Maximizing Lifetime Vector in Wireless Sensor Networks

Liang Zhang, Shigang Chen, *Senior Member, IEEE, Member, ACM*, Ying Jian, Yuguang Fang, *Fellow, IEEE, Member, ACM*, and Zhen Mo

***Abstract*—Maximizing the lifetime of a sensor network has been a subject of intensive study. However, much prior work defines the network lifetime as the time before the first data-generating sensor in the network runs out of energy or is not reachable to the sink due to network partition. The problem is that even though one sensor is out of operation, the rest of the network may well remain operational, with other sensors generating useful data and delivering those data to the sink. Hence, instead of just maximizing the time before the first sensor is out of operation, we should maximize the lifetime vector of the network, consisting of the lifetimes of all sensors, sorted in ascending order. For this problem, there exists only a centralized algorithm that solves a series of linear programming problems with high-order complexities. This paper proposes a fully distributed algorithm that runs iteratively. Each iteration produces a lifetime vector that is better than the vector produced by the previous iteration. Instead of giving the optimal result in one shot after lengthy computation, the proposed distributed algorithm has a result at any time, and the more time spent gives the better result. We show that when the algorithm stabilizes, its result produces the maximum lifetime vector. Furthermore, simulations demonstrate that the algorithm is able to converge rapidly toward the maximum lifetime vector with low overhead.**

***Index Terms*—Network lifetime, sensor networks.**

I. Introduction

WE STUDY the problem of maximizing the lifetime of long-term low-rate monitoring sensor networks that collect data from fields for ecosystem study, environmental monitoring, seismic measurement, etc. Such sensor networks are designed to gather tens of thousands of data points from each selected location over a period of weeks or months. Battery-powered sensor nodes are limited in computation capability, memory space, communication bandwidth, and, above all, energy supply. A sensor network cannot carry out its task after the nodes' energy is exhausted. Hence, maximizing the operational lifetime of a sensor network is critical.

The lifetime maximization problem has received a lot of investigation in the past under different definitions of lifetime for a sensor network. Much prior work tries to maximize the time before the first sensor in the network runs out of energy [1]–[8]. Others maximize the time before the energy of the first replay node is depleted [9], the time before the first loss of coverage [10], [11], or the time before the network is partitioned [12]. Also related are the studies that prolong the average node lifetime [13] or the time before a certain percentage of nodes run out of power [14], although they do not maximize the minimum lifetime in the network.

In reality, the operational lifetime of the network is not limited to the smallest lifetime of all nodes. When one sensor is out of operation or a few sensors are partitioned from the sink, the rest of the network can still work, as long as useful data generated by other sensors can reach the sink. We believe the lifetime of a sensor network should include the lifetimes of all sensors that produce useful data. A sensor's lifetime is the duration from the time when it begins to generate the first data packet to the time when it generates the last packet that is deliverable to the sink. The network lifetime can be defined as the vector of all sensors' lifetimes sorted in ascending order, which is called the *lifetime vector*. The value of the lifetime vector is determined by the nodes' forwarding policies that specify how packets are forwarded from the sensors through the network to the sink. More specifically, for every node, its forwarding policy specifies the proportion of packets that should be forwarded on each outgoing link toward the sink.

Hou *et al.* [15], [16] define the problem of maximizing a sensor network's lifetime as to find the packet forwarding policies for all nodes that collectively produce the lexicographically largest lifetime vector, called the *maximum lifetime vector*. In less precise terms, it first maximizes the smallest lifetime of all nodes, then maximizes the second smallest lifetime of all nodes, and so on. Hou *et al.* show that this problem can be modeled as a series of linear programming (LP) problems. After solving the LP problems, the sink uploads the optimal packet forwarding policies to the sensors. Based on its forwarding policy, each sensor forward its packets. Such a solution is, however, a centralized one. It requires solving $O(|N|)$ LP problems of size $O(|E|)$, where $|N|$ is the number of sensors in the network, $|E|$ is the number of links, and LP has high-order polynomial complexity. The computation overhead can be prohibitively high for large sensor networks that need to be operational soon after deployment. Collecting the complete information about the network and uploading the complete forwarding policies to all nodes require significant amount of transmissions in the network, particularly for nodes around the sink. To avoid these problems, a distributed algorithm that spreads the overhead evenly on all nodes becomes important.

This paper presents the first distributed solution for the problem of maximizing the lifetime vector of a sensor network.

Manuscript received September 10, 2011; revised July 19, 2012; accepted September 04, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Liu. Date of publication December 11, 2012; date of current version August 14, 2013. This work was supported in part by the US National Science Foundation under Grants CPS 0931969 and CNS 0916391.

L. Zhang is with Juniper Networks, Sunnyvale, CA 94089 USA (e-mail: liangz@juniper.net).

S. Chen and Z. Mo are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: {sgchen, zmo}@cise.ufl.edu).

Y. Jian is with Google, Mountain View, CA 94043 USA (e-mail: yingj@google.com).

Y. Fang is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: fang@ece.ufl.edu).

Digital Object Identifier 10.1109/TNET.2012.2227063

Our strategy is to design a distributed progressive algorithm that works in a series of iterations. Each iteration produces a lifetime vector that is better than the lifetime vector from the previous iteration. The sequence of lifetime vectors produced from the iterations approaches to the optimal solution. A distributed progressive algorithm is practically attractive because a result is available at any time and is getting better as more time is spent. We show that when the algorithm stabilizes, its result produces the maximum lifetime vector. We have performed thousands of simulation runs on random networks of various sizes and compared them to Hou's centralized algorithm as well as other related algorithms. The results demonstrate that our algorithm rapidly converges to the maximum lifetime vector and its overhead is small. For networks of thousands of nodes, it produces near-optimal results in 10–30 iterations—one iteration requires each node to transmit two small control messages. The algorithm scales well as its overhead increases slowly with respect to network size. When used as a centralized algorithm, it is two to three orders of magnitude faster than Hou's linear programming solution for random networks of thousands of nodes; the performance gap increases for larger networks. We also compare the proposed algorithm with other existing algorithms that maximize the smallest sensor lifetime in the network or perform minimum-power routing. The proposed algorithm produces much better lifetime vector.

The rest of this paper is organized as follows. Section II gives the network model and the problem statement. Section III lays down the theoretical foundation for our algorithm. Section IV proposes our distributed progressive algorithm for maximizing the lifetime vector. Section V gives a few illustrative examples. Section VI presents the simulation results. Section VII draws the conclusion.

II. Network Model and Problem Definition

A. Sensor Network Model

Let N be the set of sensor nodes, among which a subset S of nodes that are located in places of interest will generate data. They are called *data sources*. Other nodes in $N - S$ help to forward packets from data sources to the sink although they do not have to generate data themselves. Note that data sources will also forward packets from others if they are on the routing paths to the sink.

Let $g_i, i \in N$, be the *source rate* at which node i generates new data packets. $g_i > 0$ if $i \in S$; $g_i = 0$ if $i \notin S$. We assume that the source rates are set low enough to not cause congestion in the network. The sink may consist of multiple geographically dispersed base stations. We assume the base stations are connected to a data server. It makes no difference to which base station a data packet is routed.

Two nodes are neighbors if they can receive packets from each other (to support DATA/ACK exchange). There may be multiple routing paths from each node to the sink. Let D_i be the set of neighbors that node i use as the next hops to the sink. They are called *downstream neighbors* of node i. $\forall j \in D_i, (i, j)$ is called an *outgoing link* of i. Let U_i be the set of *upstream neighbors*, which use i as the next hop on their routing paths to the sink. $\forall k \in U_i$, (k, i) is called an *incoming link* of i. If i is a downstream neighbor of k, then k must be an upstream neighbor of i. Let $E = \{(i, j) \mid \forall i \in N, j \in D_i\}$. We call the graph consisting of all these links as the *routing graph* of the sensor network, which contains all routing paths from data sources to the sink. The routing paths should be acyclic.

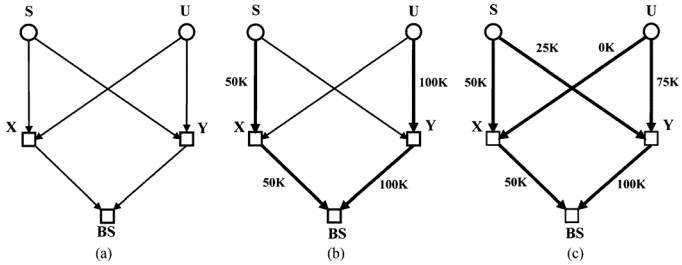


Fig. 1. (a) Meshy routing graph (b) Single-path routing (c) All-paths routing.

Any suitable routing protocol may be used to generate the routing paths. For example, if geographical routing [17]–[20] is used, D_i may consist of all or a selected subset of neighbors that are closer to the sink (based on Euclidean distance to the closest base station), and U_i may consist of all or a selected subset of neighbors that are further away from the sink. Numerous localization methods have been proposed to establish geographic locations among sensors; relative locations [21] or virtual locations [19] may also be used. Another alternative approach is to use hop counts, where $D_i(U_i)$ may consist of all or a selected subset of neighbors that are closer to (farther away from) the sink based on hop count. The hop counts from nodes to a sink can be learned from a distance-vector protocol or through a broadcast from the sink, in which the broadcast message carries a counter that is increased by one after each hop and, upon the first receipt of the message, each node takes the counter value of the message and forwards the message to neighbors except for the one from which it receives the message.

An example is given in Fig. 1(a), where s and u are data sources, x and y are nonsource nodes, and BS stands for a base station. Using geographical routing, we have $D_s = \{x, y\}$, $D_u = \{x, y\}$, $D_x = \{\text{BS}\}$, $U_x = \{s, u\}$, and so on. There are two routing paths from s to BS: $s \rightarrow x \rightarrow$ BS, and $s \rightarrow y \rightarrow$ BS. Similarly, there are two routing paths from u to BS. In a large, complex network, there may be numerous different paths from each data source to the sink.

Since sensors around the sink have to forward others' data, they are likely to exhaust their energy first and prevent the rest of the network from reaching the sink. The proposed algorithm in this paper deals with how to make the best use of available energy. It cannot make a physical energy bottleneck going away. To address this issue, we have to increase the energy availability around the sink by deploying more sensors there, using larger batteries, or perform in-network data aggregation to reduce the amount of traffic going through those nodes.

