On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks: NP-Completeness and Approximation Algorithm

Yan Wu, Sonia Fahmy, Ness B. Shroff
E-mail: {wu26, fahmy}@cs.purdue.edu, shroff@ece.osu.edu

***Abstract*—Energy efficiency is critical for wireless sensor networks. The data gathering process must be carefully designed to conserve energy and extend the network lifetime. For applications where each sensor continuously monitors the environment and periodically reports to a base station, a tree-based topology is often used to collect data from sensor nodes. In this work, we study the construction of a data gathering tree to maximize the network lifetime, which is defined as the time until the first node depletes its energy. The problem is shown to be NP-complete. We design an algorithm which starts from an arbitrary tree and iteratively reduces the load on bottleneck nodes (nodes likely to soon deplete their energy due to high degree or low remaining energy). We show that the algorithm terminates in polynomial time and is *provably* near optimal.** [1]

I. Introduction

Recent advances in micro-electronic fabrication have allowed the integration of sensing, processing, and wireless communication capabilities into low-cost and low-energy wireless sensors [1], [2]. An important class of wireless sensor network applications is the class of continuous monitoring applications. These applications employ a large number of sensor nodes for continuous sensing and data gathering. Each sensor *periodically* produces a small amount of data and reports to a base station. This application class includes many typical sensor network applications such as habitat monitoring [3] and civil structure maintenance [4].

The basic operation in such applications is *data gathering*, i.e., to collect sensing data from the sensor nodes and transmit to a base station for processing. In this process, data aggregation can be used to fuse data from different sensors to eliminate redundant transmissions. The critical issue in data gathering is conserving sensor energy and maximizing sensor lifetime. For example, in a sensor network for seismic monitoring or radiation level control in a nuclear plant, the lifetime of each sensor significantly impacts the quality of surveillance.

For continuous monitoring applications, a tree-based topology is often used to gather and aggregate sensing data. The tree is constructed after initial node deployment, and is rebuilt upon significant topology changes. We study the problem of tree construction for *maximizing the network lifetime*. Network lifetime is defined as the time until the first node depletes its energy. We prove that this problem is NP-complete, and too computationally expensive to solve exactly. By exploiting the unique structure of the problem, we obtain an algorithm which starts from an arbitrary tree and iteratively reduces the load on *bottleneck nodes*, i.e., nodes likely to soon deplete their energy due to either high degree or low remaining energy. We show that the algorithm terminates in polynomial time and is *provably* "near optimal" (i.e., close to optimal).

The remainder of this paper is organized as follows. Section II reviews related work on data gathering and aggregation. Section III describes the system model and formulates the problem. Section IV gives our tree construction algorithm and discusses implementation issues. Simulation results are presented in Section V, and Section VI concludes the paper.

II. Related Work

The problem of efficient data gathering and aggregation in a sensor network has been extensively investigated in the literature. Krishnamachari *et. al.* [5] argue that a data-centric approach is preferable to address-centric approaches under the many-to-one communication pattern (multiple sensor nodes report their data to a single base station). In directed diffusion [6], a network of nodes coordinate to perform distributed sensing tasks. This achieves significant energy savings when intermediate nodes aggregate their responses to queries. Kalpakis *et. al.* [7] model data gathering as a network flow problem, and derive an efficient schedule to extend system lifetime. Hou *et. al.* [8] study rate allocation in sensor networks under a lifetime requirement.

For continuous monitoring applications with a periodic traffic pattern, a *tree-based topology* is often adopted because of its simplicity [9]–[11]. Compared to an arbitrary network topology, a tree-based topology saves the cost of maintaining a routing table at each node, which can be computationally expensive for sensor nodes with limited resources. A number of studies have investigated tree construction for data gathering [12]–[15]. Goel *et. al.* [12] study the problem of constructing efficient trees to send aggregated information to a sink. The goal is to reduce the total amount of data transmitted. They propose a randomized algorithm that approximates the optimal tree. Enachescu *et. al.* [13] consider a grid of sensors,

[1] Yan Wu and Sonia Fahmy are with the Department of Computer Science, Purdue University. Ness B. Shroff is with the departments of ECE and CSE, The Ohio State University. This research has been sponsored in part by NSF grants 0238294 and 0207728, an ARO MURI grant W911NF-07-10376 (SA08-03), an Indiana 21st century grant, and a Tellabs foundation fellowship.

and propose a simple randomized tree construction scheme that achieves a constant factor approximation to the optimal tree. Thepvilojanapong *et. al.* [14] present a data gathering protocol that efficiently collects data while maintaining constant local state, and making only local decisions. Khan and Pandurangan [16] propose a distributed algorithm that constructs an approximate minimum spanning tree (MST) in arbitrary networks. In contrast to these approaches, we are motivated by applications with strict *coverage* requirements. For these applications, minimizing the total energy consumption may be insufficient, since some nodes may deplete their energy faster than others and cause loss of coverage.

III. System Model and Problem Definition

Consider a sensor network with N nodes $(v_1, v_2, \dots v_N)$ and one base station v_0. The nodes monitor the environment and periodically report to the base station. Time is divided into epochs, and each sensor node generates one B-bit message per epoch. The messages from all the sensors need to be collected at each epoch and sent to the base station for processing. The nodes are powered by batteries and each sensor v_i has a battery with finite, non-replenishable energy $E(i)$. The energy values $E(i)$ of different sensor nodes can be different, for reasons such as heterogeneous sensor nodes, non-uniform node energy consumption, or redeployment of nodes. As with many practical systems [17], [18], the base station is connected to an unlimited power supply, hence $E(0) = \infty$. The amount of energy required to send/receive one bit of data is α_t/α_r.

A. Assumptions

We make the following assumptions about our system:

(1) Connectivity: We assume that the sensor nodes and the base station form a connected graph, i.e., there is a path from any node to any other node in the graph. This can be achieved by setting the transmission power level to be above the critical threshold [19]–[21], which ensures that the network is connected with probability one as the number of nodes in the network goes to infinity. For simplicity, we do not consider dynamic adjustment of the transmission power levels, and assume that all nodes transmit at the same fixed power level.

