that constructs a query routing tree with minimum hot-spots (balanced tree). This translates in a reduced number of packet collisions during query execution. *MHS* has significantly reduced overhead for constructing a tree and the resulting tree deviates far less from the optimally balanced tree when compared to *ETC* as is shown in our experiments.

3. SYSTEM MODEL

In this section we formalize our system model and the basic terminology that will be utilized in the subsequent sections. We give formal definitions and proposition that we use in our algorithms and use to define the balanced tree construction problem.

Let V denote a set of n sensing devices $\{v_1, v_2, ..., v_n\}$ in a wireless sensor network. Now let $G = (V, E)$ denote the network graph that represents the implicit network edges E of the nodes in V. The edges in E are implicit, because there is no explicit connection between adjacent sensor nodes, but nodes are considered neighbors if they are within communication range (i.e., a fundamental assumption underlying the operation of a radio network).

We assume that nodes can pose continuous queries over the sensor network. The sensor node that issues the continuous query Q is called querying node and is denoted as q. It actually supports any type of continuous query (e.g. aggregate queries) as long as the query produces a continuous result which is percolated to the querying node. We will use the terms querying node and sink interchangeably throughout the paper.

Assuming that the nodes have a restricted communication range data will have to travel over a multi-hop path toward the querying node. In this case a routing tree T is created connecting every node over a multi-hop path with the sink. The querying node q is the root of this tree and receives the information needed to answer the query. We will denote as d_v the depth of v in this tree.

In this work we want to construct shortest-path (minimum height) trees in order to minimize query execution latency. Given a wireless sensor network, represented as a graph $G = (V, E)$, and a sink node q, we know in advance that the depth d_v of node v will be equal to the shortest hop distance from v to q in G. We can thus define the subset of nodes $V_d \subseteq V$ that are at depth d.

Definition 1. Given a tree $T = (V, A)$ and an edge $(u, v) \in A$, the node at the smaller depth ($min\{d_u, d_v\}$) is called *parent*, and the other is called *child*. We say that parent u adopted child v. Note that since we deal with trees there can be only one parent per child.

Definition 2. Given a graph $G(V, E)$ and a tree construction request $T = (V, A)$, the set of *candidate parents* $P_c \subseteq V$ of node $c \in V$ is $P_c = \{p | (c, p) \in E \text{ and } d_p = d_c - 1\}$.

To minimize collisions during query execution, we need to minimize the number of children (*degree*) of a node in the tree T. In order to minimize the degrees of all nodes we need to balance the degrees between the nodes, keeping all degrees at a balanced minimum. In a minimum height tree the *degrees* of depth d can not be balanced with *degrees* of a different depth d', since each node belongs to a predefined depth and has *candidate parents* only in the immediate smaller depth. This lets us divide the definition and the problem of balancing into pair of tree depths.

Definition 3. A balanced tree T is a tree where at each depth d the variation of the degrees of depth d is the minimum possible.

The node degree variation of a set of nodes measures how different their degrees are. Formally, we use the *Coefficient of Variation* (COV) in order to express this variation. Generally, *COV* is used as a normalized measure of dispersion of a distribution. It is defined as the ratio of the standard deviation (σ) to the mean (μ): $\frac{\sigma}{\mu}$. The coefficient of variation is useful because the deviation of data must always be understood in the context of the mean of the data. It is thus very suitable for comparing data of widely different means.

For ease of exposition consider the following directed tree $T = (V, E)$ with $V = \{s_1, s_2, s_3, s_4\}$ and $E = \{(s_2, s_1), (s_3, s_1), (s_4, s_2)\}$, where the pairs in the E set represent the edges of the tree. Node A is the root of the tree and has two children s_2 and s_3. In other words, node A is at depth zero and has $degree_1 = 2$. The mean value of degrees in this depth is $\mu = degree_1$ and the standard deviation σ is $\sigma = \sqrt{(degree_1 - \mu)^2} = 0$ Thus $COV = 0$ thus the tree in depth zero is perfectly balanced. At depth zero the tree is always perfectly balanced since there is only one parent at this depth: the root. Similarly, for depth one $\mu = (degree_2 + degree_3)/2$, $\sigma = \sqrt{\frac{(degree_2 - \mu)^2 + (degree_3 - \mu)^2}{2}} = 0.5$ and $COV = \frac{\sigma}{\mu} = 1$. Note that COV can not be always zero depending on the connections in the network. The formal definition of our (*balanced tree construction problem*):

Definition 4. Given a network $G = (V, E)$ and a sink, construct a minimum height tree $T = (V, A)$ with $A \subseteq E$ that minimizes the coefficient of variation COV_d for each depth $1 \le d \le height(T)$.

