Distributed Progressive Algorithm for Maximizing Lifetime Vector in Wireless Sensor Networks

Liang Zhang[†] Shigang Chen[†] Ying Jian[†] Yuguang Fang[‡]
[†]Department of Computer & Information Science & Engineering, University of Florida
[‡]Department of Electrical and Computer Engineering, University of Florida

Abstract— **Maximizing the operational lifetime of a sensor network is a critical problem in practice. Many prior works define the network's lifetime as the time before the first sensor in the network runs out of energy. However, when one sensor dies, the rest of the network can still work, as long as useful data generated by other sensors can reach the sink. More appropriately, 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 progressive algorithm which iteratively produces a series of lifetime vectors, each better than the previous one. 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 towards the maximum lifetime vector with low overhead.** [1]

I. Introduction

Sensor networks have a wide range of applications in habitat observation, seismic monitoring, battlefield sensing, etc. Battery-powered sensor nodes are limited in computation capability, memory space, communication bandwidth, and above all, energy supply. The network cannot carry out its task after the nodes' energy is exhausted. Hence, maximizing the operational lifetime of a sensor network is a critical problem.

What is exactly the lifetime of a sensor network? Many prior works [1], [2], [3], [4], [5], [6], [7], [8], [9], [10] define the network's lifetime as the time before the first sensor in the network runs out of energy, or before the first loss of coverage [11]. This definition simplifies the problem of maximizing lifetime to a linear programming problem or an NP-hard non-polynomial programming problem if the sink is allowed to move [10]. However, in reality, the operational lifetime of the network is not limited to the smallest lifetime of all nodes. When one sensor dies, the rest of the network can still work, as long as useful data generated by other sensors can still reach the sink. It is not true that, since sensors around the sink forward others' data, they will always exhaust their energy first and prevent the rest of the network from reaching the sink. One can deploy more sensors around the sink, use larger batteries to boost the energy level there, or perform in-network data aggregation.

An appropriate definition for 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's 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' *packet 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 towards the sink.

Hou et al. [12], [13] 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. Our strategy is to design a distributed progressive algorithm that works in a series of iterations, each producing a result (in our case, a lifetime vector and its corresponding forwarding policies) that is better than the previous one. The sequence of results approaches to the optimal solution. A distributed progressive algorithm is practically attractive because a result

[1]This work was supported in part by the US National Science Foundation under grant CNS-0644033. Y. Fang is also with the National Key Laboratory of Integrated Services Networks, Xidian University, Xian, China. His work was partially supported by the Changjiang Scholar Chair Professorship and the 111 Project under B08038.

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 with 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 to 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. DPA produces much better lifetime vector.

II. Network Model and Problem Definition

A. Sensor Network Model

We study the problem of maximizing the lifetime vector of long-term low-rate monitoring sensor networks that collect data from fields for ecosystem study, environmental monitoring, seismic measurement, etc. Operating under battery power, such sensor networks are designed to gather tens of thousands of data points from each selected location over a period of weeks or months. When *raw information* from each location is of interest, power consumption cannot be substantially reduced by in-network aggregation techniques that are suitable for max/min/avg queries, but not for collecting temporal/spatial data of the whole field (in case that raw data must be made available for future analysis).

Let N be the set of sensor nodes, among which the subset S that generate new data are called *data sources*, which may be the aggregation nodes representing local clusters [12], [13]. 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. Assume the base stations are externally connected to a data collector. It makes no difference which base station a data packet is routed to.

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.

B. 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. The *source volume* $v(i)$ of a node i is defined as the number of new data packets generated by i. 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, γ_i be the amount of energy that node i spends on sending a packet. The *energy constraint* is given below.

$$\sum_{k \in U_i} \alpha \times v(k,i) + \beta_i \times v(i) + \sum_{j \in D_i} \gamma_i \times v(i,j) \le e_i, \quad \forall i \in N \tag{1}$$

We say a node i is *exhausted* if

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

The *volume conservation constraint* depends on the application model. If the application requires raw data to be delivered from sources to the sink, then the number of packets sent by a node is equal to the number it receives, i.e.,

$$\sum_{j \in D_i} v(i,j) = v(i) + \sum_{k \in U_i} v(k,i), \quad \forall i \in N. \tag{2}$$

