Energy-Efficient Collection of Sparse Data in Wireless Sensor Networks Using Sparse Random Matrices

XIAOHAN YU, Zhejiang Gongshang University
SEUNG JUN BAEK, Korea University

We consider the energy efficiency of collecting sparse data in wireless sensor networks using compressive sensing (CS). We use a sparse random matrix as the sensing matrix, which we call Sparse Random Sampling (SRS). In SRS, only a randomly selected subset of nodes, called the source nodes, are required to report data to the sink. Given the source nodes, we intend to construct a data gathering tree such that (1) it is rooted at the sink and spans every source node and (2) the minimum residual energy of the tree nodes after the data collection is maximized. We first show that this problem is NP-complete and then develop a polynomial time algorithm to approximately solve the problem. We greedily construct a sequence of data gathering trees over multiple rounds and propose a polynomial-time algorithm to collect linearly combined measurements at each round. We show that the proposed algorithm is provably near-optimal. Simulation and experimental results show that the proposed algorithm excels not only in increasing the minimum residual energy, but also in extending the network lifetime.

CCS Concepts: • **Networks → Sensor networks**; **Network performance modeling**;

Additional Key Words and Phrases: Wireless sensor networks (WSNs), data collection, compressive sensing (CS), energy efficiency, sparse sensing matrices

ACM Reference format:
Xiaohan Yu and Seung Jun Baek. 2017. Energy-Efficient Collection of Sparse Data in Wireless Sensor Networks Using Sparse Random Matrices. *ACM Trans. Sen. Netw.* 13, 3, Article 22 (August 2017), 36 pages.
https://doi.org/10.1145/3085576

1 INTRODUCTION

In this article, we consider the data collection problem in Wireless Sensor Networks (WSNs) (Lindsey et al. 2002; Di Francesco et al. 2011). The WSN is regarded as a key enabling technology for data acquisition and sensing in Internet of Things (IoT) architectures (Gubbi et al. 2013; Perera et al. 2014). A WSN consists of low-cost, low-power, and energy-constrained sensors which acquire and transmit information to the sink through wireless links (Akyildiz et al. 2002;

This work is supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIP) of Republic of Korea, No. NRF-2016R1A2B1014934, and by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0126-17-1046, Research of Network Virtualization Platform and Service for SDN 2.0 Realization).

Authors' addresses: X. Yu, School of Information and Electronic engineering, Zhejiang Gongshang University, Hangzhou, China; email: yuxiaohan188@126.com; S. J. Baek, Dept. of Computer Science and Engineering, Korea University, Seoul, Korea; email: sjbaek@korea.ac.kr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax + 1 (212) 869-0481, or permissions@acm.org.
© 2017 ACM 1550-4859/2017/08-ART22 $15.00

https://doi.org/10.1145/3085576

Bouabdallah et al. 2009; Jongerden et al. 2010; Cheng et al. 2010). A major limiting factor of the performance of WSNs is the energy of battery-operated sensors. A sensor will cease to operate if it depletes its battery energy. Our goal in this article is to design an energy-efficient data gathering strategy.

Fundamental approaches for energy efficiency include balancing the traffic loads imposed on sensors, as in Liang and Liu (2007), Wu et al. (2010), and Stanford and Tongngam (2009), to list a few. One approach for energy efficiency is to build a spanning tree such that the minimum residual energy of nodes is maximized. For example, Liang and Liu (2007) considered an online data-gathering problem for WSN processing multiple queries. A greedy algorithm called Maximum Network Lifetime (MNL) was proposed to maximize the minimum residual energy after each query is answered. MNL constructs a tree by starting from the sink and iteratively adding a node with the maximum residual energy to the tree, and it is shown to perform well with various types of queries. Another line of approach is to maximize the network *lifetime* or minimum operation time of sensors, where the operation time of a sensor is defined as the time until its energy is depleted. For example, Stanford and Tongngam (2009) studied the lifetime maximization problem with data aggregation. Their method combines a linear program with the exact computation of minimum cost arborescence. They proved that the proposed approach achieves at least $(1 - \epsilon)$ times the optimum lifetime. Wu et al. (2010) focused on distributing the network loads by building a spanning tree which maximizes the network lifetime. They proved that the problem is NP-complete and proposed an approximation algorithm which iteratively improves nodes with low energy levels. Note, however, in Wu et al. (2010) it is assumed that the sink aggregates data from all the sensors, and the energy cost of transmissions over links is fixed across the network. In this article, we investigate a more general problem; we will consider maximizing the minimum residual energy for the data collection from an arbitrary *subset* of nodes. In addition, we assume that the energy cost per link may vary over the links so as to capture the varying link qualities across the network.

Recently, Compressive Sensing (CS) (Donoho 2006; Baraniuk 2007; Candès and Wakin 2008) has drawn much attention in the literature of WSNs (e.g., Luo et al. (2009), Wang et al. (2007), Ebrahimi and Assi (2014), Karakus et al. (2013), and Haupt et al. (2008)). The idea behind CS is to reliably recover a high-dimensional signal (i.e., the original data vector) from the measurements of a significantly lower dimension. Luo et al. (2009) proposed the Compressive Data Gathering (CDG) to collect data in WSNs using the CS technique. In CDG, the sink collects the linear combinations of the sensed data instead of the individual data samples. Once a sufficient number of linear combinations are collected, the sink is able to recover the original sparse data vector by solving an ℓ_1-based convex optimization (Eldar and Kutyniok 2012). The main advantage of CDG is that it not only reduces the energy consumption of the sensors, but it also evenly distributes loads across the network. Wang et al. (2007) proposed a class of sparse random sensing matrices that do not compromise recovery performance. In their scheme, each node aggregates one measurement as a linear combination of raw data, as in Luo et al. (2009). The sink can recover the original data by collecting these measurements from any m nodes. However, in Wang et al. (2007), the total number of aggregated measurements is n because there are n nodes and each node aggregates one measurement. This is redundant because the sink needs only m measurements. Ebrahimi and Assi (2014) applied the sparse random matrix proposed in Wang et al. (2007) to CDG and proposed the Minimum Spanning Tree Projection (MSTP) scheme. MSTP aimed to minimize the total energy consumption by constructing multiple MSTs. Each MST is rooted at a randomly selected node and is used to gather one measurement. The aggregated measurement is sent to the sink via the shortest path. Karakus et al. (2013) compared CS-based schemes with conventional data-gathering techniques in WSNs from the perspective of network lifetime. They showed that the network lifetime can be significantly prolonged provided that the original data are sparse and the network is dense. However, they only considered dense sensing matrices, whereas we adopt sparse sensing matrices to further improve energy efficiency. For a more comprehensive survey on the use of CS in WSNs, readers are referred to Haupt et al. (2008).

All of the preceding works considered data collection from every node (i.e., construction of a spanning tree). However, we consider data collection from an arbitrary *subset* of nodes. In an earlier work, Fürer and Raghavachari (1994) proposed an algorithm to construct a tree that spans an arbitrary subset of nodes and minimizes the maximum degree of the nodes in an unweighted graph. Our setup is more general than that of Fürer and Raghavachari (1994) because our goal is to maximize the minimum residual energy in the *weighted* graphs. Specifically, we consider a weighted graph in which the energy consumption of a node is a function of the weights of the edges incident on it. Moreover, each node may initially have a different amount of energy stored at its battery. Our algorithm takes into account all of the initial energy settings, the residual energy of the nodes, and the edge weights.

Contributions: We investigate the energy-efficient collection of compressive data using SRS. The goal is to maximize the minimum residual energy at the retrieval of a data vector; the problem, however, proves to be hard because the set of source nodes changes randomly over multiple rounds. Instead, we take a greedy approach of maximizing the minimum residual energy in a round-by-round manner; namely, at each round, we aim to construct a data-gathering tree which spans all the source node associated with that round. We first prove that even the optimization for a single round is NP-complete. We then propose a polynomial time algorithm called Approximately Maximum min-Residual Energy Steiner Tree (AMREST) to build a data-gathering tree for each round. Using AMREST, we iteratively find good routes which circumvent the sensors with low residual energy. We show that AMREST is provably efficient such that (i) there exists a performance limit which polynomial-time algorithms cannot universally exceed over all possible systems; (ii) it is guaranteed that AMREST performs close to that limit and the gap is only a constant. In the special case where the initial energy reserves and communication costs per link are uniform across the network, we show that the optimality gap is constant irrespective of the network topology. Simulation and experimental results demonstrate that AMREST performs well in maximizing the minimum residual energy as well as extending network lifetime.

Paper Organization: In Section 2, we propose sparse random sampling and present three examples of stable recovery using sparse sensing matrices. We formulate our problem in Section 3 and propose AMREST in Section 4. The performance analysis is provided in Section 5. In Section 6, we present simulation results. Section 7 provides experimental results on a WSN test bed. Section 8 concludes the paper.

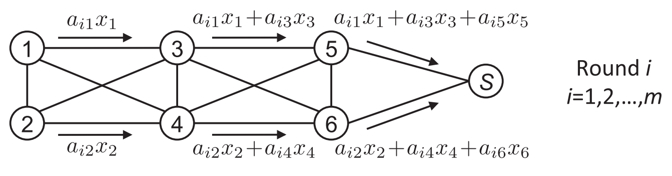
2 SPARSE RANDOM SAMPLING (SRS)

We first briefly introduce the CS technique. In CS, the following *linear* model is considered:

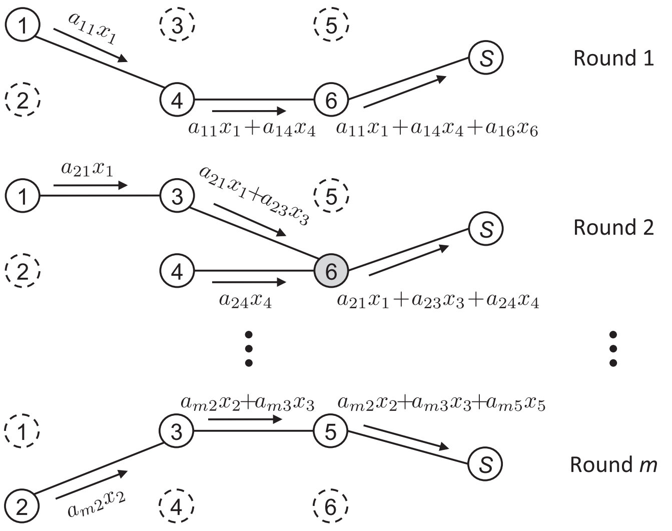
$$\mathbf{y} = \mathbf{A}\mathbf{x}, \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^m$ is the *measurement vector*, $\mathbf{x} \in \mathbb{R}^n$ is the data vector, and $\mathbf{A} \in \mathbb{R}^{m\times n}$ is called the *sensing matrix*. A vector $\mathbf{x}$ is said to be k-sparse in a basis Ψ if $\mathbf{x} = \Psi\mathbf{c}$, and the number of nonzero entries in $\mathbf{c}$ does not exceed k. Suppose $\mathbf{A}$ is a Gaussian random matrix; that is, each entry of $\mathbf{A}$ is drawn independently and identically distributed (i.i.d.) from the Gaussian distribution $N(0, \frac{1}{m})$. It is well known that if $m = O(k\log(n))$ and $k \ll n$, the k-sparse vector $\mathbf{x}$ can be recovered with high probability via linear programming (Candès and Wakin 2008; Eldar and Kutyniok 2012).

Consider a WSN with n sensors. The sensor data is represented by $\mathbf{x}$ where the jth entry x_j denotes the raw data of the jth node of the network. In CDG, the sink collects data according to



(a) Traditional CDG.



(b) Sparse Random Sampling (SRS).

Fig. 1. Comparison of the data collection between the traditional CDG and SRS.

the linear model given by Equation (1), where **A** is a Gaussian random matrix. We shall define the ith round as the process of collecting y_i given by

$$y_i = \sum_{j=1}^{n} a_{ij}x_j,\ i = 1, 2, \ldots, m, \tag{2}$$

where a_{ij} denotes the entry in the ith row and jth column of **A**. Thus, there is a total of m rounds of data collection. Consider an example of data collection under CDG: See Figure 1(a). The sink will collect m measurements $y_1, y_2, \ldots, y_m$ over m rounds. Hence, every node transmits for $m = O(k \log n)$ times. Thus, CDG significantly reduces and evenly distributes the network loads, as compared to collecting every individual measurement, which requires on average $O(n)$ number of transmissions per node (Luo et al. 2009).

Recently, *sparse* random matrices were shown to have similar recovery performance as the Gaussian random matrix (Wang et al. 2007; Li et al. 2013; Gilber and Indyk 2010). A sparse random matrix is a matrix whose entries are zero independently with a fixed probability. Importantly, if $a_{ij} = 0$, we will *not* need the data x_j when collecting y_i because y_i is a linear combination given by Equation (2). That is, during the ith round, we only need to collect data from the sensors with nonzero a_{ij}. Thus, by using sparse random matrices, we effectively select the nodes at random for the data collection. We will refer to such selection as *Sparse Random Sampling* (SRS). The selected nodes are called the *source nodes*. The examples in Figure 1 compare the data collection between

SRS and the traditional CDG. In the example, the sensing matrix for SRS is given by

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & 0 & a_{14} & 0 & a_{16} \\ a_{21} & 0 & a_{23} & a_{24} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_{m2} & a_{m3} & 0 & a_{m5} & 0 \end{bmatrix}. \tag{3}$$

At each round, only a random subset of nodes transmit data, as opposed to the traditional CDG in which every node transmits during data collection. This leads to reduced energy consumption and latency associated with the data collection. At the first round of the example in Figure 1(b), we have $a_{12} = a_{13} = a_{15} = 0$; see the first row of $\mathbf{A}$ in Equation (3). We can put the nonsource nodes (i.e., node $2, 3$, and 5) into "sleep" mode, which further saves energy. Meanwhile, at the second round, node 6 (the shaded node in Figure 1(b)) is not a source node (i.e., $a_{26} = 0$), but was forced to stay "awake" so that the source nodes could reach the sink.

In large-scale WSNs, it is of key interest to efficiently collect data over source and nonsource nodes under SRS. We aim to construct a data collection tree which not only spans the source nodes, but also evenly distributes the network loads across the network based on the *residual* battery capacity of the sensors so as to prolong the operation time of the sensors. In this article, our objective is to maximize the *minimum residual energy* among the sensors at the completion of retrieving a data vector $\mathbf{x}$ using SRS. Later, we will show that our approach is also effective in extending network operation time; e.g., retrieving as many data vectors as possible without depleting the battery of any sensor. To our knowledge, the problem of maximizing the minimum residual energy associated with sparse sensing matrices has not been explored yet.

Next, we provide three examples of sparse sensing matrices which guarantee the stable recovery of the original data vector $\mathbf{x}$. All of those sparse matrices are obtained by randomly "thinning" the entries of $\mathbf{A}$ with a fixed probability. Here, we introduce a key parameter used throughout this article called *retention probability* $s \in (0, 1]$, which will denote the probability that an entry of $\mathbf{A}$ is nonzero.