B. Lifetime, Volume, and Lifetime Vector

For an arbitrary data source s, we define its lifetime t_s to be the period from the time it begins generating data to the time when it runs out of energy or its packet can no longer reach the sink. The latter case happens when s is partitioned from the sink because other nodes on its routing paths run out of energy.

We define the *source volume* $\mu(i)$ of a node $i \in N$ to be the total number of data packets that are generated by i and delivered to the sink in its lifetime. Obviously, $\mu(i) > 0$ if $i \in S$,

and it is zero otherwise. The lifetime of a data source s and its volume are connected by the following formula:

$$t_s = \mu(s)/g_s. \tag{1}$$

For example, in Fig. 1(a), suppose s produces a new data packet per minute, i.e., $g_s = 1$. If the energy at s allows it to produce 50 K packets and the energy at nodes on its routing paths can deliver all these packets to BS, then t_s is 50 K min. When data sources share routing paths, their volumes (i.e., lifetimes) are dependent on each other. If we deliver more packets for one data source (so that it has a larger volume and can operate longer), another competing source may have to suffer.

Each data source has its own lifetime. If we put the lifetimes of all sources in a vector $\langle t_s, s \in S\rangle$ and sort them in ascending order, we have the *lifetime vector* of the sensor network. As we will demonstrate through an example shortly, different ways of forwarding packets in the network will result in different lifetime vectors. Our goal is to find the optimal way of forwarding packets in order to achieve the lexigraphically maximum lifetime vector, which is defined as follows.

Consider two feasible lifetime vectors, $T = \langle t_1, t_2, \ldots, t_{|S|}\rangle$ and $T' = \langle t'_1, t'_2, \ldots, t'_{|S|}\rangle$, respectively, where $t_i \le t_j$ and $t'_i \le t'_j$, $\forall i, j \in [1, |S|]$, $i < j$. We say $T = T'$ if $t_i = t'_i$, $\forall i \in [1, |S|]$, and $T > T'$ if $\exists k \in [1, |S|]$, $t_k > t'_k$ and $t_i = t'_i$ $\forall i \in [1, k-1]$.

Let Π be the set of all feasible lifetime vectors that can possibly be achieved in a sensor network. The maximum lifetime vector T^* is the largest in Π: $\forall T \in \Pi, T^* \ge T$. Our goal is to design a distributed algorithm to achieve the maximum lifetime vector. Intuitively, it is to first maximize the smallest lifetime among all sources, then maximize the second smallest, and so on.

C. Decoupling Routing

If we choose a single routing path for each data source, we will have a path selection problem. In Fig. 1(b), suppose x and y are energy bottlenecks in the network, and they have enough energy to forward 50 K packets and 100 K packets, respectively. If $g_s = g_u = 1$ packet per minute, the lifetime of s is 50 K min, and that of u is 100 K min, using the routing paths shown by bold arrows in Fig. 1(b). The lifetime vector is thus $\langle 50K, 100K\rangle$.

There is a serious problem: Once the optimal routing path for a source is determined, this information has to be sent to all nodes on the path so that they can create a routing entry for packets from that source. However, the scheme requires per-source routing entries, which may not be acceptable for memory-constrained sensors.

To address the above problem, we decouple routing from lifetime maximization. The lifetime maximization algorithm should be able to work with any underlying routing protocol. For example, if geographic routing is used, nodes simply forward packets to neighbors that are closer to the sink. No routing entries are needed. There may exist numerous paths from a source to the sink. Packets from the source should be able to use all these paths in order to fully explore the opportunities of lifetime maximization. An example is shown in Fig. 1(c), where we label each link with the number of packets sent over the link. Source s sends 50 K packets through x to the sink, and 25 K more through y. The resulting lifetime vector is $\langle 75K, 75K\rangle$, larger than what's achieved in Fig. 1(b). The problem is how we determine the number of packets transmitted over each link so that the maximum lifetime vector can be achieved in a complex network.

D. Volume Schedule

The *volume* $v(i, j)$ of a link (i, j) is defined as the number of packets transmitted on the link over the lifetime of the sensor network. All link volumes and source volumes together form a *volume schedule*. There are many possible volume schedules, but not all of them can be actually realized. A volume schedule is *feasible* only if it satisfies the following energy and volume conservation constraints.

Let e_i be the energy available at node i. Let α be the amount of energy that a node spends on receiving a data packet from an upstream neighbor, β_i be the amount of energy that node i spends on producing a new data packet, and γ_i be the amount of energy that node i spends on sending a packet. The *energy constraint* is given as follows:

$$\sum_{k\in U_i} \alpha \times v(k,i) + \beta_i \times \mu(i) + \sum_{j\in D_i} \gamma_i \times v(i,j) \le e_i \qquad \forall i \in N. \tag{2}$$

We say a node i is *exhausted* if

$$\sum_{k\in U_i} \alpha \times v(k,i) + \beta_i \times \mu(i) + \sum_{j\in D_i} \gamma_i \times v(i,j) = e_i.$$

That is, its energy will be used up after all packets are delivered in the network according to the volume schedule. The *volume conservation constraint* depends on the application model.

- If each node simply forwards all packets that it receives from upstream neighbors, then the number of packets sent out by a node will be equal to the number of packets it receives plus the number of packets that it generates locally, i.e.,

$$\sum_{j\in D_i} v(i,j) = \mu(i) + \sum_{k\in U_i} v(k,i) \qquad \forall i \in N. \tag{3}$$

- For periodic min/max/avg measurement among readings from data sources, each node will aggregate information from packets received from its upstream neighbors and then send out a single packet downstream. For example, suppose the application is to find out the minimum reading of all sensors. As the readings are sent from sensors toward the sink, each node will perform aggregation by finding the smallest value among its local reading and the readings received from upstream neighbors. It will only send this smallest value downstream

$$\sum_{j\in D_i} v(i,j) = \max\{\max_{k\in U_i}\{v(k,i)\}, \mu(i)\} \qquad \forall i \in N. \tag{4}$$

E. Packet Forwarding Based on Volume Schedule

The volume schedule specifies the number of packets from each source that can be delivered to the sink. According to (1), this determines the lifetimes of all sources, i.e., the lifetime vector. Moreover, the volume schedule also specifies how many of these packets are delivered on each link. This information tells how to forward packets through the network to reach the

sink, and thus gives an implementation of the lifetime vector. To implement a volume schedule, each node i simply does the following: 1) It generates new packets at its source rate g_i for $\mu(i)$ packets, and 2) it forwards the received packets to downstream neighbors in weighted round robin, using the volumes on the outgoing links as the weights. Therefore, the packet rates on the outgoing links are proportional to the volumes on the links. This is called the *volume-rate property*

$$\frac{r(i,j)}{v(i,j)} = \frac{r(i,j')}{v(i,j')} \qquad \forall j, j' \in D_i, v(i,j) \neq 0, v(i,j') \neq 0 \tag{5}$$

where $r(i,j)$ is the packet rate on link (i,j).

Our goal is to design a full distributed algorithm that finds a feasible volume schedule to produce the maximum lifetime vector. Once we find the volume schedule for the maximum lifetime vector, the nodes know how to forward their received packets based on the volumes of their adjacent links.

F. Additional Definitions

We give additional definitions that are needed by the proof of a theorem in Section III. The volume of a (directed) path is defined as the minimum volume of the links on the path. A path in the routing graph is called a *forwarding path* if its volume is greater than zero. Otherwise, it is called a *nonforwarding path*.

Node $s \in S$ is a *feeding source* of node $i \in N$ if there is a forwarding path from s to i. Furthermore, node s is a *restricted* feeding source of node i if there is an exhausted node on every forwarding path from s to i. Node s is an *unrestricted* feeding source of node i if there is no exhausted node on at least one forwarding path from s to i, where the *path* referred in this definition includes s but excludes i. Node s is a *potential source* of node i if it is not a feeding source of i, but there exists a nonforwarding path from s to i, and the path has no exhausted node.

III. Necessary and Sufficient Conditions for Maximizing Lifetime Vector

This section establishes the necessary and sufficient conditions for maximizing the lifetime vector in a theorem. We explain a basic technique used in the proof, called *volume shift*, which increases the lifetime (or volume) of a data source at the expense of another.

In Fig. 2, consider a feasible volume schedule where nodes s and w are two unrestricted feeding sources of node i. Suppose the volume of s is larger than the volume of w. Let P_1 and P_2 be two forwarding paths that do not have any exhausted node. We show that *the lifetime of an unrestricted feeding source* (w) *can be increased at the expense of the lifetime of another* (s). To do so, we simply let w send more packets down the path P_2 and let s send less down the path P_1, such that their combined number of packets to node i stays the same. More specifically, we decrease the source volume of s by a certain amount and decrease the volumes on the links of P_1 accordingly. We then increase the source volume of w by the same amount and increase the volumes on the links of P_2 accordingly. The amount of change should be small enough such that its addition on P_2 does not violate the energy constraint. The above operation is called a *volume shift* from s to w with respect to i. After volume shift, the volume schedule remains feasible and the lifetime of s is decreased, the lifetime of w is increased, while the lifetimes of all other sources remain unchanged. Obviously, in order to improve the lifetime vector, we shall always perform a volume shift from a node with a larger lifetime to a node with a smaller lifetime.

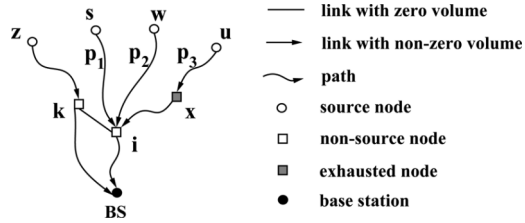


Fig. 2. There is no exhausted node on P_1 or P_2; nodes s and w are unrestricted feeding sources of i. There is an exhausted node x on P_3; node u is a restricted feeding source of i. There is no forwarding path from z to i; node z is a potential source of i.

Not only can a volume shift be performed between two unrestricted feeding sources, but also it can be performed from a restricted feeding source u to an unrestricted feeding source s, or from an unrestricted feeding source s to a potential source z, but not the other way around—more specifically: 1) *A volume shift cannot be performed from an unrestricted feeding source s to a restricted feeding source u* because we cannot add any additional volume to P_3 that has an exhausted node x; 2) *a volume shift cannot be performed from a potential source z to an unrestricted feed source s* because the volume of any path from z to i is zero, and thus nothing can be shifted out.