If it requires periodic measurement of min/max/avg among readings from sources that have not exhausted yet and remain reachable to the sink, then a node will send a packet for each set of packets received from its upstream neighbors or generated locally. The constraint becomes

$$\sum_{j \in D_i} v(i,j) = \max\{\max_{k \in U_i}\{v(k,i)\}, v(i)\} \quad \forall i \in N. \tag{3}$$

C. Maximum Lifetime Vector Problem

The volume schedule specifies how many packets are forwarded through each node, each link and each path, and thus it determines the lifetime vector. For an arbitrary feasible volume schedule, we can calculate the *lifetime* of each data source $s \in S$ as follows:

$$t_s = v(s)/g_s. \tag{4}$$

The *lifetime vector* of the sensor network is defined as $(t_s, s \in S)$ sorted in ascending order.

Each feasible volume schedule produces a *feasible lifetime vector*. All feasible lifetime vectors form the lifetime space. One lifetime vector T is *greater* than another T' if T is lexicographically larger — for some $x \in [1..|S|]$, T and T' share the common $(x-1)$ smallest elements but the xth smallest element in T is greater than the xth smallest element in T'.

The *maximum lifetime vector problem* is to find a feasible volume schedule that produces the largest (or say, maximum) lifetime vector. Intuitively, its goal is to first maximize the smallest lifetime of all sources, then the second smallest, and so on.

Once we find the volume schedule for the maximum lifetime vector, the nodes must know their packet forwarding policies that will realize the volume schedule. To implement a volume schedule, each node i simply does the following: 1) it generates new packets at its source rate g_i for $v(i)$ 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*.

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

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

D. Routing Graph

The routing graph should be a DAG (directed acyclic graph) to avoid cyclic routing because packets that are routed in a cycle waste energy and may not be able to timely reach the sink. Moreover, any feasible volume schedule based on a routing graph with cycles can be transformed into a feasible volume schedule on the routing graph with cycles removed, producing an equal or larger lifetime vector. This can be shown by the following procedure, assuming the application model (2): Identify a routing loop and find the link (i,j) with the smallest volume $v(i,j)$. Deduct the link volumes along the loop by $v(i,j)$ and remove (i,j). Repeat the above procedure until all loops are removed. The resulting volume schedule still satisfies the volume conservation constraint and the energy constraint. No source volume has been changed, and thus the lifetime vector produced by the new volume schedule on the acyclic routing graph remains the same. Furthermore, since some link volumes have been reduced, which may leave room for increasing some source volumes to produce a larger lifetime vector. The same reasoning can be applied to model (3) with added details on how to reduce volumes along a cycle and on other links. But the key point is the same — when removing a cycle, link volumes only need to be reduced.

The acyclic routing graph can be easily constructed when packets are forwarded based on hop counts or the nodes' geographic locations to the sink. For example, D_i may consist of all or a selected subset of neighbors that are closer to the sink (based on the hop count or 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.

III. Necessary and Sufficient Conditions for Maximizing Lifetime Vector

This section establishes the theoretical foundation of our distributed algorithm for maximizing the lifetime vector.

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 *non-forwarding path*.

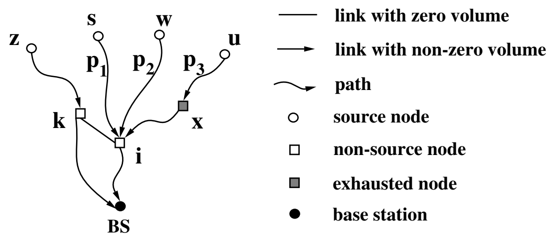

Fig. 1: There is no exhausted node on P_1 or P_2; nodes s and w are unrestricted feeding sources of i. There is an exhausted node x on P_3; node u is a restricted feeding source of i. There is no forwarding path from z to i; node z is a potential source of i.

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 non-forwarding path from s to i, and the path has no exhausted node.

We will establish the necessary and sufficient conditions for maximizing the lifetime vector in a theorem below. Below we explain a basic technique used in the proof, called *volume shift*. Understanding this technique will also help one to understand the design of the algorithm.

Consider the routing graph in Fig. 1. Suppose s and w are two unrestricted feeding sources of node i. 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 can be increased at the expense of the lifetime of another*. To do so, we simply decrease the source volume of s, then decrease the volumes on the links of P_1, increase the source volume of w, and finally increase the volumes on the links of P_2, all by the same tiny amount, which 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. It is easy to see that, 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. It is obvious that, 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.