(2) Energy expenditure: Measurements show that among all the sensor node components, the radio consumes the most significant amount of energy. In Section IV, we will show that the computational complexity of our scheduling algorithm is very low. Therefore, in this work, we only account for energy consumption of the radio.

(3) Data aggregation: We adopt a simple data aggregation model as in previous work [5]–[7], [22]. We assume that an intermediate sensor can aggregate multiple incoming B-bit messages, together with its own message, into a single outgoing message of size B bits. This models applications where we want updates of the type min, max, mean, and sum (e.g., event counts).

(4) Orthogonal transmissions and sleep/wake scheduling: Measurements show that for short range radio communications in sensor networks, a significant amount of energy is wasted due to overhearing, collision, and idle listening. To conserve energy, we assume that the system adopts a channel allocation scheme such that transmissions do not interfere with each other. Such orthogonality can be achieved via joint frequency/code allocation and time slot assignment. In [23], we have given an example solution for a cluster hierarchy topology. Similar arguments can be made for the tree topology considered in this paper. Further, because the traffic is periodic, we assume that a sensor node puts the radio into sleep mode during idle times, and turns it on only during message transmission/reception.

B. The maximum-lifetime tree problem

We consider a connected network G of N nodes. Each node monitors the environment and periodically generates a small amount of data. To gather the data from the sensor nodes, we need to construct a tree-based topology after node (re)deployment. For critical applications like seismic monitoring or radiation level control in a nuclear plant, we need to both maintain complete coverage and save redeployment cost. This requires that all the nodes remain up for as long as possible. To this end, we formulate the following optimization problem.

For any network G, there exist multiple possible data gathering trees. For example, Fig. 1 shows two data gathering trees for the same network. Each tree T has a lifetime $L(T)$, where $L(T)$ is defined as the time until the first node depletes its energy[2]. Our goal is to find the tree that maximizes the network lifetime:

(A) $\max L(T)$
such that $T \in A(G)$,

where $A(G)$ is the set of data gathering trees for G.

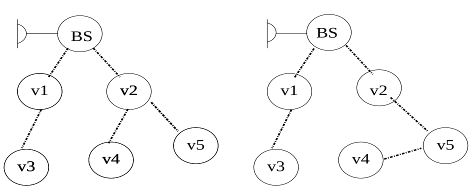


Fig. 1. Two data gathering trees for the same network

To obtain an explicit form of the above problem, we must characterize the energy dissipation for each sensor node in a given tree T. Let $C(T, i)$ be the number of children for node v_i in T, and $D(T, i)$ be the degree of node v_i in T. During an epoch, node v_i needs to:

- receive one B-bit message from each child, and
- aggregate the received messages with its own message into a single B-bit message, and transmit this aggregate message to its parent.

[2]Here, we assume that we will lose the corresponding coverage if a node dies, i.e., there are no redundant nodes. If the network has redundancy, we can consider all nodes covering the same area (e.g., nodes near the same bird nest) as a single node whose initial energy equals the sum of energy of all the relevant nodes, and the following results still apply.

Hence, in each epoch, the energy consumption of node v_i is $\alpha_r BC(T,i) + \alpha_t B$, and its lifetime (in epochs) is

$$L(T,i) = \frac{E(i)}{\alpha_r BC(T,i) + \alpha_t B}.$$

The network lifetime is the time until the first node dies, i.e.,

$$L(T) = \min_{i=1\ldots N} L(T,i) = \min_{i=1\ldots N} \frac{E(i)}{\alpha_r BC(T,i) + \alpha_t B} \quad (1)$$

Because the base station v_0 is connected to a power supply, its lifetime is infinite and can also be written as

$$L(T,0) = \frac{E(0)}{\alpha_r BC(T,0) + \alpha_t B}.$$

So we can include v_0 in Equation (1) as

$$L(T) = \min_{i=0\ldots N} \frac{E(i)}{\alpha_r BC(T,i) + \alpha_t B}. \quad (2)$$

Since T is a tree, we have

$$C(T,i) = D(T,i) - 1 \quad (3)$$

for all nodes except the base station. Combining Equation (2) with Equation (3) and extracting the constant $\alpha_r B$ from the denominator, we can write Problem (A) as

(B) $\max \min_{i=0\ldots N} \dfrac{E(i)}{D(T,i)+c}$
such that $T \in A(G)$.

where $c = \frac{\alpha_t}{\alpha_r} - 1$ is a non-negative constant because the transmission power is larger than the reception power.

In Problem (B), the goal is to maximize the minimum of $\frac{E(i)}{D(T,i)+c}, i = 0\ldots N$. This is a load balancing problem. Intuitively, for this kind of problem, a good solution would be that nodes with larger capabilities (large $E(i)$) should hold more responsibilities by serving more child nodes (large $D(T,i)$). In other words, we want to construct a tree such that the degree of a node is "proportional" to its energy.

IV. Solution and Implementation

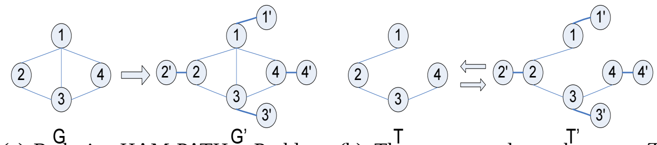
The difficulty in solving Problem (B) is illustrated by the following proposition, which shows that it is NP-complete.

Proposition 1: Problem (B) is NP-complete.

Proof: Clearly, the problem is in NP, since we can verify in polynomial time if a candidate solution is a tree and achieves the lifetime constraint.

To show it is NP-hard, we reduce from the Hamiltonian path problem, which is known to be NP-complete [24]. The reduction algorithm takes as input an instance of the Hamiltonian path problem. Given a graph G, it constructs an auxiliary graph G' in the following manner. For each vertex i in G, add a vertex i', then draw an edge between i and i' (Fig. 2).

Then in G', set the energy as follows: $E(1') = \infty$, $E(1) = E(2) = \ldots = E(N) = E(2') = \ldots = E(N') = 1$. In this manner, G' becomes an instance of Problem (B). The construction of G' and setting the energy values can be easily done in polynomial time. To complete the proof, we show that G has a Hamiltonian path if and only if G' has a tree whose lifetime is greater than or equal to $\frac{1}{3+c}$.