4. CENTRALIZED ALGORITHM

We present a centralized algorithm, *COPT*, to solve the problem from Definition 4. *COPT* aids at grasping the intuition behind our distributed algorithm, presented later on, and also serves as the ground truth in our experimental evaluation. *COPT* makes use of a routine that solves a smaller subproblem called *Balanced Assignment Problem*. As described in the previous section, we can connect nodes only of neighboring depths. Balancing the tree T would thus mean balancing each tree depth d individually.

Assignment problems have been studied as early as 1865 by Jacobi [10] and exist in various disciplines. The assignment of each node in V_{d+1} of depth $d+1$ to some node in V_d of depth d is a type of assignment problem [2]. In literature a similar problem to the one defined in Definition 4 has been studied, that only minimizes the maximum degree of the nodes in V_d. Given a bipartite graph $G = (V_{d+1}, V_d, C)$ the proposed algorithms return an assignment $A \subseteq C$ such that $max\{degree(v), \forall v \in V_d\}$ is minimized. Such algorithms are proposed in [2, 6, 5, 4].

Our subproblem of finding a balanced assignment between V_d and V_{d+1} is defined as follows:

Definition 5. Balanced Assignment Problem: Given the bipartite graph $G = (U, W, C)$ with bipartition (U, W) we are looking for an assignment $A \subseteq C$ such that COV_W is minimized.

Note, that in our tree construction efforts we deal with sets of nodes of consecutive tree levels, thus we could substitute $U = V_{d+1}$ and $W = V_d$. For the next few paragraphs we need two more notations: Given a set of edges E and a set of nodes V, $nodes(V, E)$ denotes all the nodes $v \in V$ incident to the edges in E and $edges(E, V)$ denotes the edges $e \in E$ incident to nodes in V.

We propose Max_Degree that adapts the *Cardinality* algorithm proposed by Chang and Ho [4] to serve as a routine in our solution to the *Balanced Assignment* problem. Max_Degree formally it solves the following problem: Given a bipartite graph $G = (U, W, E)$ and a partial assignment A_{part} return an assignment $A \subseteq E$ with $A_{part} \subseteq A$, such that $max\{degree(v), \forall v \in W - node(W, A_{part})\}$ is minimized. We repeat Max_Degree for each node in W that can reduce its degree further. The node of W that forces Max_Degree to terminate is identified and the edges to its children are put inside the partial result A_{part}. In each repetition of the Max_Degree process we feed the partial result of the previous call. Next we present the algorithm in Algorithm 1.

Algorithm 1 . Max_Degree

Input: bipartite graph $G(U, W, E)$ with bipartition (U, W) and a partial solution A_{part}
Output: new A_{part} for next iteration
1: $A \leftarrow A_{part}$
2: $maxDegree \leftarrow 0$;
3: $U' \leftarrow U - nodes(U, A_{part})$;
4: $C' \leftarrow C \cap edges(C, U')$ ▷ process unconnected children
5: $v \leftarrow nil$;
6: **for all** $s \in U'$ **do**
7:     set all vertices in U' and W unscanned;
8:     erase labels of all vertices in U' and W;
9:     label s by "start";
10:     $flag \leftarrow$ true;
11:     **while** flag is true **do**
12:         **if** there is a labeled and unscanned vertex i **then**
13:             **if** $i \in U$ **then**
14:                 identify all edges $(i, j) \in C$;
15:                 label each unlabeled node j by "i";
16:                 mark i as scanned;
17:             **else if** $i \in W$ and $degree(i) \geq maxDegree$ **then**
18:                 identify all edges $(j, i) \in A$;
19:                 label each j by "i";
20:                 mark i as scanned;
21:             **else if** $i \in W$ and $degree(i) \leq maxDegree$ **then**
22:                 Path$\leftarrow$ backtrack from i to s following labels;
23:                 $A \leftarrow (A - Path) \cup (Path - A)$;
24:                 mark i as scanned;
25:             **end if**
26:         **else**
27:             find edge $(s, t) \in C'$ with $t \in W$ and $degree(t) = maxDegree$;
28:             $A \leftarrow A + (s, t)$;
29:             $v \leftarrow t$;
30:             $maxDegree \leftarrow maxDegree + 1$;
31:             $flag \leftarrow$ false;
32:         **end if**
33:     **end while**
34: **end for**
35: $Y \leftarrow all(i, v) \in A$;   ▷ A is the solution to the assignment problem where the maximum degree of the nodes not in A_{part} is minimized
36: $A_{part} \leftarrow A_{part} + Y$;
37: output(A_{part});