Theorem 1: A feasible volume schedule produces the maximum lifetime vector if and only if the following conditions are met.

1) There is an exhausted node on every path from any source to the sink.
2) All unrestricted feeding sources of a node must have the same lifetime, which should be no less than the lifetimes of the restricted feeding sources of the same node, and no greater than the lifetimes of the potential sources of the same node.

Proof: First, we prove that the conditions are necessary. If a feasible volume schedule does not meet either condition, we show that a larger lifetime vector can be produced by modifying the volume schedule. If the first condition is not true on a path P from a source s to the sink, we can improve the lifetime of s by increasing its source volume as well as the volume of P by a tiny amount, which results in a larger lifetime vector. Next, consider the second condition.

- If an unrestricted feeding source s has a greater lifetime than another unrestricted feeding source w of a node i, we can perform a volume shift (Fig. 2) from s to w such that the lifetime of w is slightly increased (but still below that of s), which results in a larger lifetime vector. Note that the volume shift only changes the lifetimes of two nodes, s and w.
- If an unrestricted feeding source s has a smaller lifetime than a restricted feeding source u, we can perform a volume shift from u to s to increase the lifetime vector.
- If an unrestricted feeding source s has a greater lifetime than a potential source z, we can perform a volume shift from s to z to increase the lifetime vector.

Second, we prove that the conditions are sufficient. The lifetime space, consisting of all feasible lifetime vectors, is convex and compact, which can be easily seen from the linear (or max) nature of the energy constraint (2) and the volume conservation constraint (3) or (4), as well as the lifetime definition (1).

Radunovic and Le Boudec showed that a max-min vector always exists in a convex, compact space. Moreover, it is unique and must be lexicographically largest in the space [22]. Hence, we only need to show that a feasible volume schedule that meets the two conditions produces the max-min vector, satisfying the following requirement: The lifetime t_s of one source s cannot be increased without decreasing lifetime t_w of another source w, for which $t_w \leq t_s$. We can show that the above requirement is indeed satisfied based on the following facts: The lifetime of any source is determined by its source volume; these two quantities are proportional to each other. Due to the first condition, each path from s to the sink has an exhausted node, so that its lifetime (i.e., source volume) cannot be increased without decreasing the lifetime (source volume) of another node, which must be a feeding source because a potential source does not send any competing packets and thus has no volume to decrease. Due to the second condition, the lifetimes of other feeding sources are not greater than the lifetime of s. If we increase the lifetime (source volume) of s, at least one of those nodes has to pay, by lowering its lifetime (source volume). □

The above theorem gives us some guideline for designing a distributed algorithm that generates a volume schedule to maximize the lifetime vector. Based on the first condition, data sources should aggressively set their source volumes to the highest values that their paths to the sink allow.

According to (1), the lifetime of a source s is equal to its volume $\mu(s)$ divided by its rate g_s. For unrestrictive feeding sources of a node, the second condition requires their volumes to be allocated in proportional to their rates, so that they will have the same lifetimes.

IV. Distributed Progressive Algorithm

After the sink is deployed, before the sources actually generate data packets and deliver them to the sink, a distributed progressive algorithm (DPA) is executed to produce a volume schedule, based on which the data packets will be forwarded.

A. Rate Schedule, Volume-Bound Distribution, Volume Schedule

DPA iteratively refines a volume schedule, $\{v(i,j),\ \forall (i,j) \in E,\ \mu(i),\ \forall i \in N\}$. To accomplish this task, we need to introduce a couple of auxiliary concepts. A *rate schedule*, $\{r(i,j),\ \forall (i,j) \in E\}$, is defined by assigning a rate value to each link in the routing graph. A *volume-bound distribution*, $\{b(i,j),\ \forall (i,j) \in E,\ b(i),\ \forall i \in N\}$, is defined by assigning a volume bound to each link and each node, where *volume bound* $b(i,j)$ specifies the maximum volume that is allowed on link (i,j) and *source volume bound* $b(i)$ specifies the maximum volume that is allowed to be generated from a node i. One of the key operations of DPA is to compute volume bounds. Volume bounds are constrained by the energy at the nodes. Essentially, we will convert energy constraints into volume bounds before the volumes are set.

DPA begins with an initial rate schedule that can be arbitrarily set. From the rate schedule and energy availability at the nodes, it computes a volume-bound distribution based on the second condition in Theorem 1. From the volume-bound distribution, it sets a volume schedule, based on which it will in turn derive a new rate schedule. This completes the first iteration of the algorithm. As shown in Fig. 3, in each subsequent iteration, DPA

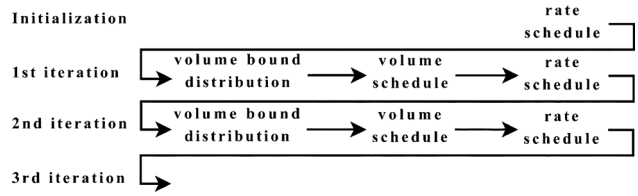


Fig. 3. Iterations of DPA.

repeats the above computation of a new volume-bound distribution (based on the rate schedule from the previous iteration), then a new volume schedule, and finally a new rate schedule. Each iteration produces a better volume schedule whose lifetime vector is larger than the previous one.

The rate schedule, volume-bound distribution, and volume schedule are stored and computed in a fully-distributed way. Each node only maintains the rates, volume bounds, and volumes of its adjacent links with a space complexity of $O(|D_i| + |U_i|)$. Because each directed link is shared by a pair of upstream-downstream nodes. Some properties of the link will be set by the upstream node and then sent to the downstream node, while other properties will be set by the downstream node and then sent to the upstream node. Details are given as follows.

Node i will set its *outgoing rates*, $r(i,j), j \in D_i$, by distributing the total incoming rate among the outgoing links. It will learn the *incoming rates*, $r(k,i), k \in U_i$, from upstream neighbors k who set those rates. (We want to stress that the link rates here are auxiliary variables used to facilitate the computation of volumes. They have nothing to do with the actual data-packet rates on the links at the time when DPA is executed. In fact, DPA can be executed at the beginning of the deployment before any data packets are transmitted.)

Node i will set its *outgoing volumes* $v(i,j)$ by distributing the total incoming volume among the outgoing links. It will learn the *incoming volumes* $v(k,i)$ from upstream neighbors k who set those volumes.

Node i will set its *incoming volume bounds* $b(k,i)$ by distributing its forwarding capacity among the incoming links. It will learn the *outgoing volume bounds* $b(i,j)$ from downstream neighbors j who set those bounds.

In the rest of the section, we will describe the details of DPA, which consists of *initialization phase* and *iterative phase* with each iteration having two steps. The first step computes volume bounds based on link rates. The second step determines link volumes from volume bounds and then computes new links rates, which sets the stage for the next iteration.

B. Initialization Phase

This phase arbitrarily sets up a rate schedule. The distributed computation for initializing link rates is described as follows. The sink broadcasts an INIT packet backward in the routing graph. When a node k receives INIT, it forwards the packet to its upstream neighbors. If k has no upstream neighbor (i.e., k is a leaf in the routing graph), it distributes its source rate evenly among its outgoing links, i.e., $r(k,i) \leftarrow \frac{g_k}{|D_k|}, \forall i \in D_k$, where "$\leftarrow$" is the assignment operator. Node k then sends those outgoing rates to its downstream neighbors in a RATE packet. After a node i learns $r(k,i)$ in RATE packets from all upstream neighbors k, it first computes its outgoing rates as

$r(i,j) \leftarrow \frac{\sum_{k\in U_i} r(k,i)+g_i}{|D_i|}, \forall j \in D_i$, and then sends those rates to downstream neighbors j in a RATE packet. The initialization phase terminates when the sink receives RATEs from all neighbors. Intuitively, this phase begins with a wave of INITs traveling backward in the routing graph to all nodes, and the INIT packets are turned around at leaf nodes to form a reverse wave of RATEs that assign the initial rates of all links subject to the flow conservation constraint.

In total, at most $|N|$ INIT packets and $|N|$ RATE packets are transmitted. Each node i sends one INIT of size $O(1)$ and one RATE packet of size $O(|D_i|)$. The initialization phase completes within the maximum round trip time between the sink and any source in the network.

C. Iterative Phase—Step 1: From Rates to Volume Bounds

The first step of each iteration is to set volume bounds based on link rates. Each node i must appropriately set its incoming volume bounds, $b(k,i), \forall k \in U_i$, and the source volume bound, $b(i)$, such that it does not receive more packets than it is able to forward. We describe two *volume-capacity constraints* that the volume bounds must satisfy, and then present the distributed algorithm that sets all volume bounds.

First, a node i should not receive and forward more packets than the downstream neighbors can handle. If the application model is characterized by (3), then the combined incoming volume bound (set by i) should not exceed the combined outgoing volume bound (set by downstream neighbors)

$$\sum_{k\in U_i} b(k,i) + b(i) \le \sum_{j\in D_i} b(i,j) \tag{6}$$

where $b(i,j)$ is learned by i from j. If the application model is characterized by (4), then the constraint becomes

$$\max\{\max_{k\in U_i}\{b(k,i)\}, b(i)\} \le \sum_{j\in D_i} b(i,j). \tag{7}$$

Second, node i should not receive and forward more packets than its energy allows

$$\sum_{k\in U_i} \alpha \times b(k,i) + \beta_i \times b(i) + \sum_{j\in D_i} \gamma_i \times b'(i,j) \le e_i \tag{8}$$

where

$$b'(i,j) = \begin{cases} \left(\sum_{k\in U_i} b(k,i) + b(i)\right) \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')}, \\ \qquad \text{for application model (3)} \\ \max\{\max_{k\in U_i}\{b(k,i)\}, b(i)\} \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')}, \\ \qquad \text{for application model (4).} \end{cases}$$

As we have explained in Section III, the second condition of Theorem 1 requires that volume allocation should be made in proportion to the incoming rates (which must be adjusted for restricted feeding sources, as will be discussed shortly in Step 2). Hence, we have the following *rate-bound property*:

$$\frac{b(k,i)}{r(k,i)} = \frac{b(k',i)}{r(k',i)} = \frac{b(i)}{g_i} \quad \forall k,k' \in U_i \cup \{i\}, r(k,i)\neq 0, r(k',i)\neq 0, g_i \neq 0. \tag{9}$$

If $r(k,i)=0$, then $b(k,i)=0$. If $g_i=0$, then $b(i)=0$.