THEOREM 2.1 (BARANIUK ET AL. 2008). *Suppose the original data vector* $\mathbf{x} \in \mathbb{R}^n$ *is* k*-sparse and each entry in the sensing matrix* $\mathbf{A}$ *is independently drawn from the distribution*

$$a_{ij} = \sqrt{1/m} \times \begin{cases} +1 & \text{with probability } s/2, \\ 0 & \text{with probability } 1-s, \\ -1 & \text{with probability } s/2, \end{cases}$$

with $s \in [\frac{1}{3}, 1]$. *Then,* $\mathbf{A}$ *has the same recovery performance as the Gaussian random matrix; i.e.,* $m = O(k \log n)$ *is sufficient to guarantee the stable recovery of* $\mathbf{x}$.

Theorem 2.1 implies that, on average, up to two-thirds of the entries in $\mathbf{A}$ can be zero without compromising recovery performance because s can be decreased to $\frac{1}{3}$.

THEOREM 2.2 (WANG ET AL. 2007). *Suppose the original data vector* $\mathbf{x} \in \mathbb{R}^n$ *is* k*-sparse under the basis* Ψ *and satisfies* $\frac{\|\mathbf{x}\|_\infty}{\|\mathbf{x}\|_2} \le h$. *Suppose each entry in the sensing matrix* $\mathbf{A}$ *is independently drawn from the distribution*

$$a_{ij} = \sqrt{1/sm} \times \begin{cases} +1 & \text{with probability } s/2, \\ 0 & \text{with probability } 1-s, \\ -1 & \text{with probability } s/2, \end{cases} \tag{4}$$

where $s \in (0, 1]$, and the number of measurements satisfies

$$m = \begin{cases} O\left(\frac{h^2 k^2 (1+\gamma)}{s\epsilon^2} \log n\right) & \text{if} \quad h^2/s \geq \Omega(1), \\ O\left(\frac{k^2(1+\gamma)}{\epsilon^2} \log n\right) & \text{if} \quad h^2/s \leq O(1), \end{cases} \tag{5}$$

where γ and ϵ are two positive constants. Then, given $\mathbf{y}$, $\mathbf{A}$ and Ψ, one can produce an estimate $\hat{\mathbf{x}}$ such that $\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 \leq (1 + \epsilon)\|\mathbf{x}\|_2^2$ with probability at least $1 - n^{-\gamma}$.

Theorem 2.2 is applicable when the peak-to-total energy of the data vector is bounded by the parameter h. We observe that $m = O(k^2 \log n)$ in Theorem 2.2. Recall that $m = O(k \log n)$ when the Gaussian random matrix is used as the sensing matrix. This implies that, in order to guarantee the stable recovery, an increased number of measurements is needed if the matrix in Equation (4) is used as the sensing matrix.

Li et al. (2013) developed a class of sparse random matrices which guarantees stable recovery with the same order of measurements $m = O(k \log n)$ as that of the Gaussian random matrix.

Theorem 2.3 (Li et al. 2013). *Suppose the original data vector $\mathbf{x} \in \mathbb{R}^n$ is k-sparse and nonnegative. Let $a_{ij} = b_{ij} q_{ij}$, where b_{ij} is independently and identically drawn from the α-stable maximally skewed distribution with unit scale, and q_{ij} satisfies*

$$q_{ij} = \begin{cases} 1 & \text{with probability } s, \\ 0 & \text{with probability } 1 - s, \end{cases} \tag{6}$$

where $s \in (0, 1]$. Then, given $\mathbf{y}$ and $\mathbf{A}$, it suffices to use $m = \frac{k \log(n/\epsilon)}{1 - e^{-(1-s)k}}$ measurements such that $\mathbf{x}$ can be recovered with an arbitrary small error with probability at least $1 - \epsilon$.

We observe that $m = O(k \log n)$ if $s = \Theta(k^{-1})$; i.e., one may gather data from only $\Theta(k^{-1})$ fraction of the nodes without degrading recovery performance. In Theorem 2.3, the data vector is required to be nonnegative, which applies to many types of signals in practice (e.g., a bounded signal can be transferred to a non-negative signal (Li et al. 2014)). Any of the aforementioned matrices can be used for SRS; we will, however, propose an algorithm which can be used for arbitrary $s \in (0, 1]$ in the sequel.

In this article, we assume that the original data vector is sparse in a certain basis. In real implementations, it is a nontrivial problem to make data vectors sparse in some basis. Although such sparsification techniques are beyond the scope of this article, we will briefly mention some related issues. The sparsification problem was investigated in an early work by Quer et al. (2009). It is not straightforward to achieve *incoherence* between the random projection of sensors' data through routing and the sparse representation of the signal, which is required for CS recovery. The authors considered several popular transformations and found that none is able to sparsify real datasets while being incoherent with respect to the routing matrix at the same time. Later, the authors proposed a sparsification framework under the CS model in Quer et al. (2012) called SCoRe1. SCoRe1 is able to effectively self-adapt to the real-time data vectors by exploiting a feedback control loop which can estimate the signal reconstruction error. The key technique for SCoRe1 is to use the Principle Component Analysis (PCA) to capture the spatial and temporal characteristics of real data vectors measured by WSNs. Their work showed that, under some statistical assumptions, the use of CS for data collection in WSNs is legitimate through random sampling of sensors' data, similar to SRS followed by a combined use of CS and PCA for recovery.

3 PROBLEM FORMULATION

Consider a weighted graph $G = (V, L)$ where $V := \{v_1, v_2, \ldots, v_n\}$ is the set of nodes including the sink v_1 and L is the set of edges. Each edge $l \in L$ is associated with a weight through the function $w : L \rightarrow [\alpha, \beta]$, where $0 < \alpha \leq \beta$. The weights are related to our model for energy consumption, which will be explained later. Recall that our goal is to maximize the minimum residual energy at the completion of retrieving the data vector $\mathbf{x}$. In order to retrieve $\mathbf{x}$, we need to collect a total number of m measurements. Denote S_i the set of source nodes for the ith round. In other words, S_i consists of nodes whose ith coefficient is nonzero; i.e., $S_i := \{v_j : a_{ij} \neq 0\}$ where a_{ij} is the ith coefficient of node v_j. The data-gathering tree for the ith round is denoted by $T_i = (V_{T_i}, L_{T_i})$ where $V_{T_i} \subseteq V$ is the set of nodes in T_i and $L_{T_i} \subseteq L$ is the set of edges in T_i.

The random coefficients in $\mathbf{A}$ should be known to the network. However, there exists a practical issue in disseminating all the entries of $\mathbf{A}$ to the entire network; it may incur significant communication overhead. Later, we will outline a method which uses Pseudo-Random Number Generators (PRNG) in order to minimize such overhead.

We assume that all the nodes, including the sink, use the identical PRNG. The following procedure is performed at network initialization:

- The sink generates a seed, say c, for PRNG. Note that a seed is a number used to initialize PRNG.
- The sink broadcasts c to the network.
- Each sensor generates its own seed through combining the sink's seed and its node ID (e.g., a concatenation of the seed and the ID).
- Denote the seed of node i by c_i. Node i will generate random coefficients a_{ji} of $\mathbf{A}$ in round j, using the PRNG initialized by seed c_i.

Note that seeds c_i, $i = 1, \ldots, n$, are known to the network, which enables every node to generate the same set of random numbers as other nodes. Thus, sensing matrix $\mathbf{A}$ is effectively known to the network during the entire rounds of data aggregation. The communication overhead is simply that of broadcasting c and thus is present only at system initialization. During data collection, the sink will compute and broadcast the routing information of T_i to each node at the beginning of the ith round. Node v_j in tree T_i will generate a_{ji} as its coefficient. After receiving data from all of its children in T_i, v_j linearly combines the received data and its own data $a_{ij}x_j$ and transmits the result to its parent. Note that $a_{ij} \neq 0$ if v_j is a source node, and $a_{ij} = 0$ if v_j is not a source node.

Next we introduce the model for energy consumption. Denote $\mathcal{I} := \{I(v_1), I(v_2), \ldots, I(v_n)\}$ the set of initial energy where $I(v_j)$ is the initial energy of node v_j. We assume the sink is a special node and has an infinite amount of initial energy (i.e., $I(v_1) = \infty$). Suppose each node transmits data through a data packet of size b bits. Furthermore, we assume that the energy consumed by transmitting and receiving one bit of data via an edge is approximately equal (Raghunathan et al. 2002). Although this is a somewhat simplified model for energy expenditure, we are able to capture diverse channel characteristics among nodes by reflecting varying energy costs per link to edge weights. Specifically, node v_i consumes r_l amount of energy to transmit one bit data to node v_j through the link $l = (v_i, v_j)$. Node v_j consumes the same amount of energy r_l to receive one bit of data from node v_i. Thus, node v_i consumes br_l amount of energy to transmit one packet to node v_j through the edge $l = (v_i, v_j)$, and node v_j consumes br_l amount of energy to receive one packet from node v_i. Denote $w(l) = br_l$ for any $l \in L$. Then, given a data-gathering tree $T = (V_T, L_T)$, the energy consumed by node v is given by $\sum_{l \in L_v \cap L_T} w(l)$ when the data is gathered through T. The notations used in this article are summarized in Table 1.

Table 1. Notations

s	Retention probability
$G(V,L)$	Network graph, where V is the set of nodes and L is the set of edges
V_T	Set of nodes in the tree T
L_T	Set of edge in the tree T
S_i	Set of source nodes for the ith round
$\mathcal{I}$	Set of initial energy, i.e., $\mathcal{I} := \{I(v) : v \in V\}$
l_v	Edge that is incident on node v
L_v	Set of edges that are incident on the node v
$l_v^\dagger$	Edge incident on v which has the largest weight, i.e., $l_v^\dagger := \{l_v \in L_v : w(l_v) \geq w(l_v'), \forall l_v' \in L_v\}$
$\mathcal{T}_m$	Collection of m trees for data gathering
$\mathcal{T}(S)$	Collection of trees in which any tree T satisfies: (1) T spans S; (2) all leaves of T are in S
$D(\mathcal{G},v)$	Degree of node v in graph $\mathcal{G}$
$E(T,v)$	Residual energy of node v if data is gathered through T
$R(T)$	Minimum residual energy of nodes in T
$T(p)$	Path in T that connects two endpoints of the path p
$F(v)$	Component in the forest F that contains node v
$F(p)$	Union of components $F(v)$ for each node v in the path p, i.e., $F(p) = \cup_{v \in p} F(v)$

The objective is to find a collection $\mathcal{T}_m = \{T_1, T_2, \ldots, T_m\}$ which maximizes the minimum residual energy:

$$(\mathbf{P}_0) \quad \underset{\mathcal{T}_m}{\text{maximize}} \;\; R(\mathcal{T}_m), \qquad \text{subject to } S_i \subseteq V_{T_i}, \;\; i = 1, 2, \ldots, m.$$

where $R(\mathcal{T}_m) := \min_{v \in V} E(\mathcal{T}_m, v)$ and $E(\mathcal{T}_m, v)$ is the residual energy of node v after measurements are gathered through trees in $\mathcal{T}_m$. $(\mathbf{P}_0)$ is quite hard to solve; e.g., we will later show that $(\mathbf{P}_0)$ is NP-complete even for $m = 1$. Instead, we take a greedy approach, such that we attempt to maximize the minimum residual energy during each round. Specifically, given S_i, we will construct a tree T_i subject to $S_i \subseteq V_{T_i}$ so that the minimum residual energy of nodes in T_i is maximized. A similar greedy approach was adopted in Liang and Liu (2007) in a different context; the authors intended to maximize the number of queries answered by the WSN without expiring any node's energy. They also proposed to greedily maximize the minimum residual energy of the nodes for each data collection query. The proposed approach is outlined in Algorithm 1.

While such greedy approximations may work reasonably well in practice, the optimality gap can be significant. For example, when we greedily maximize the minimum residual energy at each round, we may consume much energy in total, which can be burdensome to the network in the long run. This can be partly mitigated by putting a limit on the total energy consumed by the network at each round. We discuss this issue in detail in Section 4.5. Another optimistic aspect of our approach is that the number of rounds is at most $m = O(k^2 \log n)$. Thus, we expect that the gap between our approximation and the solution to $(\mathbf{P}_0)$ grows relatively slowly with increasing network size n.

Next, let us consider the construction of the data-gathering tree for a single round. Without loss of generality, we consider the ith round and let $S := S_i$. We will construct a tree $T = (V_T, L_T)$ which

ALGORITHM 1: Greedy Round-by-Round Data Collection

```
1: $\mathcal{I}_i$: set of the initial energy for the $i$th round.
2: $T_i$: the data gathering tree for the $i$th round.
3: **Input** the graph $G$, the vector of initial energy $\mathcal{I}$ and the sensing matrix $\mathbf{A}$.
4: **Output** the data gathering tree $T_i$ $(i = 1, 2, \ldots, m)$ for collecting the $i$th measurement.
5: **Initialize** $\mathcal{I}_1 = \mathcal{I}$.
6: Compute the set of source nodes for collecting the $i$th measurement:
   $S_i = \{v_j : a_{ij} \neq 0\}, i = 1, 2, \ldots, m$.
7: **for** $i = 1$ **to** $m$ /* measurements are collected round by round.*/ **do**
8:     Compute the data gathering tree $T_i$ based on AMREST described in Algorithm 2.
9:     Update the initial energy of nodes for the next round:
       $I_{i+1}(v) = I_i(v) - \sum_{l \in L_{T_i} \cap L_v} w(l), \ \forall v \in V_{T_i}$.
10:    Collect the $i$th measurement $y_i$ by broadcasting the information of $T_i$ and the $i$th row of $\mathbf{A}$ to nodes.
11: **end for**
```

spans S. Denote $E(T, v)$ the residual energy of node v if T is applied to gather data. We have that

$$E(T, v) = I(v) - \sum_{l \in L_T \cap L_v} w(l). \tag{7}$$

Our goal is to find a tree $T \in \mathcal{T}(S)$ such that the minimum residual energy of nodes in T is maximized:

$$\textbf{(P)} \quad \underset{T \in \mathcal{T}(S)}{\text{maximize}} \ R(T) := \min_{v \in V_T} \left[I(v) - \sum_{l \in L_T \cap L_v} w(l) \right].$$

Theorem 3.1. *(**P**) is NP-complete.*

Proof. See Appendix A. □

The rest of this article is devoted to developing an algorithm to approximately solve **(P)**.

4 PROPOSED ALGORITHM

The key idea behind our algorithm is to repeatedly reduce the loads of the nodes having small residual energy. Specifically, we begin by constructing a random tree $T \in \mathcal{T}(S)$. Next, we search for a path p, in which all the nodes are not in T except for the endpoints, such that the load of a node which has small residual energy can be reduced by adding p and deleting an edge in T that connects to this node. This process is repeated until we cannot find such a path to "improve" the nodes that have small residual energy.

4.1 Basic Notions

We first introduce basic notions related to the proposed algorithm. Some of the notions were adopted from Fürer and Raghavachari (1994) and Wu et al. (2010), which we modify and generalize to fit our model. Given an arbitrary tree $T = (V_T, L_T)$, we partition V_T into three disjoint subsets as follows:

— $V_b(T) = \{v \in V_T : R(T) \leq E(T, v) < R(T) + w(l_v^\dagger)\}$. V_b is the set of nodes whose residual energy is close to the minimum residual energy of nodes in T. The nodes in V_b are called *bottleneck* nodes.
— $V_p(T) = \{v \in V_T : R(T) + w(l_v^\dagger) \leq E(T, v) < R(T) + 2w(l_v^\dagger)\}$. The nodes in V_p are called *pseudo-bottleneck* nodes. These nodes are close to becoming bottleneck nodes.