Due to lack of space we focus mainly on the changes made to the initial *Cardinality* algorithm to get Max_Degree. For details on *Cardinality* please see the work of Chang and Ho [4]. The main change is in the input and the output. The input is a partial solution A_{part} containing already some edges $e \in C$. The output of *Car-*

dinality is the edge list A constructed during the for-loop (Lines 6 - 34). Instead, the solution of Max_Degree, denoted as A_{part}, only uses the input partial solution augmented by the edges from A that connect to node v (Lines 35 - 36). Node v is the node that forced the last increase in the $maxDegree$ variable in Lines 29-30.

The next step is to present the *Balanced Assignment Approximation* (BAA) shown in Algorithm 2. It uses the Max_Degree algorithm as a subroutine and uses it to minimize the degree of every node in set W until no more degree can be decreased. After every call to the routine, the output A_{part} is fed as input to the next routine call. The routine just minimizes the degree of the nodes $u \notin A_{part}$. The Max_Degree routine is called iteratively until $degree(u) = 1, \forall u \notin A_{part}$. BAA performs the task of minimizing the variation of the node degrees better than *Cardinality*. Starting from the parent with maximum minimum degree and minimizing its degree, forces the nodes with lower degrees to share the assignments with the high degree nodes. BAA calls *Cardinality* $|W|$ times and in every run i it iterates $|C| - i$ nodes in the worst case. The complexity of *Cardinality* is $O(|U||C|)$ [4] thus the time complexity of BAA is bound above by $(O(|W||U|log|C|))$.

Algorithm 2 . BAA

Input: bipartite graph $G = (U, W, C)$
Output: assignment $A \subseteq C$ such that COV_W is minimized
1: $reducedDegree \leftarrow$ inf
2: $A_{part} \leftarrow 0$
3: **while** $degree(u) > 1, \forall u \notin A_{part}$ **do**
4:     $A_{part} \leftarrow Max\_Degree(G, A_{part})$
5: **end while**

Our centralized algorithm (*COPT*) solves the *Balanced Assignment* problem for each set of neighboring depths V_d and V_{d+1}. Combining the solution for each depth gives us a the query routing tree. A good upper bound for the complexity of *COPT* given a graph $G = (V, E)$ is $O(|V|^2log|E|)$. Although *COPT* is very easy to implement in a distributed fashion, its high communication cost makes it unattractive. In each loop of Max_Degree we scan a node. In a distributed implementation of BAA this would translate into a message between two nodes passing the process control to the next node. Thus, the number of messages exchanged in a decentralized version of *COPT* would be $O(|V|^2log|E|)$. Sending all the needed information to the sink q and perform the algorithm centrally, as presented above, would require only $O(|V|)$. We only use *COPT* as ground truth to compare our distributed algorithm against in our evaluation in Section 6.

5. DISTRIBUTED ALGORITHM

We propose *Minimum Hot-Spot* (*MHS*), a distributed algorithm that creates a balanced query routing tree connecting all nodes of the network. First we show how this problem can be broken down per tree depth in section 5.1 and determine the minimum possible communication cost needed to construct a tree. In section 5.2 we describe how to solve the subproblem per tree depth and in section 5.3 we present the whole *MHS* process.

5.1 Problem Breakdown

As in the centralized version and as explained in Section 3 optimal balancing between tree depths can be done independently. Nodes of depth d interact with nodes in depth $d + 1$ to determine parent-child connections. At the end we combine the parent-children assignments of each tree depth to get the final balanced tree.

Given a graph $G = (V, E)$ and a querying node r, we have a set of nodes V_d at distance d from r, a set of nodes V_{d+1} at distance $d+1$ from r, and the subset of edges $C \in E$ connecting V_d and V_{d+1}. We solve the *Balanced Assignment* problem (Definition 5) for V_d and V_{d+1} using the minimum amount of messages exchanged.

The minimum number of messages needed to connect a child to a parent is a request from the child and an acknowledgement from the parent. Thus the minimum number of messages to connect two bipartite sets is $2 * |C|$. The minimum amount of message guarantees the minimum possible overhead for constructing a tree. Our distributed algorithm, *MHS*, only needs this minimum number of messages. This is made possible using our novel method called *sequential greedy parent selection*.

