

# Coverage without Location Information

Ossama Younis

Applied Research, Telcordia Technologies  
One Telcordia Dr., Piscataway, NJ 08854  
Email: oyounis@research.telcordia.com

Marwan Krunz and Srinivasan Ramasubramanian

Dept. of Electrical and Computer Engineering  
University of Arizona, Tucson, AZ 85721  
Email: {krunz,srini}@ece.arizona.edu

**Abstract**—When sensors are redundantly deployed, a subset of sensors should be selected to actively monitor the field (referred to as a “cover”), while the rest of the sensors should be put to sleep to conserve their batteries. We consider networks in which all the nodes are not aware of their locations or the relative directions of neighbors. We develop several geometric and density-based tests for deciding whether a sensor should turn itself off without degrading the quality of field coverage. These tests rely on estimated neighbor distances and locally advertised two-hop neighborhood information. We design an algorithm (LUC) that exploits these tests for computing covers. Based on LUC, we propose two *distributed* protocols (LUC-I and LUC-P) that periodically select covers and switch between them to extend “coverage time” and tolerate unexpected failures. Our protocols are highly efficient in terms of message overhead and processing complexity. We implement LUC-I in TinyOS and evaluate it using the TOSSIM simulator. Experimental results indicate that our approach significantly prolongs coverage time.

## I. INTRODUCTION

Several applications, such as environmental monitoring, require sensors be redundantly deployed to accommodate unexpected failures and improve the fidelity of received measurements. Redundancy means that some parts in the field are covered by more than one sensor at the same time. If idle sensors are not put to sleep, then redundant node deployment does not necessarily improve the *coverage time* of the field, defined as the time until the fraction of the area that is monitored by at least one sensor falls below a given threshold. This is because the sensor’s radio expends a significant portion of its battery lifetime in idle-listening to support data forwarding, and thus active sensors tend to die at roughly the same time. For example, the powers consumed by the radio of the MICA2 mote [7] during idle-listening and reception are almost the same, as reported in [2]. It was also reported in [13] that in the WINS Rockwell seismic sensor the power consumed in the receive and idle-listening modes are 0.36 mW and 0.34 mW, respectively. In contrast, the energy consumed in the sleep mode of the MICA2 is three orders of magnitude less than that during idle-listening.

Therefore, the network topology should be controlled by selecting a subset of nodes to actively monitor the field and putting the remaining nodes to sleep. More specifically, if the set of nodes in the network is  $V$ , it is required to select a subset  $V_A \subseteq V$  that covers the entire area covered by  $V$  ( $V_A$

is referred to as a “cover”). The remaining set of nodes  $V_S = V - V_A$  can be put to sleep and later activated to form new covers. Besides prolonging coverage time, reducing the number of active nodes also reduces channel-contention and collisions in the network.

Most proposed protocols for selecting sensor covers assume that nodes can estimate their locations (via localization techniques) or at least the directions of their neighbors (e.g., [9], [16], [19], [20], [25]). Equipping every node with a GPS is not cost effective, so localization is typically performed by estimating distances between neighboring nodes (e.g., RADAR [3]) and triangulating positions using a small set of location-aware *anchor* nodes (e.g., [5]). In this work, we focus on applications in which network-wide localization is unnecessary and possibly infeasible. Localization is unnecessary in applications that do not require reporting the location of events. An example is a warfare scenario, where the detection of any radiation or chemical activity is enough to alert the troops to evacuate. Localization may also be infeasible due to the failure or bad distribution of anchor nodes. Note that node localization can be performed based only on distance estimates and arbitrary selection of anchors. However, in practice, this approach may result in failure to place some nodes due to inaccurate estimation of distances [23]. Thus, we need new redundancy check tests that are based on “distance ranges” and not on “locations.”

**Contributions.** In this work, we develop four tests for determining node redundancy, assuming that nodes are *not* aware of their locations or the relative directions of neighbors. Two of the proposed tests are geometrically provable, while the other two are based on the dense random deployment. To determine if a node  $v$  is redundant, our tests exploit the two-hop neighborhood information advertised by  $v$ ’s one-hop neighbors, as well as the estimated neighbor distances. We propose a location-unaware coverage algorithm (LUC) that incorporates our tests. Then, we design two computationally efficient *distributed* protocols for periodically selecting new active covers and switching between them. We refer to these protocols as LUC-I and LUC-P (for iterative- and probabilistic-LUC, respectively). We implement LUC-I in TinyOS [17] and evaluate it in the context of a multi-hop network application using the TOSSIM simulator [17]. To the best of our knowledge, our work is the first to study cover selection in the absence of information about the relative

locations of nodes.

