Constructing Maximum-Lifetime Data-Gathering Forests in Sensor Networks

Yan Wu, Zhoujia Mao, Sonia Fahmy, *Senior Member, IEEE*, and Ness B. Shroff, *Fellow, IEEE*

***Abstract*—Energy efficiency is critical for wireless sensor networks. The data-gathering process must be carefully designed to conserve energy and extend 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 first study the construction of a data-gathering tree when there is a single base station in the network. The objective is 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 that 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 then extend our work to the case when there are multiple base stations and study the construction of a maximum-lifetime data-gathering forest. We show that both the tree and forest construction algorithms terminate in polynomial time and are *provably* near optimal. We then verify the efficacy of our algorithms via numerical comparisons.**

***Index Terms*—Data-gathering, network lifetime, tree/forest construction.**

I. Introduction

RECENT advances in microelectronic 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].

Manuscript received October 02, 2008; revised September 21, 2009; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Rubenstein. . This research was supported in part by NSF grants 0238294-CNS (CAREER), 0721434-CNS, and 0721236-CNS and ARO MURI awards W911NF-07-10376 (SA08-03) and W911NF-08-1-0238. Part of this work (studying the single-base-station case) was published in the proceedings of IEEE INFOCOM 2008.

Y. Wu is with the Microsoft Corporation, Seattle, WA.

Z. Mao is with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: maoz@ece.osu.edu).

S. Fahmy is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA.

N. B. Shroff is with the departments of Electrical and Computer Engineering and Computer Science and Engineering, The Ohio State University, Columbus, OH 43210 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2010.2045896

The fundamental operation in such applications is *data gathering*, i.e., collecting sensing data from the sensor nodes and conveying it 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- or forest-based topology is often used to gather and aggregate sensing data. The tree or forest is constructed after initial node deployment and is rebuilt upon significant topology changes. We first 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 hence too computationally expensive to solve exactly. By exploiting the unique structure of the problem, we obtain an algorithm that 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 precise definition will be given in Section IV-A).

In many sensor network applications, there may be multiple base stations to which the sensor nodes report. Each base station selects a group of sensors to construct a "local" data-gathering tree. We assume that the base stations have no energy constraint. We thus extend the tree construction problem to construct a data-gathering forest for a network with multiple base stations. Each base station should construct a tree that does not intersect with trees constructed by other base stations, and the subset of nodes a base station chooses to construct a tree is not fixed. Hence, it is infeasible to run the tree construction algorithm independently at each base station. This is analogous to network clustering, which cannot be executed independently at each cluster head [4], [5]. Moreover, as will be shown in the paper, running the original tree construction algorithm iteratively could result in poor overall performance. Thus, we need to intelligently extend our framework to construct a maximum-lifetime data-gathering forest.

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 for tree construction. Section IV gives our tree construction algorithm and discusses implementation issues. Section V extends the system model to forest case. Section VI gives our forest construction algorithm and its implemen-

tation. Simulation results are presented in Section VII, and Section VIII 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. Chang and Tassiulas [6] studied energy efficient routing in wireless ad hoc networks and proposed a maximum-lifetime routing scheme. Since their focus was on general wireless ad hoc networks, nodes do not collaborate on a common task. Hence, intermediate nodes do not aggregate received data. Krishnamachari *et al.* [7] 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 [8], 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.* [9] model data gathering as a network flow problem and derive an efficient schedule to extend system lifetime. Hou *et al.* [10] 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 [5], [11], [12]. 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 [13]–[16] problems. Goel *et al.* [13] 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.* [14] consider a grid of sensors and propose a simple randomized tree construction scheme that achieves a constant factor approximation to the optimal tree. Thepvilojanapong *et al.* [15] present a data-gathering protocol that efficiently collects data while maintaining constant local state and making only local decisions. Khan and Pandurangan [17] 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 could cause a loss of coverage.

III. System Model and Problem Definition For Tree Construction

Consider a sensor network with N nodes $(v_1, v_2, \ldots v_N)$ and one base station v_0. (Our notation is summarized in Table I.) 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, nonreplenishable energy $E(i)$. The energy values $E(i)$ of different sensor nodes can be different for reasons such as heterogeneous sensor nodes, nonuniform node energy consumption, or redeployment of nodes. As with many practical systems [18], [19], the base station is connected to an unlimited power supply, hence $E(0) = \infty$. The amount of energy required to send/receive 1 bit of data is α_t/α_r.

TABLE I
List of Symbols

N	number of nodes
$E(i)$	amount of non-replenishable energy node v_i has
E_m	$\min_{i=0,\ldots,N} E(i)$
α_t	amount of energy required to send one bit of data
α_r	amount of energy required to receive one bit of data
c	$\alpha_t/\alpha_r - 1$
ϵ	algorithm parameter
$A(G)$	set of data gathering trees for network G
$L(T)$	lifetime of data gathering tree T
$C(T,i)$	number of children for node v_i in T
$D(T,i)$	degree of node v_i in T
$L(T,i)$	lifetime of node v_i in T
$r(T,i)$	inverse lifetime of node v_i in T
M	number of base stations in network G
$B(G)$	set of data gathering forests for network G
$L(F)$	lifetime of data gathering forest F
$D(F,i)$	degree of node v_i in F
$r(F,i)$	inverse lifetime of node v_i in F

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 levels to be above the critical threshold [20]–[22], which ensures that the network is connected with probability 1 as the number of nodes in the network goes to infinity. For simplicity, we do not consider dynamically adjusting 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 works [7]–[9], [23]. 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 lis-

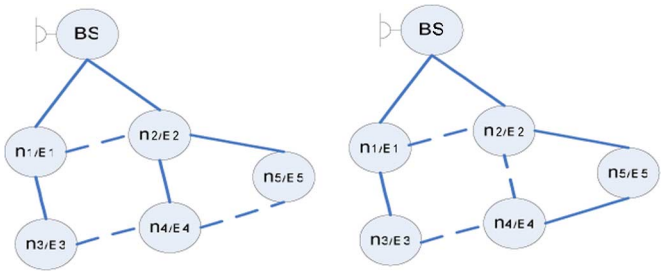

Fig. 1. Two data-gathering trees for the same network.

tening. 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 [24], we have given an example solution for a cluster hierarchy topology. Similar arguments can be made for the tree topology considered in this paper. Furthermore, 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. 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.[1] 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.

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.