The distributed computation of Step 1 is described as follows. The sink begins the process of setting volume bounds after the rate initialization phase terminates (at the time when the sink receives RATEs from all upstream neighbors), or after Step 2 completes (at the time when the sink receives VOL_RATE packets from all upstream neighbors—to be described in Section IV-D). The sink sets its incoming volume bounds to be infinite and sends a BOUND packets to upstream neighbors, carrying the volume bounds of its incoming links. After a node i receives BOUNDs from all downstream neighbors $j \in D_i$ and learns all outgoing volume bounds $b(i,j)$, it sets the incoming volume bounds, $b(k,i), k \in U_i$, and its source volume bound $\mu(i)$ as large as possible, based on (9) subject to the constraints of (6)–(8). Node i then sends its incoming volume bounds to the upstream neighbors in a BOUND packet.

In total, $|N|$ BOUND packets are transmitted. Each node i only transmits one packet of size $O(|U_i|)$.

D. Iterative Phase—Step 2: From Volume Bounds to Volumes and Rates

Next we discuss how to set the volumes of all links based on the volume bounds from Step 1. We first discuss how the volumes should be assigned, and then present the distributed algorithm to do so. Each node i should set the outgoing volumes, $v(i,j), \forall j \in D_i$, and its source volume $\mu(i)$, subject to the following *bound constraint*:

$$\mu(i) \le b(i), \quad v(i,j) \le b(i,j) \qquad \forall j \in D_i, \forall i \in N. \tag{10}$$

The first condition of Theorem 1 requires us to set the source volume as high as possible. Hence, we should assign

$$\mu(i) \leftarrow b(i). \tag{11}$$

In addition to (10), outgoing link volumes are also subject to the volume conservation constraint in (3) or (4). A node cannot send more packets than it receives. If it does not receive enough incoming volumes, its outgoing volumes may have to be set lower than what the volume bounds allow. If the volume conservation constraint is (3), to satisfy this constraint, node i assigns its outgoing volumes as follows:

$$v(i,j) \leftarrow \left(\sum_{k\in U_i} v(k,i) + \mu(i)\right) \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \qquad \forall j \in D_i \tag{12}$$

where $v(k,i)$ is set by upstream neighbor k and learned by i from k. If the volume conservation constraint is (4), node i assigns the outgoing volumes to be

$$v(i,j) \leftarrow \max\{\max_{k\in U_i}\{v(k,i)\}, \mu(i)\} \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \qquad \forall j \in D_i. \tag{13}$$

First, we prove by induction that using (12) will satisfy the bound constraint (10). Consider the base case with $U_i = \emptyset$. By (12), (6), and the fact that $U_i = \emptyset$, we have

$$\begin{aligned} v(i,j) &= \mu(i) \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \\ &\le \sum_{j'\in D_i} b(i,j') \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \\ &= b(i,j). \end{aligned}$$

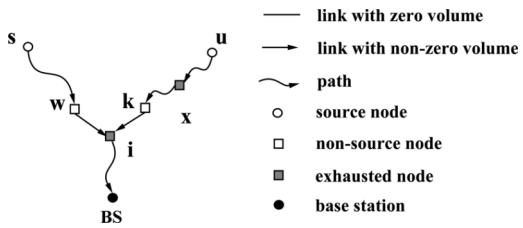


Fig. 4. There is no exhausted node from s to i; node s is an unrestricted feeding sources of i. There is an exhausted node x from u to i; node u is a restricted feeding source of i. The upstream bottleneck x may prevent source u from fully utilizing the volume bound set by i on link (k,i).

Next, we make the inductive assumption that $v(k,i) \leq b(k,i), \forall k \in U_i$, and prove the case when $U_i \neq \emptyset$. (This is a valid inductive assumption for a DAG routing graph, which has no loop for circular reasoning.) Together with (6) and (11), we have

$$\begin{aligned} v(i,j) &= \left(\sum_{k\in U_i} v(k,i) + \mu(i)\right) \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \\ &\leq \left(\sum_{k\in U_i} b(k,i) + b(i)\right) \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \\ &\leq \sum_{j'\in D_i} b(i,j') \frac{b(i,j)}{\sum_{j'\in D_i} b(i,j')} \\ &= b(i,j). \end{aligned}$$

By similar induction proof, it can be shown that using (13) will also satisfy the bound constraint (10). This result, together with (8), ensures that the assigned volumes satisfy the energy constraint required in (2)—to see this, one has to use the fact that $v(i,j) \leq b'(i,j)$ due to (12) and (13) and (10), where $b'(i,j)$ is defined in (8). Consequently, the resulting volume schedule is feasible.

After we set the link volumes, we assign new link rates based on the rate-volume property in (5), setting the stage for the next iteration. For application model (3)

$$r(i,j) \leftarrow \left(\sum_{k\in U_i} r(k,i) + g_i\right) \frac{v(i,j)}{\sum_{j'\in D_i} v(i,j')} \qquad \forall j \in D_i. \tag{14}$$

For application model (4)

$$r(i,j) \leftarrow \max\{\max_{k\in U_i} r(k,i), g_i\} \frac{v(i,j)}{\sum_{j'\in D_i} v(i,j')} \qquad \forall j \in D_i. \tag{15}$$

We have one additional issue that must be handled. As shown in Fig. 4, the volume bound assigned by i on link (w,i) for an unrestricted feeding source s will be fully utilized. However, the *volume bound* assigned by i on link (k,i) for a restricted source u may not be fully utilized due to an upstream bottleneck x that may set a tighter bound on the source volume of u. In this case, the *volume* $v(k,i)$, which is set by k and constrained by the limited upstream energy at x, is smaller than the volume bound $b(k,i)$. When this happens, we shall reduce $b(k,i)$ to match $v(k,i)$, and allow $b(w,i)$ to be larger, which will in turn allow s to have a larger source volume and thus a larger lifetime. Since volume bounds are set at Step 1 based on link rates, we can achieve the reduction of $b(k,i)$ by artificially reducing the rate $r(k,i)$.

More specifically, after the link rates are calculated based on (14) and (15), they may be reduced by multiplying a *reduction factor* $f(i)(\in (0,1])$, which has an initial value of 1 and is updated at each iteration as follows. Suppose node i is not a direct neighbor of the sink. If i is exhausted, i.e., $\sum_{k\in U_i} \alpha \times v(k,i) + \beta_i \times \mu(i) + \sum_{j\in D_i} \gamma_i v(i,j) = e_i$, or it was exhausted in one of the previous iterations, then it updates $f(i)$

$$\begin{aligned} B(i) &\leftarrow \sum_{j\in D_i} b(i,j)/f(i) \\ \mu(i) &\leftarrow \sum_{j\in D_i} v(i,j) \\ &\quad \times \frac{e_i}{\sum_{k\in U_i} \alpha \times v(k,i) + \beta_i \times \mu(i) + \sum_{j\in D_i} \gamma_i v(i,j)} \\ f(i) &= \frac{\mu(i)}{B(i)} \end{aligned} \tag{16}$$

where $B(i)$ and $\mu(i)$ are the would-be volume bound and volume on all outgoing links, respectively, if the rate reduction had not been preformed to reduce the outgoing volume bound in previous iterations. Clearly, the value of $f(i)$ will stabilize at an exhausted node i only when the volume $\sum_{j\in D_i} v(i,j)$ matches the bound $\sum_{j\in D_i} b(i,j)$. After updating $f(i)$, node i reduces the outgoing rates as follows:

$$r(i,j) \leftarrow r(i,j) \times f(i) \qquad \forall j \in D_i. \tag{17}$$

For the example in Fig. 4, both i and x will perform the above operation. When x does so, its rate reduction will propagate downstream, causing the reduction of $r(k,i)$, which in turn causes the reduction of $b(k,i)$ and the increase of $b(w,i)$.

The distributed computation of Step 2 for setting volumes/rates is a natural continuation of Step 1. After a node with no upstream neighbor receives BOUND (defined Step 1) from all downstream neighbors, it is able to assign its source volume by (11) and outgoing volumes by (12) and (13). It then updates the link rates by (14)–(17). After that, it sends the outgoing volumes/rates to the downstream neighbors by a VOL_RATE packet. After a node i receives VOL_RATE packets from all upstream neighbors k and learns $v(k,i)$, it is able to assign its source volume by (11), the outgoing volumes by (12) and (13), and the new outgoing rates by (14)–(17). It sends the outgoing volumes/rates to downstream neighbors in VOL_RATE. When the sink receives VOL_RATE from all upstream neighbors, it knows that Step 2 is completed.

Step 2 transmits $|N|$ packets. Each node i sends only one packet of size $O(|D_i|)$. Each iteration, including Steps 1 and 2, completes within the maximum round trip time between the sink and any source in the network.

E. Property

DPA carries out three computations to set volume bounds, volumes, and rates, respectively. We show that all three computations lead to better lifetime vectors.

First, consider the computation of volume bounds. The total forwarding capability of a node, which is determined by (6)–(8), is distributed as volume bounds based on the rate-bound property in (9), which essentially performs volume shift from feeding sources with larger lifetime (i.e., volume divided by rate) to those with smaller lifetime. Such volume shift increases the lifetime vector. The only problem is that a volume bound may not be fully turned into volume if there is an upstream exhausted node that sets a tighter volume bound. This problem is solved by rate reduction, which contiguously updates a reduction factor by (16) until the volume matches the bound.

Second, the volume assignments in (10), (12), and (13) are aggressive in the sense that they try to fully utilize all volume bounds, by setting the source volumes as high as possible and by forwarding all incoming volumes at each node.

Third, the rate reduction in (14)–(17) artificially decreases the link rates if the volume bounds are not fully turned into the volumes. In subsequent iterations, due to (9), decreased rates lead to decreased volume bounds on those links, allowing other links that can fully utilize their bounds to have higher volume bounds.

In summary, the volume bound computation performs volume shift from large-lifetime sources to small-lifetime sources; the volume computation and the rate reduction technique ensure that the volume bounds are fully utilized. Together, they improve the lifetime vector as DPA executes through its iterations. As the lifetime vector moves increasingly closer to its maximum value, the room for improvement becomes smaller and smaller. Our simulations will show that DPA converges rapidly.