— $V_s(T) = \{v \in V_T : E(T, v) \geq R(T) + 2w(l_v^\dagger)\}$. The nodes in $V_s(T)$ are called *safe* nodes. The safe nodes have relatively large residual energy.

We partition the nodes not in T into two disjoint subsets: $V_c(T)$ and $V_{nc}(T)$.

— $V_c(T) = \{v \in V_T^c : I(v) \geq R(T) + 3w(l_v^\dagger)\}$. The nodes in V_c are called *candidate* nodes. Candidate nodes have sufficient residual energy and are used to reduce the loads of bottleneck nodes.
— $V_{nc}(T) = \{v \in V_T^c : I(v) < R(T) + 3w(l_v^\dagger)\}$. The nodes in V_{nc} are called *noncandidate* nodes.

We briefly explain how the candidate nodes are used. Consider a path p, in which all the intermediate nodes between two nodes in T are candidate nodes. Suppose $p = (x, v_1, \ldots, v_k, y)$ where $x, y \in V_T$ and $v_1, \ldots, v_k$ are candidate nodes. Suppose we add p to T and delete edge (x, v_1) to generate a new tree. Then, any intermediate node in p never becomes a bottleneck node because p is a path and thus the degree of any candidate node is at most two in T. Hence, the residual energy of the candidate node v_i $(i = 1, \ldots, k)$ is decreased by at most $2w(l_{v_i}^\dagger)$. Therefore, by definition, the residual energy of v_i is at least $R(T) + w(l_{v_i}^\dagger)$. Thus, $V_c(T)$ is the set of nodes that can be safely added to T during such a modification. Note that, in this article, we either delete edge (x, v_1) or (v_k, y). Such augmented paths are used to reduce the loads of bottleneck nodes, as will be detailed later. In summary, we have that $V_T = V_b \cup V_p \cup V_s$ and $V_T^c = V_c \cup V_{nc}$.

Based on the aforementioned node partitioning, we introduce the following notions:

— *Valid path*: We refer to a path $p = (u, h_1, h_2, \ldots, h_k, v)$ as a *valid path* if $u, v \in V_s \cup V_p$ and $h_1, h_2, \ldots, h_k \in V_c$ (i.e., the endpoints are pseudo-bottleneck nodes or safe nodes), but every intermediate node is a candidate node. A valid path may contain only two nodes (e.g., $p = (u, v)$), which reduces to an edge in L.
— *Improvement*: Suppose $p = (u, \ldots, v)$ is a valid path and $T(p)$ contains a bottleneck or pseudo-bottleneck node b. We reduce the energy consumption of b by adding p to T and deleting an edge in $T(p)$ that is incident on b. We call such a modification an *improvement*, and we say b is *improved* by adding p.
— *Block*: Suppose $p = (u, \ldots, v)$ is a valid path and $T(p)$ contains a bottleneck node b. Also suppose that adding p to T makes u (or v or both) a bottleneck node. We say u (or v or both) *blocks* b from p in this case. Note that only pseudo-bottleneck nodes may block b.
— *Unblock*: In order to improve a bottleneck node b which is blocked by a pseudo-bottleneck node u, we need to reduce the energy consumption of u before improving b. We say the process of reducing the energy consumption of u the *unblocking* process. The details of the unblocking process will be discussed in Section 4.4.

We will describe the proposed algorithm based on these notions.

4.2 Outline

In this subsection, we introduce the AMREST algorithm. The flow chart of AMREST is shown in Figure 2. An outline of AMREST follows.

Step 1 (Build an initial tree). We initially construct a random spanning tree T. Given T, we repeatedly remove leaves that are not source nodes until all the leaves are source nodes. Note that, in this article, removing a node from a graph means that all of the edges incident on this node are also removed.

Step 2 (Generate a forest). We generate a forest that contains only safe nodes or pseudo-bottleneck nodes; later, we will repeatedly connect the components of the forest using valid paths in order to reduce the load of bottleneck nodes. We generate forest F by removing $V_b \cup V_p$ from T.

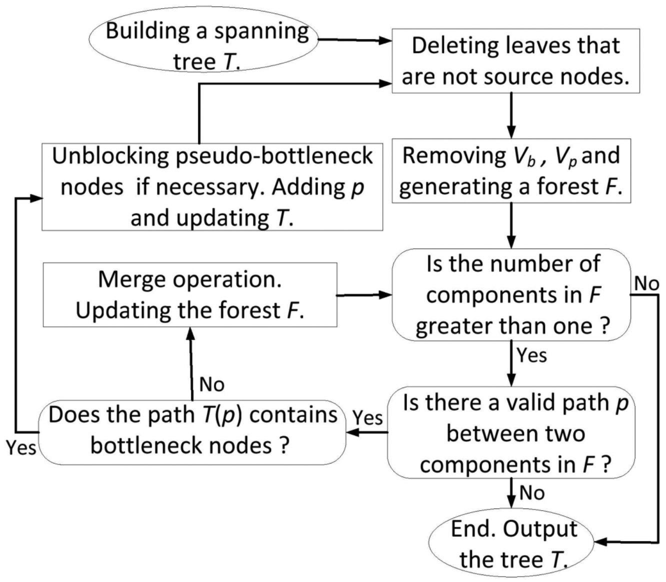

Building a spanning tree T.
Deleting leaves that are not source nodes.
Removing V_b, V_p and generating a forest F.
Unblocking pseudo-bottleneck nodes if necessary. Adding p and updating T.
Merge operation. Updating the forest F.
Is the number of components in F greater than one ?
No
Yes
Is there a valid path p between two components in F ?
Yes
No
Does the path $T(p)$ contains bottleneck nodes ?
No
Yes
End. Output the tree T.

Fig. 2. Flow chart of AMREST.

Step 3 (Find a valid path). Because improving a bottleneck node involves adding a valid path and deleting an incident edge on it, we first need to find a valid path. Specifically, given the forest F, we search for a valid path between two different components. If there is no valid path between any two different components in F, we terminate the algorithm and output T. Suppose there exists a valid path between two different components F_1 and F_2. We choose the valid path $p_i = (u, \ldots, v)$ that satisfies the rule described in Line 11 of AMREST. In AMREST, we define parameter $iter\_res$ in order to track the number of iterations in which a node is added to the data-gathering tree. For example, if node v is a safe node, we have $iter\_res(v) = 0$ since safe nodes are retained in the data-gathering tree after Step 2. If node v is a pseudo-bottleneck node which is added during the ith iteration, we have $iter\_res(v) = i$. Depending on whether $T(p_i)$ contains a bottleneck node, two cases may occur:

Case 1. $T(p_i)$ contains no bottleneck nodes. Go to Step 4.

Case 2. $T(p_i)$ contains at least one bottleneck node. Go to Step 5.

Step 4 (Update the forest). Our goal is to find a bottleneck node to improve; however, there are no bottleneck nodes in $T(p_i)$ at this point. We intend to find a new valid path by adding some pseudo-bottleneck nodes to forest F. Specifically, we merge every component which contains nodes in $T(p_i)$ together with $T(p_i)$ into a single component. In other words, the newly generated component is a subgraph of T induced by the nodes in $F(T(p_i)) \cup T(p_i)$. Note that some nodes in $T(p_i)$ may not be in F but may have been added to forest F via the merging operation. After the merging operation, we return to Step 3.

Step 5 (Update T). Since a bottleneck node has been found, we can reduce its load via an improvement. Specifically, let $b \in T(p_i)$ denote the bottleneck node that has the minimum residual energy among all the bottleneck nodes in $T(p_i)$. We desire to improve the bottleneck node b. Two cases may occur depending on the residual energy of u and v:

Case 1. Both u and v are safe nodes. In this case, we directly improve b by adding p_i to T, deleting an incident edge on b. Note that, in $T(p_i)$, there exist two edges which are incident on b. We improve b by deleting one of the two edges that has the larger weight.

ALGORITHM 2: Approximately Maximum min-Residual Energy Steiner Tree (AMREST)

```
1: Input a weighted graph $G = (V, L, W)$, the initial energy $I(v)$ for all $v \in V$ and the set of source nodes $S$.
2: Output a tree $T \in \mathcal{T}(S)$.
3: Initialize $T$ as a random spanning tree.
4: loop
5:   Given a tree $T$, repeatedly delete leaves that are not source nodes until all the leaves are source nodes.
6:   Compute $V_b, V_p, V_s, V_c$ based on $T$; Generate a forest $F$ by removing $V_b \cup V_p$; Set
     $iter\_res(v_s) = 0, \forall v_s \in V_s$.
7:   label_path=0 /*indicator for valid paths*/.
8:   Initialize $i = 1$.
9:   while the number of components in $F$ is larger than 1 do
10:    if there exists a valid path between different components $F_1$ and $F_2$ in $F$. then
11:      label_path=1; Let $p_i = (u, \ldots, v)$ be the valid path between $F_1$ and $F_2$ such that for any other
         valid path $p' = (u', \ldots, v')$ between $F_1$ and $F_2$, we have that $iter\_res(u') \geq iter\_res(u)$ or
         $iter\_res(v') \geq iter\_res(u)$.
12:      if $T(p_i)$ contains at least one bottleneck node. then
13:        Find the bottleneck node $b$ that has the minimum residual energy among nodes in $T(p_i)$.
14:        If adding $p_i$ makes $u$ (or $v$) a bottleneck node, unblock $u$ (or $v$) by using the Algorithm 3.
15:        Add $p_i$ to $T$, and delete the edge in $T(p_i)$ that connects to $b$ and has the larger weight.
16:        Break out of the while loop.
17:      else
18:        Merge every component that contains nodes in $T(p_i)$ together with $T(p_i)$ into a single
           component /* the forest F is updated */.
19:        Set $iter\_res(v) = i$ for each pseudo-bottleneck node $v$ that is newly added to the forest $F$;
           $i = i + 1$.
20:      end if
21:    else
22:      label_path=0. Break out of the while loop.
23:    end if
24:  end while
25:  if label_path=0 then
26:    Terminate and output the tree $T$.
27:  end if
28: end loop
```

Case 2. u or v or both are pseudo-bottleneck nodes. In this case, b may be blocked by u (or v or both). We first unblock u (or v or both), then make an improvement of b as in *Case 1*. Details of the unblocking procedure are given in Section 4.4.

In both *Case 1* and *Case 2*, T is updated. After updating T, we return to Step 2.

A pseudo-code of AMREST is presented in Algorithm 2. Next, we will provide an illustrative example of AMREST.

4.3 Example

Consider the graph shown in Figure 3(a) where s_1 is the sink. Figure 3(a) shows the initial tree T in which every leaf is a source node. Nodes h_1 and h_2 are not in T and thus are candidate nodes. Assume the following initial energy reserves for nodes b_1, b_2 and u_2:

$$I(b_1) = I(b_2) = 4, \quad I(u_1) = 7.$$

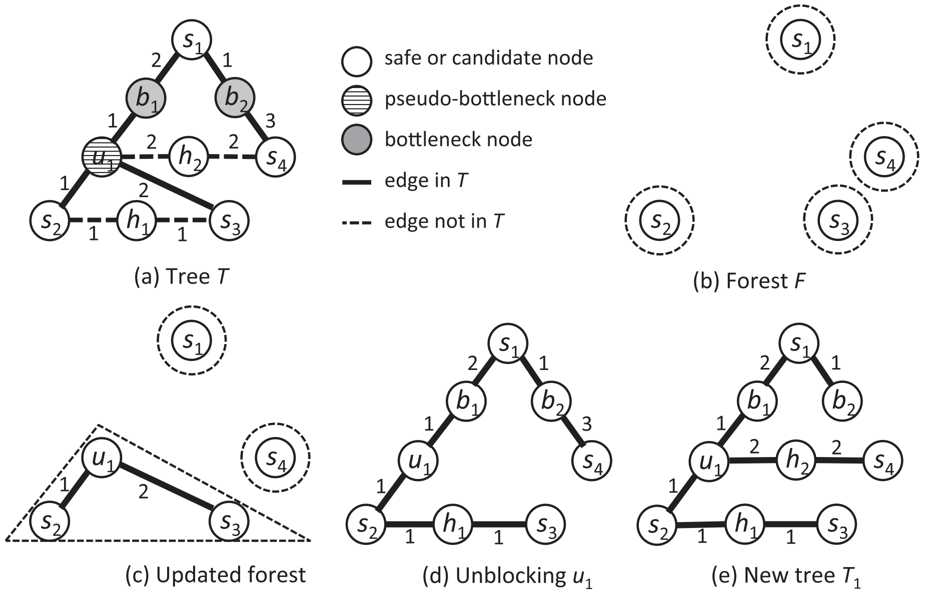


Fig. 3. Example of the AMREST algorithm.

Assume that the initial energy of the remaining nodes is sufficiently large. It follows that the residual energy of b_1, b_2, u_1 is given by

$$E(T, b_1) = 1, \quad E(T, b_2) = 0, \quad E(T, u_1) = 3,$$

where we have that $R(T) = E(T, b_2) = 0$.

We will explain a single outer loop (Lines 4–28). We remove $V_b \cup V_p$ from T and generate a forest F which contains components $\{s_1\}, \{s_2\}, \{s_3\}$ and $\{s_4\}$: see Figure 3(b). During the first *while* iteration at Line 9, we find a valid path $p_1 = (s_2, h_1, s_3)$ between $\{s_2\}$ and $\{s_3\}$. Since there is no bottleneck node in $T(p_1) = (s_2, u_1, s_3)$, we merge every component that contains nodes in $T(p_1)$ (i.e., $\{s_2\}$ and $\{s_3\}$) together with $T(p_1)$ into a single component (Line 18): see Figure 3(c). There are now three components in the forest: $\{s_1\}$, $\{s_4\}$ and $\{s_2, u_1, s_3\}$.

Next, we iterate the second *while* loop at Line 9. We find a valid path $p_2 = (u_1, h_2, s_4)$ between $\{s_2, u_1, s_3\}$ and $\{s_4\}$. There exist two bottleneck nodes in $T(p_2)$; hence, we move to Line 13. We choose to improve b_2 since it has smaller residual energy than b_1. Then we find that adding p_2 reduces the residual energy of u_1 from 3 to 1, which makes u_1 a bottleneck node (i.e., u_1 blocks b_2 from p_2). Hence, we first improve u_1 (i.e., unblock u_1) by adding p_1 and deleting (u_1, s_3): see Figure 3(d). Note that we chose to delete (u_1, s_3) since the weight of (u_1, s_3), which is 2, is larger than that of (u_1, s_2), which is 1. Finally, we improve b_2 by adding p_2 and deleting (b_2, s_4). This is a simple example of unblocking; the general unblocking procedure will be detailed in the next subsection. The new tree T_1 is shown in Figure 3(e). Consequently, $R(T_1) = E(T_1, b_1) = 1$; hence, the minimum residual energy is increased from $R(T) = 0$ to $R(T_1) = 1$ after the single iteration of AMREST.