[1]Here, we assume that we will lose the corresponding coverage if a node dies, i.e., there are no redundant nodes. In case of 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

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

Therefore, we can include v_0 in (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 (2) with (3) and extracting the constant $\alpha_r B$ from the denominator, we can write Problem (A) as

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

where $c = \frac{\alpha_t}{\alpha_r} - 1$ is a nonnegative 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 For Tree Construction

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 the problem is NP-hard, we reduce from the Hamiltonian path problem, which is known to be NP-complete [25]. The reduction algorithm takes as input an instance of the Hamiltonian path problem. Given a graph G, we construct 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: $E(1') = \infty$, $E(1) = E(2) = \cdots = E(N) = E(2') = \cdots = E(N') = 1$. In this manner, G' becomes an instance of Problem (B). The construction of G' and setting the energy values can be easily done in

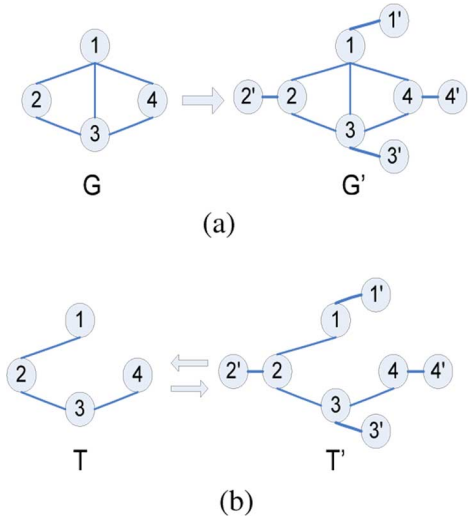

Fig. 2. Problem (B) is NP-complete. (a) Reducing HAM-PATH to Problem (B). (b) The correspondence between T and T'.

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 $\frac{1}{3+c}$.

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. Furthermore, since T is a Hamiltonian path, the maximal degree in T is no larger than 2. However, 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 [26] 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) 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 1. 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 1.

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 refer to this modification as 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

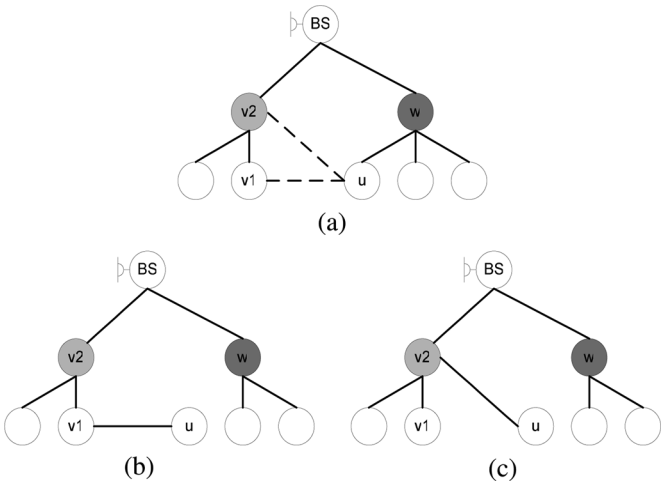

Fig. 3. The notion of improvement. (a) The tree. (b) Adding (u, v_1) and deleting (w, u) is an improvement. (c) Adding (u, v_2) and deleting (w, u) is not an improvement.

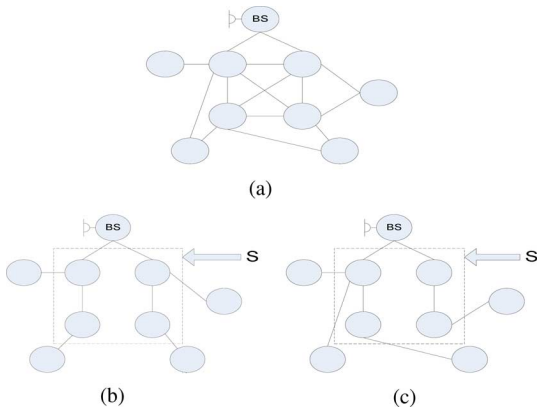

Fig. 4. Bounding r^* from below. (a) The graph. (b) T and S. (c) An arbitrary tree X.

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 gray node) is a bottleneck node, v_2 (the light gray 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.

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).

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 (4), $\sum_{i\in S} D(X,i) \geq \sum_{i\in S} D(T,i) - |S| + 1$, and the inverse lifetime of X is

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

Since X is an arbitrary spanning tree, (5) holds for any spanning tree including the optimal one. Hence, (5) gives a lower bound for the minimum maximal inverse lifetime r^*, which is equivalent to an upper bound for the maximum minimal lifetime. Furthermore, 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 (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

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

Combined with (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, $r^* > r(T) - \epsilon - \frac{2}{E_m}$. ■

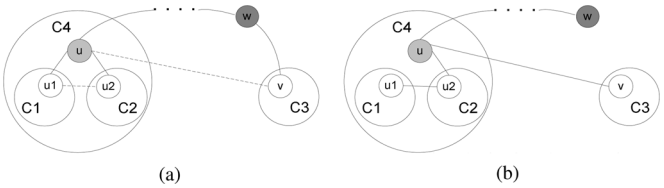

Fig. 5. Unblock a blocking node. (a) Node u blocks w from (u, v). (b) Unblock the blocking node.

B. 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, from 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) 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 $S(= V_1 + V_2)$ that 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 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 nonblocking. 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 $u_1 \rightarrow u_2 \rightarrow v \rightarrow u_1$. 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 that 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 gray 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 1 and make it nonblocking. 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 nonblocking *within its own component*. The following proposition formalizes this idea.

Proposition 2: A blocking node merged into a component can be made nonblocking 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 nonblocking within C. There are two cases.

- If C_1 and C_2 are both basic components, then both u_1 and u_2 are nonblocking. Thus, we can add (u_1, u_2) and remove one edge incident on u, making u nonblocking. 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 nonblocking by applying improvements in C_1, and make u_2 nonblocking 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 nonblocking. 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) Iterative Approximation Algorithm: The approximation 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) Algorithm 1 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 that 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 $1', 2', \ldots N'$ and edges $(1, 1'), \ldots (N, N')$. Clearly, P' is connected and acyclic, thus T' is a tree. Furthermore, since P is a Hamiltonian path, the maximal degree in P is no larger than 2. However, P' is constructed by adding one edge to each vertex in P, so 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 $D(T', i) \leq 3, i = 1 \ldots N$. Otherwise, if $D(T', j) > 3$ for some $j, 1 \leq j \leq N$, then $r(T') \geq r(T', j) \geq 4 + c$.