The rest of the paper is organized as follows. Section II briefly surveys related work. Section III introduces our redundancy check tests. Section IV presents the LUC algorithm and its associated protocols. In Section V, we study the properties of our protocols via simulation. Section VI describes our implementation of LUC-I in TinyOS and evaluates its performance in the context of a multi-hop network application. Finally, Section VII gives concluding remarks.

## II. RELATED WORK

Several protocols were proposed for selecting sensor covers. These protocols are either centralized (e.g., [6], [9], [12]) or distributed (e.g., [19], [20], [25]). They target either field coverage, where a whole area is to be monitored, or target coverage, where a set of targets in the field are to be monitored.

Cardei et al. [6] computed a number of set covers that maximize the lifetime of the sensor network. They proposed two *centralized* heuristic techniques for target coverage; one uses linear programming and the other is a greedy approach. We use the greedy approach in [6] as a baseline for comparison with our protocols (see Section V). Meguerdichian et al. [12] proposed centralized algorithms for achieving both deterministic and statistical coverage using Voronoi diagrams. Slijepcevic and Potkonjak [15] proposed a centralized heuristic to compute a disjoint maximal set of covers.

Several distributed algorithms were recently proposed. Wang et al. proposed CCP [19], which probabilistically provides different degrees of coverage according to the application requirements. Zhang and Hou proposed OGDC [25], which determines the minimum set of working nodes by reducing their overlap. They provided necessary conditions on the ratio between the sensing and transmission ranges to guarantee that coverage implies connectivity, and studied the case where the sensing range is non-uniform. Tian and Georganas [16] proposed a simple approach for selecting covers based on checking the *sponsored area*, defined as the area covered by other working neighbors. If the union of all sponsored areas includes the sensing area of a sensor, then this sensor decides to go to sleep. Gupta et al. [9] and Iyengar et al. [10] proposed algorithms for selecting connected covers. These algorithms do not enforce constraints on the relation between sensing and transmission ranges. Yan et al. [20] proposed a protocol for collaborative sleep and wakeup among neighboring nodes, assuming that the network is synchronized. Ye et al. [21] proposed the PEAS protocol that provides fault-tolerance coverage using randomized sleep/wakeup schedules. PEAS focused on maintaining network connectivity by periodically awakening nodes to probe the active ones. Kumar et al. [11] provided theoretical bounds on the number of nodes required to achieve  $k$ -coverage.

All the aforementioned protocols assumed that nodes can estimate their locations and/or the directions of their neighbors.

They also assumed that the employed localization mechanisms can provide reasonably accurate location estimates of *all* the nodes, which may be difficult in large-scale networks [23]. In this work, we do not make any of these assumptions.

## III. DETERMINING NODE REDUNDANCY

Below, we introduce our system model and describe our proposed tests for determining node redundancy.

### A. System Model

We consider sensor nodes for which  $R_t$  is the maximum transmission range and  $R_s$  is the maximum sensing range (i.e., the distance from the sensor after which an event or phenomenon is not detectable). We assume the following:

- 1) Nodes are randomly and redundantly deployed. They have similar batteries and energy consumption rates.
- 2) Nodes have omni-directional antennae and do not possess localization capability. Thus, node locations and relative directions of neighbors cannot be estimated.
- 3)  $R_t \geq 2R_s$ . Under this condition, coverage implies connectivity [25]. An example where this assumption holds is the MICA2 mote [7], which has a maximum communication range of about 1000 feet and a sensing range of about 100 feet [20].
- 4) A node can estimate the distances to its one-hop neighbors. This can be achieved using well-known approaches, such as time of flight or RF signal strength [3], [24]. If transmission ranges are short, these approaches can provide reasonably accurate estimates of one-hop distances. For example, the Cricket sensor [7] uses time of flight of packets for accurate ranging based on ultrasound and RF beacons (effect of distance inaccuracy is evaluated in Section V).
- 5) Links can be asymmetric due to radio range irregularity [26]. A node decides whether it is redundant or not based on the one-hop neighbors it is aware of.