5.2 Sequential Greedy Parent Selection

For every pair of node sets V_d and V_{d+1} we propose *sequential greedy parent selection* for solving the *Balanced Assignment* problem. *Sequential greedy parent selection* makes the child nodes perform the parent selection sequentially. The order in which the children pick their parent greatly affects the quality of the outcome.

We order the children in ascending order of their number of candidate parents, since some nodes have many choices and some others only one. Nodes with more candidate parents should pick a parent at a later timepoint. This way we avoid forcing a node, with only a few candidate parents, choose between two parents that already have too many children. Making the children select parents in sequential order also has a positive side effect: collisions are reduced *during* the tree construction process.

To implement the above in a distributed environment we implicitly define time-slots. Time-slot i is reserved for children that have i number of candidate parents. For nodes that belong to the same time-slot we add a small random timeout that shifts their parent selection inside their time-slot. We do not need to achieve absolute order or absolute uniform spread inside a given timeslot. Formally, the time-point t, where node v will select one of its candidate parents from set $P \in V_d$ to connect to, is given by

$$t = timeslotsize * (|P| + timeslotsize * rand) \qquad (1)$$

where $rand$ is a random number in the range $(0, 1)$. The value of *timeslotsize* is a network parameter and is not query-specific. Too large values would add to query latency and very small values would not exploit the *sequential parent selection* intuition. *timeslotsize* is set in respect to the time needed to communicate a unit of data between two neighboring nodes ($timeunit$), the size of an acknowledgement message (ack_size) and the density of the network, i.e., average number of neighbors per node (avg_neigh). In particular,

$$timeslotsize = 2 * timeunit * ack\_size * avg\_neigh$$

After defining the order of parent selection comes the actual selection. To get a balanced assignment the nodes select the candidate parent with the least children. They send an *adoption request* to the chosen parent. A node that receives an *adoption request* always accepts it and sends back an *adoption acknowledgement*. The number of children a node has adopted is, thus, equal to the number of *adoptions acknowledgments* this node sent out.

Information about how many children a parent has is constructed using *snooping*. Node $v \in V_{d+1}$ needs to estimate the number of children of its candidate parents. Every time a candidate parent of v transmits an *adoption acknowledgment*, node v *snoops* this message and increases the counter for this candidate parent. Obviously, the nodes that are early in the selection order will not have sufficient information to make a sophisticated selection, but remember that

those nodes have a limited amount of choices. The nodes choosing parent last have almost complete information about their candidate parent and have a larger pool of candidate parents to choose from. Note that using snooping, the process of counting children per candidate parent does not impose any communication overhead.

5.3 Constructing an *MHS* Tree

Everything starts as soon as a node r receives a query Q. Node r creates a tree construction request $tcr(Q, u, d(u))$ containing the query Q, the neighbor node u who forwarded the tree construction request, and the shortest hop distance $d(u)$ from u to node r. Initially $u = r$ and $d(u) = 0$.

Every node v stores its distance to the root $d(v)$ and the set of candidate parents in a heap we call *Candidate Parent Heap* (CPH). This heap is populated during the dissemination of the tree constructions requests, where node v can tell its neighbors and what neighbor is 1 hop closer to the root using the received tcrs. The top element of the heap is the parent with the least number of adoptions. Note that the heap does not hold duplicates. We describe the *MHS* algorithm in an event-driven fashion: events followed by actions.

Algorithm 3 . *MHS* - query dissemination

Event: node i receives a tree construction request $tcr(Q, u, d(u))$
Actions:
1: **if** $d(i) = \emptyset$ **then**
2:   $d(i) \leftarrow d(u) + 1$
3:   $CPH \leftarrow u$
4:   broadcast($tcr(Q, i, d(i))$)
5: **else if** $d(u) \leq d(i) - 2$ **then**
6:   $d(i) \leftarrow d(u) + 1$
7:   $CPH \leftarrow \emptyset$
8:   $CPH \leftarrow u$
9:   broadcast($tcr(Q, i, d(i))$)
10: **else if** $d(n) = d(i) - 1$ **then**
11:   $CPH \leftarrow u$
12: **end if**

With every tree construction request $tcr(Q, u, d(u))$ received, node i checks whether it received the same request before (Step 1 Algorithm 3). If not, it sets its depth $d(i)$, adds the node u to its candidate parent heap CPH and forwards the updated tree construction request (Steps 2-4). Otherwise, node i checks whether u is closer to the root than the nodes already in the CPH (Step 5). In this case, it clears the CPH and adds the new node u. It also updates its own distance to the root $d(i)$ and re-forwards the tree construction request with the updated information (Steps 6-9). Otherwise, if the tree construction request comes from a different parent node, it adds it to the candidate parent heap CPH (Step 11).