Furthermore, in T', vertices $1', 2' \ldots N'$ are all leaves. We construct T by removing $1', 2' \ldots N'$ and corresponding edges $(1, 1'), \ldots (N, N')$ from T'. T is still a tree, and it spans G. Because $D(T', i) \leq 3, i = 1 \ldots N$, then in T, $D(T, i) \leq 2, i = 1 \ldots N$. 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, $P = NP$. ■

C. Implementation

In many sensor systems for continuous monitoring applications [18], [19], the base station is a Pentium-level PC, which has a high computational capability and sufficient memory compared to the sensor nodes. Furthermore, 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.[2]

We first construct a tree rooted at the base station, following [27]. After the completion of this process, each node will report the identity of its neighbors[3] to the base station. The transmis-

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

[3]To mitigate transmission errors, when a node detects its neighbors, it can choose those with which it has good link quality.

sion 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 to its own parent. To guarantee that all the information is received by the base station, reliable data delivery mechanisms such as 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. System Model and Problem Definition for Forest Construction

We now extend our framework to the case of multiple base stations. We use the same assumptions as in Section III-A. Consider a large sensor network G with M base stations $v_0, v_1, \ldots, v_{M-1}$, and N sensor nodes $v_M, v_{M+1}, \ldots, v_{M+N-1}$. Each base station $v_i (i = 0, 1, \ldots, M-1)$ selects N_i sensor nodes to construct a local data-gathering tree, where $\sum_{i=0}^{M-1} N_i = N$. Each sensor node is contained only in one tree. Each sensor node $v_i (i = M, M+1, \ldots, N+M-1)$ has a battery with finite, nonreplenishable energy $E(i)$. The base stations are connected to unlimited power supplies, hence $E(0) = E(1) = \cdots = E(M-1) = \infty$. Nodes monitor the environment and periodically report to the base station in their local tree.

We need to construct a data-gathering forest with M trees $T_0, T_1, \ldots, T_{M-1}$. Each tree T_i uses base station v_i as the root node. We need to maintain complete coverage and save deployment cost, which requires that all nodes in G remain operational for as long as possible.

For a network G, there are two degrees of freedom to construct the required forest: Each base station v_i chooses a disjoint set of $N_i (i = 0, 1, \ldots, M-1)$ nodes to form a tree with the constraint $\sum_{i=0}^{M-1} N_i = N$; after choosing the set of nodes to construct a local tree for each base station, there still exist multiple possible data-gathering trees. For example, Fig. 6 shows two data-gathering forests for the same network. Algorithm 1 cannot be run *independently* at each base station since each base station cannot independently select the set of nodes to form its local tree. Moreover, running Algorithm 1 iteratively (for one base station first, and then for the next base station on the remaining nodes, and so on) is insufficient since Algorithm 1 does not prescribe how to choose a subset of nodes to construct a local tree. We therefore must extend our framework for this new scenario. As before, each forest F has a lifetime $L(F)$, which is defined as the time until the first node depletes its energy. Our goal is to find the forest that maximizes the network lifetime

(D) $\max L(F)$
such that $F \in B(G)$

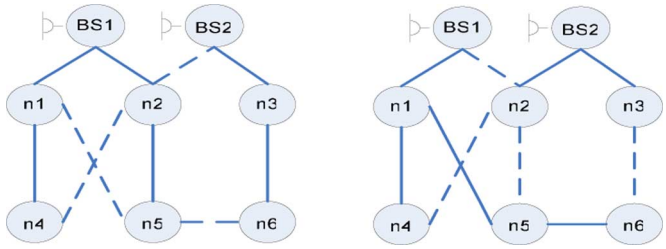

Fig. 6. Two data-gathering forests for the same network.

where $B(G)$ is the set of data-gathering forest for G.

Similar to the tree case, we can write Problem (D) as

(E) $\max \min_{i=0,1,\ldots,N+M-1} \frac{E(i)}{D(F,i)+c}$
such that $F \in B(G)$

where $\alpha_t(\alpha_r)$ is the amount of energy required to send (receive) 1 bit of data, $c = \frac{\alpha_t}{\alpha_r} - 1$, and $D(F,i)$ is the degree of v_i in forest F. Since the energy of base stations is infinite, we consider all nodes including the base stations in Problem (E).

VI. Solution and Implementation For Forest Construction

From Proposition 1, we know that Problem (E) is NP-complete, so we search for an approximate solution. Similar to the tree case, we transform Problem (E) into an equivalent form. Let $r(F,i) = \frac{D(F,i)+c}{E(i)}$ be defined as the inverse lifetime of node v_i in F. Then, Problem (E) can be written as

(F) $\min_{F \in B(G)} \max_{i=0,1,\ldots,N+M-1} r(F,i)$.

Note that Problem (F) is equivalent to Problem (D), so we will study Problem (F) instead of Problem (D).

A. Approximation Bound

Similar to the tree case, our solution starts from an arbitrary forest and iteratively makes "improvements" by reducing the degree of the *bottleneck nodes*, i.e., nodes with a large inverse lifetime, at each step. Upon termination, we will bound r^* from below and show that the resulting forest has an inverse lifetime close to the lower bound.

Given a forest F and an arbitrary ϵ, let $r(F) = \max_{i=0,1,\ldots,N+M-1} r(F,i)$, $k = \lceil \frac{r(F)}{\epsilon} \rceil$, i.e., $(k-1)\epsilon < r(F) \leq k\epsilon$. As with the tree case (Section IV-A1), we classify nodes into three disjoint subsets.

- $V_1 = \{v_i : (k-1)\epsilon < r(F,i) \leq k\epsilon\}$, i.e., V_1 contains the bottleneck nodes.
- $V_2 = \{v_i : (k-1)\epsilon - \frac{1}{E(i)} < r(F,i) \leq (k-1)\epsilon\}$. These nodes will become bottlenecks if their degree increases by 1.[4]
- $V_3 = V - V_1 - V_2$, i.e., all the remaining nodes. These nodes will not become bottlenecks even if the degree is increased by 1.

From the above categories, we can see that the smaller the value of ϵ, the fewer the nodes in V_1, which indicates that the

[4]Note that the definition of V_2 must take into account the impact of the initial energy. This is different from the MDST problem [26] where only the node degrees are considered.

bottleneck nodes are more precisely partitioned and the approximation will be tighter.