4.4 Unblocking Procedure

In Line 14 of AMREST, we may need to unblock u or v. In this subsection, we explain details of the unblocking procedure. We will extend the unblocking technique for the node-degree

ALGORITHM 3: Unblock u with $iter\_res(u) = k$

```
1: $\tilde{P}$: Set of valid paths that will be added to the component.
2: $\tilde{L}$: Set of edges in $F(u)$ that will be deleted.
3: $\tilde{V_i}$: Set of nodes with $iter\_res(v) = i$.
4: **Input** the component $F(u)$ which contains $u$.
5: **Initialize** $\tilde{P} = \{p_k\}$ and $\tilde{L} = \{l_u\}$ where $l_u \in T(p_k)$ is the edge that is incident on $u$ and has the larger weight.
6: Suppose among all the nodes in $F(u)$, there are $q$ different $iter\_res$ values that are not larger than $k$: $k = j_1 > j_2 > \cdots > j_q = 0$.
7: **for** $i = 2$ **to** $q - 1$ **do**
8:     **if** there exists a node $v \in p_t \cap V_{j_i}$, where $p_t \in \tilde{P}$, such that adding $p_t$ makes $v$ a bottleneck node. **then**
9:         $\tilde{P} = \tilde{P} \cup \{p_{j_i}\}$.
10:        $\tilde{L} = \tilde{L} \cup \{l_v\}$ where $l_v \in T(p_{j_i})$ is the edge that connects to $v$ and has the larger weight.
11:    **end if**
12: **end for**
13: Add every path in $\tilde{P}$ to the component $F(u)$ and delete every edge in $\tilde{L}$.
14: **Output** the updated component.
```

minimization for unweighted graphs in Fürer and Raghavachari (1994) to our case of residual energy optimization for weighted graphs.

In the unblocking procedure, we find a sequence of improvements to reduce the energy consumption of the blocking node. Analogous to the parameter *iter* in Fürer and Raghavachari (1994), we will use the parameter $iter\_res$ to track how the residual energy of a blocking node is iteratively improved as follows. In order to improve a node, say u, we add a valid path p to T and delete an edge from u. However, this may cause an endpoint of p, say v, to become a new bottleneck node. Thus, we iterate a similar operation in order to improve v by adding another valid path p' to T, and so on. At each iteration, the iteration count is assigned to the node to be improved as its $iter\_res$ value. We say an improvement is in the jth *level* if the improvement is made for a node v with $iter\_res(v) = j$. By assigning $iter\_res$ to the nodes, we can keep track of the level in which we intend to improve the corresponding node; this enables us to prove some useful lemmas in the sequel.

Let us specifically explain how we unblock u with $iter\_res(u) = k$. Initially, we add $p_k = (u_k, \ldots, v_k)$ to $\tilde{P}$ and add the edge in $T(p_k)$, which is incident on u and has the larger weight to $\tilde{L}$. Suppose $iter\_res(u_k) = j < k$ and adding p_k makes u_k a new bottleneck node. In this case, we also need to make the improvement in the jth level to improve u_k. Hence, we add $p_j = (u_j, \ldots, v_j)$ to $\tilde{P}$ and add the edge in $T(p_j)$ which is incident on u_k and has the larger weight to $\tilde{L}$. This process is repeated until adding a valid path does not create new bottleneck nodes. So far, we have found $\tilde{P}$ and $\tilde{L}$. We finally unblock u by adding every path in $\tilde{P}$ to the tree and deleting every edge in $\tilde{L}$. The pseudocode of the unblocking procedure is shown in Algorithm 3.

We show that a blocking node can be always unblocked.

LEMMA 4.1. *Any pseudo-bottleneck node u which blocks a bottleneck node can be unblocked through Algorithm 3.*

PROOF. See Appendix B. □

In AMREST, T is updated during each outer loop (i.e., Lines 4–28). Some candidate nodes may be added to T during the update of T. We show that every added node will never become a bottleneck node.

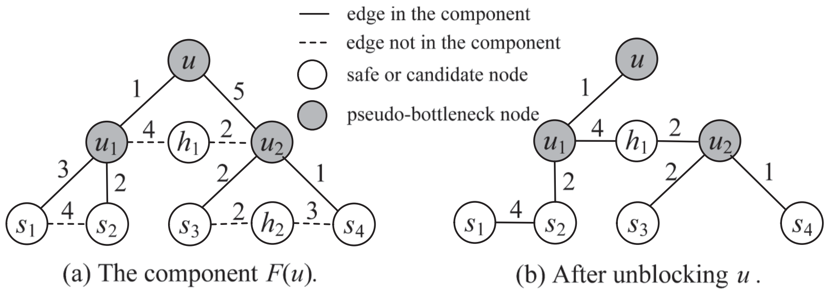


Fig. 4. Example of unblocking the pseudo-bottleneck node u.

Lemma 4.2. *Suppose there exist some candidate nodes which are added to T during the outer loop at Line 4 of AMREST. Then, the residual energy of any newly added candidate node $v \in V_c(T)$ is at least $R(T) + w(l_v^\dagger)$.*

Proof. See Appendix C. □

Lemma 4.2 implies the following: Let T' be the tree obtained by updating T. Because a bottleneck node is improved and no new bottleneck nodes are generated, we have that $R(T') \geq R(T)$. Namely, the minimum residual energy never decreases after the update of T.

Here, we mention key differences between our unblocking procedure and that in Fürer and Raghavachari (1994):

— During the unblocking procedure in Fürer and Raghavachari (1994), any node which is not in T can be selected to improve other nodes. However, the node selection is more restrictive in our case; the improvements involve the valid paths only; that is, every intermediate node selected in the added paths must be a candidate node that has sufficiently large residual energy. The difference is because our problem generalizes and extends that of Fürer and Raghavachari (1994); Fürer and Raghavachari (1994) deals with unweighted graphs without residual energy constraints.
— Our unblocking procedure can immediately halt after adding a valid path $p = (u, \ldots, v)$ as long as u and v are not bottleneck nodes. This can occur even if u and v are pseudo-bottleneck nodes. By contrast, in Fürer and Raghavachari (1994), the improvements must be repeatedly made until both u and v become a node with degree smaller than $\Delta - 2$, where Δ is the maximum degree of nodes in the tree; nodes with this property are analogous to the *safe* nodes in our setup. As a result, in their case, the unblocking procedure must hierarchically propagate to the initial blocking nodes. Consequently, our unblocking procedure may require fewer improvement steps than in Fürer and Raghavachari (1994) given the same network topology.
— Unlike (Fürer and Raghavachari 1994), we consider weighted graphs. As a result, there is an opportunity to delete edges with large weights connected to pseudo-bottleneck nodes at each improvement. Suppose a valid path is added to the tree in order to improve the pseudo-bottleneck node u, creating a cycle containing u. Thus, we can remove one of the two edges incident on u which are in the cycle. We propose to remove the one with the larger weight (Line 10 of Algorithm 3), which helps to further reduce the energy burdens on u.

Figure 4 shows an illustrative example of the unblocking procedure. The component in the forest that contains u is shown in Figure 4(a). We assume that $iter\_res(u) = 3 > iter\_res(u_2) = 2 > iter\_res(u_1) = 1$. The valid paths chosen during the 1st, 2nd, and 3rd iterations are given by

$p_1 = (s_1, s_2)$, $p_2 = (s_3, h_2, s_4)$, and $p_3 = (u_1, h_1, u_2)$, respectively. In order to unblock u, we make an improvement of u by adding p_3 and deleting (u, u_2). Hence, we add p_3 to $\tilde{P}$ and add (u, u_2) to $\tilde{L}$. Suppose adding p_3 makes u_1 a bottleneck node. We also need to improve u_1 by adding p_1 and deleting (s_1, u_1). Thus, we add p_1 to $\tilde{P}$ and add (s_1, u_1) to $\tilde{L}$. Note that adding p_3 does not make u_2 a bottleneck node; hence, we do not need to improve u_2. So far, we have found $\tilde{P}$ and $\tilde{L}$. We add every path in $\tilde{P}$ and delete every edge in $\tilde{L}$. Figure 4(b) shows the component after unblocking u. Finally, the residual energy of the pseudo-bottleneck node is increased by 5.

4.5 Mitigating Inefficiency of Greedy Approach

There may exist deleterious effects of our greedy approach which maximizes the minimum residual energy at each round. AMREST attempts to improve bottleneck nodes by iteratively augmenting possibly long paths to the tree. Such long paths may incur excessive energy consumption across the network, which may be harmful in the long run over the rounds. A similar issue was addressed in Li et al. (2001) which introduces a novel framework to mitigate that effect. The authors proposed the max – min $zP_{\min}$ algorithm, which finds a route between two nodes to maximize the minimum residual energy along the route. A salient feature of max – min $zP_{\min}$ is that the algorithm explores the tradeoff between the *total* energy cost and the minimum residual energy, where the total cost is defined as the sum of edge weights. Specifically, max – min $zP_{\min}$ maximizes the minimum energy under the constraint that the total cost of the resulting route does not exceed zP_{min}. Here, $z \geq 1$ is a parameter and P_{min} is the minimum total cost between the nodes.

Motivated by the max – min $zP_{\min}$ framework, we consider a simple modification of AMREST as follows:

(1) Initially, we build an approximately optimal Steiner tree spanning the source nodes by using, for example, the algorithm in Robins and Zelikovsky (2000). The AMREST procedure proceeds with that tree instead of the initialization in Line 3 of Algorithm 2. Denote the total energy cost of that tree by C^*. $z \geq 1$ is a given parameter.
(2) In the beginning of Loop in Line 4 of Algorithm 2, check if the total cost of the current tree exceeds zC^*.
(3) If yes, terminate the procedure and output the tree from the previous loop. If no, continue the loop.

Note that we begin with the approximately minimum cost C^* by initially building an approximately optimal Steiner tree. Due to Step (2), the algorithm will check if the total cost exceeds zC^* before adding a valid path to the tree. Thus, modified AMREST guarantees that the total energy consumption of the output tree is at most zC^*. Thus, the modified algorithm enables us to strike a balance between the minimum possible total cost and the minimum residual energy. Note that z can be adaptively determined over the rounds by using a method similar to that in Li et al. (2001).

5 ALGORITHM ANALYSIS

We first analyze the performance of AMREST. Let $R(T^*)$ be the optimal solution to **(P)** defined in Section 4. Denote $I_{\max}$ and $I_{\min}$ the maximum and minimum initial energy in $\mathcal{I}$ respectively.

Theorem 5.1. *For an arbitrary weighted graph with an arbitrary set of source nodes $S \in V$ and initial energy setting $\mathcal{I}$, the output T of AMREST spans S and satisfies*

$$R(T) \geq R(T^*) - (2\beta + \alpha + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}), \tag{8}$$

where $\Delta(G)$ denotes the maximum degree of G.

Proof. See Appendix D. □

THEOREM 5.2. *Suppose there exists a polynomial time algorithm such that the output T' satisfies*

$$R(T') > R(T^*) - \{\beta + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}\} \quad (9)$$

for an arbitrary weighted graph with an arbitrary set of source nodes S and initial energy setting $\mathcal{I}$. We then have that P=NP[1].

PROOF. See Appendix F. □

Theorem 5.1 implies that the gap of the minimum residual energy between AMREST and the optimal solution depends on the maximum degree and the initial energy setting. By combining Theorems 5.1 and 5.2, we can make the following statement. Let T denote the output tree from AMREST. If there exists a polynomial-time algorithm which outputs T' such that

$$R(T') - R(T) > \alpha + \beta \quad (10)$$

for arbitrary weighted graphs and arbitrary sets of source nodes and initial energy settings, then P=NP. In other words, any polynomial algorithm cannot outperform AMREST over all network graphs and initial energy configurations by more than $\alpha + \beta$, unless P=NP. The result shows that there may exist algorithms that outperform AMREST under some initial energy and network settings; however, it is hard to find an algorithm which universally outperforms AMREST over all systems by more than a constant. We obtain a stronger statement on the performance of AMREST in a "uniform" setting in which the initial energy settings and edge weights are identical. The following corollary is a direct consequence of Theorems 5.1 and 5.2.

COROLLARY 1. *Suppose $I_{max} = I_{min}$ and $\alpha = \beta$. Let T and T' denote the output of AMREST and an arbitrary polynomial-time algorithm, respectively. Unless P=NP, the following holds for any given graph and set of source nodes:*

$$R(T) \geq R(T') - 2\beta. \quad (11)$$

In other words, AMREST is near-optimal irrespective of the network size, and the optimality gap is a constant.

Next, we analyze the computational complexity of AMREST. During each outer loop (Lines 4–28), a bottleneck node b is improved and some candidate nodes are added to T. The residual energy of b is increased to at least $R(T) + \alpha$. As for a safe node or an added candidate node v, the residual energy of v is at least $R(T) + w(l_v^\dagger)$. Consider a pseudo-bottleneck $u \in V_p(T)$ whose residual energy has changed during the loop. Two cases may occur:

- The degree of u remains unchanged. In other words, among the edges incident on u, one edge in T is deleted, and one edge not in T is added. Thus, the residual energy of u is decreased by at most $w(l_u^\dagger) - \alpha$. From the definition of pseudo-bottleneck nodes, the residual energy of u is at least $R(T) + \alpha$.
- The degree of u is increased by one. This implies that one edge not in T but incident on u is added, and u remains a pseudo-bottleneck node after adding the edge. Thus, the residual energy of u is at least $R(T) + w(l_u^\dagger)$.

As a result, the residual energy of u is at least $R(T) + \alpha$. In summary, for any node whose residual energy is changed during an outer loop, its residual energy is at least $R(T) + \alpha$ after this outer loop. Thus, after $|V_b(T)|$ loops, the minimum residual energy is at least $R(T) + \alpha$. In other words, the minimum residual energy is increased by at least α after $|V_b(T)|$ loops. In the worst case, there are $O(n)$ bottleneck nodes in T and $R(T) = 0, I(b) = I_{\max}$, which results in $O(\frac{I_{\max}}{\alpha} n)$ outer loops.

[1]Here, P and NP denote complexity classes.

We then evaluate the complexity of each outer loop (Lines 4–28). The complexity of Line 5 and Line 6 are both given by $O(n)$. In Line 10, checking a valid path between u and v can be implemented as follows. Given $G = (V, L)$, let $G' = (V_c, L_c)$ be the subgraph induced by candidate nodes. Let F_i' be the ith component in G'. For a node $u \in V_b \cup V_p$, denote $C(u)$ the set of components in G' that connect to u; that is, $C(u) = \{i : \exists(u, x) \in L, \forall x \in F_i'\}$. There exists a valid path between u and v if $C(u) \cap C(v) \neq \emptyset$. In the worst case, the complexity of constructing $C(u)$ for all $u \in V_b \cup V_p$ is given by $O(|L|)$.

The complexity of Algorithm 3 is given by $O(n)$ since there are at most $n - 1$ *while* iterations. In Line 14, Algorithm 3 is invoked only once during each outer loop. In Line 18, we apply Tarjan's union-find algorithm to form the forest during each *while* iteration, which is of complexity $|L|A^{-1}(|L|, n)$ where $A^{-1}(|L|, n)$ is the inverse Ackermann's function (Fürer and Raghavachari 1994; Tarjan 1975; Cormen et al. 2009). Thus, the complexity of each outer loop is given by $O(|L|A^{-1}(|L|, n))$[2]. In conclusion, the complexity of AMREST is given by $O(\frac{I_{\max}}{\alpha}|L|nA^{-1}(|L|, n))$.