### B. Redundancy Check Tests

Let  $N(v, r)$  denote the set of neighbors of node  $v$  that lie within a range  $r$ . Discovering such a set relies on an approach that is described later in Section IV. A node can be in one of three states: ACTIVE, ASLEEP, or UNDECIDED. All nodes start in the UNDECIDED state. Let  $V_U$  denote the set of undecided nodes. Note that  $V = V_A \cup V_S \cup V_U$ . Define  $wgt(v)$  to be the weight of a node  $v$  in the network (e.g., its remaining battery). We will use different definitions for  $wgt(v)$  in our LUC-I and LUC-P protocols, described in Section IV.

We propose two geometrically proven tests (RTest-D1 and RTest-D2) and two density-based tests (RTest-H1 and RTest-H2) for determining node redundancy. RTest-D1 and RTest-D2 decide that a node is redundant only if its sensing region is *geometrically* covered by active nodes. These tests assume

that node  $v$ 's sensing capability is uniform in all directions and thus  $v$ 's sensing range can be approximated by a circle  $C(v)$ . RTest-H1 and RTest-H2 decide that a node is redundant if certain conditions on node density and distribution are satisfied. They only require a node's sensing region be convex but not necessarily circular.

**RTest-D1:** Node  $v$  is redundant if  $\exists$  three nodes  $v_i$ ,  $1 \leq i \leq 3$ , where: (1)  $v_i \in N(v, R_s) \forall i$ , (2)  $v_i \in V_A \forall i$ , (3) the  $v_i$ 's are pairwise neighbors within range  $R_s$ , (4)  $v$  lies inside the triangle formed by the  $v_i$ 's, and (5) the circumference of  $C(v)$  is covered by the  $C(v_i)$ 's.

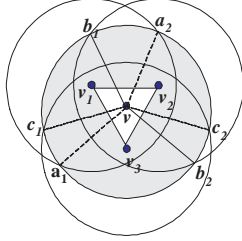


Fig. 1. Demonstrating RTest-D1 where  $v$  lies inside the triangle formed by three of its neighbors.

RTest-D1 can be explained in the context of Figure 1 (we use three neighbors in this test for tractability). Nodes  $v_1$ ,  $v_2$ , and  $v_3$  are active neighbors of node  $v$  and are pairwise neighbors within range  $R_s$  (the first three conditions). Two conditions need to be satisfied. First,  $v$  should lie inside the virtual triangle formed by the lines connecting the three neighbors. Second, the sectors where the  $C(v_i)$ 's intersect with  $C(v)$  should completely cover  $C(v)$ . For example, in Figure 1,  $C(v_1)$ ,  $C(v_2)$  and  $C(v_3)$  intersect  $C(v)$  in sectors  $a_1va_2$ ,  $b_1vb_2$ , and  $c_1vc_2$ , respectively. The challenge is how to determine these sectors in the absence of location information. We propose a simple approach to solve this problem. An estimate of the distance between any two neighbors of  $v$  can be determined (see Section IV-A.1), as well as two-hop neighborhood. Thus,  $v$  can compute relative coordinates of the  $v_i$ 's as follows. Node  $v$  assumes that it resides at  $(0,0)$  and that  $v_1$  resides at  $(d_1,0)$ , where  $d_1$  is the estimated distance between  $v$  and  $v_1$ . It then uses the distance between itself and  $v_2$  ( $v_3$ ) and the distance between  $v_1$  and  $v_2$  ( $v_3$ ) to assign coordinates for  $v_2$  ( $v_3$ ). Based on these coordinates,  $v$  can determine whether or not it lies inside the triangle  $v_1v_2v_3$  and can compute the intersection sectors.

**Lemma 1:** RTest-D1 provides a sufficient condition for the redundancy of node  $v$ .

**Proof.** It is trivial to show that if the conditions in RTest-D1 are satisfied then  $v$ 's sensing range is covered. However, these conditions may not be satisfied even though  $v$ 's sensing range is covered by active nodes. Thus, RTest-D1 provides a sufficient but not necessary condition for redundancy.