(a) Reducing HAM-PATH to Problem (B)
(b) The correspondence between T and T'

Fig. 2. Problem (B) is NP-complete

Suppose G has a Hamiltonian path T. Construct T' in G' by adding vertices $1', 2', \ldots N'$ and edges $(1,1'), \ldots (N,N')$ as depicted in Fig. 2. Clearly, T' is connected and acyclic, thus T' is a tree. Further, since T is a Hamiltonian path, the maximal degree in T is no larger than 2. But T' is constructed by adding one edge to each vertex in T, so the maximal degree in T' is no larger than 3. Therefore, the lifetime of T' is

$$L(T') = \min \frac{E(i)}{D(T',i)+c} \geq \frac{1}{3+c}.$$

Similarly, if G' has a spanning tree T' with $L(T') \geq \frac{1}{3+c}$, then we have $D(T',i) \leq 3, i = 1\ldots N$. Otherwise, if $D(T',j) > 3$ for some $j, 1 \leq j \leq N$, then

$$L(T') \leq L(T',j) < \frac{E(j)}{D(T',j)+c} \leq \frac{1}{4+c},$$

which is contradictory.

We further observe that in T', vertices $1', 2' \ldots N'$ are all leaves. We construct T by removing $1', 2' \ldots N'$ and the corresponding edges $(1,1'), \ldots (N,N')$ from T'. T is still a tree and it spans G. Since in T' we have $D(T',i) \leq 3, i = 1\ldots N$, it is easy to see that in T, $D(T,i) \leq 2, i = 1\ldots N$. Thus, T is a spanning tree with maximal degree no larger than 2, which is exactly a Hamiltonian path. ■

Since Problem (B) is NP-complete, we next try to find an approximate solution. However, in the current form of Problem (B), the variable $D(T,i)$ is in the denominator and is hard to tune. Hence, we transform the problem into an equivalent form. Let $r(T,i) = \frac{D(T,i)+c}{E(i)}$, i.e., $r(T,i)$ is the *inverse lifetime* for node i in tree T. Correspondingly, define the inverse lifetime of a tree T as $r(T) = \max_{i=0\ldots N} r(T,i)$. We write Problem (B) as:

(C) $\min_{T\in A(G)} \max_{i=0\ldots N} r(T,i)$,

i.e., maximizing the minimal lifetime is equivalent to minimizing the maximal *inverse lifetime*. Note that in $r(T,i)$, the variable $D(T,i)$ is in the numerator, while the denominator is the constant $E(i)$ which does not change during the operation of the algorithm. Note that Problem (C) is an equivalent formulation to Problem (A). In the remainder of the paper, we will study Problem (C), and we refer to the minimum maximal inverse lifetime as r^*.

A. Two building blocks of the algorithm

Considerable work has been done on the Minimum Degree Spanning Tree (MDST) problem, i.e., finding a spanning tree whose maximal degree is the minimum among all spanning trees. Problem (C) can be viewed as a generalization of the MDST problem, where the capacity of a node ($E(i)$) needs to be considered in the tree construction. Frer and Raghavachari [25] studied the MDST problem and proposed an approximation algorithm. Our solution utilizes hints from their approach. Essentially, our solution starts from an arbitrary tree, and iteratively makes "improvements" by reducing the degree of the *bottleneck nodes*, i.e., nodes with a large inverse lifetime (or short lifetime), at each step. Upon termination, we will bound r^* from below, and show that the resulting tree has inverse lifetime close to the lower bound. In this section, we describe two building blocks of our algorithm: (1) the notion of "an improvement," and (2) the technique to bound r^* from below.

1) The notion of an improvement: Given a tree T and an arbitrary $\epsilon > 0$, let $k = \lceil \frac{r(T)}{\epsilon} \rceil$, i.e., $(k-1)\epsilon < r(T) \leq k\epsilon$. We classify the nodes into three disjoint subsets:

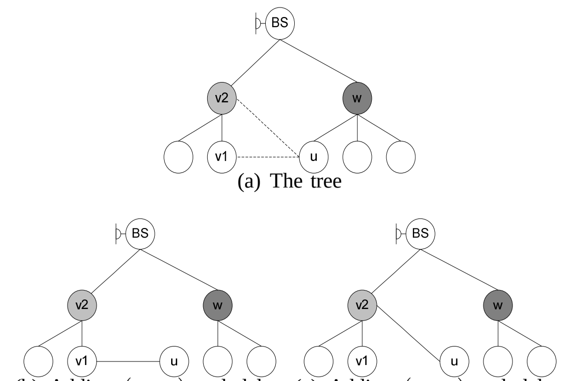
- $V_1 = \{v_i : (k-1)\epsilon < r(T,i) \leq k\epsilon\}$, i.e., V_1 contains the bottleneck nodes that are our "target" in each step.
- $V_2 = \{v_i : (k-1)\epsilon - \frac{1}{E(i)} < r(T,i) \leq (k-1)\epsilon\}$. These nodes are "close" to becoming bottleneck nodes in the sense that they will become bottlenecks if their degree increases by one. We should not increase the degree of these nodes in the algorithm.
- $V_3 = V - V_1 - V_2$, i.e., all the remaining nodes. These nodes are "safe" nodes as they will not become bottlenecks even if the degree is increased by one.

Consider an edge (u,v) that is not in T. A unique cycle C will be generated when we add (u,v) to T. If there is a bottleneck node $w \in V_1$ in C, while both u and v are in V_3 ("safe nodes"), then we can add (u,v) to T and delete one of the edges in C incident on w. This will reduce the degree of the bottleneck node w by one. We call this modification an *improvement*, and we say that w *benefits* from (u,v). We will use this method as a building block to increase the network lifetime in our algorithm.

In the above example, if either u or v or both are in V_2, then the above modification will turn u or v or both into bottleneck node(s). Thus, while reducing the degree for one bottleneck node, we produce additional bottleneck(s). This is undesirable and we say that w is *blocked* from (u,v) by u (or v or both). A node is *blocking* if it is in V_2.

We illustrate the notion of improvement using an example. Fig. 3(a) shows a tree, where solid lines correspond to edges in the tree, and dotted lines correspond to edges not in the tree. For simplicity, we set the initial energy for all nodes in this example to be 1, so $r(T,i) = \frac{D(T,i)+c}{E(i)} = D(T,i) + c$ for all nodes except the base station.