Theorem 2: When DPA stabilizes the link volumes, the resulting volume schedule produces the maximum lifetime vector.

Proof: Let G be the subgraph consisting of all paths from sources to the first encountered exhausted nodes or to the sink if no exhausted nodes are encountered. Rate reduction has no impact on the link rates inside G. When link volumes are stabilized in G, link rates and volume bounds must also be stabilized because their linear interdependency in (5), (9), and (12)–(15). We prove by induction that

$$v(i,j) = b(i,j) \qquad \forall (i,j) \in G. \tag{18}$$

Consider the base case with $U_i = \emptyset$. Node i is not exhausted and hence

$$\beta_i \times \mu(i) + \sum_{j \in D_i} \gamma_i \times v(i,j) < e_i.$$

By (11)–(13), it can be rewritten as

$$\beta_i \times b(i) + \sum_{j \in D_i} \gamma_i \times b'(i,j) < e_i$$

where $b'(i,j)$ is defined in (8). Therefore, the real constraint for the value of $b(i)$ is (6) and (7). Since we should set $b(i)$'s value as large as possible, we have

$$b(i) = \sum_{j \in D_i} b(i,j).$$

By (12) and (13), we have $v(i,j) = b(i,j)$. Next, we make inductive assumption that $v(k,i) = b(k,i), \forall k \in U_i$, and prove the case when $U_i \neq \emptyset$. (This is a valid inductive assumption for a DAG routing graph, which has no loop for circular reasoning.) The proof is similar to the base case, except that $v(k,i)$ and $b(k,i)$ are included in the formulas.

To prove that the resulting volume schedule achieves the maximum lifetime vector, we have to show that the two conditions in Theorem 1 are satisfied.

First, we prove by contradiction that the first condition holds. If not, the sink will be in G. Consider an arbitrary link $(i, sink)$. We have proved earlier that $v(i, sink) = b(i, sink)$, which is not possible because $b(i, sink)$ is infinity.

Second, we prove that the second condition of Theorem 1 holds. Let s and w be two unrestricted feeding sources of node i. Let P_1 be a path from s to i that has no exhausted node. Let P_2 be a path from w to i that has no exhausted node. Both P_1 and P_2 are in G. Hence, for each link on the paths, its volume is equal to its volume bound. By (3)–(5), (9), (14), (15), and (18), the ratio of volume to rate is kept constant over the links of P_1 and equal to $\frac{\mu(s)}{g_s}$, the lifetime of s. Similarly, the ratio of volume to rate is kept constant over the links of P_2 and equal to $\frac{\mu(w)}{g_w}$, the lifetime of w. Moreover, these two ratios have to be the same when P_1 and P_2 intersect at i due to (9) and (18).

By assigning link rates in proportion to link volumes, (14) and (15) attempt to equalize the links' volume-to-rate ratios, which means, after each iteration, the rate is shifted away from downstream links with smaller volume-to-rate ratios to links with larger volume-to-rate ratios. The construction of the initial rate schedule ensures that every routing path is a forwarding path and there is no potential feeding source at the beginning. An unrestricted feeding source may become a potential feeding source by shifting its rate away from a path. When an unrestricted feeding source of i has a larger lifetime than other unrestricted feeding sources of i (due to routing paths that are not through i), the node's downstream link toward i will have smaller volume-to-rate ratio than its other links. Only in this case, due to (14)–(15), its rate will be contiguously shifted away from feeding i, and eventually turns itself into a potential source of i.

We have proved earlier the first condition of Theorem 1 that there is an exhausted node on every path from a source to the sink. Let G' be the subgraph consisting of all paths from sources to the last encountered exhausted nodes, and C' be the set of those last encountered exhausted nodes, which forms a *cut* of the network that separates the sink from all sources. When link volumes are stabilized in G', link rates and volume bounds must also be stabilized because their linear interdependency in (5), (9), and (12)–(15). We prove by contradiction that

$$v(i,j) = b(i,j) \qquad \forall (i,j) \in G'. \tag{19}$$

Suppose,$\exists (k,i) \in G'$,$v(k,i) < b(k,i)$. Based on the definition of G', any path from i to the sink must contain an exhausted node. By (8), (10), (12), (13), and the above assumption, we have

$$\sum_{k \in U_i} \alpha \times v(k,i) + \beta_i \times \mu(i) + \sum_{j \in D_i} \gamma_i \times v(i,j) < e_i.$$

In addition, from (3), (4), (6), and (7), we have: $\exists (i,j) \in E$, $v(i,j) < b(i,j)$. Following the same token, we know that j must not be exhausted, and it also has a downstream link whose volume is smaller than bound. Repeating the above reasoning, we can construct a path all the way to the sink without passing

an exhausted node, which contradicts with the fact that any path from i to the sink must contain an exhausted node.

By (3)–(5), (9), (14), (15), and (19), the ratio of volume to rate is kept constant on any path segment in G' that does not contain an exhausted node (which performs rate reduction). It is easy to see that the lifetime of a restricted feeding source u of a node i can only be equal to or smaller than that of an unrestricted source because, due to rate reduction, the radio of volume to rate will decrease when we traverse a path backward from i to u and cross an exhausted node. □

F. Termination Conditions

By Theorem 2, we shall terminate DPA when it has stabilized the link volumes, which can be detected by adding a flag that is transitively carried by the control messages. The flag is initially unset. A node sets the flag if it changes a link volume by an amount that is not negligibly small. It is up to the application requirement to decide on how small is negligible. The sink will stop if it does not receive a flag that is set. Alternatively, DPA may also be terminated artificially after a certain number of iterations, or when the resulting lifetime vector meets the application requirement.

G. Overhead

DPA has a flooding-based design. Flooding would be considered as inefficient for point-to-point tasks such as routing a packet from a source to a destination. However, for a global task such as building a volume schedule that involves every node and every link, flooding is the obvious choice that allows every node to participate in the distributed computation.

While the flooding design itself may appear noninnovative, the novelty of DPA is in the details that establishes the constraints and formulas for nodes to perform *localized operations*—iteratively computing their individual volume bounds from rates, volumes from volumes bounds, and rates from volumes with reduction—yet globally, as a *net outcome*, produce a progressively better lifetime vector, approaching to the optimal result.

During each iteration, node i sends two control packets, one BOUND of size $O(|U_i|)$ and one VOL_RATE of size $O(|D_i|)$. Upstream/downstream neighbors represent a subset of all nodes within the communication range of i. The packet size is limited when we choose a small number of upstream/downstream neighbors for routing purpose. We performed many simulations in Section VI, which shows that DPA converges quickly toward the optimal lifetime vector. To achieve no more than 5% deviation from the optimal, for networks of 1000 nodes, less than 25 iterations are needed. In addition, the overhead (i.e., number of iterations) increases slowly with network size.

If the network is designed to collect tens of thousands of data packets from each source, the small overhead of DPA (in tens of control packets per node) is negligible. If the number of iterations is predetermined, we can take the small energy consumption of DPA into account by reducing the nodes' energy (e_i) for an appropriate amount.

H. Centralized or Distributed

DPA can be trivially turned into a centralized one that is implemented at the sink if it has the complete information about the network. Comparing with the existing approach [15], [16] of solving $O(|N|)$ linear programs (each having high-order complexity), this centralized algorithm is remarkably efficient as it only needs to run for a small number of iterations (each having linear complexity $O(|E|)$) to get close-to-optimal results. However, we believe a distributed implementation is more relevant to sensor network deployment in general when the complete information of the network is not readily available.

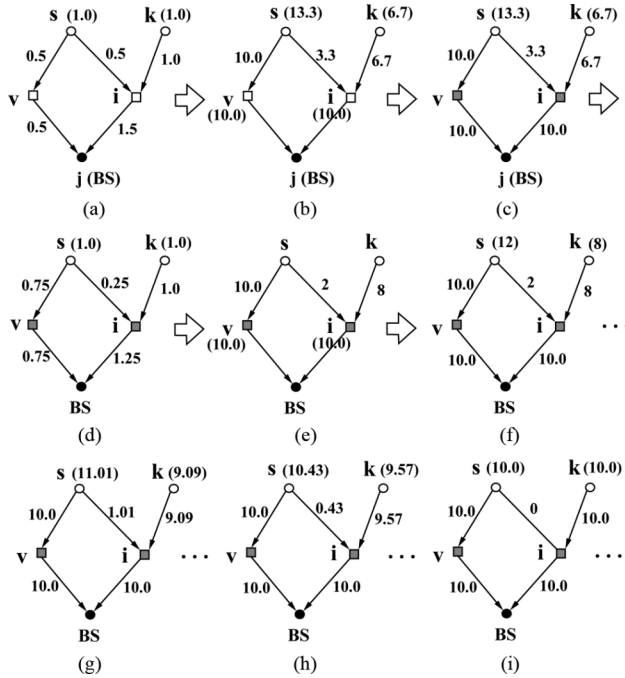


Fig. 5. Progressively improving lifetime vector. (Volume and volume bound are in thousands.) (a) Rate. (b) Volume bound. (c) Volume. (d) Rate. (e) Volume bound. (f) Volume. (g) Volume after fifth iteration. (h) Volume after 10th iteration. (i) Volume for max lifetime vector.

I. Topology Change

During network operation, unexpected node failures may change the topology of routing graph and reduce the lifetimes of affected sources. When node failure is detected, DPA may be reexecuted to produce an updated volume schedule. However, if small topology changes are frequent, invoking DPA frequently may cause considerable overhead. One way to reduce overhead is to perform DPA periodically or upon triggering after a certain number of node failures are detected. We will evaluate the impact of small topology changes on lifetime vector by simulation.

V. Examples

This section gives a few examples to illustrate how DPA works.

A. Iterations of DPA

In Fig. 5, data sources s and k generate new data at the same rate of one packet per minute. Suppose v and i are energy bottlenecks in the network. Each of them is able to forward 10 000 packets before its energy is exhausted. This maximum number of packets that a node is able to forward is called the *volume capacity*. Assume the energy available to sources s and k is plentiful.

The top two rows of plots in the figure show the rate schedule, the volume-bound distribution, and the volume schedule after

the first two iterations of the algorithm, respectively, while the third row presents only the volume schedules after the fifth and 10th iterations, as well as the optimal volume schedule that produces the maximum lifetime vector.