We empirically evaluated the conservativeness of RTest-D1 by randomly placing three neighbors of node  $v$  within range

$R_s$  and computing the success ratio over 10,000 experiments (we say that RTest-D1 fails if the three neighbors form a correct cover but the test cannot determine that). RTest-D1 showed a success ratio of about 57%.

**RTest-D2:** Node  $v$  is redundant if  $\exists$  three nodes  $v_i$ ,  $1 \leq i \leq 3$ , where (1)  $v_i \in N(v, 0.618R_s) \forall i$ , (2)  $v_i \in V_A \forall i$ , and (3) the  $v_i$ 's are pairwise non-neighbors within range  $R_s$  (i.e.,  $v_i \notin \bigcup_{j \neq i} N(v_j, R_s), \forall i$ ).

**Lemma 2:** RTest-D2 provides a sufficient condition for the redundancy of node  $v$ .

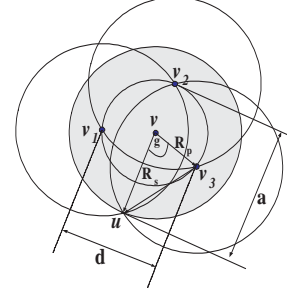


Fig. 2. Determining the probing range ( $R_p$ ) in RTest-D2.

**Proof.** Geometrically, if the centers of three circles lie sufficiently close to the center of a fourth one, then the three circles completely cover the fourth one. The problem is how to determine the radius  $R_p$  of the largest probing circle in which the three centers of  $C(v_1)$ ,  $C(v_2)$ , and  $C(v_3)$  lie and completely cover  $C(v)$ . Consider the scenario depicted in Fig. 2. The worst-case organization (which results in the smallest  $R_p$ ) occurs when  $v_1$  lies on the boundary of the circle of radius  $R_p$  (probing circle),  $v_2$  lies on the boundary of  $C(v_1)$  and the probing circle, and  $v_3$  lies on the boundary of  $C(v_2)$  and the probing circle, such that  $C(v_3)$  barely covers the remaining region of  $C(v)$ . Let  $a \stackrel{\text{def}}{=} R_s + R_p$ . From basic geometry, we have:

$$a = \frac{1}{d} \times \sqrt{d^2(-d + 2R_s)(d + 2R_s)} = \sqrt{4R_s^2 - d^2},$$

where  $d \stackrel{\text{def}}{=} \|v_1v_3\|$  as shown in Fig. 2. To compute  $d$ , consider the angle  $g$  in Fig. 2. Clearly  $g = \cos^{-1}(\frac{R_p}{2R_s})$ . Since  $\cos^{-1}(x) = \sin^{-1}\sqrt{1-x^2}$ ,  $g = \sin^{-1}\sqrt{1 - \frac{R_p^2}{4R_s^2}}$ , which results in  $d = 2R_p \sin(g) = 2R_p \sqrt{1 - \frac{R_p^2}{4R_s^2}}$ . Thus,  $R_s + R_p = \sqrt{4R_s^2 - 4R_p^2(1 - \frac{R_p^2}{4R_s^2})}$ . This results in the following quartic equation:  $R_p^4 - 5R_s^2R_p^2 - 2R_s^3R_p + 3R_s^4 = 0$ . Solving for  $R_p$  yields:  $R_p = (\sqrt{5}/2 - 1/2)R_s \approx 0.618R_s$ . Thus, if the conditions provided in RTest-D2 are satisfied for node  $v$ , we can assert that  $v$  is redundant. However, the converse is not true (i.e., RTest-D2 is a sufficient condition for redundancy).

Our empirical evaluation of the conservativeness of RTest-D2 indicates a success ratio of 3.2%. However, this test is still useful since it is computationally cheap and shows significant

success when individually applied in dense settings, as shown in Section V.

Since RTest-D1 and RTest-D2 only provide sufficient conditions for determining node's redundancy, we provide other tests (RTest-H1 and RTest-H2) that exploit conditions on node density to improve the confidence in determining redundancy.

**RTest-H1:** A node  $v$  is redundant if (1)  $\exists S = \{v_i \in V_A : v_i \in N(v, R_s), 1 \leq i \leq M, M \geq 4\}$ , and (2)  $N(v, R_s) \subseteq \bigcup_{i=1}^M N(v_i, R_s)$ .