Consider an edge (u, v) that is not in F. Since F is a forest, either a unique cycle C or a unique root-to-root path P will be generated when we add (u, v) to F. We define an *improvement* and *blocking* in the same manner as in the tree case: If there is a bottleneck node $w \in V_1$ in C or P, while nodes u and v are in V_3, then we add (u, v) to F and delete one of the edges in C or P incident on w. We call this an *improvement* and say that w *benefits* from (u, v). 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). Then, we say w is *blocked* from (u, v) by u (or v or both). A node is *blocking* if it is in V_2.

Now, we extend the approximation framework for the tree construction to the forest case. We begin by assuming that we have already found a nonempty subset of nodes S that satisfies the following property.

The connected components produced by removing S from F are disconnected in both F and G.

We break the set S into partitions as follows.

Lemma 2: Suppose that we can find set S for a forest F in network G with M disjoint trees $T_0, T_1, \ldots, T_{M-1}$. Then, we have the following.

1) There exists a nonempty set P such that $P \subset S$, and P is contained and only contained in one of the trees, T_k, $k \in \{0, 1, \ldots, M-1\}$.
2) $S = S_0 \bigcup S_1 \bigcup \ldots \bigcup S_{L-1}$, where $L \leq M$. S_j is disjoint, nonempty, and $S_j \subset T_k$ for $j = 0, 1, \ldots, L-1$ and $k \in \{0, 1, \ldots, M-1\}$. Furthermore, if $S_{j_1} \subset T_{k_1}$, $S_{j_2} \subset T_{k_2}$ and $j_1 \neq j_2$, then $k_1 \neq k_2$.

Proof: 1) Since S is nonempty, it contains at least one node, say v_i, $i \in \{0, 1, \ldots, N+M-1\}$. Since $v_i \in F$ and $F = T_1 \bigcup T_2 \bigcup \ldots \bigcup T_{M-1}$, there exists $k \in \{0, 1, \ldots, M-1\}$ such that $v_i \in T_k$. Let $P = \{v_i\}$, then $P \subset S$ and $P \subset T_k$. Suppose $P \subset T_k$ and $P \subset T_l$, $k \neq l$, then $T_k \bigcap T_l \neq \Phi$, which contradicts to the fact that the trees are disjoint. Thus, P is contained and only contained in one tree T_k.

2) Since S is nonempty, then $1 \leq |S| \leq N + M$ and each node in S can form a set by itself, say P_i. Thus, $S = \bigcup_i P_i, i = 0, 1, \ldots, |S| - 1$. From 1), $P_i \subset T_k, i = 0, 1, \ldots, |S| - 1, k \in \{0, 1, \ldots, M-1\}$. We place the sets contained by the same tree into one group, then S is divided into groups. Since P_i is nonempty, contained and only contained by one tree by 1), these groups are nonempty and disjoint, i.e., $S = S_0 \bigcup S_1 \bigcup \ldots \bigcup S_{L-1}$. By the property of groups, $S_j \subset T_k, j = 0, 1, \ldots, L-1, k \in \{0, 1, \ldots, M-1\}$ and $L \leq M$. ∎

From Lemma 2, the resulting components of T_k produced by removing S_j can only be connected through S_j for T_k. Let $D(F, i)$ be the degree of node i in forest F. Then, we can count $\sum_{i \in S_j} D(F, i)$ edges incident on S_j. Since S_j is part of tree T_k, then at most $|S_j| - 1$ of these edge are within S_j, and these edges are counted twice. Hence, the number of generated components that are produced by removing S_j from T_k is

$$O_j \geq \sum_{i \in S_j} D(F, i) - 2(|S_j| - 1).$$

There are $M - L$ trees that contain no nodes in S, and these trees form entire components by themselves. Thus, the total number of generated components in a forest F is

$$\begin{aligned} O &= (M - L) + \sum_{j=0}^{L-1} O_j \\ &\geq (M - L) + \sum_{j=0}^{L-1} \left(\sum_{i \in S_j} D(F, i) - 2(|S_j| - 1) \right) \\ &= (M - L) + \sum_{i \in S} D(F, i) - 2 \sum_{j=0}^{L-1} (|S_j| - 1) \\ &= (M - L) + \sum_{i \in S} D(F, i) - 2(|S| - L). \end{aligned}$$

In an arbitrary forest X with M trees, we can connect these O components only through S. This is because there is no edge between these components in G. We need to connect these components and vertices in S. This requires

$$\begin{aligned} \sum_{j=0}^{L-1} (O_j + |S_j| - 1) &= O - (M - L) + |S| - L \\ &\geq \sum_{i \in S} D(F, i) - (|S| - L) \end{aligned} \tag{7}$$

edges. This is because if we take a component as a single node, then for the tree T_k, there are $O_j + |S_j|$ "nodes," and we need $O_j + |S_j| - 1$ edges to connect them as a tree. Furthermore, all these edges must be incident on vertices in S, then by (7), $\sum_{i \in S} D(X, i) \geq \sum_{i \in S} D(F, i) - |S| + L$, and the inverse lifetime of X is

$$\begin{aligned} r(X) &= \max_{i=0,1,\ldots,N+M-1} r(X, i) \\ &\geq \max_{i \in S} r(X, i) = \max_{i \in S} \frac{D(X, i) + c}{E(i)} \\ &\geq \frac{\sum_{i \in S} (D(X, i) + c)}{\sum_{i \in S} E(i)} \\ &\geq \frac{\sum_{i \in S} (D(F, i) + c) - |S| + L}{\sum_{i \in S} E(i)} \\ &\geq \min_{i \in S} r(F, i) - \frac{|S| - L}{\sum_{i \in S} E(i)}. \end{aligned} \tag{8}$$

From (8), we observe that if (F, S) is chosen such that $\min_{i \in S} r(F, i) \approx r(F)$, then F is a good approximation.

Lemma 3: For a forest F with M disconnected trees in a connected network G, if there is a subset S such that: 1) the components produced by removing S from F are disconnected in both F and G; and 2) if S consists of nodes exclusively from V_1 and V_2, then $r(F) \leq r^* + \frac{2}{E_m} + \epsilon - \frac{L}{|S| E_m}$, where $E_m = \min_{i=0,1,\ldots,N+M-1} E(i), L \in \{1, 2, \ldots, M\}$.