We now examine the first iteration of the algorithm. In Fig. 5(a), DPA begins with an initial rate schedule, where each node splits its rate (received from upstream or locally generated) evenly among its outgoing links. For example, $r(s,i) = 0.5$ and $r(k,i) = 1.0$. The source rate is shown in the parentheses beside the source node.

In Fig. 5(b), the volume capacity of a node is shown in the parentheses beside the node. It is determined either by the local energy if the node itself is a bottleneck (in the case of v or i), or by downstream energy bottleneck (in the case of s or k). A node should set the volume bounds on its incoming links in such a way that their sum is equal to the node's volume capacity and their values are proportional to the link rates. Because the ratio of $r(s,i)$ and $r(k,i)$ is 1:2, node i splits its volume capacity in the same ratio, 3.3 thousand packets for link (s,i) and 6.7 thousand packets for link (k,i). The volume capacity of s is the sum of the volume bounds on its outgoing links, which is 13.3 thousand packets.

Node s sets its source volume to 13.3 thousands, as shown in the parentheses beside the node in Fig. 5(c). Then, it sets the volumes on the outgoing links to be the same as the volume bounds. *Because the source rates of s and k are both 1, their lifetimes are equal to their source volumes in this example*.

Next we examine the second iteration of the algorithm. Based on the volume-rate property in (5), from the volume schedule in Fig. 5(c), we can derive the new rate schedule in Fig. 5(d) for the next iteration. Note that $r(s,i)$ is changed from 0.5 in the previous iteration to 0.25. It means that s has directed some of its traffic away from i to v in an effort to equalize its lifetime with k. From the new rate schedule, the algorithm calculates the volume-bound distribution in Fig. 5(e) and then derives the volume schedule in (f). It is evident that a volume shift from s to k with respect to i has happened when compared to the previous volume schedule. It results in a lifetime vector (8,12) larger than the previous one (6.7, 13.3).

The volume schedules after the fifth and the 10th iterations are shown in Fig. 5(g) and (h), respectively. Their lifetime vectors are (9.09, 11.01) and (9.57, 10.43), respectively, approaching to the maximum lifetime vector (10, 10), which is shown in Fig. 5(i).

B. Rate Reduction

Next, we show an example with a restricted feeding source in Fig. 6, where a new node k', whose limited energy allows it to forward only 5000 packets, is inserted between k and i.

The first iteration of the algorithm is illustrated by the top row of plots in the figure. Let us take a close look at Fig. 6(b). Node i's volume capacity is divided between (s,i) and (k',i) in proportion to the link rates. Hence, the volume bound $b(k',i)$ is 6.7 thousand packets. However, node k' cannot fully utilize this volume bound because its energy can only forward 5 thousand packets. Consequently, the volume bound $b(k,k')$ is only 5000 packets, and the source volume of k is also limited to 5000, as shown in Fig. 6(c). The volume bound on (k',i) is not fully utilized. Solving this problem requires rate reduction.

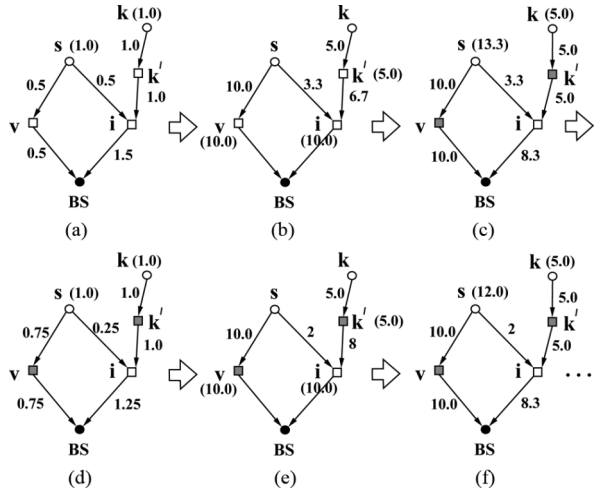


Fig. 6. Problem caused by restricted feeding sources. (Volume and volume bound are in thousands.) (a) Rate. (b) Volume bound. (c) Volume. (d) Rate. (e) Volume bound. (f) Volume.

To illustrate the necessity of rate reduction (described in Section IV-D), we first demonstrate what will happen without it. The second iteration of the algorithm is illustrated by the next row of plots. In Fig. 6(d), s directs some of its traffic away from link (s,i), as we have explained in the previous example. Shown in Fig. 6(e), this results in a larger volume bound for link (k',i), which is now 8000, compared to 6.7 thousand in the previous iteration. However, the volume bound for link (k,k') remains 5000 due to the bottleneck k'. Hence, the source volume of k is still limited to 5000 in Fig. 6(f). The new lifetime vector (5.0, 12.0) is smaller than the previous one (5.0, 13.3). If we continue this example, we will see that the lifetime vector will become smaller after each iteration.

The cause of the problem is that k is a restricted feeding source of i, with an exhausted node k' on its forwarding path, which prevents it from fully utilizing the volume bound assigned on link (k',i) by node i. To solve this problem, node i should reduce the volume bound on (k',i) and increase the volume bound on (s,i) such that the unrestricted feeding source s can have a larger lifetime.

We use Fig. 7 to explain the solution of rate reduction. We know that the volume bounds should be set in proportion to the link rates. In order to reduce the volume bound $b(k',i)$, we can artificially reduce the link rate $r(k',i)$. More specifically, each exhausted node k' maintains a new variable $f(k')$, called the *rate-reduction factor*. The exact formula for calculating the value of $f(k')$ is given in Section IV-D. In this simple example, when we see the volume bound $b(k',i)$ is 6.7 thousand in Fig. 7(b) and the volume $v(k',i)$ is 5000 in Fig. 7(c), we want to reduce the volume bound by a factor of $\frac{v(k',i)}{b(k',i)}$ so that the two will match. To achieve that, we set the value of $f(k',i)$ to $\frac{v(k',i)}{b(k',i)}$, and artificially reduce the link rate $r(k',i)$ by multiplying it with $f(k',i)$. The result is shown in Fig. 7(d), where $r(k',i)$ is reduced to $\frac{5.0}{6.7} = 0.75$, which leads to a reduced volume bound $b(k',i)$ in Fig. 7(e), now 7.5 thousand instead of 8000, and an increased volume bound $b(s,i)$, now 2.5 thousand instead of 2 thousand. This allows the lifetime of s to be 12.5 in Fig. 7(f), instead of 12 in Fig. 6(f), where the rate reduction on link (k',i) was not performed.

With rate reduction, after the fifth, 10th, and 12th iterations, the lifetime of source s is improved to 14.94, 14.97, and 15.0,

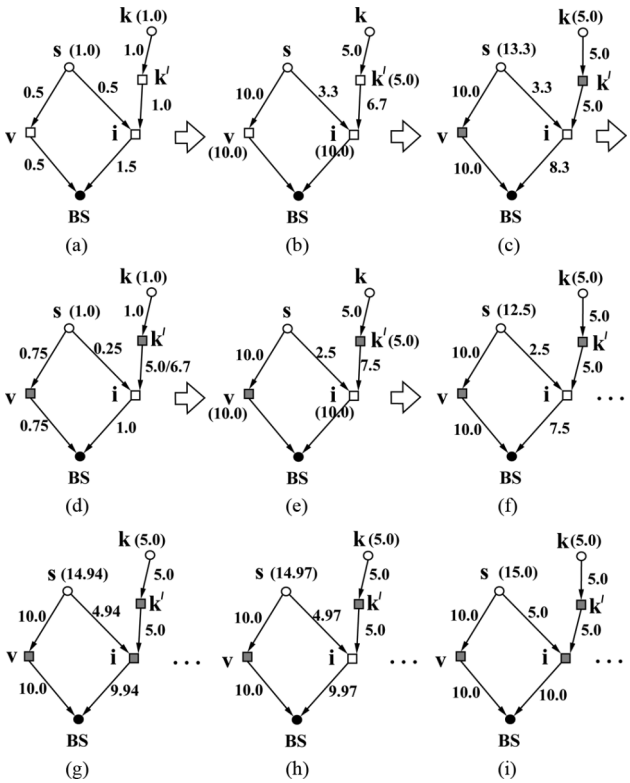


Fig. 7. Rate reduction. (Volume and volume bound are in thousands.) (a) Rate. (b) Volume bound. (c) Volume. (d) Rate. (e) Volume bound. (f) Volume. (g) Volume after fifth iteration. (h) Volume after 10th iteration. (i) Volume after 12th iteration.

respectively, as shown in Fig. 7(g)–(i), practically achieving the maximum lifetime vector (5.0, 15.0).

VI. Simulation

In this section, we develop a simulation software to evaluate the performance of DPA. We want to point out that the proposed algorithm is designed to run at the application layer. It is executed before the network begins to operate and deliver data packets. The link quality does not affect the execution of the algorithm as long as control messages can get through the link, possibly after retransmissions when necessary—if control messages cannot get through a link, then this link should not be used in the topology in the first place.

A. A Simple Illustrative Test Case

The first simulation is performed on the routing graph shown in Fig. 8, where a circle represents a source node, a square represents a nonsource node, and the two numbers beside a node are the initial energy (in joules) and the source rate (in packets/min), respectively. DPA itself does not dictate how the routing graph should be constructed. Instead, it can work with any routing graph that contains the potential routing paths it can choose from (see Section II). DPA works at the application level; it is independent of which MAC protocol is used. Suppose $\alpha = \beta = 0.000012$ J/packet and $\gamma = 0.0000432$ J/packet, which are chosen based on the parameters in [23] and will be used in all our simulations.

Table I shows the lifetime vectors after the first, second, 10th, and 20th iterations of DPA, as well as the maximum lifetime vector (MLV) in the last column, which is computed numerically based on Hou's centralized algorithm [16]. The result

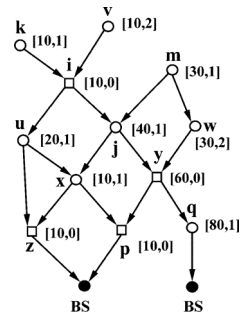


Fig. 8. Simple illustrative test case.