Let $\epsilon = 1$. According to the above definition, w (the dark grey node) is a bottleneck node, v_2 (the light grey node) is a blocking node and all other nodes are safe. We can add (u, v_1) and delete (w, u). This is an improvement as it reduces the degree of the bottleneck node w. In contrast, adding (u, v_2) and deleting (w, u) do not prolong the network lifetime, because doing so produces another bottleneck node v_2 while reducing the degree of w.



(a) The tree

(b) Adding (u, v_1) and deleting (w, u) is an improvement

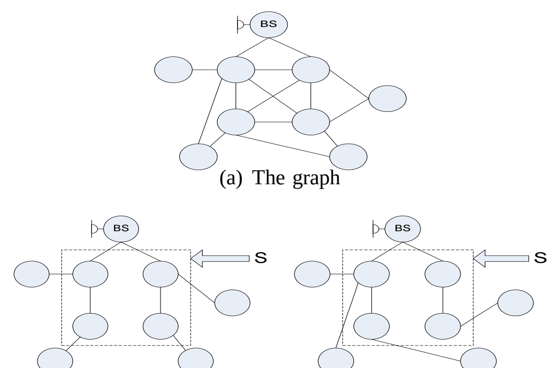
(c) Adding (u, v_2) and deleting (w, u) is not an improvement

Fig. 3. The notion of improvement

2) Method for bounding r^* from below: We note that given a tree T, if we can find a subset of nodes S that satisfies the following property:

the components produced by removing S from T are also disconnected in G,

then in any spanning tree X, we can connect these components only through S. This is because there is no edge between these components in G. Hence, in X any edge external to these components must be incident on some vertex in S (see Fig. 4).



(a) The graph

(b) T and S

(c) An arbitrary tree X

Fig. 4. Bounding r^* from below

Now let us assume that we have already found such an S, and study the components generated by removing S from T. We can count $\sum_{i \in S} D(T,i)$ edges incident on S. Since T is a tree, at most $|S|-1$ of these counted edges are within S and counted twice. Hence, the number of generated components is:

$$O \geq \sum_{i \in S} D(T,i) - (|S|-1) + 1 - |S|.$$

In an arbitrary spanning tree X, we need to connect these O components and the vertices in S. This requires

$$O + |S| - 1 \geq \sum_{i \in S} D(T,i) - (|S|-1) \tag{4}$$

edges. According to the discussion above, all these edges must be incident on some vertex in S. Thus, by Equation (4), $\sum_{i\in S} D(X,i) \geq \sum_{i\in S} D(T,i) - |S| + 1$ and the inverse lifetime for X is

$$\begin{aligned} r(X) &= \max_{i=0\ldots N} r(X,i) \geq \max_{i\in S} r(X,i) = \max_{i\in S} \frac{D(X,i)+c}{E(i)} \\ &\geq \frac{\sum_{i\in S}(D(X,i)+c)}{\sum_{i\in S} E(i)} \geq \frac{\sum_{i\in S}(D(T,i)+c) - |S| + 1}{\sum_{i\in S} E(i)} \\ &\geq \min_{i\in S} r(T,i) - \frac{|S|-1}{\sum_{i\in S} E(i)}. \end{aligned} \quad (5)$$

Since X is an arbitrary spanning tree, Equation (5) holds for any spanning tree including the optimal one. Hence, Equation (5) gives a lower bound for the minimum maximal inverse lifetime r^*, which is equivalent to an upper-bound for the maximum minimal lifetime. Further, we observe that if (T,S) is chosen such that

$$\min_{i\in S} r(T,i) \approx r(T),$$

i.e., $r(T,i)$ for all $i \in S$ are close to $r(T)$, then Equation (5) implies that T is a good approximation to the optimal tree. Specifically, we have the following lemma.

Lemma 1: For a tree T, if there is a subset S such that (1) the components produced by removing S from T are also disconnected in G, and (2) S consists of nodes exclusively from V_1 and V_2, then $r(T) \leq r^* + \frac{2}{E_m} + \epsilon$, where $E_m = \min_{i=0\ldots N} E(i)$.

Proof: Since S consists of nodes exclusively from V_1 and V_2, we have $r(T,i) > (k-1)\epsilon - \frac{1}{E(i)}, \forall i \in S$, but $(k-1)\epsilon < r(T) \leq k\epsilon$, thus

$$r(T,i) > r(T) - \epsilon - \frac{1}{E(i)} \geq r(T) - \epsilon - \frac{1}{E_m}, \forall i \in S. \quad (6)$$

Combined with Equation (5), for any tree X, we have

$$\begin{aligned} r(X) &\geq \min_{i\in S} r(T,i) - \frac{|S|-1}{\sum_{i\in S} E(i)} \\ &> r(T) - \epsilon - \frac{1}{E_m} - \frac{|S|-1}{\sum_{i\in S} E(i)} \\ &> r(T) - \epsilon - \frac{2}{E_m}. \end{aligned}$$

Since X is arbitrary, this holds for any tree. Hence, $r^* > r(T) - \epsilon - \frac{2}{E_m}$. ■

B. The approximation algorithm

The approximation algorithm starts from an arbitrary tree, and iteratively makes improvements as described in Section IV-A1, by reducing the degree of the bottleneck nodes (V_1) in each iteration. Upon termination, we will show that the resulting tree includes S, which consists of nodes exclusively from V_1 and V_2. Hence by Lemma 1, the resulting tree is a good approximation to the optimal tree.

We first describe the operations in a single iteration of the algorithm.

1) A single iteration: Given a tree T, we remove the nodes in V_1 and V_2, which will generate a forest with several components. If there are no edges between these components in G, we terminate the algorithm. In this situation, we have found an $S(= V_1 + V_2)$ which consists of nodes exclusively from V_1 and V_2. By Lemma 1, T is already a good approximation to the optimal tree.