RTest-H1 says that if  $v$  has  $M \geq 4$  active neighbors within distance  $R_s$  and if every neighbor of  $v$  is also a neighbor of one or more of these  $M$  active neighbors, then  $v$  is considered redundant. The choice of four neighbors stems from the fact that the network is “sufficiently” dense (as defined below) when each node has neighbors in all directions (north, south, east, and west). The density model defined below is an extension of the work in [4].

**Lemma 3:** Assume that  $n$  nodes with sensing range  $R_s$  are deployed uniformly and independently in a field  $F = [0, L]^2$ . Let  $F$  be divided into  $5L^2/R_s^2$  square cells, each of side length  $R_s/\sqrt{5}$ . Let  $R_s^2 n = aL^2 \ln L$ , for some  $a > 0$  such that  $R_s \ll L$  and  $n \gg 1$ . If  $a \geq 10$ , then  $\lim_{L \rightarrow \infty} E(\mu_0(n)) = 0$ , where  $\mu_0(n)$  is a random variable that denotes the number of empty cells in  $F$ .

The lemma states that if the network area is divided into small square regions (cells), and if  $n$  and  $R_s$  satisfy a certain constraint, then every cell will contain at least one sensor asymptotically almost surely (a.a.s.). Consequently, every node (except those at the borders) will have at least one neighbor in each of the four main directions. The proof of Lemma 3 is a straightforward extension of the one provided in [4] for one-dimensional networks. We omit it here for brevity<sup>1</sup>.

**Lemma 4:** If the density model provided in Lemma 3 is satisfied, then RTest-H1 correctly determines whether  $v$  should be put to sleep.

**Proof.** Consider the scenario depicted in Fig. 3. Assume a worst-case configuration in which all the active neighbors of node  $v$  ( $v_1, v_2, v_3$ , and  $v_4$ ) are placed in  $B$ . Also assume that one undecided neighbor  $v_5$  is placed on the boundary of  $A$  while the rest of the neighbors are all asleep. We prove Lemma 4 by contradiction. Assume that  $v$  is put to sleep although cell  $C$  is not covered. Now, consider a node  $u$  that resides in  $D$ . If  $u$  is active, then  $C$  is covered. If  $u$  is not active, then  $v_5$  must be included in the probing neighborhood of  $u$ 's active neighbors. This also implies that  $C$  will be covered, which contradicts the original supposition.

The above argument may only be violated at the cells on the corners of the network area. This does not affect the coverage of the field since the number of cells is supposedly  $\gg 1$ . Our

<sup>1</sup>We chose  $R_s/\sqrt{5}$  as our cell dimension to allow a node to completely cover neighboring cells in the four main directions. This is different from the cell definition used in [4].

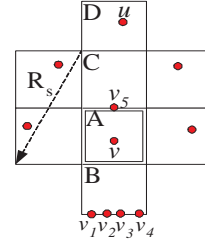


Fig. 3. Correctness of RTest-H1.  $\{v, u, v_1, \dots, v_5\}$  identify nodes while  $\{A, B, C, D\}$  identify square cells within the network area.

simulation experiments (Section V) indicate that RTest-H1 is the most effective in redundancy elimination.

**RTest-H2:** Assume that nodes are uniformly and redundantly deployed. A node  $v$  is considered redundant if  $\exists S = \{v_i : v_i \in N(v, R_s), v_i \in (V_A \cup V_U), 1 \leq i \leq \gamma\}$ , where  $\gamma$  is a constant, such that (1)  $wgt(v) < wgt(v_i) \forall v_i \in S$ , and (2)  $N(v, R_s) \subseteq \bigcup_{i=1}^{\gamma} N(v_i, R_s)$ .

Unlike previous tests, RTest-H2 aims at turning off “weak” nodes in the network. It checks if node  $v$  has at least  $\gamma$  active or undecided neighbors within distance  $R_s$ , and if every other neighbor of  $v$  is also a neighbor of one or more of these  $\gamma$  neighbors. Node  $v$  is then put to sleep if its weight is the smallest among the  $\gamma$  neighbors. The rationale for this test is that if nodes are uniformly deployed, then  $v$  will have neighbors in all directions (as assumed in RTest-H1), which are likely to cover the entire sensing region of  $v$ . Putting  $v$  to sleep early will force “stronger” neighbors to become active. We recommend  $\gamma > 4$  to avoid leaving uncovered holes (we use  $\gamma = 6$  in Section V).