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

$$r(F, i) > r(F) - \epsilon - \frac{1}{E(i)} \geq r(F) - \epsilon - \frac{1}{E_m} \quad \forall i \in S. \tag{9}$$

Combined with (8), for any forest X, we have

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

Since X is arbitrary, this holds for any forest with M trees including the optimal forest. Hence, $r^* > r(F) - \epsilon - \frac{2}{E_m} + \frac{L}{|S|E_m}$. ■

It is easy to see that the larger the value of L, the better the bound we can achieve. Compared to Lemma 1 (a tree is a forest with $M = L = 1$), as the number of base stations M increases, the value of L is likely to increase, and the gap between the approximation and optimal solution for that forest becomes tighter, which means that the approximation becomes better.

Note that an alternative approach for solving Problem (D) can proceed as follows.

1) Construct an auxiliary graph G' by adding a virtual base station v' with infinite energy and adding edges $(v', v_0), (v', v_1), \ldots, (v', v_{M-1})$.
2) Execute Algorithm 1 on G' and find the solution tree T'. If, in T', a base station $v_i (i = 0, 1, \ldots, M-1)$ is the child of sensor node $v_j (j = M, M+1, \ldots, N+M-1)$, then remove edge (v_i, v_j) and add edge (v', v_i).
3) Remove v' and all edges incident on v'.

With this approach, however, we will lose the approximation improvement stated in Lemma 3, specifically the $\frac{L}{|S|Em}$ term.

B. Approximation Algorithm

So far, all of our results have been predicated on the existence of the set S. We next provide an approximation algorithm. We will prove that this algorithm terminates in finite steps and finds set S upon termination.

The algorithm starts from an arbitrary forest and reduces the degree of the bottleneck nodes in each iteration. The algorithm terminates when all components produced by removing S (which consists of nodes exclusively from V_1 and V_2) have no edges between each other. Then, from Lemma 3, the resulting forest is a good approximation to the optimal forest. The intuition behind this algorithm is that since the lifetime of the forest is determined by the nodes with the largest inverse lifetime, the goal should be to equalize the lifetime or inverse lifetime of all nodes.

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

1) Single Iteration: Adding an edge to a forest may either result in a unique cycle or a unique root-to-root path. If we get a cycle, then the situation is the same as the tree case. Thus, we only study the situation when we get a root-to-root path after adding a new edge.

Suppose that two components produced by removing S from F belong to different trees in F. Let (u, v) be an edge in G but not in F between these two components. Furthermore, let nodes u and v belong to the trees T_i and T_j, respectively. There are two cases.

- Consider the unique path from u to the root of T_i and the unique path from v to root of T_j. If there is a bottleneck node on either of these paths, then pick one bottleneck, say w, remove an edge incident on w, and add (u, v). This new type of improvement reduces the degree of the bottleneck node by moving a subtree of one tree to another tree.
- If there are no bottleneck nodes on these two paths, then the path can either contain nodes from V_2 or V_3 from S. We refer to the path from root of $T_i \rightarrow u \rightarrow v \rightarrow$ root of T_j as the root-to-root path. Then, we merge the components and nodes along the root-to-root path into a newly generated component called a *composite* component and proceed to the next iteration.

After finding a bottleneck node, we may not be able to reduce its degree directly if composite components are involved (for the same reason as in the tree construction case). Similar to Algorithm 1, we use the *unblock* operation when we have the situation that reducing the degree of a bottleneck node will increase the degree of nodes in V_2. If we want to unblock a node in V_2 of a composite component that consists of two basic component belonging to two trees, then we delete an incident edge on this node and add an edge in G that connects these two components. We can continue this unblocking operation until there is no blocking according to Proposition 2.

2) Iterative Approximation Algorithm:

Algorithm 2 Approximation Algorithm for Forest Construction

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

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

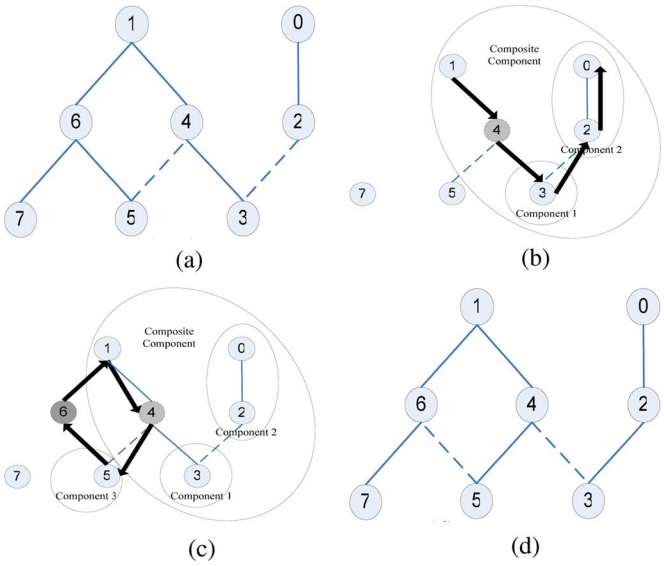

Fig. 7. Illustration of Algorithm 2: (a) initial forest, (b) combining components, (c) improvement, and (d) final forest.

Algorithm 2 starts from an arbitrary forest and iterates until it outputs forest F. We give an example to illustrate the operation of the algorithm. Fig. 7(a) shows a connected graph with an initial spanning forest that contains two trees $\{0,2\}$ and $\{1,3,4,5,6,7\}$. The dotted lines are links in the network but not in the forest. Nodes 0 and 1 are base stations. Let $E(i)=1$ for $i=2,3,4,5,6,7$; $c=1$; and $\epsilon=1$. Then in the first iteration, $V_1=\{6\}, V_2=\{4\}$. By removing V_1, V_2, we get components $\{0,2\},\{1\},\{3\},\{5\},\{7\}$. In Fig. 7(b), we check link $(2,3)$, which connects components 1 and 2. This link is on the root-to-root path $1\rightarrow 4\rightarrow 3\rightarrow 2\rightarrow 0$. Since $V_1=\{6\}$, there is no node in V_1 along this path, so we combine nodes and components along this path into a composite component. In Fig. 7(c), we check $(4,5)$ which connects component 3 and the composite component. This link is in the cycle $1\rightarrow 4\rightarrow 5\rightarrow 6\rightarrow 1$ of tree $\{1,3,4,5,6,7\}$, and node 6 is in V_1, then we perform an improvement. We first delete an edge incident on node 6 and add $(4,5)$. Since node 4 is in V_2, we "unblock" in the composite component and release node 4. The result is in Fig. 7(d). In the second iteration, $V_1=\{2,6\}$, and $V_2=\{3,4,5,7\}$. We remove nodes in V_1 and V_2 of Fig. 7(d) and find components $\{0\}$ and $\{1\}$ that are disconnected both in G and F. Thus, the algorithm terminates, finding $S=\{2,3,4,5,6,7\}$.