TABLE I
Data Source Lifetimes (in Days)

sources	1st iter.	2nd iter.	10th iter.	20th iter.	MLV
k	41.9	41.9	41.3	41.9	41.9
v	41.9	41.9	41.3	41.9	41.9
u	36.9	63.2	131.2	129.4	125.8
x	33.6	60.7	109.0	121.6	125.8
m	129.9	146.2	158.9	157.3	157.3
j	108.8	154.2	160.2	157.3	157.3
w	151.0	140.5	156.7	157.3	157.3
q	335.5	200.5	239.5	251.7	251.6

TABLE II
Data Source Volumes (in Thousands of Packets)

sources	1st iter.	2nd iter.	10th iter.	20th iter.	MLV
k	60.4	60.4	59.4	60.4	60.4
v	120.8	120.8	118.8	120.8	120.8
u	53.1	90.9	188.9	186.3	181.2
x	48.3	87.5	156.9	175.1	181.2
m	187.0	210.6	228.8	226.5	226.4
j	156.6	222.0	230.6	226.5	226.4
w	434.8	404.6	451.2	453.0	452.9
q	483.1	288.7	344.8	362.5	362.3

demonstrates that the sequence of lifetime vectors produced by DPA converges rapidly toward MLV. Table II shows the source volumes that are assigned by DPA to the source nodes after the first, second, 10th, and 20th iterations, as well as the optimal source volumes that produce MLV. Recall that the source volume is the number of packets that a source can successfully deliver to the sink over its lifetime. Source q has a larger lifetime than w, but a smaller source volume. That is because its source rate is smaller.

B. Simulation Setup for Complex Cases

Unless specified otherwise, each sensor network used in the remaining simulations has 500 nodes that are randomly deployed in a 1000×1000 area. There are 100 data sources randomly selected from the 500 sensors. Their source rates are all one packet per minute. When the network size is not 500 nodes, we change the deployment area proportionally while keeping the same node density. The sink consists of four base stations, evenly spaced along one edge of the deployment area. Each sensor has a transmission range of 100 and an initial energy of 5 J. The routing graph is constructed based on hop count. Each sensor picks its downstream neighbors from those neighbors that are one hop closer to the sink. Each of the data points used to produce the figures in this section is the average of 100 simulation runs on different random networks.

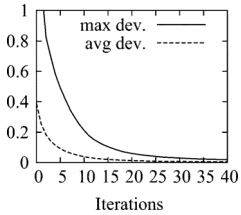


Fig. 9. Max deviation and avg deviation of lifetime vector with respect to the number of iterations that DPA has performed.

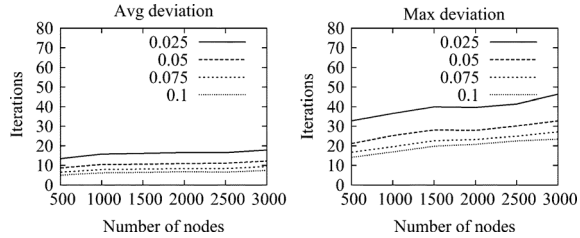


Fig. 10. DPA scales well. Its overhead grows slowly with the network size.

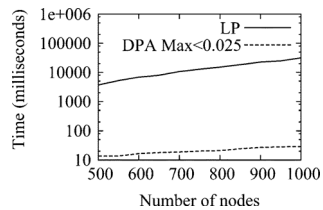


Fig. 11. Comparison of running time between LP and DPA.

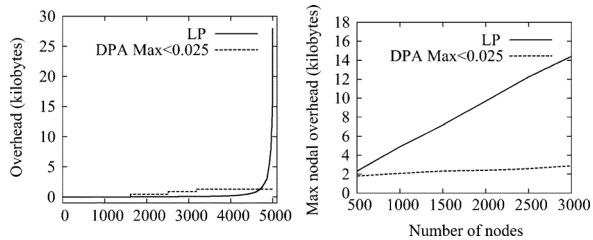


Fig. 12. *(left)* Comparison of nodal overhead distribution between LP and DPA. *(right)* Comparison of maximum nodal overhead between LP and DPA.

Each sensor uses up to three downstream neighbors, which are chosen from the neighbors that are one hop closer to the sink. If there are more than three such neighbors, the sensor picks three among them randomly.

C. Convergence Speed of DPA

The first simulation studies how quickly DPA converges its lifetime vector to the MLV, which is computed numerically based on Hou's centralized algorithm [16]. Consider the lifetime vector V_x produced by DPA after the xth iteration. We measure how much V_x deviates from MLV by the following two metrics. Let $t_x(s)$ be the lifetime of source s in V_x. Let $t_*(s)$ be the lifetime of s in MLV. The *max deviation* of V_x is defined as

$$\max_{s\in S}\left\{\frac{|t_x(s)-t_*(s)|}{t_*(s)}\right\}$$

and the *avg deviation* is defined as

$$\frac{1}{|S|}\sum_{s\in S}\frac{|t_x(s)-t_*(s)|}{t_*(s)}.$$

Fig. 9 shows the avg/max deviations of lifetime vectors produced by DPA on 500-node sensor networks. The deviations drop quickly to an insignificant level after a small number of iterations. For example, the avg/max deviations are merely 0.066 and 0.013, respectively, after 20 iterations—that means, in the worse case, the lifetime of any source deviates from its optimal value by no more than 6.6%, and on the average case, the lifetime of a source deviates from the optimal by 1.3%.

D. Scalability of DPA

We evaluate the scalability of DPA on random networks of 500–3000 nodes (with 20% being sources). We set a *target* (avg or max) deviation to be 0.025, 0.05, 0.075, or 0.1. We then count the number of iterations that DPA has to perform in order to produce a lifetime vector whose deviation is bounded by the target value. The simulation results are presented in Fig. 10. It shows that the *overhead* for DPA to satisfy a target deviation, *which is measured by the number of iterations*, grows slowly with the network size. Recall that a node sends at most two small control packets in each iteration. Even for a network of 3000 nodes, only 12 iterations are needed to achieve an avg deviation of 5%, and 32 iterations are needed for a max deviation of 5%.

E. Comparison to Hou's Centralized Algorithm

We use *LP* to stand for Hou's centralized algorithm [16] based on iterative linear programming. Our DPA can also be used as a centralized algorithm when the information about the network is available at the sink. The target max deviation for DPA is set to be 0.025. Fig. 11 compares the running times of the two algorithms. It shows that DPA is orders of magnitude faster than LP, and the gap widens when the network size increases.

Next we compare the communication overhead of the two algorithms when LP is used as a centralized algorithm while DPA is used as a distributed algorithm. For LP, the sink has to collect network information, including, for each node, source rate (4 B), node energy (4 B), transmission power γ (4 B), node ID, and IDs of its downstream neighbors (2 B each). The sink has also to download the resulting volume schedule to the network, which includes, for each node, its source volume and the volumes of its outgoing links (4 B each). For DPA, in every iteration, a node sends out the volumes/rates of its outgoing links and the volume bounds of its incoming links (4 B each).

The communication overhead of DPA spreads evenly among all nodes. The communication overhead of LP concentrates on nodes surrounding the sink. For 5000-node random networks, the left plot in Fig. 12 shows the nodal communication overhead in ascending order. The overhead is measured by the number of bytes that a node has to transmit. Clearly, some nodes in LP (at the right end of the figure) bear a huge burden of communication overhead.

The right plot in Fig. 12 shows the maximum nodal overhead with respect to network size. The maximum nodal overhead of LP increases much faster than that of DPA.

F. Comparison to Other Centralized and Distributed Solutions

We compare DPA with two additional algorithms: SLP (following the same name used in [15]) that is a linear programming

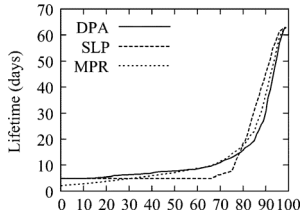


Fig. 13. Network lifetimes of DPA, SLP, and MPR.

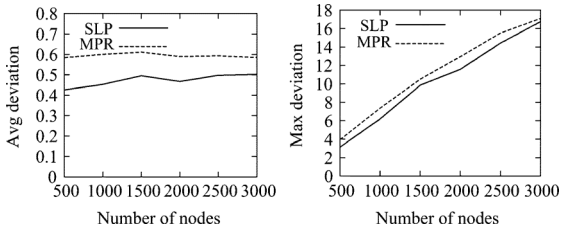


Fig. 14. Avg and max deviations of SLP and MPR.

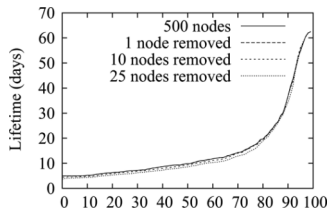


Fig. 15. Impact of small topology changes.

solution for maximizing the minimum lifetime of all sources, and Minimum-Power Routing (MPR [24], [25]) that is a distributed algorithm for energy-efficient routing.

First we run DPA, SLP, and MPR on 100-node random networks (with all nodes being sources). Fig. 13 compares the lifetime vectors produced by the algorithms. Each curve represents the lifetime vector in ascending order generated from one of the three algorithms. The smallest lifetime in the vector produced by DPA is more than 100% larger than the smallest lifetime in the vector by MPR. For SLP, the result shows that maximizing the minimum *lifetime* of sources does not maximize the *lifetime vector* of the network. DPA produces far better source lifetimes in the lower three quarters of the vector. Second, we compare the algorithms on larger networks. Fig. 14 shows the avg/max deviations of the lifetime vectors produced by SLP and MPR on networks of 500–3000 nodes (with 20% being sources). The deviations are large when comparing with those of DPA, which can be made arbitrarily small.

G. Topology Change

We evaluate the impact of unexpected topology change on lifetime vector. Initially, there are 500 nodes in the network, among which 100 sensors are data sources. See the simulation setup in Section VI-B. The lifetime vector of the 100 data sources is shown by the line of "500 nodes" in Fig. 15, where the horizontal axis shows the sources in ascending order of their lifetimes, and the vertical axis shows the lifetimes. To simulate the impact of a node failure, we randomly select a nonsource node and remove it from the topology. The removal may cause the lifetimes of some sources to be shortened. The resulting lifetime vector is shown by the line of "1 node removed," which is the average of 100 simulation runs, each run removing one randomly selected node. Similarly, we repeat the simulation with 10 or 25 nodes removed. The results show that although small topology changes worsen the lifetime vector, they do so by a relatively small amount.

VII. Conclusion

We have proposed a distributed progressive algorithm for maximizing the lifetime vector in a wireless sensor network. Its design is based on the necessary and sufficient conditions that we have proved for the maximum lifetime vector. The algorithm is totally decentralized, and it improves the lifetime vector iteratively toward the maximum value. Our simulations show that the algorithm rapidly converges toward the maximum lifetime vector and significantly outperforms the existing algorithms.