In case that there are some edges between these components in G, let (u,v) be an edge between two components. We consider the cycle that would be generated had (u,v) been added to T. There are two cases:

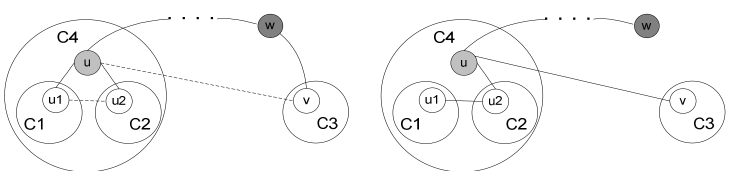
- If the cycle contains a bottleneck node w, then we add (u,v) to T and remove one edge incident on w. This is an improvement because both u and v are in V_3 and non-blocking. Thus, we have successfully reduced the degree of a bottleneck node within this iteration. We move on to the next iteration with the updated T as the input.
- If there is no bottleneck node in the cycle, the situation becomes complex and we discuss it in detail below.

If there is no bottleneck node in the cycle, then it must contain some node(s) from V_2. We merge these nodes along with all the components on the cycle into a single component. We call this newly generated component a *composite* component, to differentiate it from the *basic* components originally generated after removing V_1 and V_2 from T. As shown in Fig. 5(a), C_1 and C_2 are two basic components generated by removing V_1 and V_2 from T, and (u_1,u_2) is an edge between them. Node u is in V_2. By adding (u_1,u_2) to T, we get a cycle $u_1 \to u_2 \to v \to u_1$. Since $u \in V_2$, there is no bottleneck node in this cycle. We thus merge u and all components on the cycle (C_1 and C_2 in this example) into a single composite component C_4.

After this merge operation, we go back and check if there are edges between the components (basic or composite). If there are no such edges, the algorithm terminates. Otherwise, we choose an edge between two components. We consider the corresponding cycle that would be generated and repeat the above process. Since the graph is finite, eventually we will either find an S which consists of nodes exclusively from V_1 and V_2, or we will find a bottleneck node in the cycle.

After finding a bottleneck node, however, we may not be able to easily reduce its degree if composite components are involved. This is because, due to the merging of the components, some composite components may contain nodes in V_2. If the chosen edge happens to be between one or two nodes from V_2, then we cannot simply add it, because that would generate another bottleneck node(s). For example, in Fig. 5(a), C_1, C_2 and C_3 are basic components, hence u_1, u_2 and v are all in V_3 by construction of the algorithm. $u \in V_2$ (the light grey node) is in the cycle produced had (u_1,u_2) been added, and C_4 is the composite component generated by merging C_1, C_2 and u. A bottleneck node $w \in V_1$ (the light grey node) is in the cycle produced if (u,v) was added. If we add (u,v) and delete one edge incident on w, then u would become a bottleneck node.

The above problem can be solved in the following manner. Since u is in the cycle produced had (u_1, u_2) been added, and both u_1 and u_2 are in V_3, we can add (u_1, u_2) and remove one edge incident on u (e.g., (u, u_1)). This will decrease the degree of u by one and make it non-blocking. Then, we add (u, v) and remove one edge incident on w, which reduces the degree for bottleneck node w. In other words, we first "unblock" u *within its own component* C_4, then use edge (u, v) to make an improvement as described in Section IV-A1. This procedure can be recursively applied if C_1, C_2 are composite components and u_1, u_2 are blocking, since a blocking node can be made non-blocking *within its own component*. The following proposition formalizes this idea.



(a) Node u blocks w from (u, v) (b) Unblock the blocking node

Fig. 5. Unblock a blocking node

Proposition 2: A blocking node merged into a component can be made non-blocking by applying improvements within this component.

Proof: Let u_1 be a node in component C_1, and u_2 be a node in component C_2. Let u be a blocking node that is merged into component C when edge (u_1, u_2) is checked, along with C_1 and C_2. We need to show that u can be made non-blocking within C. There are two cases:

- If C_1 and C_2 are both basic components, then both u_1 and u_2 are non-blocking. Thus, we can add (u_1, u_2) and remove one edge incident on u, making u non-blocking. The improvement is within C.
- If C_1 or C_2 or both are composite components, then u_1 or u_2 or both could be blocking. Under this situation, if we can make u_1 non-blocking by applying improvements in C_1, and make u_2 non-blocking by applying improvements in C_2, then we apply the above improvement to "unblock" u. This is because C_1 and C_2 are disjoint from the construction of the algorithm, hence improvements within one component do not interfere with those in another.
  Thus, we check u_1, C_1 and u_2, C_2. We recursively repeat this checking process and eventually we will get to the basic components, in which all nodes are non-blocking. We then reverse the process and unblock the nodes in a bottom-up manner, until u_1 and u_2 are unblocked. Then, we unblock u by adding (u_1, u_2) and removing one edge incident on u. Note that all the improvements are within C. ■

Based upon this, in a single iteration, we will reduce the degree for some bottleneck node, otherwise we will find an S and terminate the algorithm.

2) The iterative approximation algorithm: We can now give the approximation algorithm. The algorithm starts from an arbitrary tree (line 1), and proceeds with the iterations. Lines 2-14 correspond to an iteration. Finally, it outputs the solution in line 15.

Algorithm 1 Approximation Algorithm

Input: A connected network G and a positive parameter ϵ
Output: A data gathering tree of G that approximates the maximum-lifetime tree

1: Find a spanning tree T of G.
2: **loop**
3: Let $k = \lceil \frac{r(T)}{\epsilon} \rceil$.
4: Remove V_1 and V_2 from T. This will generate a forest with several components. Let F be the set of components in the forest.
5: **while** there is an edge (u, v) connecting two different components of F and no bottleneck nodes are on the cycle generated if (u, v) was added to T **do**
6: Merge the nodes and the components on the cycle into a single component.
7: **end while**
8: **if** there is a bottleneck node in the cycle **then**
9: Follow the procedure in Proposition 2 and find a sequence of improvements to reduce the degree of the bottleneck node.
10: Make the improvements and update T.
11: **else**
12: Break out of the loop. {no edge connecting two different components of F.}
13: **end if**
14: **end loop**
15: Output the tree T as the solution.

The following proposition gives the quality of the approximation algorithm.