The following proposition characterizes the quality of the approximation and proves the existence of set S. We omit the proof since it is a direct extension of Proposition 3.

Proposition 4: Algorithm 2 terminates in finite time, and after termination, it finds S and the resulting forest F has $r(F)\leq r^*+\frac{2}{E_m}+\epsilon-\frac{L}{|S|E_m}$.

3) Computational Complexity: We now analyze the computational complexity of our algorithms.[5] For any bottleneck node i in V_1, we have $(k-1)\epsilon<\frac{D(F,i)+c}{E(i)}\leq k\epsilon$. Hence, $(k-1)\epsilon E(i)<D(F,i)+c$. Since M base stations must not be in V_1, then for each tree T_j, there are at most N_j nodes in V_1, where N_j is the number of sensor nodes in T_j. For tree T_j, there are at most N_j edges incident on N_j nodes (in the case of N_j edges, at least one edge that is incident on the base station in T_j is counted). Thus, at most $\sum_{j=0}^{M-1}N_j=N$ edges in F are incident on nodes in V_1 (in the case of N edges, at least M edges that are incident on the base stations are counted). For a forest with N nodes and M base stations, the sum of degrees of nodes in V_1 is at most $2N-M$. Thus, we have

$$(k-1)\epsilon\sum_{i\in V_1}E(i)<\sum_{i\in V_1}(D(F,i)+c)\leq 2N-M+cN. \tag{10}$$

Therefore

$$\epsilon\sum_{i\in V_1}E(i)<\frac{2N-M+cN}{k-1} \tag{11}$$

and

$$\begin{aligned}(k-1)\epsilon E_m|V_1|\leq(k-1)\epsilon\sum_{i\in V_1}E(i)<2N-M+cN\\ \Rightarrow|V_1|<\frac{(2+c)N-M}{(k-1)\epsilon E_m}.\end{aligned} \tag{12}$$

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

$$\begin{aligned}\sum_{i\in V_1}\left\lceil\frac{\epsilon}{\frac{1}{E(i)}}\right\rceil&\leq\sum_{i\in V_1}\left(\frac{\epsilon}{\frac{1}{E(i)}}+1\right)=\epsilon\sum_{i\in V_1}E(i)+|V_1|\\&<\frac{(2+c)N-M}{k-1}\left(1+\frac{1}{\epsilon E_m}\right)\\&<\frac{(2+c)N}{k-1}\left(1+\frac{1}{\epsilon E_m}\right)\end{aligned} \tag{13}$$

iterations by (11) and (12).

We can see from (13) that the upper bound on the total number of iterations decreases as the number of base stations increases. Since the inverse lifetime cannot exceed $\frac{N+c}{E_m}$, k cannot exceed $\lceil\frac{N+c}{E_m\epsilon}\rceil$. Summing up the right-hand side of the above equation 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 $O(|E|\alpha(|E|,N))$ time (as in [26]) using Tarjan's disjoint set union-find algorithm [28], where $|E|$ is the number of edges and $\alpha(\cdot)$ is the inverse Ackerman function. Therefore, the complexity of the entire algorithm is $O((1+\frac{1}{\epsilon E_m})|E|N\alpha(|E|,N)\log(\frac{N}{E_m\epsilon}))$. The computational complexity of the optimal solution (exhaustive search) for a forest with N nodes will increase exponentially as the number of base stations M grows. Since the computational complexity of our approximation algorithm remains at the same level as M increases, the complexity improvement between our approximation scheme and the exhaustive search scheme becomes more pronounced from the tree $(M=1)$ to the forest case.

[5]Note that we cannot directly follow the complexity analysis in [26]. Therein, in each iteration, the number of bottleneck nodes is reduced by one. Here, due to the impact of E_i, the size of V_1 may or may not decrease in each iteration.

We note that the parameter ϵ appears both in the approximation range (as given in Proposition 4) and in the algorithm complexity. It affects the tradeoff 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 will be large. Alternatively, choosing a large ϵ will reduce the computation time, but degrade the approximation quality. We will quantitatively study the impact of ϵ via simulations in Section VII-B.

C. Implementation

We use the following protocol for the forest construction. Each base station starts gathering local information by broadcasting a beacon. This is the same as the protocol used in the tree computation in Section IV-C; the only difference here is that the beacon also contains the identity of the base station. If a node receives beacons from several base stations, it selects one as its root and ignores others. This broadcast process terminates once every node is assigned a level and finds a parent to which to report. After that, each node reports the identity of its neighbors and that of the neighbors' root base station. Each base station receives topology information of a subset of nodes that select it as the root base station. With the received information, each base station constructs its local tree. We call two trees "neighboring trees" if there are edges in the network connecting them. Since all nodes report their neighbors' root base station, then each base station knows the neighboring trees of its local tree.

Each base station knows the node with maximum inverse lifetime in its local tree and broadcasts its identity and this inverse lifetime to its neighboring trees. These beacons between neighboring trees are sent through their adjacent nodes. Each base station waits for a period of time to confirm that there are no more beacons, and then it calculates the maximum inverse lifetime of the forest and learns the categories of the nodes in its local tree. Thus, base stations locally remove nodes in V_1 and V_2 and generated components. Then, each base station checks connecting edges between the components of its local tree or its neighboring trees. If a base station needs to perform an "improvement" or "unblock" operation, it broadcasts a beacon with its identity and indicates that it has such operations to perform. We adopt a policy that the base station with the smallest identity value "wins." After a period of time, all base stations know which base station should perform the operations if there are edges between components in this iteration. After that, each base station waits for a period of time to ensure that the operation has been completed, and then updates its local tree.

VII. 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 100×100 m^2 field.[6] Each node is assigned a randomly generated initial energy level between 1 and 10 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 [8], the transmission power is about two times the reception power, then $c = \frac{\alpha_t}{\alpha_r} - 1 = 1$. All the simulation results are averaged over 100 runs, with each run using a different randomly generated topology. Table II summarizes the simulation parameters and other system constants.