6 SIMULATION

In this section, we numerically evaluate the performance of AMREST. We use MATLAB software to simulate data aggregation and the associated energy consumption. The sensor nodes are uniformly and randomly deployed in a unit square area. The sink is located at the top right corner. We assume that there exists an edge between two nodes if the distance between the two nodes is no greater than 0.5. Each node has the same initial energy (100 Joules). The weight of each edge is independently and uniformly drawn from the range $[1, 10]$. We shall define the network lifetime used in the simulations as follows.

Definition 6.1. Consider a sequence of n-dimensional data vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$. We define the *lifetime* of the network as the number of retrieved data vectors until any node in the network depletes its energy.

We first consider the special case $s = 1$ (i.e., the case where $\mathbf{A}$ is dense). The SRS scheme with $s = 1$ reduces to the traditional CDG; hence, we set $m = k \log n$ where $k = \log n$. As a baseline scheme, we consider the MST scheme, in which an MST is used as the data-gathering tree during each round. We will make comparison with the MNL scheme (Liang and Liu 2007) as well. In the MNL scheme, the data-gathering tree initially consists of the sink only. During each iteration, a node, which has the maximum residual energy among all the nodes not in the tree, is added to the tree. The process is repeated until every node is included in the tree. Figure 5 compares the minimum residual energy for $s = 1$. The minimum residual energy plotted in the figure is normalized with respect to the initial energy. For a fixed value of n, we run the simulation 100 times and compute the average minimum residual energy after m measurements are collected. We observe that AMREST outperforms the other schemes, increasing the minimum residual energy by 11–45% and 2–7% compared to MST and MNL, respectively. We find that the minimum residual energy decreases as n increases. This is because m, the number of transmissions per node, increases as n increases. Furthermore, we observe that the minimum residual energy of AMREST and MNL decreases significantly more slowly in n than in MST. This is because both AMREST and MNL attempt to balance the residual energy among the nodes. However, AMREST results in a more gradual decrease in the minimum residual energy as the network size increases, as compared to MNL.

[2]Note that the inverse Ackermann function is an extremely slowly increasing function; e.g., it increases even slower than iterative logarithm (Boissonnat and Yvinec 1998). For any practical network size, $A(|L|, n)$ can be effectively regarded as a constant; e.g., $A(|L|, n)$ is at most 5 for any practical finite numbers $|L|$ and n (Boissonnat and Yvinec 1998).

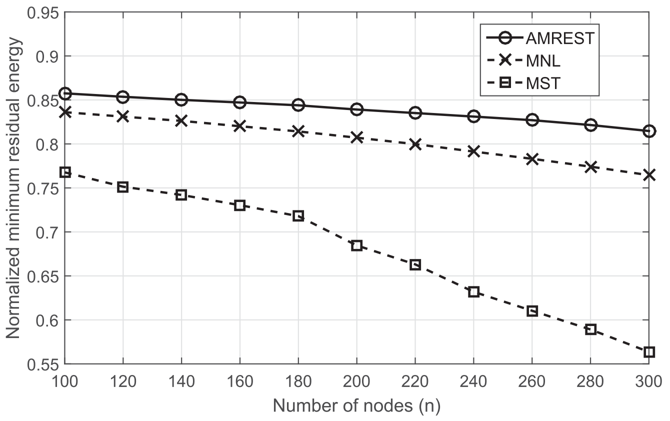


Fig. 5. Comparison of the minimum residual energy for the case $s = 1$.

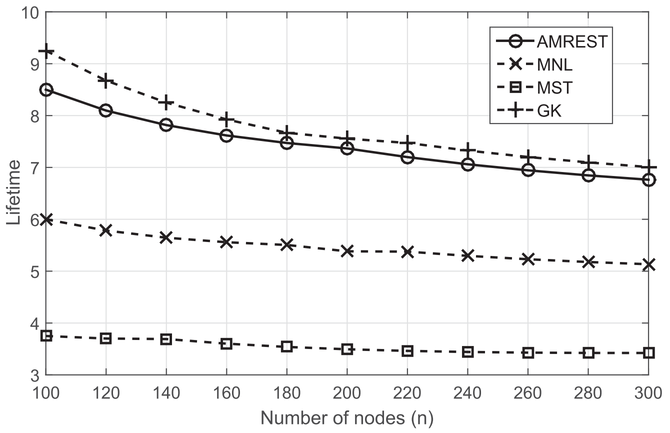


Fig. 6. Comparison of the lifetime for the case $s = 1$.

Figure 6 compares the network lifetime for $s = 1$. We observe that the lifetime of AMREST is very close to that of the GK algorithm (Stanford and Tongngam 2009), which was dedicated to directly maximize network lifetime with the approximation ratio $(1 - \epsilon)$. Note, however, that GK is applicable only for the case $s = 1$. We observe that AMREST increases the lifetime by up to 123% and 42% relative to MST and MNL, respectively. This shows that our algorithm, which iteratively maximizes the minimum residual energy over multiple rounds, can be also very effective in prolonging network lifetime.

Next, we compare the performance for $s < 1$. We will compare AMREST and the modified MST scheme defined as follows. In modified MST, a minimum spanning tree is first constructed. We then repeatedly delete leaves which are not source nodes until all the leaves are source nodes. Note that we were unable to compare AMREST with MNL or GK when $s < 1$ because those algorithms are designed for the case $s = 1$.

Figure 7 shows the distribution of the minimum residual energy for $s = 0.5$, $n = 100$, and $m = 10$. We observe that the minimum residual energy of AMREST is significantly greater than that of the modified MST. Furthermore, we observe that the variance of the distribution for AMREST is significantly smaller than that of modified MST. This implies that the deviation of the residual energy

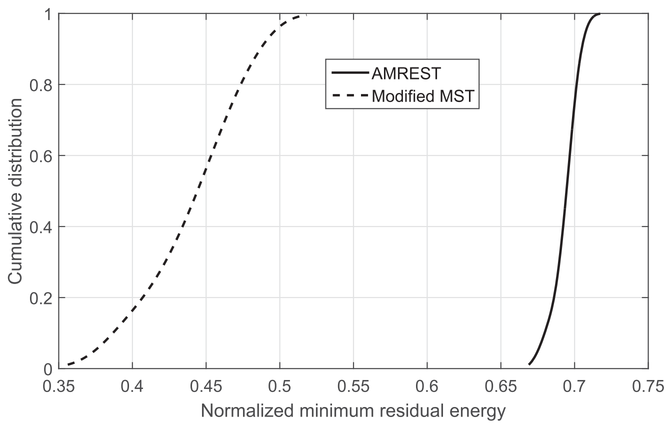


Fig. 7. Cumulative distribution function of the minimum residual energy. The minimum residual energy in the horizontal axis is normalized with respect to the initial energy.

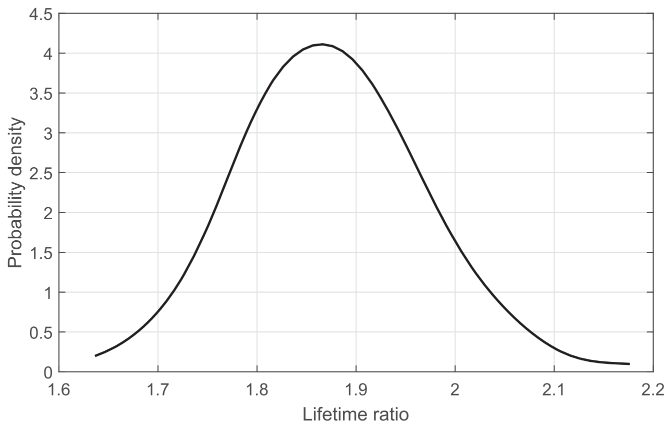


Fig. 8. Probability density function of the lifetime ratio.

with respect to the random selection of source nodes is smaller for AMREST. We hence conclude that the performance of AMREST is less sensitive to the choice of source nodes as compared to the modified MST. Figure 8 compares the distribution of the ratio of the lifetime between AMREST and the modified MST for $s = 0.5$, $n = 100$, and $m = 10$. We observe that the lifetime achieved by AMREST is at least 60% larger than that of the modified MST scheme. Furthermore, the gain of the lifetime reaches up to 115%.

Figures 9 and 10, respectively, compare the minimum residual energy and lifetime against the retention probability s. We set $n = 100, m = 10$ in Figure 9, and set $n = 100, m = 1$ in Figure 10. We observe that AMREST increases the minimum residual energy and lifetime by 22–37% and 102–188%, respectively. We also observe that the minimum residual energy and lifetime both decrease as s increases. This is because for larger s, a greater number of nodes are required to participate in data gathering during each round. Moreover, we observe that the lifetime of AMREST decreases faster in s for smaller values of s. A small s implies that the average number of source nodes in the network is small. Thus, many nonsource nodes must be included in the data-gathering tree in order to maintain the network connectivity. Now suppose we slightly increase s, which can be

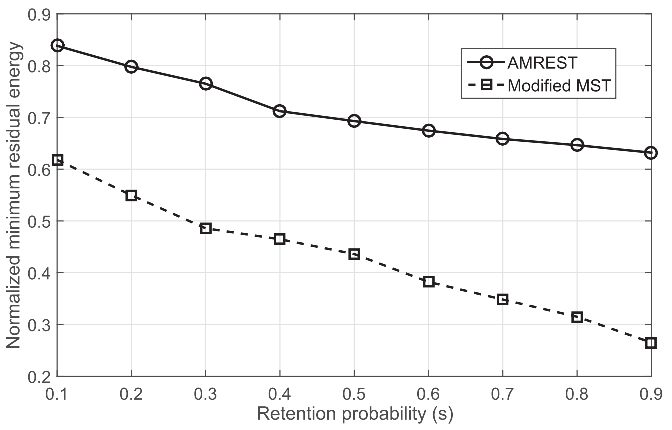


Fig. 9. Comparison of the minimum residual energy with varying s.

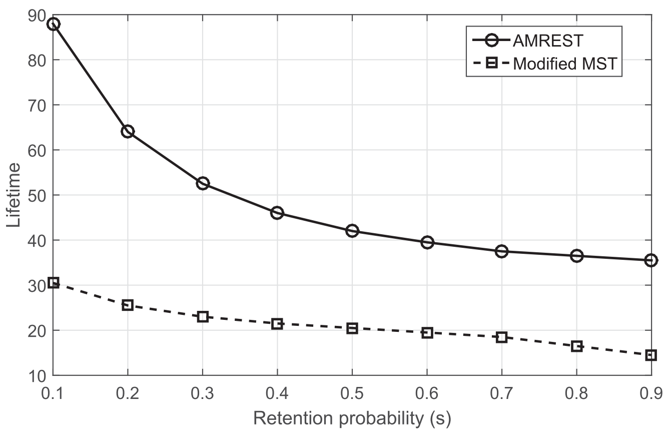


Fig. 10. Comparison of the lifetime with varying s.

hypothetically done by promoting some randomly selected nonsource nodes to new source nodes. If s is small, it is likely that these new source nodes are not adjacent to other source nodes. Thus, a relatively large number of non-source nodes need to be added to the data-gathering tree to keep the source nodes connected. This phenomenon will be prominent when s is small, which explains a steeper drop in the lifetime for smaller values of s.

Next, we evaluate the performance associated with the sparse matrices proposed in Theorem 2.2 and Theorem 2.3 (i.e., the matrices in Equations (4) and (6)). Figures 11 and 12 show the minimum residual energy and the lifetime, respectively. For the matrix in Equation (4), we set $h = 1/\sqrt{n}$, $k = \log n$, $s = 1/\sqrt{n}$ and $m = k^2 \log n = \log^3 n$. As for the matrix in Equation (6), we set $k = \log n$, $s = \frac{1}{\log n}$ and $m = k \log n = \log^2 n$. We observe that both of the minimum residual energy and lifetime under the same scheme decreases as n increases. This is because the average number of source nodes for each round is $sn = O(\sqrt{n})$ in Theorem 2.2 and $sn = O(n/\log n)$ in Theorem 2.3, both of which are increasing in n. Furthermore, m is also increasing in n. We observe that AMREST increases the minimum residual energy by 24–78% and 8–31% for the sensing matrices in

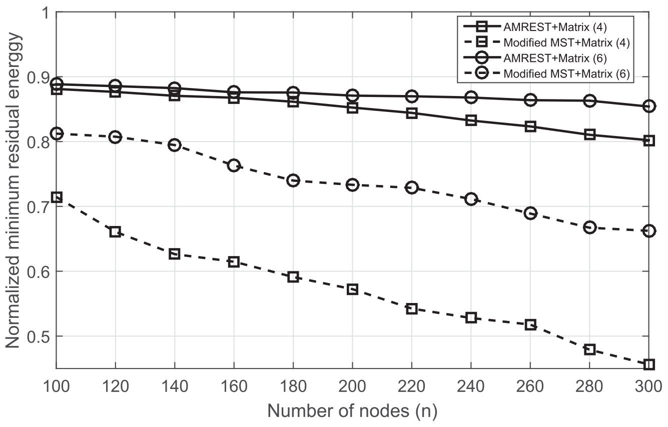


Fig. 11. Comparison of the minimum residual energy.

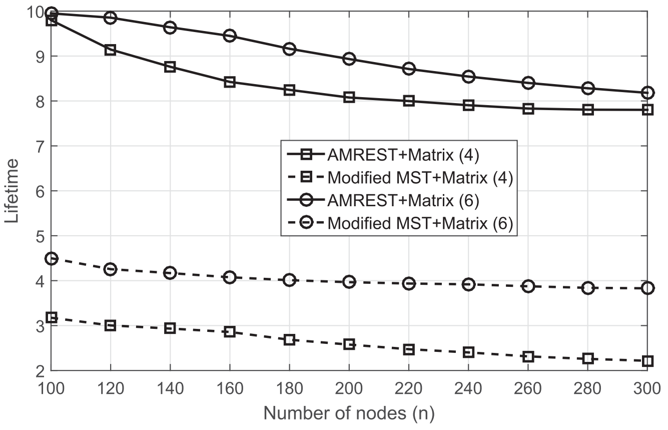


Fig. 12. Comparison of the lifetime.

Equations (4) and (6), respectively. The gains in network lifetime are 216–255% and 105–122% for the sensing matrices in Equations (4) and (6), respectively.

Finally, we evaluate the performance of the proposed algorithm under different network topology; specifically, with more sparse networks compared to the previous setup. We will shrink the communication range R to 0.15. In this case, the network is critically connected when $n = 100$; hence, the network topology results in a sparse graph. In the following simulations, we set the sparsity of the data vector $k = \sqrt{n}$ and the initial energy of each sensor to 1,000 Joules. Figure 13 compares the performance between cases $R = 0.15$ and $R = 0.5$ with $s = 0.5$. We observe that the performance under more sparse network is worse: If the network is barely connected, many non-source nodes must participate in the aggregation to keep the network connected. By contrast, if the network is very dense, the aggregation tree can be potentially formed by the source nodes only, and thus the aggregation requires relatively fewer nodes. Figure 14 compares the performance between cases $R = 0.15$ and $R = 0.5$ for different values of retention probability s with $n = 100$. We observe a trend in performance similar to that in Figure 13. In addition, the minimum residual energy decreases more slowly in s for larger R. This is because if the network is critically connected, there may exist nodes which are included in the tree irrespective of s in order to maintain the

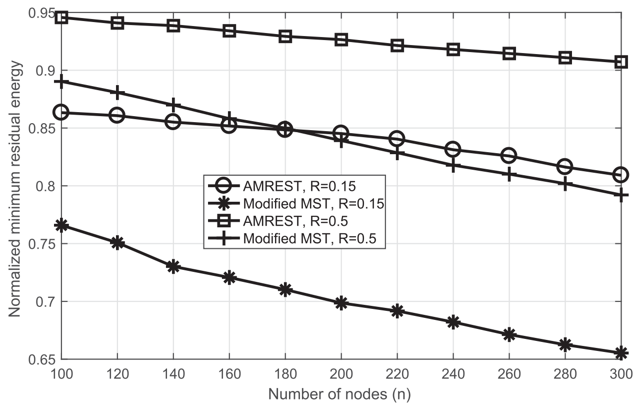


Fig. 13. Comparison of the minimum residual energy against the number of nodes n under different values of communication range R.