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, i) *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; ii) *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 a 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.

The proof is omitted due to space limitation. The above theorem gives us some guideline for designing a distributed algorithm that generates a volume schedule to maximize the lifetime vector. Below we give intuitive interpretation.

Based on the first condition, data sources should aggressively set their source volumes to the highest values that their paths to the sink allow.

The lifetime of a source s, which is $v(s)/g_s$, can be interpreted as the average volume assigned to each unit of rate. The second condition requires that each unit of rate received by a node i from an unrestricted feeding source deserves the same amount of volume allocation. In other words, for unrestricted feeding sources, node i should allocate volumes in proportion to their rates (that i receives and forwards). However, each unit of rate from a restricted feeding source (which encounters an exhausted node on its forwarding path) may receive less volume allocation at node i. Moreover, a source should always direct its packets to paths that have highest volume allocation per unit of rate.

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, $\{v(i,j),\ \forall(i,j) \in E,\ v(i),\ \forall i \in N\}$. 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.

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

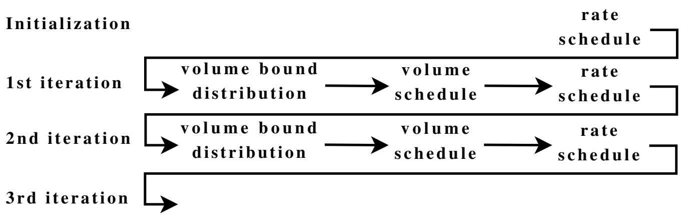

Fig. 2: Iterations of DPA

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. 2, in each subsequent iteration, 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 below.

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. The volume bounds are subject to two *volume-capacity constraints*.

First, a node i should not receive and forward more packets than the downstream neighbors can handle. If the application model is characterized by (2), then the combined incoming volume bound (set by i) should not exceed the combined outgoing volume bound (set by downstream neighbors).

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

where $b(i,j)$ is learned by i from j. If the application model is characterized by (3), then the constraint becomes

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

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

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

where

$$b'(i,j) = \begin{cases} (\sum_{k \in U_i} b(k,i) + b(i)) \frac{b(i,j)}{\sum_{j' \in D_i} b(i,j')}, \\ \qquad \text{for application model (2);} \\ \max\{\max_{k \in U_i}\{b(k,i)\}, b(i)\} \frac{b(i,j)}{\sum_{j' \in D_i} b(i,j')}, \\ \qquad \text{for application model (3).} \end{cases}$$

Because of (6)-(7), $b'(i,j) \leq b(i,j)$, and therefore the above constraint is more relax than one that replaces $b'(i,j)$ with $b(i,j)$.

As we have explained in the previous section, 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*.

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

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 $v(i)$ as large as possible, based on (9) subject to the constraints of (6)-(7) and (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. Each node i should set the outgoing volumes, $v(i,j), \forall j \in D_i$, and its source volume $v(i)$, which are subject to the following *bound constraint*.

$$v(i) \leq b(i), \quad v(i,j) \leq b(i,j), \quad \forall j \in D_i, \forall i \in N \tag{10}$$

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

$$v(i) \leftarrow b(i) \tag{11}$$

In addition to (10), outgoing link volumes are also subject to the volume conservation constraint in (2) or (3). 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 (2), to satisfy this constraint, node i assigns its outgoing volumes as follows.

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

where $v(k,i)$ is set by upstream neighbor k and learned by i from k. If the volume conservation constraint is (3), node i assigns the outgoing volumes to be

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

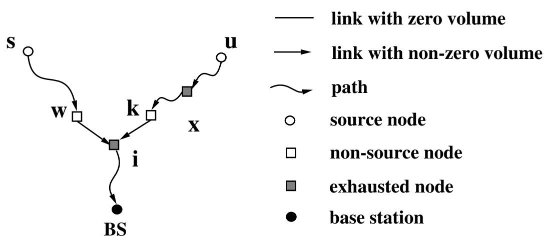

Fig. 3: There is no exhausted node from s to i; node s is an unrestricted feeding sources of i. There is an exhausted node x from u to i; node u is a restricted feeding source of i. The upstream bottleneck x may prevent source u from fully utilizing the volume bound set by i on link (k,i).