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

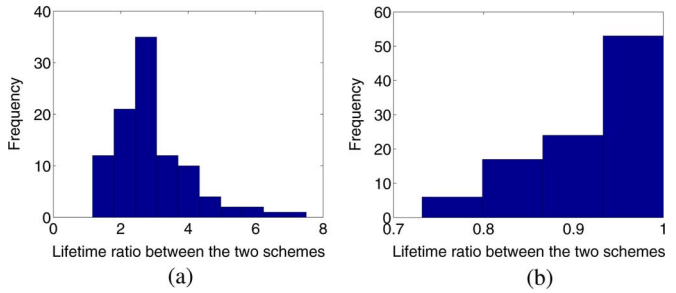

Fig. 8. Comparing our scheme to two schemes for tree construction. (a) Histogram of the lifetime ratios between our scheme and the random scheme. (b) Histogram of the lifetime ratios between our scheme and the optimal scheme.

TABLE II
Simulation Parameters and System Constants

Number of nodes	100
Field	100×100
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

1) Tree Construction: 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.

The base station is located at the center of the field, i.e., its coordinate is (50, 50). For each run, we compute the lifetime ratio between our scheme (Algorithm 1) and the random scheme. We give the histogram over 100 runs in Fig. 8(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 to the optimal solution. To do this, we enumerate all the trees for a given graph, find the one with maximum lifetime, and compare to our scheme. Because of the high complexity of the enumeration, we set the number of nodes to 10, the area to 10×10 m^2, and the transmission range to 6.5. We show the histogram of the lifetime ratios in Fig. 8(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.

2) Forest Construction: We first compare the lifetime performance of Algorithm 2 to the initial (random) forest that we

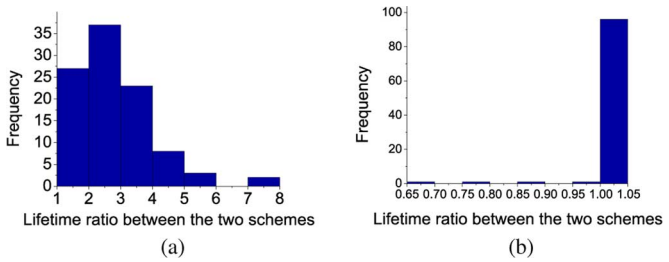

Fig. 9. Comparing our scheme to two schemes for forest construction. (a) Histogram of the lifetime ratios between our scheme and the random scheme. (b) Histogram of the lifetime ratios between our scheme and the optimal scheme.

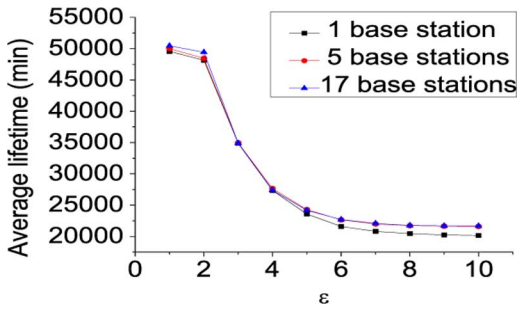

Fig. 10. Impact of ϵ on lifetime.

described in Section VI-C. We place four base stations in the field with coordinates (25, 25), (75, 25), (25, 75), (75, 75). For each simulation run, we compute the lifetime ratio between Algorithm 2 and the random scheme. The random scheme is implemented as follows: We first randomly arrange the nodes, and then construct M trees using Breadth First Search (BFS) with the arranged order of nodes. We show the histogram over 100 runs in Fig. 9(a). It can be seen that our scheme (Algorithm 2) significantly outperforms the random scheme. For most runs, the lifetime achieved by our scheme is two to three times larger than the random scheme.

We also compare our scheme to the optimal solution. To find the optimal solution, we use exhaustive search to enumerate all the spanning forests for a given graph and find the one with the maximum lifetime. Due to the complexity of the enumeration, we set the number of nodes to 6, the number of base stations to 2, the field to 10×10 m^2, and the transmission range to 6.5. The base stations are located at (2.5,2.5) and (7.5,7.5). We depict the histogram of the lifetime ratio in Fig. 9(b). It can be seen that for most runs (above 90%), our scheme results in the optimal spanning forest.

B. Impact of ϵ on Lifetime

Fig. 10 depicts the impact of ϵ on network lifetime. Three lines represent the following three cases: tree with base station at (50,50); forest with five base stations at (25, 25), (75, 25), (25, 75), (75, 75), (50, 50); and forest with 17 base stations at (12.5, 12.5), (37.5, 12.5), (62.5, 12.5), (87.5, 12.5), (12.5, 37.5), (37.5, 37.5), (62.5, 37.5), (87.5, 37.5), (12.5, 62.5), (37.5, 62.5), (62.5, 62.5), (87.5, 62.5), (12.5, 87.5), (37.5, 87.5), (62.5, 87.5), (87.5, 87.5), (50, 50). We vary the value of ϵ, execute our algorithm, and compute the lifetime for each randomly generated topology. For each value of ϵ, we compute the average lifetime over 100 runs and show the result in Fig. 10. We observe that for all cases, the trend is that the network lifetime achieved by our algorithm decreases as ϵ increases. This is consistent with the analytical result given by propositions 3 and 4. We also observe that when ϵ is small (i.e., good approximation), the more base stations in the network, the longer the lifetime we can achieve.

VIII. Conclusion

In this work, we study the construction of data-gathering trees and forests to maximize the network lifetime of a wireless sensor network. The data-gathering tree problem turns out to be NP-complete and hard to solve exactly. However, by investigating its structure, we provide a polynomial time algorithm, which is *provably* close to optimal. We then extend our solution to the construction of a data-gathering forest. This results in a better approximation ratio for extending the network lifetime and in decreasing the load on the central base station. Simulations show that both our schemes successfully balance the load and significantly extend the network lifetime. Furthermore, our schemes have a low computational burden, which is important for online implementation.

Our work can be extended in several directions. First, our definition of network lifetime mainly applies to application scenarios with strict coverage requirements. We will extend our framework to consider other definitions of network lifetime, e.g., time until network partitioning. Second, our implementation of tree construction leverages a centralized base station. In the forest case, the implementation is decentralized among trees, but each base station still makes centralized decisions in its local tree. For applications where a powerful base station is unavailable, distributed implementations of these algorithms are needed. We plan to investigate a fully distributed implementation of the algorithms. Third, we will quantitatively study the impact of wireless transmission errors. Fourth, our work assumes that all nodes transmit at the same power and do not dynamically adjust the transmission power level. Data gathering in the case when nodes dynamically adjust their transmission power levels is an open issue that we plan to investigate. Finally, we will study the joint optimization of lifetime and propagation delay (i.e., tree depth).