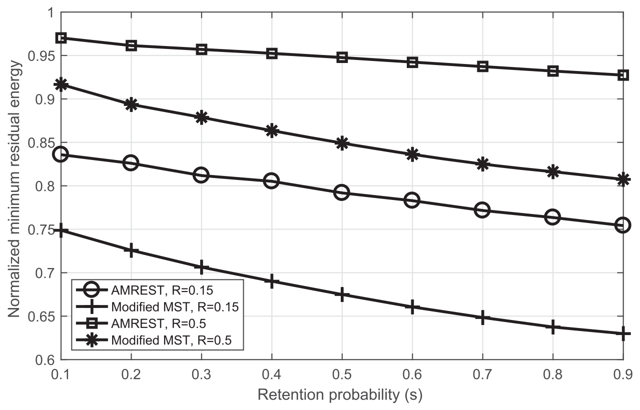


Fig. 14. Comparison of the minimum residual energy against the retention probability s under different values of communication range R.

network connectivity. Hence, we conclude that the size of tree, which directly affects the energy performance, is relatively less sensitive to s.

7 EXPERIMENTS

7.1 Setup

In this section, we present experimental results on the proposed algorithm. IoT-LAB is one of the largest public test beds for WSNs and Internet of Things (Adjih et al. 2015). It provides large-scale open-source experimental platforms for sensing, computing, and communications, allowing researchers to perform hands-on design and monitoring without the need to build individual platforms, which typically is a complex and time-consuming task. Command-line or Web-based interfaces to the platform are available, and users can launch experiments across six sites in which 2,728 wireless sensors are accessible. Users can collect sensor readings on temperature, atmospheric pressure, luminosity, and seismic data. The test bed also provides tools to monitor power, current, or voltage consumption and RSSI at the sensor nodes.

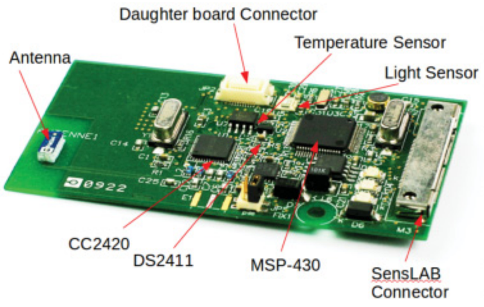


Fig. 15. WSN430 Open Node. Image from `www.iot-lab.info`.

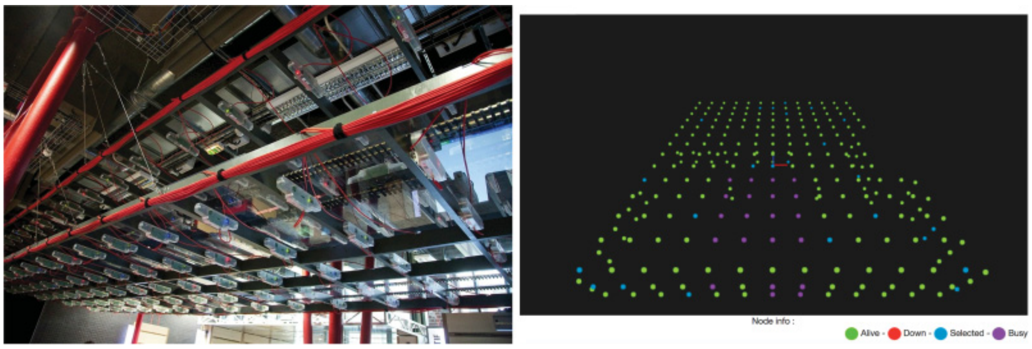


Fig. 16. (Left) Photograph of IoT-LAB environment of EuraTech located at Lille, France. (Right) 3-D sensor layout diagram of a testbed at Lille via Web interface for node selection and status check. Images from `www.iot-lab.info`.

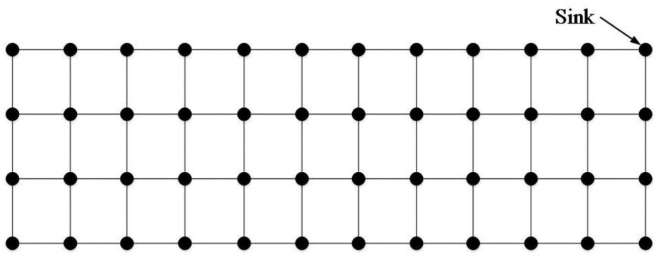


Fig. 17. Grid network for experiments.

Our experimental setup on IoT-LAB platform is as follows. For the hardware platform, we chose *WSN430 open node* by IoT-LAB which uses 16-bit MSP430F1611 microcontroller with 10kB RAM (Texas_Instruments 2009). The node is equipped with TI CC2420 chipset (Texas_Instruments 2006) for radio communications, which is a low-power 2.4GHz transceiver compatible with IEEE 802.15.4 PHY standard. Figure 15 is a photograph of the WSN430 open node. Figure 16 shows an example of the 3-D layout of the sensors in IoT-LAB. We consider a grid topology of dimension 4×12, shown in Figure 17; hence, there are a total of 48 nodes. The spacing between two nodes is 0.60m.

We implemented our test program on Contiki, which is a lightweight open-source OS designed primarily for IoT devices (Dunkels et al. 2004). For the MAC protocol, Contiki OS uses a

Table 2. Testbed Setup

Parameter	Value
microcontroller	MSP430F1611
radio chip	TI CC2420
PHY	IEEE 802.15.4@2.4 GHz
transmit power	0 dBm
OS	Contiki
MAC	X-MAC
protocol stack	Rime
topology	4×12 regular grid
node spacing	0.60m

compatibility version of X-MAC (Buettner et al. 2006), which is an asynchronous duty-cycled MAC protocol running on top of 802.15.4 PHY. The protocol stack for packet communications in our test bed is Rime (Dunkels 2007), which is a simplified communication stack with "thin" protocol layers suitable for sensor networks. Table 2 summarizes our test bed environment.

Similar to other aggregation methods such as CDG, the packet size in AMREST is fixed. In our experiment, the nodes generate and transmit hypothetical sensing data that is 8-byte long and is aggregated through the routing tree constructed by AMREST. Random sensing matrix $\mathbf{A}$ is generated in a distributed manner using PRNG, according to the method described in Section 3. For packet transmissions, we leverage the *reliable unicast* mechanism provided by the Rime protocol stack. This is because every data packet must be eventually delivered to the sink. Thus, occasional *retransmissions* of data packets may occur in the experiment. Later, we will discuss such retransmission issues in further detail.

Next, we will explain how we calculate the energy consumption of nodes based on measurements. There are two issues in real-time measurement of power consumption and residual battery levels. First, real-time measurements of energy expenditures are not readily available at WSN430 nodes at runtime; instead, the power consumption can be read from a log file at the server through an SSH interface. Second, WSN430 nodes are operated in DC power mode, hence it is of no use to read the residual battery level. Our workaround for these issues is as follows. Each node maintained variable `residual_level` which represents a hypothetical residual battery level of that node. At every transmission attempts, `residual_level` is decremented by variable `energy_per_tx`, which represents the energy expenditure per transmission. `energy_per_tx` is determined based on measurements as follows.

A measurement of a node's power consumption is shown in Figure 18. The plot shows the power consumption associated with periodic packet transmissions at every 3.5 seconds from a WSN430 transmitter to a receiver. For the simplicity of our power calculation, only the power consumption at transmission epochs (marked peaks in Figure 18) will be counted as the actual transmit (TX) and receive (RX) power expenditures.[3] At each transmission, we will randomly generate `energy_per_tx` from the empirical distribution of TX power consumption. The distribution is obtained from measurements such as those in Figure 18. A similar procedure is done for estimating RX power consumption (e.g., by defining variable `energy_per_rx`).

[3]Perhaps due to some system activities associated with real-time systems (e.g., context switching), we observe periodic overshoots in power consumption in Figure 18. These activities cannot be controlled by test bed users. For simplicity, we will ignore these overshoots and the idle power. Specifically, we will assume that a perfect power control/sleep scheduling is in action; hence, power consumption other than packet TX/RX is negligible.

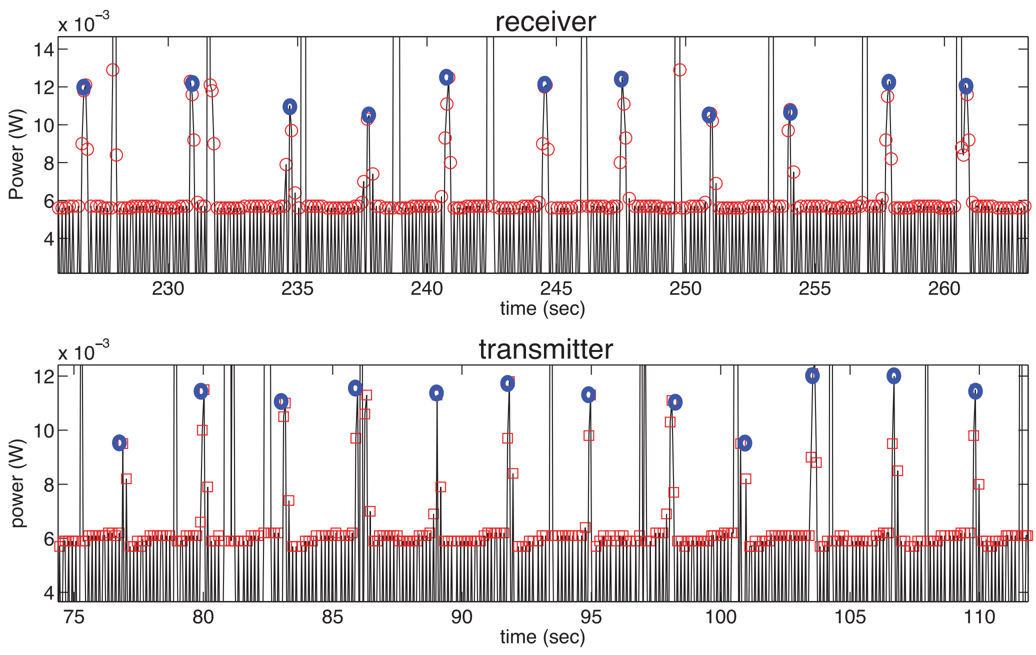


Fig. 18. Power consumption in periodic packet transmissions between a WSN430 transmitter and a receiver. The transmission epochs correspond to periodic peaks (marked) at every 3.5 seconds; the intermittent "overshoots" outside the plot are presumably due to other system activities.

We observe that, in Figure 18, the RX power consumption is similar to that for TX. This observation is consistent with our assumption that TX and RX power are approximately equal, hence it demonstrates the practicality of our assumption. It also matches the technical specification of TI CC2420 such that active RX consumes a current of 18.8mA whereas active TX consumes 17.4mA (Texas_Instruments 2006). Note that, in order to implement AMREST, we need to assign edge weights $w(u, v)$ for nodes u and v which represents the energy consumption per packet transfer. We will simply set weight $w(u, v)$ as the mean of `energy_per_tx` and `energy_per_rx` of nodes u and v.

7.2 Results

In our 4×12 grid network in Figure 17, the sink node is located at the top right corner. We set the sparsity parameter at $k = \sqrt{n}$, which makes the required number of measurements $m = \sqrt{n} \log n$ with $n = 48$ in our experiment. The initial energy level of the sensors is set to 0.35 Joules. In AMREST, the nodes should keep track of the other nodes' energy levels. In practice, unexpected expenditures of energy may occur (e.g., packet retransmissions). This causes a mismatch between the actual energy level of a node and the other nodes' belief on that node's energy level, whereas AMREST is run based on such beliefs on energy profiles. Thus, it is necessary that nodes occasionally update their residual energy information to the network. In order to prevent nodes from sending too frequent updates, we will make the following method for updates.

A node will notify its energy level to the network only if the actual energy level substantially deviates from the other nodes' beliefs. Specifically, a node will send updates only if (i) the error between the current energy level and belief exceeds τ_1%, and (ii) the number of rounds since the last update exceeds threshold τ_2. Note that the error in (i) is relative to the current energy level.

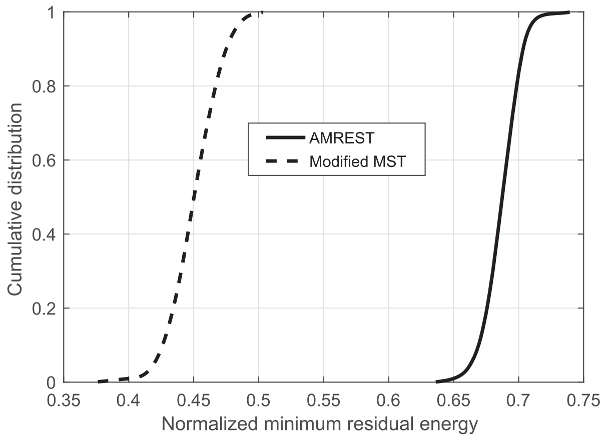


Fig. 19. Cumulative distribution function of the minimum residual energy.

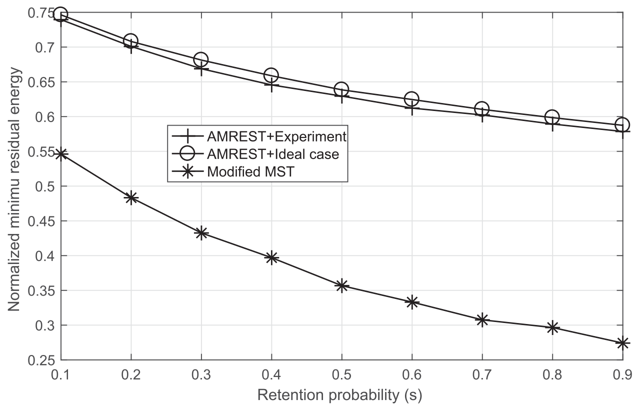


Fig. 20. Comparison of the minimum residual energy in experiments and the minimum residual energy in the ideal case.

The idea is that, due to (i), a node with large energy levels will be relatively insensitive to a few retransmissions; also, (ii) prevents the nodes with low energy levels from sending updates too often. Thus, our update method can adapt to nodes' residual energy levels without causing excessive communication overheads. In our experiment, we set $\tau_1 = 5\%$ and $\tau_2 = 10$ rounds.