Proposition 3: (1) The algorithm terminates in finite time, and after termination, the tree T which it finds has $r(T) \leq r^* + \frac{2}{E_m} + \epsilon$;

(2) If there is a polynomial time algorithm which finds a tree T' with $r(T') < r^* + \frac{1}{E_m}$ for all graphs and energy settings, then $P = NP$.

Proof:(1) We first show that the algorithm terminates in finite time. Clearly, each iteration will finish in finite time, so it suffices to show the algorithm terminates after a finite number of iterations. We show this by contradiction. Suppose the algorithm never stops. In each iteration, we will reduce the degree for some node i with $r(T, i) \in ((k-1)\epsilon, k\epsilon]$. Because the network is finite, all nodes will have an inverse lifetime smaller than $(k-1)\epsilon$ within a finite number of iterations. Repeating this process, within a finite number of iterations, all nodes will have inverse lifetime smaller than $(k-2)\epsilon, (k-3)\epsilon \ldots$. However, by definition, the inverse lifetime cannot be smaller than r^*. Thus, the algorithm must terminate in finite time.

The algorithm terminates when there is no edge between the components in F, i.e., there exists S consisting of nodes exclusively from V_1 and V_2. Thus, by Lemma 1, we have $r(T) \leq r^* + \frac{2}{E_m} + \epsilon$.

(2) Similar to Proposition 1, we reduce from the Hamiltonian path problem. Given a graph G, we want to decide if it contains a Hamiltonian path. To this end, we construct an auxiliary graph G' as in Fig. 2 and adopt the same setting of energy values.

We show that the proposition is true by contradiction. Suppose that there is a polynomial algorithm which finds a tree T' with $r(T') < r^* + \frac{1}{E_m}$ for all graphs and energy settings. Running this algorithm on G' will generate a tree T' with $r(T') < r^* + 1$. We will show that G contains a Hamiltonian path if and only if $r(T') < 4 + c$.

Suppose G has a Hamiltonian path P. We construct P' in G' by adding vertices $1', 2', \dots N'$ and edges $(1, 1'), \dots (N, N')$. Clearly, P' is connected and acyclic, thus T' is a tree. Further, since P is a Hamiltonian path, the maximal degree in P is no larger than 2. But P' is constructed by adding one edge to each vertex in P, so that the maximal degree in P' is no larger than 3. Therefore, the inverse lifetime of P' is $r(P') = \max \frac{D(P',i)+c}{E(i)} \leq 3 + c$. Since P' is one particular data gathering tree for G', for G' we have $r^* \leq r(P') \leq 3+c$. Thus, $r(T') < r^* + 1 \leq 4 + c$.

Similarly, if $r(T') < 4+c$, then we have $D(T', i) \leq 3, i = 1 \dots N$. Otherwise, if $D(T', j) > 3$ for some $j, 1 \leq j \leq N$, then $r(T') \geq r(T', j) \geq 4 + c$.

Further, in T', vertices $1', 2' \dots N'$ are all leaves. We construct T by removing $1', 2' \dots N'$ and corresponding edges $(1, 1'), \dots (N, N')$ from T'. T is still a tree and it spans G. Because $D(T', i) \leq 3, i = 1 \dots N$, then in T, $D(T, i) \leq 2, i = 1 \dots N$. Thus, T is spanning tree with maximal degree no larger than 2, which is exactly a Hamiltonian path for G.

Thus, G contains a Hamiltonian path if and only if $r(T') < 4 + c$. This means for any graph G, we can decide if it contains a Hamiltonian path by running the algorithm on the constructed auxiliary graph G' and checking if $r(T') < 4+c$. This can be done in polynomial time. Hence, we can decide if a graph contains a Hamiltonian path in polynomial time. If this is true, $P = NP$. ■

We analyze the computational complexity of our algorithm. For any bottleneck node i, we have $(k-1)\epsilon < \frac{D(T,i)+c}{E(i)} \leq k\epsilon$. Hence, $(k-1)\epsilon E(i) < D(T, i) + c$. For a tree, the sum of degrees of the vertices is $2(N-1)$. So we have

$$\begin{aligned}(k-1)\epsilon \sum_{i \in V_1} E(i) &< \sum_{i \in V_1} (D(T,i) + c) \leq 2(N-1) + cN \\ &< (2+c)N. \end{aligned} \tag{7}$$

Therefore,

$$\epsilon \sum_{i \in V_1} E(i) < \frac{(2+c)N}{k-1}, \tag{8}$$

and

$$\begin{aligned}(k-1)\epsilon E_m |V_1| &\leq (k-1)\epsilon \sum_{i \in V_1} E(i) < (2+c)N \\ &\Longrightarrow \quad |V_1| < \frac{(2+c)N}{(k-1)\epsilon E_m}. \end{aligned} \tag{9}$$

In each iteration, the degree of some bottleneck node i will decrease by one, so its inverse lifetime will decrease by $\frac{1}{E(i)}$. After $\lceil \frac{\epsilon}{\frac{1}{E(i)}} \rceil$ iterations, its inverse lifetime will be lower than $(k-1)\epsilon$. For all the bottleneck nodes to have their inverse lifetime decreased to lower than $(k-1)\epsilon$, we need a total of

$$\begin{aligned}\sum_{i \in V_1} \lceil \frac{\epsilon}{\frac{1}{E(i)}} \rceil &\leq \sum_{i \in V_1} (\frac{\epsilon}{\frac{1}{E(i)}} + 1) = \epsilon \sum_{i \in V_1} E(i) + |V_1| \\ &< \frac{(2+c)N}{k-1}(1 + \frac{1}{\epsilon E_m}). \end{aligned} \tag{10}$$

iterations, where the last "$<$" comes from Equations (8) and (9).

Since the inverse lifetime cannot exceed $\frac{N+c}{E_m}$, k can be no larger than $\lceil \frac{N+c}{E_m \epsilon} \rceil$. Summing up the right hand side of Equation (10) over k, the algorithm will terminate in $O((1 + \frac{1}{\epsilon E_m}) N \log(\frac{N}{E_m \epsilon}))$ iterations. Each iteration can be completed in $O(M\alpha(M, N))$ time as in [25] using Tarjan's disjoint set union-find algorithm [26], where M is the number of edges and $\alpha(\cdot)$ is the inverse Ackerman function. Therefore, the complexity of the entire algorithm is $O((1 + \frac{1}{\epsilon E_m}) M N \alpha(M, N) \log(\frac{N}{E_m \epsilon}))$.