In this paper, we decouple routing from lifetime maximization so that the proposed algorithm can work with any underlying routing protocol, which may be predetermined based on considerations other than lifetime in practice. In our future work, we will study the related problem of finding the optimal routing strategy for the purpose of lifetime vector maximization. Some revelant heuristics can be found in [26] in the context of max-min rates. Our future work will study both heuristic and exact distributed solutions for finding the optimal routing graph.

VIII. Acknowledgement

The authors would like to thank the anonymous reviewers for their constructive comments.

References

[1] D. Luo, X. Zhu, X. Wu, and G. Chen, "Maximizing lifetime for the shortest path aggregation tree in wireless sensor networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1566–1574.

[2] H. Wang, N. Agoulmine, M. Ma, and Y. Jin, "Network lifetime optimization in wireless sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 7, pp. 1127–1137, Sep. 2010.

[3] Y. Yun and Y. Xia, "Maximizing the lifetime of wireless sensor networks with mobile sink in delay-tolerant applications," *IEEE Trans. Mobile Compu.*, vol. 9, no. 9, pp. 1308–1318, Sep. 2010.

[4] G. Zussman and A. Segall, "Energy efficient routing in ad hoc disaster recovery networks," in *Proc. IEEE INFOCOM*, 2003, vol. 1, pp. 682–691.

[5] A. Sankar and Z. Liu, "Maximum lifetime routing in wireless ad-hoc networks," in *Proc. IEEE INFOCOM*, 2004, vol. 2, pp. 1089–1097.

[6] J. Zhu, S. Chen, B. Bensaou, and K.-L. Hung, "Tradeoff between lifetime and rate allocation in wireless sensor networks: A cross layer approach," in *Proc. IEEE INFOCOM*, 2007, pp. 267–275.

[7] Y. Wu, S. Fahmy, and N. B. Shroff, "On the construction of a maximum-lifetime data gathering tree in sensor networks: NP-completeness and approximation algorithm," in *Proc. IEEE INFOCOM*, 2008, pp. 356–360.

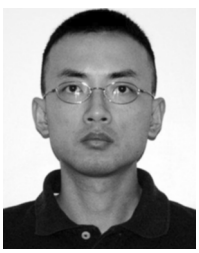
[8] Y. Shi and T. Hou, "Theoretical results on base station movement problem for sensor networks," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 1–5.

[9] F. Wang, D. Wang, and J. Liu, "Traffic-aware relay node deployment: Maximizing lifetime for data collection wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1415–1423, Aug. 2011.

[10] D. Blough and P. Santi, "Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks," in *Proc. ACM MobiCom*, 2002, pp. 183–192.

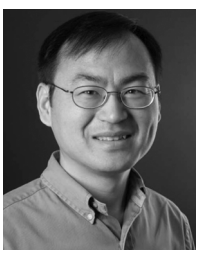
[11] M. Bhardwaj and A. Chandrakasan, "Bounding the lifetime of sensor networks via optimal role assignments," in *Proc. IEEE INFOCOM*, 2002, vol. 3, pp. 1587–1596.

[12] Q. Li, J. Aslam, and D. Rus, "Online power-aware routing in wireless ad-hoc networks," in *Proc. ACM MobiCom*, 2001, pp. 97–107.
[13] S. Cui, R. Madan, A. J. Goldsmith, and S. Lall, "Joint routing, MAC, and link layer optimization in sensor networks with energy constraints," in *Proc. IEEE ICC*, 2005, vol. 2, pp. 725–729.
[14] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proc. ACM MobiCom*, 2001, pp. 70–84.
[15] Y. T. Hou, Y. Shi, and H. Sherali, "On lexicographic max-min node lifetime for wireless sensor networks," in *Proc. IEEE ICC*, Jun. 2004, vol. 7.
[16] Y. T. Hou, Y. Shi, and H. D. Sherali, "Rate allocation in wireless sensor networks with network lifetime requirement," in *Proc. ACM MobiHoc*, 2004, pp. 67–77.
[17] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *Proc. 3rd DialM*, Aug. 1999, pp. 48–55.
[18] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. ACM MobiCom*, Aug. 2000, pp. 243–254.
[19] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proc. ACM MobiCom*, Apr. 2003, pp. 96–108.
[20] S. Chen, G. Fan, and J. Cui, "Avoid void in geographic routing for data aggregation in sensor networks," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 169–178, Jul. 2006.
[21] Y. Wang, S. Lederer, and J. Gao, "Connectivity-based sensor network localization with incremental Delaunay refinement method," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2401–2409.
[22] B. Radunovic and J. L. Boudec, "A unified framework for max-min and min-max fairness with applications," in *Proc. 40th Annu. Allerton Conf. Commun., Control, Comput.*, 2002.
[23] W. Heinzelman, "Application-specific protocol architecture for wireless networks," Ph.D. dissertation, MIT, Cambridge, MA, 2000.
[24] S. Doshi, S. Bhandare, and T. Brown, "An on-demand minimum energy routing protocol for a wireless ad hoc network," *Mobile Comput. Commun. Rev.*, vol. 6, no. 3, pp. 50–66, Jul. 2002.
[25] J. Gomez, A. Campbell, M. Naghshineh, and C. Bisdikian, "Conserving transmission power in wireless ad hoc networks," in *Proc. IEEE Int. Conf. Netw. Protocols*, Nov. 2001, pp. 24–34.
[26] R. Liu, K. Fan, Z. Zheng, and P. Sinha, "Perpetual and fair data collection for environmental energy harvesting sensor networks," *IEEE/ACM Trans. Netw.*, vol. 19, no. 4, pp. 947–960, Aug. 2011.



Liang Zhang received the B.E. and M.E. degrees in computer science from Tsinghua University, Beijing, China, in 1999 and 2001, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2009.

He had worked with Oracle R&D Center, Beijing, China, from 2002 to 2003. Since 2008, he has been working with Juniper Networks, Sunnyvale, CA. His research interests include sensor networks, multihop wireless networks, and network security.



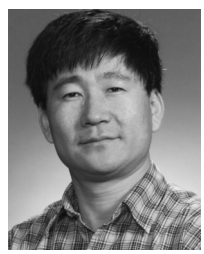
Shigang Chen (M'02–SM'12) received the B.S. degree from the University of Science and Technology of China, Hefei, China, in 1993, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively, all in computer science.

He is an Associate Professor with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville. After graduation, he had worked with Cisco Systems, San Jose, CA, for three years before joining the University of Florida in 2002. He served on the technical advisory board for Protego Networks, Sunnyvale, CA, from 2002 to 2003. He published more than 100 peer-reviewed journal/conference papers. He holds 11 US patents. His research interests include computer networks, Internet security, wireless communications, and distributed computing.

Dr. Chen is an Associate Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING, *Computer Networks*, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. He has been serving in the steering committee of IEEE IWQoS since 2010. He received the IEEE Communications Society Best Tutorial Paper Award in 1999 and an NSF CAREER Award in 2007.

Ying Jian received the B.E. and M.E. degrees in computer science from Tsinghua University, Beijing, China, in 2001 and 2004, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2008.

After graduation, he worked with Microsoft for two years and then joined Google, Mountain View, CA, in 2010. His research interests include QoS and security in multihop wireless networks and sensor networks.



Yuguang "Michael" Fang (S'92–M'97–SM'99–F'08) received the Ph.D. degree in systems engineering from Case Western Reserve University, Cleveland, OH, in 1994, and the Ph.D. degree in electrical engineering from Boston University, Boston, MA, in 1997.

He was an Assistant Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, from 1998 to 2000. He then joined the Department of Electrical and Computer Engineering, University of Florida (UF), Gainesville, in 2000 as an Assistant Professor, got an early promotion to an Associate Professor with tenure in 2003, and to a Full Professor in 2005. He holds a University of Florida Research Foundation (UFRF) Professorship from 2006 to 2009, a Changjiang Scholar Chair Professorship with Xidian University, Xi'an, China, from 2008 to 2011, and a Guest Chair Professorship with Tsinghua University, Beijing, China, from 2009 to 2012. He has published over 300 papers in refereed professional journals and conferences.

Dr. Fang is a member of the Association for Computing Machinery (ACM). He served as the Editor-in-Chief for *IEEE Wireless Communications* from 2009 to 2012 and serves or has served on editorial boards of technical journals including the IEEE TRANSACTIONS ON MOBILE COMPUTING (2003–2008, 2011–present), *IEEE Network* (2012–present), IEEE TRANSACTIONS ON COMMUNICATIONS (2000–2011), IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (2002–2009), IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (1999–2001), *IEEE Wireless Communications Magazine* (2003–2009), and *Wireless Networks* (2001–present). He served on the Steering Committee for the IEEE TRANSACTIONS ON MOBILE COMPUTING from 2008 to 2010. He has been actively participating in professional conference organizations such as serving as the Technical Program Co-Chair for IEEE INOFOCOM 2014, the Steering Committee Co-Chair for QShine (2004–2008), the Technical Program Vice-Chair for IEEE INFOCOM 2005, the Technical Program Area Chair for IEEE INFOCOM (2009–2013), Technical Program Symposium Co-Chair for IEEE GLOBECOM 2004, and a member of Technical Program Committee for IEEE INFOCOM (1998, 2000, 2003–2008). He received the National Science Foundation Faculty Early Career Award in 2001 and the Office of Naval Research Young Investigator Award in 2002. He is the recipient of the Best Paper Award in IEEE GLOBECOM 2011 and the IEEE International Conference on Network Protocols (ICNP, 2006) and the recipient of the IEEE TCGN Best Paper Award in the IEEE High-Speed Networks Symposium, IEEE GLOBECOM 2002. He has also received a 2010–2011 UF Doctoral Dissertation Advisor/Mentoring Award, a 2011 Florida Blue Key/UF Homecoming Distinguished Faculty Award, and the 2009 UF College of Engineering Faculty Mentoring Award.

Zhen Mo received the B.E degree in information security engineering and M.E. degree in theory and new technology of electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2007 and 2010, and is currently pursuing the Ph.D. degree computer and information science engineering at the University of Florida, Gainesville.

His research interests include network security and cloud computing security.