Algorithm 4 . *MHS* - request adoption

Event: node i receives no more tree construction messages
Actions:
1: $p \leftarrow CPH.pop()$
2: $t \leftarrow calculateAdoptionTimeout()$
3: broadcast_after_timeout($t, ar(i, p)$)

When a node i stops receiving tree construction requests (Algorithm 4), it pops the first entry p of the candidate parent heap CPH, which is the parent with the least number of adoptions (Step 1) so far. Node i then calculates the timeout t to wait before sending an adoption request $ar(i, p)$ to p (Step 2-3). In section 5.2 and Equa-
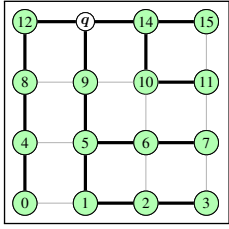
Figure 4: An *MHS* solution.
$COV_{MHS} = 3.01$
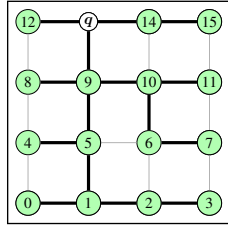
Figure 5: An *ETC* solution.
$COV_{ETC} = 4.67$

tion 1 we describe how the timeout t is calculated. A candidate parent p that receives an adoption request $ar(u, p)$ always sends back an adoption acknowledgement to the requesting node u.

When a node i receives an adoption acknowledgement $aack(p, u)$ and i is the intended receiver ($i = u$), it sets its parent to p. Otherwise, if the acknowledgement comes from one of its candidate parents, it increments its number of adoptions in the CPH.

For example consider the 4x4 grid network in Figure 4 (grey and black lines). For each depth d the node set V_d is $V_1 = \{9, 12, 14\}$, $V_2 = \{5, 8, 10, 15\}$, $V_3 = \{1, 4, 6, 11\}$, $V_4 = \{0, 2, 7\}$ and $V_5 = \{3\}$. The candidate parent list P_v of each node $v \in V$ is $P_0 = \{1, 4\}$, $P_1 = \{5\}$, $P_2 = \{1, 6\}$, $P_3 = \{2, 7\}$, $P_4 = \{5, 8\}$, $P_5 = \{9\}$, $P_6 = \{5, 10\}$, $P_7 = \{6, 11\}$, $P_8 = \{9, 12\}$, $P_{10} = \{9, 11\}$, $P_{11} = \{10, 15\}$, $P_{15} = \{14\}$. According to the size of their candidate parent list every node will have a selection time-point in the time-slot $timeslot\_1 = \{1, 5, 9, 12, 14\}$, $timeslot\_2 = \{0, 2, 3, 4, 6, 7, 8, 10, 11\}$. Notice, nodes of depth one (9, 12, 14) do not take part in the parent selection process. The random parameter inside Equation 1 could cause any sequence inside a time-slot. Assume that the final selection sequence inside $timeslot\_2$ is 11, 6, 0, 2, 3, 4, 7, 8, 10.

The solution T given by *MHS* is shown in Figure 4. The deviation of the node degrees for T is $COV_{MHS} = 3.01$, whereas for *COPT* it is $COV_{COPT} = 2.43$ (Figure 3). For *COPT* node 6 connects to node 10, whereas for *MHS* it connects to 5. This is caused by the randomness of the intra-slot sequence of parent selection (Equation 1). Had node 6 been randomly set to select and connect to a candidate parent first, it would have connected to 10. Then 11 would be forced to connect to 5 giving an even better solution. Node degrees for *MHS* are $degrees_{MHS} = \{1, 1, 2, 1, 2, 1, 0, 1, 1, 1, 0, 0, 1, 0\}$.

6. EXPERIMENTAL EVALUATION

We show with experiments that our algorithm *MHS* improves upon previous work for creating a query routing tree with minimum hot-spots. We run experiments for various network layouts and identify the cases where our algorithm prevails. Our efficiency metrics include the balance of the tree, the maximum energy spent per node and the total energy spent in the network. We compare against our centralized algorithm *COPT*, and a previously proposed distributed algorithm summarized in the next paragraph.