We note that the parameter ϵ appears both in the approximation range (as given in Proposition 3) and in the algorithm complexity. It affects the trade-off between the approximation quality and the computation time. If ϵ is chosen to be small, the approximation quality will be good, but the computation time would be large. On the other hand, choosing a large ϵ will reduce the computation time, but degrade the approximation quality. We will quantitatively study the impact of ϵ via simulations later in Section V-B and Section V-C.

C. Implementation

In many sensor systems [17], [18], the base station is a Pentium-level PC, which has a high computational capability and sufficient memory compared to the sensor nodes. Further, the base station is often connected to an unlimited power supply. Hence, it is preferable to take advantage of the computing capabilities of the base station and let it perform the tree computation[3].

In order for the base station to perform the tree computation, it first needs to obtain the neighborhood information from the nodes, so that it can construct the network graph. For this purpose, we adopt the following protocol after the network is deployed. The base station is assigned level 0, and it initiates the process of gathering neighborhood information by broadcasting a beacon. This beacon contains the identity and the level of its sender (in this case the base station). The 1-hop neighbors of the base station receive this beacon, and assign themselves level 1. They also set the base station as their parent so that they will report to it in the future. After this, they broadcast a beacon containing their own identity and

[3] Note that this *centralized* scheme is effective because the base station is much more powerful than the sensor nodes. If the base station has similar performance to the sensor nodes, a distributed implementation is more desirable.

level. The 2-hop neighbors of the base station will receive one or more such beacons, set their level to be 2 and choose a 1-hop neighbor (of the base station) as the parent to report to. This process continues and eventually every node is assigned a level and finds a parent to report to. To avoid flooding in this process, once a node is assigned a level and finds a parent, it will ignore such beacons for a sufficiently long period of time to ensure the algorithm has terminated. In this manner, a hierarchical structure is established, with the root being the base station.

Following the completion of this process, each node will report the identity of its neighbors to the base station. The transmission is hierarchical: a node reports to its parent, then the parent combines its own information with the information from its children and passes it along onto its own parent. To guarantee that all the information is received by the base station, reliable data delivery mechanisms like hop-by-hop acknowledgments can be used. The base station can construct the graph from the received information. It then computes the data gathering tree using Algorithm 1, and informs each node of its parent.

To combat the fragility of tree topologies, we must reconstruct the tree whenever a node depletes its energy or fails (e.g., due to physical damage). This computation of the tree is only done *infrequently*, i.e., we compute the tree only once after network deployment or topology change. Hence, for continuous monitoring applications where nodes are mostly static, the additional message overhead is insignificant in the long run.

V. Simulation Results

In this section, we evaluate the performance of our approximation algorithm via simulations. Unless otherwise specified, we assume that 100 nodes are uniformly dispersed in a 100 m $\times$ 100 m field[4]. The base station is located at the center of the field, i.e., its coordinate is $(50, 50)$. Each node is assigned a randomly-generated initial energy level between 1 and 10 Joules (J). There is a link between two nodes if and only if the distance between them is less than or equal to the transmission range R. Each node generates $B = 2\ bytes$ of data per minute. From previous measurements [6], the transmission power is about two times the reception power, so we set $c = 1$. All the simulation results are averaged over 100 runs, with each run using a different randomly generated topology. Table I summarizes the simulation parameters and other system constants.

TABLE I
SIMULATION PARAMETERS AND SYSTEM CONSTANTS

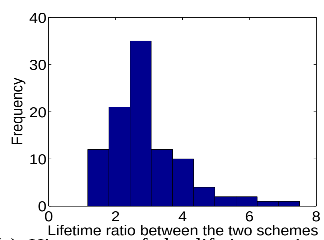
Number of nodes	100
Field	100×100
Base station location	(50, 50)
Initial energy (J)	$U(1, 10)$
B (bytes per min.)	2
Data rate (kbps)	19.2
$c = \frac{\alpha_t}{\alpha_r} - 1$	1
R (transmission range)	20
Algorithm parameter ϵ	0.5
Number of runs	100

A. Lifetime performance

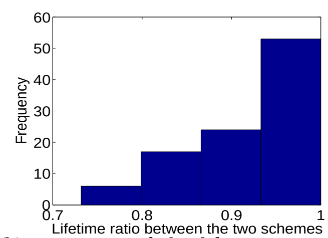
To illustrate the lifetime performance of our approximation algorithm, we compare our scheme with the initial (random) tree that we described in Section IV-C. In the random scheme, all 1-hop nodes choose the base station as the parent node. An n-hop ($n \geq 2$) node will choose an $n-1$-hop node as its parent. If there are multiple $n-1$-hop nodes within its transmission range, it *randomly* picks one of them.

For each run, we compute the lifetime ratio between our scheme and the random scheme. We show the histogram over 100 runs in Fig. 6(a). It can be seen that our scheme significantly outperforms the random scheme. For all runs, the lifetime achieved by our scheme is at least 30% larger than the random scheme, and for most runs, the lifetime of our scheme is three times larger. This confirms that it is necessary to adopt an intelligent tree construction algorithm, and validates the effectiveness of our scheme.

We also compare our scheme with the optimal solution. To do this, we enumerate all the trees for a given graph, find the one with maximum lifetime and compare with our scheme. Because of the high complexity of the enumeration, we set the number of nodes to be 10, the area to be 10×10 and the transmission range to be 6.5. We show the histogram of the lifetime ratios in Fig. 6(b). It can be seen that the performance of our scheme is close to the optimal solution. For all runs, the lifetime achieved by our scheme is at least 70% that of the optimal solution.



(a) Histogram of the lifetime ratios between our scheme and the random scheme



(b) Histogram of the lifetime ratios between our scheme and the optimal scheme

Fig. 6. Comparing our scheme with two schemes

B. Impact of ϵ on lifetime

Fig. 7 depicts the impact of ϵ on network lifetime. For each randomly generated topology, we vary the value of ϵ, execute our algorithm, and compute the lifetime. For each value of ϵ, we compute the average lifetime over 100 runs, and show the result in Fig. 7. We observe that the trend is that network lifetime achieved by our algorithm decreases as ϵ increases. This is consistent with the analytical result given by Proposition 3.