It can be shown by induction that the above assignment satisfies the bound constraint in (10). This result, together with (8), ensures that the assigned volumes satisfy the energy constraint required in (1) — to see this, one has to use the fact that $v(i,j) \leq b'(i,j)$ due to (12)-(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 below based on the rate-volume property in (5), setting the stage for the next iteration. For application model (2),

$$r(i,j) \leftarrow (\sum_{k \in U_i} r(k,i) + g_i) \frac{v(i,j)}{\sum_{j' \in D_i} v(i,j')}, \quad \forall j \in D_i \quad (14)$$

For application model (3),

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

We have one additional issue that must be handled. As shown in Fig. 3, 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)-(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., $\sum_{k \in U_i} \alpha \times v(k,i) + \beta_i \times v(i) + \sum_{j \in D_i} \gamma_i v(i,j) = e_i$, or it was exhausted in one of the previous iterations, then it updates $f(i)$:

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

where $B(i)$ and $V(i)$ 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.

$$r(i,j) \leftarrow r(i,j) \times f(i), \quad \forall j \in D_i \quad (17)$$

For the example in Fig. 3, 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)-(13). It then updates the link rates by (14)-(15), (16), and (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)-(13), and the new outgoing rates by (14)-(15), (16), and (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 Step 1 and Step 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)-(7) and (8), is distributed as volume bounds based on the rate-bound property in (9), which essentially performs volume shift from feeding sources with larger volume per unit of rate (i.e., larger lifetime) to those with smaller volume per unit of rate. 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 which 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) and (12)-(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)-(15), (16) and (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. The proof of the following theorem is lengthy and cannot fit in the space of this paper. It can be found in a web version (Appendix C) at http://www.cise.ufl.edu/∼sgchen/FullPaper.pdf.

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

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. But 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 non-innovative, 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 V, which shows that DPA converges quickly towards the optimal lifetime vector. To achieve no more than 5% deviation from the optimal, for networks of 1,000 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 pre-determined, we can take the small energy consumption of DPA into account by reducing the nodes' energy (e_i) for an appropriate amount.

H. Network Dynamics

After DPA computes a volume schedule, because of network dynamics, the nodes may not always be able to follow the schedule exactly to forward packets. For example, a wireless link may be temporarily down due to environmental noise interference. The packets to be carried by this link will have to be sent to other downstream neighbors. If all outgoing links with non-zero volumes are temporarily down, then packets have to be forwarded to outgoing links with zero volumes. Consequently, the actual lifetime vector will deviate from the one predicted by the volume schedule.

To keep up with changes, DPA may be re-executed to compute a new volume schedule. There is a tradeoff between overhead and better lifetime vector. The frequency of executing DPA is dependent on the amount of overhead allowed. For example, suppose the sink collects aggregate information from the network periodically based on the application model characterized by (3), and data packets are longer than control packets (INIT/RATE/BOUND/VOL_RATE). If DPA is allowed to consume no more than 0.5% of all energy, then the sink may execute DPA for 5 iterations each time after it receives 2,000 data packets.

The only alternative solution [12], [13] in the literature is centralized. It is much harder for a centralized algorithm to handle network dynamics because that requires the sink to collect the complete network information before each execution.

V. Simulation

In this section, we use simulations to evaluate the performance of DPA.

A. Simulation Setup

Unless specified otherwise, each sensor network used in the simulation has 500 nodes that are randomly deployed in a $1,000 \times 1,000$ area. There are 100 data sources randomly selected from the 500 sensors. Their source rates are all 1 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 4 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 joules. Suppose $\alpha = \beta = 0.000012$ Joule/packet and

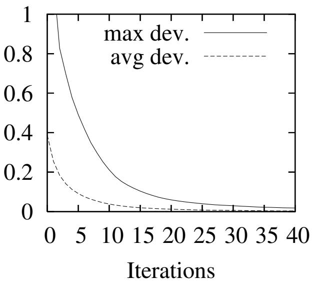

Fig. 4: max deviation and avg deviation of lifetime vector with respect to the number of iterations that DPA has performed

$\gamma = 0.0000432$ Joule/packet, which are chosen based on the parameters in [14] and will be used in all our simulations. DPA works at the application level; it is independent of which MAC protocol is used. 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.