6.1 Energy-driven Tree Construction

Andreou et al [1] propose *ETC*, which is a 2 step process where an arbitrary tree is created first and then it is reorganized into a near-balanced tree. In the first step (*Discovery Phase*) we want to get the the total number of sensors n and the height of the routing tree h at the sink q. When variables n and h are received, the sink calculates a uniform *Optimal Branching Factor* (β).

To get h and n at the sink an arbitrary query routing tree T' is

constructed using the First-Heard-From (FHF) approach. During the construction, each node v_i also records its level l_i in the tree. A node v_i also maintains a child node list children and an candidate parent list. When the initial tree is done, the sink queries the network for the total number of sensors n and the height of the tree $h = l_{max}$. They also send their candidate parent lists to their current parent under T'. The nodes reply over the initial tree back to the sink and when variables n and h are received, the sink calculates the optimal branching factor $\beta = \sqrt[h]{n}$ for the whole tree.

In the second step (*Balancing Phase*) the sink disseminates the β value to the n nodes using the initial tree T'. Upon receiving β, each sensor conducts a number of local rearrangements to its local topology in order to create a near-balanced topology. In particular, when a node v_i receives β it tries to reassign as many of its children needed such that $degree_{v_i} \leq \beta$. It uses the candidate parent lists from its children received during the *Discovery Phase* to choose a new parent $newParent$ for its children, making sure not all are reassigned to the same new parent. Each child v_j that receives a parent reassignment tries to connect to the newly assigned parent $newParent$. Notice that if a node $newParent$ can not accommodate the connect request from a child v_j, then v_j has to report back to the current parent v_i and v_i has to pick a new parent from the candidate parent list. This procedure is repeated until completion or until the alternative parents are exhausted, where the a different child is chosen for parent reassignment. If no child can be reassigned then they all stay connected to their current parent v_i.

As an example consider 4x4 grid network. *ETC* first constructs an arbitrary initial tree T' using the First Heard From technique. In our example it is the tree shown in Figure 2. Then the querying node q collects information to compute β which in this case is $\beta = 3.06$. The node degrees for T' are $degrees_{T'} = \{0, 3, 1, 0, 2, 2, 0, 0, 2, 1, 0, 0, 1, 0\}$ since all degrees are less than β no node reassigns any of its children and the resulting tree T for the *ETC* algorithm (Figure 5) is the same as T' giving $COV_{ETC} = 4.67$.

6.2 Experimental Setup

We run all the experiments on an Intel Core2 Duo 2.5Ghz processor with 3GB RAM running Ubuntu Linux. To simulate a WSN and to implement our algorithms we used the SensorSim wireless sensor network simulation framework together with the compositional discrete event simulator J-Sim [11].
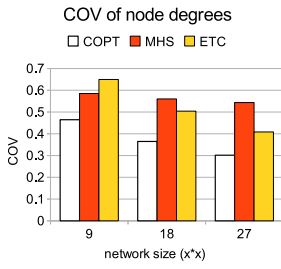
We use three layouts: grid, grid with diagonal and random. For the random layout we randomly place n nodes in a 1000x1000 area, set their communication range to $2 * 1000/\sqrt{n}$, and make sure the network is connected, resulting in a sparse connected graphs. This emulates a real world WSN application where nodes are usually placed randomly and obstacles, failures and transmission range variations prevent any predefined structured layout.

We present the performance of each algorithm in Section 6.3, and then the overhead that is needed to construct this tree, i.e. the energy consumed in 6.4. Note that the x-axis of every figure represents the square root of the number of nodes in the network. For $x = 9$ the network size is $n = 81$ and for $x = 27$ $n = 729$.