#### IV. LOCATION-UNAWARE COVERAGE

Based on the proposed redundancy check tests, we now present the LUC algorithm and present two distributed protocols to realize it in operational scenarios.

##### A. LUC Algorithm

1) *Distance Estimation:* The LUC algorithm at node  $v$  proceeds as follows. Node  $v$  discovers its neighbors within the range  $R_t$  (its maximum transmission range) and their approximate distances (estimated using the time of flight and/or received signal strength, as described in Section III-A). To accommodate uncertainties due to fading, reflection, and radio sensitivity, we use a conservative approach to estimate distances in which  $R_t$  is divided into a discrete set of  $n_d$  distances<sup>2</sup> and every range of signal strengths is mapped to one of these distances (similar to radio maps [24]). Every node broadcasts the estimated distances to its neighbors so that every node is aware of its 2-hop neighborhood. Our simulation experiments in Section V show that imprecision in distance

<sup>2</sup>We refer to  $n_d$  as the “discretization level.”

estimation does not affect the performance of our protocols. This is due to the efficiency of RTest-H1.

2) *Activation Test*: LUC uses an activation test (ATest) that returns SUCCESS (i.e., “tentatively” sets the state of an undecided node  $v$  to ACTIVE) if  $v$  has the highest weight among its undecided neighbors (note that the weight is a real number and thus no ties occur). The test is necessary to seed the network with active nodes and force “stronger” nodes to be active, giving an opportunity for putting “weaker” nodes to sleep. This has a desirable effect of keeping every individual node alive as long as possible for better reliability against unexpected node failures. Note that when ATest succeeds, the LUC algorithm (Fig. 4) gives another opportunity to a node to go to sleep (by executing RTest-H1) before making the activation final.

**ATest**: A node  $v \in V_U$  is tentatively active if for every node  $u \in N(v, R_s)$  with  $u \in V_U$ , we have  $wgt(u) < wgt(v)$ .

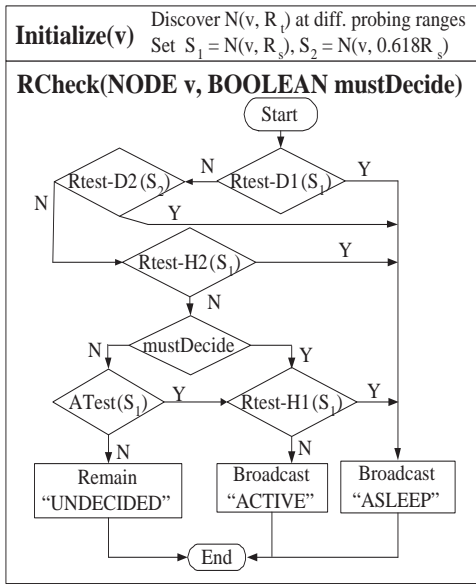


Fig. 4. The LUC algorithm.

3) *Order of Test Execution*: The flowchart of the LUC algorithm is provided in Fig. 4. LUC executes the geometric tests before the density-based ones since they guarantee no false positives (erroneously putting a necessary node to sleep) even under arbitrary node distributions. RTest-H1 is invoked only when the node is about to make a decision because it is the most aggressive test, as shown in Section V. The RCheck( $v$ ,  $mustDecide$ ) function in Fig. 4 checks whether a node must make a decision (ACTIVE/ASLEEP) or can remain in the UNDECIDED state. If  $mustDecide=1$ , then the node must make a decision regardless of the status of its neighbors.

### B. Iterative LUC (LUC-I) Protocol

We refer to this protocol as “Iterative LUC” (LUC-I) because a node does not decide to be active unless all its

neighbors that are within range  $R_s$  and having higher weights have made their decisions. However, a node can put itself to sleep once any redundancy check test is satisfied. In LUC-I, the weight of the node is selected in a way that favors activating nodes with higher residual energy. Thus, the weight of node  $v$  is set to:  $wgt(v) = \frac{e(v)}{e(v) + \sum_{i=1}^{|N(v, R_s)|} e(i)}$ , where  $e(v)$  is the residual energy of node  $v$  and  $|N(v, R_s)|$  is the number of  $v$ 's neighbors within range  $R_s$ .