[4]This topology model is for illustration purposes only. Our scheme works with general topology models.

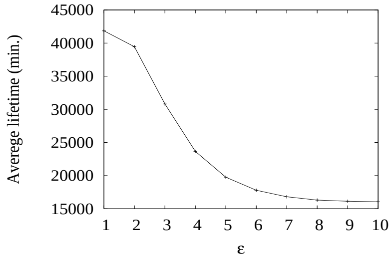


Fig. 7. Impact of ϵ on lifetime

C. Computation time

We measured the computation time of our algorithm. Our platform is a personal computer (PC) with a Pentium 4, 3.4 GHz processor and 1 GB RAM, which runs Linux version 2.6.12.6 and GNU gcc v2.7. We find that the computation time decreases as ϵ increases, and the maximum computation time is less than a few seconds for 100-node networks. Even when we increase the number of nodes to 1000, the computation takes no longer than 20 seconds to complete. This is significantly lower than the optimal computation that we compared with, which even for only 12 nodes took 239 minutes (it took about 10 minutes for 11 nodes; for 13 nodes, we aborted it before completion after two days). This shows that our algorithm incurs little computational burden and can be used for large networks.

VI. Conclusions

In this work, we have studied the construction of a data gathering tree to maximize the network lifetime of a wireless sensor network. The problem turns out to be NP-complete and hard to solve exactly. However, by investigating its structure, we give a polynomial time algorithm, which is *provably* close to optimal. Simulations show that our scheme successfully balances the load and significantly extends the network lifetime. Further, our scheme has a low computational burden, which is important for on-line implementation.

Our definition of network lifetime mainly applies to application scenarios with strict coverage requirements. We plan to extend our framework to consider other definitions of network lifetime, e.g., time until network partitioning. Further, our implementation of the algorithm leverages a centralized base station that exists in many sensor systems. For applications where a centralized base station is unavailable, a distributed approach is needed. We plan to investigate this distributed implementation of the tree construction algorithm in our future work.

References

[1] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, “Next Century Challenges: Scalable Coordination in Sensor Networks,” in *Proc. of MOBICOM*, 1999.

[2] J. Kahn, R. Katz, and K. Pister, “Next Century Challenges: Mobile Networking for ”Smart Dust”,” in *Proc. of MOBICOM*, 1999.

[3] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, “Wireless Sensor Networks for Habitat Monitoring,” in *Proc. of WSNA*, September 2002.

[4] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, “A Wireless Sensor Network for Structural Monitoring,” in *Proc. of SenSys*, 2004.

[5] L. Krishnamachari, D. Estrin, and S. Wicker, “Modeling Data-Centric Routing in Wireless Sensor Networks,” in *Proc. of INFOCOM*, 2002.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks,” in *Proc. of MOBICOM*, 2000.

[7] K. Kalpakis, K. Dasgupta, and P. Namjoshi, “Maximum lifetime data gathering and aggregation in wireless sensor networks,” in *Proc. of IEEE International Conference on Networking*, 2002.

[8] Y. T. Hou, Y. Shi, and H. D. Sherali, “Rate allocation in wireless sensor networks with network lifetime requirement,” in *Proc. of MOBI-HOC*, 2004.

[9] J. Gehrke and S. Madden, “Query Processing In Sensor Networks,” *Pervasive Computing*, vol. 3, no. 1, pp. 46–55, 2004.

[10] A. Woo, T. Tong, and D. Culler, “Taming the Underlying challenges of Reliable Multihop Routing in Sensor Networks,” in *Proc. of SenSys*, 2003.

[11] B. Hohlt, L. Doherty, and E. Brewer, “Flexible power scheduling for sensor networks,” in *Proc. of IPSN*, 2004.

[12] A. Goel and D. Estrin, “Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk,” in *Proc. of the ACM Symposium on Discrete Algorithms (SODA)*, 2003.

[13] M. Enachescu, A. Goel, R. Govindan, and R. Motwani, “Scale free aggregation in sensor networks,” in *In Proc. of the First International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, 2004.

[14] N. Thepvilojanapong, Y. Tobe, and K. Sezaki, “On the construction of efficient data gathering tree in wireless sensor networks,” in *Proc. of ISCAS*, 2005.

[15] Y. Zhang and Q. Huang, “A Learning-based Adaptive Routing Tree for Wireless Sensor Networks,” *Journal of Communications*, vol. 1, no. 2, 2006.

[16] M. Khan and G. Pandurangan, “A fast distributed approximation algorithm for minimum spanning trees,” in *Proc. of International Symposium on Distributed Computing*, 2006.

[17] G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu, “Middleware support for seamless integration of sensor and IP networks,” in *Proc. of DCOSS*, 2006.

[18] S. Arms, J. Galbreath, A. Newhard, and C. Townsend, “Remotely reprogrammable sensors for structural health monitoring,” in *Structural Materials Technology conference*, 2004.

[19] P. Gupta and P. R. Kumar, “Critical Power for Asymptotic Connectivity in wireless Networks,” *Stochastic Analysis, Control, Optimizations, and Applications: A Volume in Honor of W. H. Fleming*, 1998.

[20] S. Shakkottai, R. Srikant, and N. Shroff, “Unreliable sensor grids: Coverage, connectivity and diameter,” in *Proc. of INFOCOM*, 2003.

[21] S. Kumar, T. H. Lai, and J. Balogh, “On k-Coverage in a mostly sleeping sensor network,” in *Proc. of MOBICOM*, September 2004.

[22] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, “An Application-Specific Protocol Architecture for Wireless Microsensor Networks,” *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, October 2002.

[23] Y. Wu, S. Fahmy, and N. B. Shroff, “Energy efficient sleep/wake scheduling for multi-hop sensor networks: Non-convexity and approximation algorithm,” in *Proc. of INFOCOM*, May 2007.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[25] M. Frer and B. Raghavachari, “Approximating the minimum-degree Steiner tree to within one of optimal,” *Journal of Algorithms*, vol. 17, pp. 409–423, 1994.

[26] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. McGraw-Hill, 2001.