6.3 Quality of Query Routing Tree

As described theoretically in Section 3 and experimentally in [1], we save on energy when we construct a balanced query routing tree. The performance of an algorithms is defined by how balanced the resulting query routing tree is. The balance of the tree is given by the coefficient of variation *COV* of the degrees of the nodes (Section 3). Smaller *COV* values signify a better balanced tree.
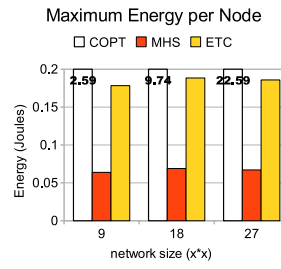
In the structured layouts (Figures 7a,b and 6a,b) *ETC* performs better with increasing network size, whereas *MHS* the performance
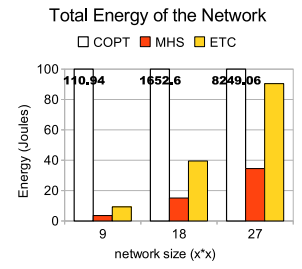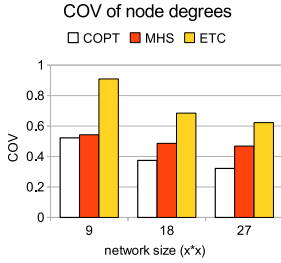
(a) Grid network — COV of node degrees
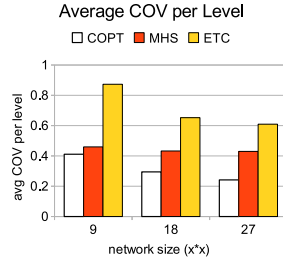

(a) Grid network — Average COV per Level

(a) Grid network — Maximum Energy per Node
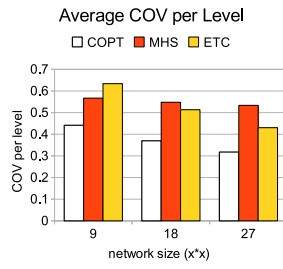
(a) Grid network — Total Energy of the Network

(b) Grid with diagonals network — COV of node degrees

(b) Grid with diagonals network — Average COV per Level

(b) Grid with diagonals network — Maximum Energy per Node
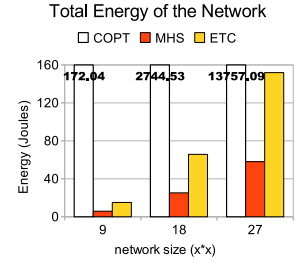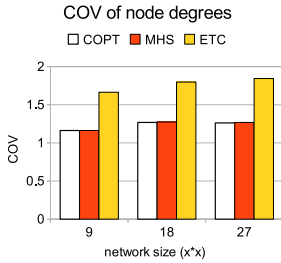
(b) Grid with diagonals network — Total Energy of the Network

(c) Random network — COV of node degrees

(c) Random network — Average COV per Level

(c) Random network — Maximum Energy per Node
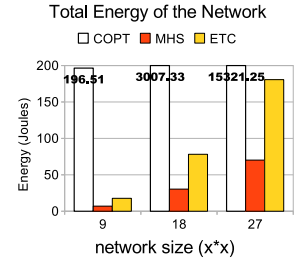
(c) Random network — Total Energy of the Network

Figure 6: Balance of Tree

Figure 7: Balance of Tree

Figure 8: Maximum energy

Figure 9: Total energy

stays almost constant. In structured layouts each node has approximately the same amount of possible parents. When the number of candidate parents varies greatly (e.g. random layout), *ETC* performs poorly deteriorating with network size (Figures 6(c) and 7(c)), while *MHS* is almost optimal. Experiments showing how much energy we save during query execution can be found in [1].

6.4 Tree Construction Overhead

To simulate the energy, we set: 0.66 Watt power consumption for transmission, 0.395 Watt power consumption for reception, and 19.2 kbps data rate. We record the maximum energy for each node (Figure 8) and the total energy consumed in the network (Figure 9).

The maximum energy per node (Fig. 8) stays almost constant with network size. This is attributed to the distributed nature of *MHS* and *ETC*. The total energy (Fig. 9) scales nicely with network size. Note that the network size is increased exponentially, thus the total energy scales linear to the network size for *MHS* and *ETC*.

For comparison, we also show the energy needed to construct our query routing tree using our centralized optimal *COPT* algorithm. This corresponds to collecting the needed information centrally at the sink, running *COPT*, disseminating the solution back to the network and letting the nodes connect to their predefined parent. The high cost of *COPT* makes it unattractive. Note that the values for the *COPT* algorithm are shown in bold text on top of the white bars, since the bars did not fit.

7. CONCLUSIONS

We present a novel distributed algorithm (*MHS*) that constructs a query routing tree that minimizes collisions during query execution. It was shown in previous work that minimizing collisions during query execution saves significant amount of energy [1]. In the same paper it is shown that balancing the node degrees of a query routing tree significantly reduces collisions during query execution.