B. Convergence Speed of DPA

The first simulation studies how quickly DPA converges its lifetime vector to the maximum lifetime vector (MLV), which is computed numerically based on Hou's centralized algorithm [13]. 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

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

and the *avg deviation* is defined as

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

Fig. 4 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%.

C. Scalability of DPA

We evaluate the scalability of DPA on random networks of 500 to 3,000 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. 5. 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 2 small

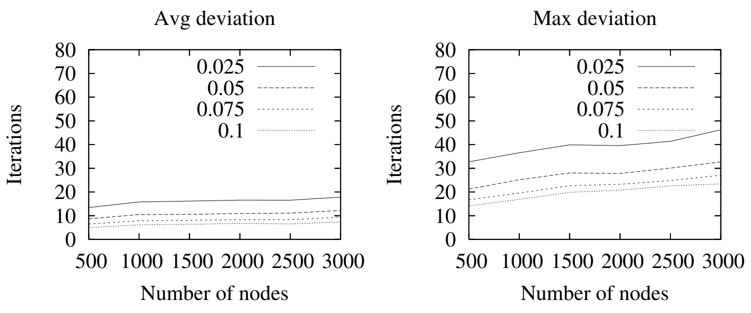

Fig. 5: DPA scales well. Its overhead grows slowly with the network size.

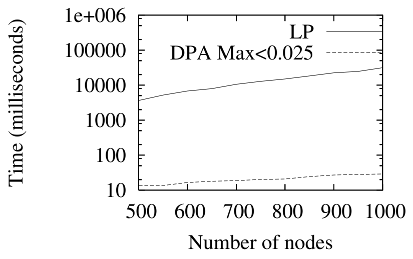

Fig. 6: Comparison of running time between LP and DPA

control packets in each iteration. Even for a network of 3,000 nodes, only 12 iterations are needed to achieve an avg deviation of 5%, and 32 iterations are needed for a max deviation of 5%.

D. Comparison with Hou's Centralized Algorithm

We use *LP* to stand for Hou's centralized algorithm [13] 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. 6 compares the running times of the two algorithms. It shows that DPA are 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 bytes), node energy (4 bytes), transmission power γ (4 bytes), node ID, and IDs of its downstream neighbors (2 bytes 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 bytes 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 bytes each).

The communication overhead of DPA spreads evenly among all nodes. The communication overhead of LP concentrates on nodes surrounding the sink. For 5,000-node random networks, the left plot in Fig. 7 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

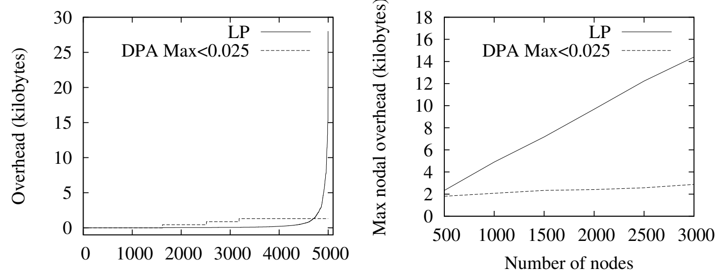

Fig. 7: *Left plot*: comparison of nodal overhead distribution between LP and DPA. *Right plot*: comparison of maximum nodal overhead between LP and DPA

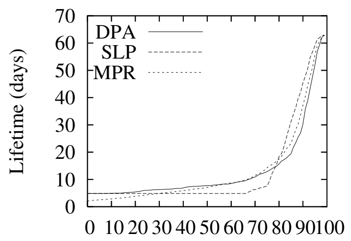

Fig. 8: Network lifetimes of DPA, SLP and MPR

communication overhead.

The right plot in Fig. 7 shows the maximum nodal overhead with respect to network size. The maximum nodal overhead of LP increases much faster than that of DPA.

E. Comparison with Other Centralized and Distributed Solutions

We compare DPA with two additional algorithms: SLP (following the same name used in [12]) that is a linear programming solution for maximizing the minimum lifetime of all sources, and MPR (Minimum-Power Routing [15], [16]) 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. 8 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 that 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. 9 shows the avg/max deviations of the lifetime vectors produced by SLP and MPR on networks of 500 to 3,000 nodes (with 20% being sources). The deviations are large when comparing with those of DPA, which can be made arbitrarily small.