Figure 19 shows the distribution of the minimum residual energy under the case $s = 0.2$. We make several observations that match our simulation results. For example, the energy levels under AMREST tend to be higher than those under the modified MST. Also the variance of energy levels under AMREST is smaller than that under the modified MST.

Next, we examine the robustness of AMREST against imprecise system information. Figure 20 compares the experimental results versus the ideal case, which we define as follows. We assume that perfect information on residual energy levels of the entire network is available at any time without any overhead. By contrast, in the experiments, AMREST may suffer from imprecise information due to, for example, unexpected retransmissions and stochastic amounts of energy consumption at other nodes. Exact information becomes available only intermittently by our update method, which incurs some communication overheads. Indeed, we observe that the performance

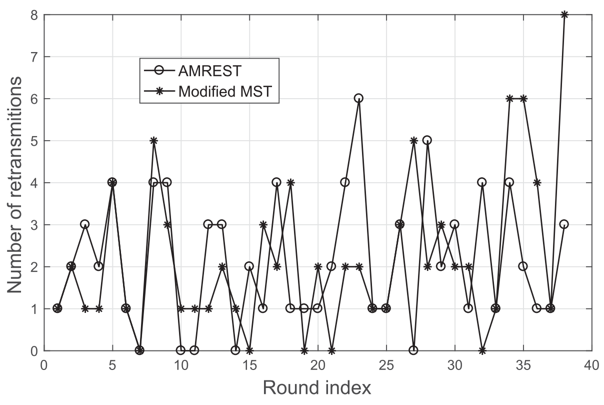


Fig. 21. Number of retransmissions over the rounds.

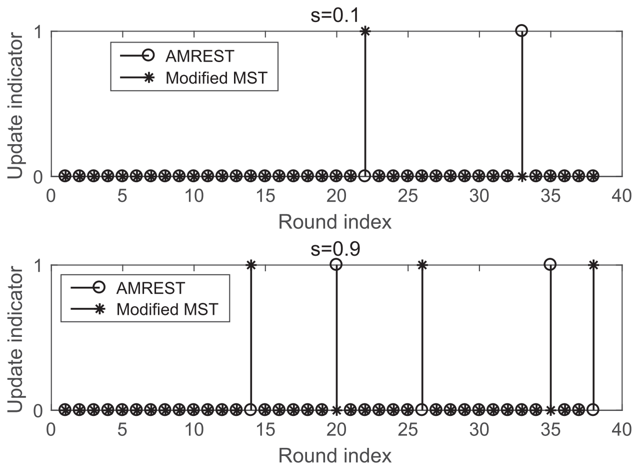


Fig. 22. Comparison of the round at which the update of residual energy occurs between $s = 0.1$ and $s = 0.9$. The value 1 represents the information of residual energy is updated.

under the experiment is only slightly worse than that under the ideal case. However, the gap is small because our method adaptively updates the information according to nodes' energy levels without causing much overhead. This demonstrates the robustness of our algorithm.

Next, we present some statistics on the updates of residual energy information. The information update is mainly caused by packet retransmissions. Figure 21 shows the number of retransmissions in the network over the rounds. The overall number of retransmissions is not high, perhaps due to short-distance and line-of-sight links among sensors in our experiment. Figure 22 shows the round at which the update of residual energy information occurs. For example, when $s = 0.1$, the information is updated at rounds 33 and 22 for AMREST and the modified MST, respectively. Overall, the updates under AMREST occur less frequently than those under modified MST, which results in a lower overhead for AMREST. This is because the energy levels under AMREST tend to be higher, and, according to our update method, the nodes with higher energy levels will update their information less often. We also observe that the updates become more frequent for larger values of s. This is because there are more source nodes in the network, which leads to higher energy consumption.

8 CONCLUSION

In this work, we proposed an energy-efficient data-gathering algorithm exploiting sparse random sampling of the sensor nodes. Our goal is to maximize the minimum residual energy at the retrieval of the original data vector. We proposed a greedy algorithm that optimizes data collection in a round-by-round manner. During each round, we randomly select source nodes according to a fixed probability. We proved that the problem is NP-complete and proposed the algorithm AMREST to approximately construct a maximum minimum residual energy Steiner tree for each round. Our analysis shows that AMREST is provably near-optimal; in case of uniform initial energy reserves and edge weights, the optimality gap achieved by AMREST is constant and does not depend on network size and topology. From the simulation results, we observe that AMREST is able to prolong the network lifetime even for the case $s = 1$, where the sensing matrix is dense. Also, when $s < 1$, our scheme significantly outperformed the baseline schemes in terms of both the minimum residual energy and network lifetime.

APPENDIXES

A PROOF OF THEOREM 3.1

We will use a reduction argument to prove NP-completeness. Suppose there exists hypothetical algorithm $\mathcal{P}$ which can solve $(\mathbf{P})$ in polynomial time. We will show that the existence of $\mathcal{P}$ would imply that one can solve the Hamiltonian path problem (Garey and Johnson 2002), which is NP-complete, in polynomial time. Note that a similar idea of the proof was used in Wu et al. (2010).

We will represent an instance of $(\mathbf{P})$ by a 4-tuple. Specifically, an instance is represented by $\mathcal{H} = (G, w, \mathcal{I}, S)$, where $G = (V, L)$ is a graph, w is the edge weight function, $\mathcal{I}$ is the set of initial energy, and $S \subseteq V$ is the set of source nodes. Suppose we are given an arbitrary graph $\mathcal{G} = (V, L)$. We will create an instance $\mathcal{H} = (G, w, \mathcal{I}, V)$ of $(\mathbf{P})$ as follows. We take $\mathcal{G}$ as the input graph G to $(\mathbf{P})$. We let $I(v_1) = \infty, I(v_2) = \cdots, I(v_n) = 1$, and set all the edge weights to a constant $\beta \in (0, \frac{1}{n})$. Let the set of source nodes $S = V$. Clearly, because the edge weights and initial energy reserves are identical across the network, a minimum-degree spanning tree in G would be the solution to $(\mathbf{P})$. This implies that, if there exists a Hamiltonian path, say T^*, then T^* is a solution to $(\mathbf{P})$. T^* yields the optimal value of $1 - 2\beta$ (assuming $n \geq 3$) because any vertex in T^* has the degree of at most 2.

Now suppose we solved $(\mathbf{P})$ using $\mathcal{P}$ and obtained the optimal value denoted by r^*. We can decide the existence of the Hamiltonian path in $\mathcal{G}$ by comparing r^* with $1 - 2\beta$ as follows. If $r^* \geq 1 - 2\beta$, it implies that there exists a Hamiltonian path which is a solution to $(\mathbf{P})$; if $r^* < 1 - 2\beta$, then there exists no Hamiltonian path because T^* would yield higher minimum residual energy given by $1 - 2\beta$, which is a contradiction.

In summary, if there exists $\mathcal{P}$, we can decide the existence of a Hamiltonian path for an arbitrary graph in polynomial time. Therefore, $(\mathbf{P})$ is NP-complete.

B PROOF OF LEMMA 4.1

We prove this lemma by induction on $iter\_res(u)$. Suppose $iter\_res(u) = 1$. This implies that u is added to the forest during the first *while* iteration (Line 9). Because all of the pseudo-bottleneck and bottleneck nodes are removed before *while* iterations start, both endpoints of p_1 are safe nodes. Note that p_i is the chosen valid path during the ith round. We can improve u by adding p_1 to T and deleting an edge in $T(p_1)$ that is incident on u. Suppose any pseudo-bottleneck node v with $iter\_res(v) \leq k$ can be improved. Consider u with $iter\_res(u) = k + 1$. Let $p_{k+1} = (u', \ldots, v')$. We have that $iter\_res(u') \leq k$ and $iter\_res(v') \leq k$. By the assumption, both u' and v' can be improved. After improving u' and v', we can improve u by adding p_{k+1} and deleting an edge in $T(p_{k+1})$ that is incident on u.

C PROOF OF LEMMA 4.2

We leverage the idea in Fürer and Raghavachari (1994) to prove this lemma. Consider a single outer loop of AMREST at Line 4. Suppose a bottleneck node b is improved by adding $p_k = (u, \ldots, v)$ and deleting an edge incident on b during the kth *while* iteration (Line 9). Suppose both u and v need to be unblocked before improving b. In the updated tree, each newly added node must be a candidate node because only the nodes in valid paths can be added to T. We will show that the added candidate nodes will have a degree of no more than two in the updated tree. This can be shown by proving the following three claims:

(a) In the procedure of unblocking u (or v), each added candidate node is involved in only one improvement.
(b) Unblocking u and v involves a disjoint set of candidate nodes.
(c) Any candidate node in p_k is *not* involved in the unblocking of u or v.

First, we prove Claim (a) by contradiction. Recall that, to unblock a pseudo-bottleneck node u in Algorithm 3, we make a sequence of improvements. Suppose that the improvements on u' and u'' are included in the unblocking procedure of u and that both of the improvements involve a candidate node h. Without loss of generality, we assume that $iter\_res(u) > iter\_res(u') = i > iter\_res(u'') = j$. This implies that both $p_i = (u_i, \ldots, v_i)$ and $p_j = (u_j, \ldots, v_j)$ contain the candidate node h. The pseudo-bottleneck node u'' is added to the forest before u' since $i > j$. Let F be the forest at the beginning of the ith iteration. Then u'' belongs to either component $F(u_i)$ or component $F(v_i)$. Without loss of generality, suppose u'' belongs to the component $F(u_i)$. This implies that u_j also belongs to $F(u_i)$, and $j \leq iter\_res(u_i)$ where the equality holds when $u'' = u_i$. Consider the valid path $p = (u_j, \ldots, h, \ldots, v_i)$, which is constructed by combining the subpath from u_j to h in p_j and the subpath from h to v_i in p_i. Obviously, p is also a valid path that connects the component $F(u_i)$ and the component $F(v_i)$ in F. Furthermore, we have that $iter\_res(u_j) < j \leq iter\_res(u_i)$, which contradicts the choice of the valid path p_i.

Second, we prove Claim (b) by contradiction. Suppose unblocking u and v both involve the same candidate node, say h. Assume that unblocking u involves the improvement on u' with $iter\_res(u') = i$, and unblocking v involves the improvement on v' with $iter\_res(v') = j$. Then, both $p_i = (u_i, \ldots, v_i)$ and $p_j = (u_j, \ldots, v_j)$ include h. Let F be the forest at the beginning of the kth iteration. Then we have that $u_i \in F(u), v_j \in F(v)$, and $i \leq iter\_res(u), j \leq iter\_res(v)$. Consider the path $p = (u_i, \ldots, h, \ldots, v_j)$ which is constructed by combining the subpath from u_i to h in p_i and the subpath from h to v_j in p_j. Obviously, p is a valid path that connects component $F(u)$ and $F(v)$. Furthermore, we have that $iter\_res(u_i) < i \leq iter\_res(u)$ and $iter\_res(v_j) < j \leq iter\_res(v)$, which contradicts the choice of p_k.

Third, Claim (c) can be proved in a similar way to Claim (b).

In summary, every added node will have a degree of no more than two in the updated tree. This implies that the energy consumption of a candidate node v added to the updated tree is at most $2w(l_v^\dagger)$. From the definition of the candidate nodes, the residual energy of v in the updated tree is at least $R(T) + w(l_v^\dagger)$.

D PROOF OF THEOREM 5.1

Given T, let $Q \subseteq V_p(T)$ be an arbitrary subset and $U := V_b(T) \cup Q$. Let $\mathcal{F}$ be the forest generated by removing U from T. Suppose G satisfies the condition that there exists no valid paths between different components in $\mathcal{F}$. In that case, we will show that the following holds:

$$R(T) \geq R(T^*) - \{2\beta + \alpha + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}\}. \tag{12}$$

We define a component in $\mathcal{F}$ which contains at least one source node as a *source component*. The set of source components in $\mathcal{F}$ is denoted by $\mathcal{F}_s$. We establish a useful inequality which relates T and $\mathcal{F}_s$ as follows.

Lemma D.1. *We have that*

$$\sum_{v \in U} D(T, v) \leq |\mathcal{F}_s| + 2|U| - 2. \tag{13}$$

For the proof of Lemma D.1, refer to Appendix E.

Consider the optimal data-gathering tree for the subgraph $(V - V_{nc}(T), L)$, which we denote by $T^\dagger = (V_{T^\dagger}, L_{T^\dagger})$. We observe that $T^\dagger$ satisfies the following constraints:

— All the leaves in $T^\dagger$ belong to S.
— $T^\dagger$ contains all the source nodes that belong to either a component in $\mathcal{F}_s$ or the set U.
— In $T^\dagger$, the source nodes that belong to different components in $\mathcal{F}_s$ are connected through some nodes in U. This is because, by assumption, G satisfies the condition that there are no paths through the nodes of $V - (U \cup V_{nc}(T))$ between any components in $\mathcal{F}$, and $T^\dagger$ does not contain nodes in $V_{nc}(T)$. Let $U^\dagger = U \cap V_{T^\dagger}$ be the subset of nodes in U which are used to connect the source nodes in $\mathcal{F}_s$ within the optimal tree $T^\dagger$.

Note that $T^\dagger$ contains all of the nodes in $U^\dagger$ and spans the source nodes in every component in $\mathcal{F}_s$. Because the nodes in $U^\dagger$ are disjoint from the nodes in the components in $\mathcal{F}_s$, and each component in $\mathcal{F}_s$ contains at least one source node, there must exist at least $|U^\dagger| + |\mathcal{F}_s| - 1$ edges in $T^\dagger$. Thus, we have that $\sum_{v \in U^\dagger} D(T^\dagger, v) \geq |U^\dagger| + |\mathcal{F}_s| - 1$. Consequently,

$$\frac{1}{|U^\dagger|} \sum_{v \in U^\dagger} D(T^\dagger, v) \geq 1 + \frac{|\mathcal{F}_s| - 1}{|U^\dagger|} \geq 1 + \frac{|\mathcal{F}_s| - 1}{|U|} \tag{14}$$

$$\geq \frac{1}{|U|} \sum_{v \in U} D(T, v) - 1 + \frac{1}{|U|}, \tag{15}$$

where the second inequality of Equation (14) follows from $|U^\dagger| \leq |U|$ and Equation (15) follows from Lemma D.1. Thus, we can bound $R(T^\dagger)$ as follows:

$$\begin{aligned} R(T^\dagger) &\leq \min_{v \in U^\dagger} E(T^\dagger, v) \leq \frac{1}{|U^\dagger|} \sum_{v \in U^\dagger} E(T^\dagger, v) \\ &= \frac{1}{|U^\dagger|} \sum_{v \in U^\dagger} \left[I(v) - \sum_{l \in L_{T^\dagger} \cap L_v} w(l) \right] \\ &\leq \frac{1}{|U^\dagger|} \sum_{v \in U^\dagger} [I(v) - \alpha D(T^\dagger, v)] \end{aligned} \tag{16}$$

$$\leq \frac{1}{|U^\dagger|}\sum_{v\in U^\dagger} I(v) - \alpha\left\{\left(\frac{1}{|U|}\sum_{v\in U} D(T,v)\right) - 1 + \frac{1}{|U|}\right\} \tag{17}$$
$$= \frac{1}{|U|}\sum_{v\in U}[I(v) - \alpha D(T,v)] + \alpha - \frac{\alpha}{|U|}$$
$$+ \frac{1}{|U^\dagger|}\sum_{v\in U^\dagger} I(v) - \frac{1}{|U|}\sum_{v\in U} I(v)$$
$$\leq \frac{1}{|U|}\sum_{v\in U}\left[I(v) - \sum_{l\in L_T\cap L_v} w(l)\right] + \alpha$$
$$+ \frac{1}{|U|}\sum_{v\in U}\left[\left\{\sum_{l\in L_T\cap L_v} w(l)\right\} - \alpha D(T,v)\right] + I_{\max} - I_{\min}$$
$$= \frac{1}{|U|}\sum_{v\in U}\left[I(v) - \sum_{l\in L_T\cap L_v} w(l)\right] + \alpha$$
$$+ \frac{1}{|U|}\sum_{v\in U}\left\{\sum_{l\in L_T\cap L_v}[w(l) - \alpha]\right\} + I_{\max} - I_{\min}$$
$$\leq R(T) + 2\beta + \alpha + \frac{1}{|U|}\sum_{v\in U}\left\{\sum_{l\in L_T\cap L_v}[\beta - \alpha]\right\} + I_{\max} - I_{\min}, \tag{18}$$
$$\leq R(T) + 2\beta + \alpha + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}, \tag{19}$$

where Equation (17) is due to Equation (15); Equation (18) holds since, by definition, $E(T,v) \leq R(T) + 2\beta$ holds for all $v \in U$; and Equation (19) follows from the definition of $\Delta(G)$.