References

[1] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. MobiCom*, 1999, pp. 263–270.
[2] J. Kahn, R. Katz, and K. Pister, "Next century challenges: Mobile networking for "Smart Dust"," in *Proc. MobiCom*, 1999, pp. 271–278.
[3] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. WSNA*, Sep. 2002, pp. 88–97.
[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. SenSys*, 2004, pp. 13–24.
[5] B. Hohlt, L. Doherty, and E. Brewer, "Flexible power scheduling for sensor networks," in *Proc. IPSN*, 2004, pp. 205–214.

[6] J. H. Chang and L. Tassiulas, "Maximum lifetime routing in wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 4, pp. 609–619, Aug. 2004.
[7] L. Krishnamachari, D. Estrin, and S. Wicker, "Modeling data-centric routing in wireless sensor networks," USC Computer Engineering Tech. Rep. CENG 02-14, 2002 [Online]. Available: http://lecs.cs.ucla.edu/Publications/papers/Bhaskar-DataCentric.pdf
[8] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proc. MobiCom*, 2000, pp. 56–67.
[9] K. Kalpakis, K. Dasgupta, and P. Namjoshi, "Maximum lifetime data gathering and aggregation in wireless sensor networks," in *Proc. IEEE Int. Conf. Netw.*, 2002, pp. 685–696.
[10] Y. T. Hou, Y. Shi, and H. D. Sherali, "Rate allocation in wireless sensor networks with network lifetime requirement," in *Proc. MobiHoc*, 2004, pp. 67–77.
[11] J. Gehrke and S. Madden, "Query processing in sensor networks," *Pervasive Comput.*, vol. 3, no. 1, pp. 46–55, 2004.
[12] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proc. SenSys*, 2003, pp. 14–27.
[13] A. Goel and D. Estrin, "Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk," in *Proc. ACM SODA*, 2003, pp. 499–505.
[14] M. Enachescu, A. Goel, R. Govindan, and R. Motwani, "Scale-free aggregation in sensor networks," in *Proc. 1st Int. Workshop Algor. Aspects Wireless Sensor Netw.*, 2004, pp. 15–29.
[15] N. Thepvilojanapong, Y. Tobe, and K. Sezaki, "On the construction of efficient data gathering tree in wireless sensor networks," in *Proc. ISCAS*, 2005, vol. 1, pp. 648–651.
[16] Y. Zhang and Q. Huang, "A learning-based adaptive routing tree for wireless sensor networks," *J. Commun.*, vol. 1, no. 2, pp. 12–21, 2006.
[17] M. Khan and G. Pandurangan, "A fast distributed approximation algorithm for minimum spanning trees," in *Proc. Int. Symp. Distrib. Comput.*, 2006, pp. 391–402.
[18] G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu, "Middleware support for seamless integration of sensor and IP networks," in *Proc. DCOSS*, 2006, pp. 101–118.
[19] S. W. Arms, J. H. Galbreath, A. T. Newhard, and C. P. Townsend, "Remotely reprogrammable sensors for structural health monitoring," in *Proc. NDE/NDT Highways, Bridges, Struct. Mater. Technol. VI*, Buffalo, NY, Sep. 16, 2004, pp. 331–338.
[20] P. Gupta and P. R. Kumar, "Critical power for asymptotic connectivity in wireless networks," in *Stochastic Analysis, Control, Optimizations, and Applications: A Volume in Honor of W. H. Fleming*. Cambridge, MA: Birkhauser, 1998, pp. 547–566.
[21] S. Shakkottai, R. Srikant, and N. Shroff, "Unreliable sensor grids: Coverage, connectivity and diameter," in *Proc. IEEE INFOCOM*, 2003, vol. 2, pp. 1073–1083.
[22] S. Kumar, T. H. Lai, and J. Balogh, "On k-coverage in a mostly sleeping sensor network," in *Proc. MobiCom*, Sep. 2004, pp. 144–158.
[23] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Trans. Wireless Commun.*, vol. 1, no. 4, pp. 660–670, Oct. 2002.
[24] 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. IEEE INFOCOM*, May 2007, pp. 1568–1576.
[25] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. San Franciso, CA: Freeman, 1979.
[26] M. Frer and B. Raghavachari, "Approximating the minimum-degree Steiner tree to within one of optimal," *J. Algor.*, vol. 17, pp. 409–423, 1994.
[27] R. Perlman, "An algorithm for distributed computation of a spanning tree in an extended LAN," in *Proc. 9th Symp. Data Commun.*, 1985, pp. 44–53.
[28] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. New York: McGraw-Hill, 2001.

Yan Wu received the B.S. degree in electrical engineering and information science from the University of Science and Technology of China, Hefei, China, in 1997; the M.S. degree in computer science from the Chinese Academy of Sciences, Beijing, China, in 2000; and the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, in 2007.

He is currently a Software Engineering with Microsoft Corporation, Seattle, WA. His research interests lie in resource allocation, optimization, and cross-layer design in wireless and sensor networks.

Zhoujia Mao received the B.S. degree in electrical and information science from the University of Science and Technology of China, Hefei, China, in 2007. He is currently working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, The Ohio State University, Columbus. His research interests include wireless networks, resource allocation, scheduling and optimization.

Sonia Fahmy (SM'05) received the Ph.D. degree from the Ohio State University, Columbus, in 1999.

She is an Associate Professor with the Computer Science Department, Purdue University, West Lafayette, IN. Her current research interests lie in the areas of Internet tomography, network security, and wireless sensor networks.

Dr. Fahmy is a Member of the Association for Computing Machinery (ACM). She received the National Science Foundation CAREER Award in 2003 and the Schlumberger Technical Merit Award in 2000. For more information, please see http://www.cs.purdue.edu/~fahmy/.

Ness B. Shroff (F'07) received the Ph.D. degree from Columbia University, New York, NY, in 1994.

He joined Purdue University, West Lafayette, IN, after receiving the Ph.D. degree. At Purdue, he became Professor of the School of Electrical and Computer Engineering in 2003 and Director of the Center for Wireless Systems and Applications (CWSA) in 2004. In 2007, he joined the Ohio State University, Columbus, as the Ohio Eminent Scholar of Networking and Communications and Professor of electrical and computer engineering and computer science and engineering. His research interests span the areas of wireless and wireline communication networks. For more information, please see http://www.ece.ohio-state.edu/~shroff/.