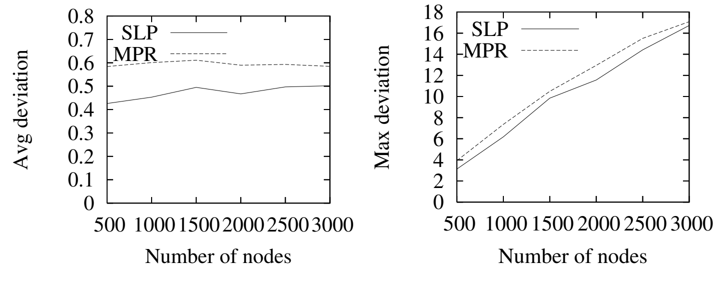

Fig. 9: Avg and max deviations of SLP and MPR

VI. Conclusion

We have proposed a distributed progressive algorithm for maximizing the lifetime vector in a wireless sensor network, the first algorithm of its kind for this problem. The design of the algorithm was based on the necessary and sufficient conditions that we have proved for producing the maximum lifetime vector. Simulations are performed to demonstrate the performance of the algorithm.

References

[1] J. Chang and L. Tassiulas, “Energy conserving routing in wireless ad-hoc networks,” *Proc. of IEEE INFOCOM’00*, 2000.

[2] Q. Li, J. Aslam, and D. Rus, “Online power-aware routing in wireless Ad-hoc networks,” *Proc. of ACM MobiCom’01*, pp. 97–107, 2001.

[3] M. Bhardwaj and A. Chandrakasan, “Bounding the lifetime of sensor networks via optimal role assignments,” *Proc. of IEEE INFOCOM’02*, vol. 3, p. 1587C1596, 2002.

[4] K. Kalpakis, K. Dasgupta, and P. Namjoshi, “Maximum lifetime data gathering and aggregation in wireless sensor networks,” *Proc. of IEEE ICN’02*, 2002.

[5] G. Zussman and A. Segall, “Energy Efficient Routing in Ad Hoc Disaster Recovery Networks,” *Proc. of IEEE INFOCOM’03*, 2003.

[6] A. Sankar and Z. Liu, “Maximum Lifetime Routing in Wireless Ad-hoc Networks,” *Proc. of IEEE INFOCOM’04*, 2004.

[7] R. Madan, Z. Q. Luo, and S. Lall, “A Distributed Algorithm with Linear Convergence for Maximum Lifetime Routing in Wireless Sensor Networks,” *Proc. of the Allerton Conference on Communication, Control and Computing*, 2005.

[8] J. Zhu, S. Chen, B. Bensaou, and K.-L. Hung, “Tradeoff between Lifetime and Rate Allocation in Wireless Sensor Networks: A Cross Layer Approach,” *Proc. of IEEE INFOCOM’07*, 2007.

[9] 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 Algorithms,” *Proc. of IEEE INFOCOM’08*, 2008.

[10] Y. Shi and T. Hou, “Theoretical Results on Base Station Movement Problem for Sensor Networks,” *Proc. of IEEE INFOCOM*, April 2008.

[11] D. Blough and P. Santi, “Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks,” *Proc. of ACM MobiCom’02*, p. 183C192, 2002.

[12] Y. T. Hou, Y. Shi, and H. Sherali, “On Lexicographic Max-Min Node Lifetime for Wireless Sensor Networks,” *Proc. of IEEE ICC’04*, vol. 7, pp. 3790–3796, June 2004.

[13] Y. T. Hou, Y. Shi, and H. D. Sherali, “Rate Allocation in Wireless Sensor Networks with Network Lifetime Requirement,” *Proc. of ACM MobiHoc’04*, pp. 67–77, 2004.

[14] W. Heinzelman, “Application-Specific Protocol Architecture for Wireless Networks,” *Ph.D. Thesis, MIT*, 2000.

[15] S. Doshi, S. Bhandare, and T. Brown, “An on-demand minimum energy routing protocol for a wireless ad hoc network,” *ACM Mobile Computing and Communications Review*, vol. 6, no. 3, July 2002.

[16] J. Gomez, A. Campbell, M. Naghshineh, and C. Bisdikian, “Conserving transmission power in wireless ad hoc networks,” *Proc. of IEEE International Conference on Network Protocols*, pp. 24–34, November 2001.