Finally, let us consider the optimal tree $T^* = (V_{T^*}, L_{T^*})$ for the graph $G = (V, L)$. Two cases may occur:

— $V_{T^*} \cap V_{nc}(T) = \emptyset$. The optimal tree for G does not contain nodes in $V_{nc}(T)$. Because $T^\dagger$ is an optimal tree for $(V - V_{nc}(T), L)$, $T^\dagger$ is also optimal for G. We have that

$$R(T^*) = R(T^\dagger)$$
$$\leq R(T) + 2\beta + \alpha + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}.$$

— $V_{T^*} \cap V_{nc}(T) \neq \emptyset$. In this case, there exists a node that is not included in T but T^*. Then, for any $v \in V_{T^*} \cap V_{nc}(T)$, we have that

$$R(T^*) \leq E(T^*, v) \leq I(v) - 2\alpha, \tag{20}$$
$$\leq R(T) + 3w(l_v^\dagger) - 2\alpha \tag{21}$$
$$\leq R(T) + 3\beta - 2\alpha = R(T) + 2\beta - \alpha + (\beta - \alpha)$$
$$\leq R(T) + 2\beta + \alpha + (\beta - \alpha)\Delta(G),$$

where Equation (20) holds since v is not a source node and thus not a leaf node, and Equation (21) follows from the definition of $V_{nc}(T)$.

In summary, we have that $R(T^*) \leq R(T) + 2\beta + \alpha + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min}$.

Now we are ready to prove Theorem 5.1. Consider the forest variable F in AMREST, and suppose the algorithm has just terminated. According to the terminating condition, there exist no valid paths between different components in F. The theorem is proved by letting $Q \subseteq V_p(T)$ be the set of pseudo-bottleneck nodes that does not belong to F, and by letting $U = V_b(T) \cup Q$, and, finally, by applying Equation (12).

E PROOF OF LEMMA D.1

Given $T \in \mathcal{T}(S)$ and $\mathcal{F}$, we will generate a *hypothetical* tree T' as follows. We construct T' by removing all the nonsource components from T and by adding hypothetical edges, according to the following procedure:

(1) Pick a nonsource component $F_i \in \mathcal{F}_s^c$. Note that F_i consists of nonleaf nodes in T because all the leaf nodes belong to S.
(2) Suppose there are e edges in T between F_i and U. Let $\mathcal{V}_i$ denote the nodes in U which are the endpoints of those e edges. We first remove the e edges and then interconnect the nodes in $\mathcal{V}_i$ with $e - 1$ hypothetical edges. This operation will preserve connectivity and acyclicity of the nodes in U.
(3) Repeat the above for every nonsource component in $F_i \in \mathcal{F}_s^c$.

By construction, T' spans S and contains both hypothetical edges and the actual edges of T. In Step 2, the removal of e edges will decrease the sum of degrees of nodes in U by e. However, at the same time, the addition of $e - 1$ hypothetical edges increases the sum of degrees of nodes in U by $2(e - 1)$. Thus, the overall sum of degrees increases by $e - 1$ after Step 2. Because F_i contains no leaf nodes, we have that $e \geq 2$. This implies that the sum of degrees of the nodes in U does not decrease after the preceding procedure, that is, we have that

$$\sum_{v \in U} D(T', v) \geq \sum_{v \in U} D(T, v). \tag{22}$$

Figure 23 shows an example of building the hypothetical T'. $\mathcal{F}$ contains two non-source components, $\{v_2\}$ and $\{v_4, v_5\}$, which we denote by F_1 and F_2, respectively. For $F_1 = \{v_2\}$, there are 3

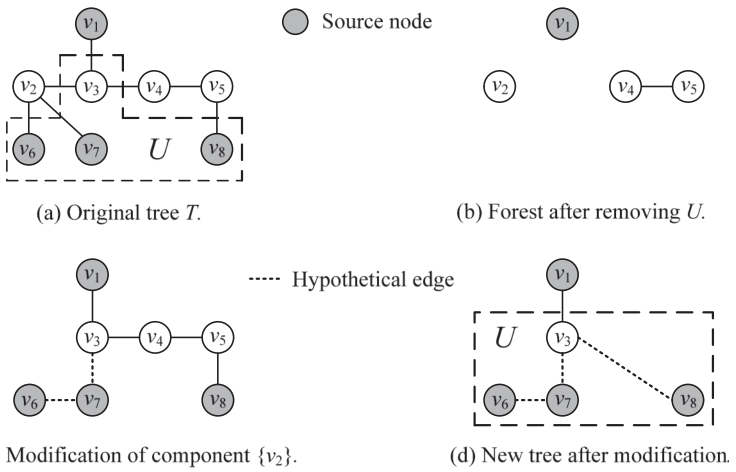

Source node
Hypothetical edge
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
U
(a) Original tree T.
(b) Forest after removing U.
(c) Modification of component $\{v_2\}$.
(d) New tree after modification.

Fig. 23. Example of the modification. Figure 23(a) shows the original tree T. Figure 23(b) shows the forest after removing U. There are three components: one source component $\{v_1\}$, and two nonsource components $F_1 := \{v_2\}$ and $F_2 = \{v_4, v_5\}$. Figure 23(c) shows the tree after the modification with respect to the non-source component $F_1 = \{v_2\}$. After the modification with respect to F_2, we obtain the hypothetical tree T', as shown in Figure 23(d).

edges in T between v_2 and U: (v_2, v_3), (v_2, v_6) and (v_2, v_7). We remove these 3 edges (Figure 23(b)) and add hypothetical edges (v_6, v_7) and (v_3, v_7) (Figure 23(c)). For $F_2 = \{v_4, v_5\}$, we remove edges (v_3, v_4), (v_5, v_8) and add the hypothetical edge (v_3, v_8) (Figure 23(d)). The new tree T' after the modification of every nonsource component is shown in Figure 23(d).

Given T', denote the number of edges (including hypothetical edges) by a such that both of their endpoints belong to U. Denote b the number of edges in T' which are incident on nodes in U. We have that $a = |U| - 1$ since T' is a tree. A similar counting argument shows that

$$b = |\mathcal{F}_s| + |U| - 1, \tag{23}$$

where $|\mathcal{F}_s|$ is the number of source components in $\mathcal{F}$. For example, in Figure 23(d), we have that $a = 3$ and $b = 4$. We observe that

$$\sum_{v \in U} D(T', v) = 2a + (b - a) = a + b. \tag{24}$$

Thus, we have that, from Equations (22), (23), and (24),

$$\begin{aligned} |\mathcal{F}_s| + |U| - 1 &= \sum_{v \in U} D(T', v) - (|U| - 1) \\ &\geq \sum_{v \in U} D(T, v) - (|U| - 1), \end{aligned}$$

which proves the lemma.

F PROOF OF THEOREM 5.2

We take a similar approach to the proof of Proposition 3 in Wu et al. (2010). Let algorithm $\mathcal{P}$ denote a polynomial time algorithm that outputs a tree T' with $R(T') > R(T^*) - (\beta + (\beta - \alpha)\Delta(G) + I_{\max} - I_{\min})$ for an arbitrary instance $\mathcal{H} = (G, w, \mathcal{I}, S)$. For the definition of the instance $\mathcal{H}$ of (**P**), refer to Appendix A. Based on the graph G, we will construct another instance $\mathcal{H}' = (G, w', \mathcal{I}', V)$ which has the same topology as $\mathcal{H}$ but different initial energy settings $I'(v_1) = I(v_1) = \infty$, $I'(v_2) = I'(v_3) = \cdots = I'(v_n) = 1$. Define the edge weight function: $w' : L \to \{\beta\}$ where $\beta \in (0, \frac{1}{n})$ is the maximum edge weight in the instance $\mathcal{H}$. The function w' implies that $\alpha = \beta$. Suppose we input $\mathcal{H}'$ to the algorithm $\mathcal{P}$ and obtain the output T'. Then, $R(T') > R(T^*) - \beta$ must hold by the assumption on T'.

Next, we show that G contains a Hamiltonian path if and only if ($\Leftrightarrow$) the output of algorithm $\mathcal{P}$ satisfies $R(T') > 1 - 3\beta$. In other words, we can check whether G contains a Hamiltonian path by inputting $\mathcal{H}'$ to algorithm $\mathcal{P}$ and checking whether $R(T') > 1 - 3\beta$. Obviously, constructing the instance $\mathcal{H}'$ and running algorithm $\mathcal{P}$ can be completed in polynomial time.

Sufficiency ($\Rightarrow$): Suppose G contains a Hamiltonian path $p = (V_p, L_p)$. We have that

$$R(p) = \min_{v \in V_p} \left[I'(v) - \sum_{l_v \in L_p} w'(l_v) \right] = 1 - 2\beta.$$

Moreover, by the optimality of T^*, we have that $R(T^*) \geq R(p) = 1 - 2\beta$. Finally, we have that

$$R(T') > R(T^*) - \beta \geq 1 - 3\beta.$$

Necessity ($\Leftarrow$): Assume that $R(T') > 1 - 3\beta$. This implies that $D(T', v) \leq 2$ for all $v \in V$. In order to see this, let us assume that $D(T', u) \geq 3$ for some $u \in V$. Then we obtain that $R(T') \leq E(T', u) = I'(u) - \beta D(T', u) \leq 1 - 3\beta$, which is a contradiction. Since T' is a spanning tree in G with maximum degree 2, it is exactly a Hamiltonian path in G. Therefore, the existence of such $\mathcal{P}$ implies that the Hamiltonian path problem can be decided in polynomial time, which completes the proof.

REFERENCES

Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and others. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on.* IEEE, 459–464.

I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. 2002. A survey on sensor networks. *IEEE Communications* 40, 8 (2002), 102–114.

Richard Baraniuk. Jul. 2007. Compressive sensing. *IEEE Signal Processing* 24, 4 (Jul. 2007), 118–120.

R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. 2008. A simple proof of the restricted isometry property for random matrices. *Constrive Approximation* 28, 3 (2008), 253–263.

Jean-Daniel Boissonnat and Mariette Yvinec. 1998. *Algorithmic Geometry.* Cambridge University Press.

F. Bouabdallah, N. Bouabdallah, and R. Boutaba. 2009. On balancing energy consumption in wireless sensor networks. *IEEE Transactions on Vehicular Technology* 58, 6 (2009), 2909–2924.

Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. 2006. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems.* ACM, 307–320.

Emmanuel J. Candès and Michael B. Wakin. Mar. 2008. An introduction to compressive sampling. *IEEE Signal Processing* 25, 2 (Mar. 2008), 21–30.

C. Cheng, C. Tse, and F. Lau. 2010. An energy-aware scheduling scheme for wireless sensor networks. *IEEE Transactions on Vehicular Technology* 59, 7 (2010), 3427–3444.

T. Cormen, C. Leiserson, R. Rivest, and C. Stein. 2009. *Introduction to Algorithms, 3rd ed.* Cambridge, MA: MIT Press.

Mario Di Francesco, Sajal K. Das, and Giuseppe Anastasi. 2011. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Transactions on Sensor Networks (TOSN)* 8, 1 (2011), 7.

D. Donoho. 2006. Compressed sensing. *IEEE Transactions on Information Theory* 52, 4 (2006), 1289–1306.

Adam Dunkels. 2007. Rime-a lightweight layered communication stack for sensor networks. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands.*

Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. 2004. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on.* IEEE, 455–462.

D. Ebrahimi and C. Assi. 2014. Compressive data gathering using random projection for energy efficient wireless sensor networks. *Ad Hoc Networks* 16 (2014), 105–119.

Y. Eldar and G. Kutyniok. 2012. *Compressed Sensing: Theory and Applications.* Cambridge University Press.

M. Fürer and B. Raghavachari. 1994. Approximating the minimum-degree Steiner tree to within one of optimal. *Journal of Algorithms* 17, 3 (1994), 409–423.

M. Garey and D. Johnson. 2002. *Computers and Intractability.* W.H. Freeman.

A. Gilber and P. Indyk. Jun. 2010. Sparse recovery using sparse matrices. *Proceedings of the IEEE* 98, 6 (Jun. 2010), 937–947.

Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (2013), 1645–1660.

J. Haupt, W. Bajwa, M. Rabbat, and R. Nowak. 2008. Compressed sensing for networked data. *IEEE Signal Processing* 25, 2 (2008), 92–101.

M. Jongerden, A. Mereacre, H. Bohnenkamp, B. Haverkort, and J. Katoen. Aug. 2010. Computing optimal schedules of battery usage in embedded systems. *IEEE Transactions on Industrial Informatics* 6, 3 (Aug. 2010), 276–286.

C. Karakus, A. Gurbuz, and B. Tavli. 2013. Analysis of energy efficiency of compressive sensing in wireless sensor networks. *IEEE Sensors Journal* 13 (2013), 1999–2008.

Ping Li, Cun-Hui Zhang, and Tong Zhang. 2013. Sparse recovery with very sparse compressed counting. *arXiv:1401.0201* (2013).

Ping Li, Cun-Hui Zhang, and Tong Zhang. 2014. Compressed counting meets compressed sensing. In *Proceedings of the 27th Conference on Learning Theory.* 1058–1077.

Qun Li, Javed Aslam, and Daniela Rus. 2001. Online power-aware routing in wireless ad-hoc networks. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking.* ACM, 97–107.

W. Liang and Y. Liu. 2007. Online data gathering for maximizing network lifetime in sensor networks. *IEEE Transactions on Mobile Computing.* 6, 1 (2007), 2–11.

Stephanie Lindsey, Cauligi Raghavendra, and Krishna M. Sivalingam. 2002. Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel Distribution Systems* 13, 9 (2002), 924–935.

C. Luo, F. Wu, J. Sun, and C. Chen. 2009. Compressive data gathering for large-scale wireless sensor networks. In *Proceedings of the International Conference on Mobile Computer Networking (MobiCom).* Beijing, China, 145–156.

Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE* 16, 1 (2014), 414–454.