We address the inefficiencies of the previously proposed algorithm and propose a simpler, purely distributed, parameter-free, cheaper and more efficient algorithm. Our resulting query trees are optimally balanced, guarantee minimum collisions and minimum latency for query execution and allow for opportunistic in-network processing. *MHS* poses the minimum possible communication overhead to the network and is parameter-free as opposed to previously proposed algorithms. Our proposed algorithm can be used for acquiring data from the nodes of any distributed systems where the main objective is to minimize the communication cost.

8. REFERENCES

[1] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras. Etc: Energy-driven tree construction in wireless sensor networks. In *Mobile Data Management*, pages 513–518, 2009.

[2] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, Philadelphia, 2009.

[3] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12:12–21, 2008.

[4] G. J. Chang and P.-H. Ho. The β-assignment problem in general graphs. *Comput. Oper. Res.*, 24(8):757–765, 1997.

[5] R. S. Chang and R. C. T. Lee. On a scheduling problem where a job can be executed only by a limited number of processors. *Comput. Oper. Res.*, 15(5):471–478, 1988.

[6] Y. L. Chen and Y. H. Chin. Scheduling unit-time jobs on processors with different capabilities. *Comput. Oper. Res.*, 16(5):409–417, 1989.

[7] H. Dai and R. Han. A node-centric load balancing algorithm for wireless sensor networks. In *Global Telecommunications Conference*, volume 1, pages 548– 552, December 2003.

[8] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. pages 376–380, 1997.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS Hawaii International Conference on System Sciences*, Washington, DC, 2000.

[10] J. C. G. Jacob. De investigando ordine systematis aequationum differentialum vulgarium cujuscunque. 5:193–216, 1890.

[11] J. Kacer. Discrete event simulations with j-sim. In *Intermediate Representation Engineering for virtual machines*, pages 13–18, Dublin, Ireland, 2002.

[12] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: a reliable bulk transport protocol for multihop wireless networks. In *SenSys'07*, pages 351–365, Sydney, Australia, 2007.

[13] S. Kim, S. Pakzad, D. E. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *IPSN*, pages 254–263, Cambridge, Massachusetts, USA, 2007.

[14] J. F. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[15] J. Li, A. Deshpande, and S. Khuller. On computing compression trees for data collection in sensor networks. *CoRR*, abs/0907.5442, 2009.

[16] S. Lindsey, C. S. Raghavendra, and K. M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Parallel Distrib. Syst.*, 13(9):924–935, 2002.

[17] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *In OSDI*, 2002.

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[19] R. Murty, A. Gosain, M. Tierney, A. Brody, A. Fahad, J. Bers, and M. Welsh. Citysense: A vision for an urban-scale wireless networking testbed. In *International Conference on Technologies for Homeland Security, Waltham, MA*, 2008.

[20] N. A. Pantazis, D. J. Vergados, D. D. Vergados, and C. Douligeris. Energy efficiency in wireless sensor networks using sleep mode tdma scheduling. *Ad Hoc Networks*, 7(2):322–343, 2009.

[21] S. Singh and C. S. Raghavendra. Pamas—power aware multi-access protocol with signalling for ad hoc networks. *SIGCOMM Comput. Commun. Rev.*, 28(3):5–26, 1998.

[22] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. M. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications ACM*, 47(6):34–40, 2004.

[23] V. Tsaoussidis and H. G. Badr. Tcp-probing: Towards an error control schema with energy and throughput performance gains. In *ICNP*, pages 12–21, 2000.

[24] Voltree-Power-Incorporated. http://www.voltreepower.com/.

[25] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *Mobile computing and networking*, pages 70–84, New York, NY, USA, 2001.

[26] T. Yan, Y. Bi, L. Sun, and H. Zhu. Probability based dynamic load-balancing tree algorithm for wireless sensor networks. In *Int. Conf. Networking and Mobile Computing*, 2005.

[27] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR, Conf. on Innovative Data Systems Research*, 2003.

[28] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, 2004.

[29] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, and G. Samaras. Mint views: Materialized in-network top-k views in sensor networks. In *MDM*, pages 182–189, 2007.

[30] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar. Microhash: An efficient index structure for flash-based sensor devices. In *FAST*, pages 31–44, 2005.

[31] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *SenSys*, pages 227–238, Baltimore, MD, USA, 2004.

[32] Q. Zhang, Z. Xie, W. Sun, and B. Shi. Tree structure based data gathering for maximum lifetime in wireless sensor networks. In *APWeb, Asia-Pacific Web Conference*, pages 513–522, Shanghai, China, 2005. Springer.