LUC-I has two phases. The first phase is a *neighbor discovery* phase that runs for  $t_{nd}$  seconds. By the end of this phase,  $v$  will have discovered its neighbors and their weights, and computed the neighbor sets  $S_1$  and  $S_2$ . In the second phase,  $v$  starts the *coverage process*, which runs for  $t_{cp}$  seconds. Whenever  $v$  receives an update from one of its neighbors changing its state to ACTIVE or ASLEEP, it executes RCheck with  $mustDecide=0$  (see Fig. 4). If  $v$  decides that it is redundant, it does not wait until the end of the  $t_{cp}$  interval and immediately goes to sleep. If  $v$  decides to become active, it broadcasts its new state and keeps checking its redundancy whenever one of its probing neighbors becomes active. This continues until the end of the  $t_{cp}$  interval (in order to allow  $v$  to *prune* itself from  $V_A$  if possible). If the  $t_{cp}$  interval expires before  $v$  has made a decision,  $v$  executes RCheck one last time with  $mustDecide = 1$ . LUC-I is re-invoked every  $t_{cu}$  seconds, which we refer to as the “cover update” interval. The pseudo-code for the LUC-I protocol is provided in Fig. 5.

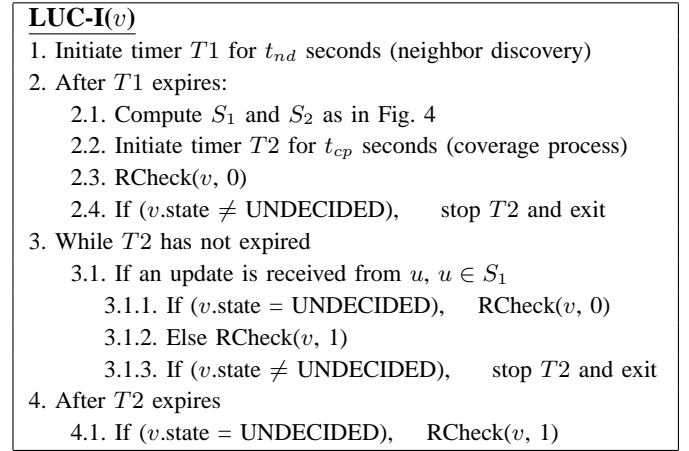


Fig. 5. Pseudo-code for the LUC-I protocol executed at node  $v$ .

### C. Probabilistic LUC (LUC-P) Protocol

Under worst-case distribution of node weights, where every node has to wait for another one, the convergence of LUC-I will be dependent on the number of nodes (Section IV-D). This motivates the need for another protocol in which nodes autonomously decide to join  $V_A$  or  $V_S$  within a fixed number of iterations, regardless of the network size. In the probabilistic LUC protocol (LUC-P), a node  $v$  is added to  $V_A$  according to an activation probability  $P_{on}$  that corresponds to the remaining

energy in  $v$ . This way, decision-making is implicitly based on node weights without needing to exchange information among neighbors. Pseudo-code for the LUC-P protocol can be found in [22].

The neighbor discovery phase in LUC-P is similar to that of LUC-I. At the start of the coverage process phase, node  $v$  initializes an activation probability  $P_{on}$  as follows:

$$P_{on} = \frac{e(v) \times P_{start}}{e(v) + \sum_{i=1}^{|N(v, R_s)|} e(i)}, \quad P_{min} \leq P_{on} \leq 1 \quad (1)$$

where  $P_{start}$  is an initial probability that is periodically doubled by every node. Increasing  $P_{start}$  leads to an increase in  $P_{on}$  and this causes the set of active nodes to grow gradually.  $P_{min}$  is the smallest allowed value for  $P_{on}$ . As described below,  $P_{min}$  ensures that LUC-P terminates in a constant number of iterations. Node  $v$  initializes timers  $T1$  and  $T2$  as in LUC-I. A new timer  $T3$  is also initialized to  $t_{at}$  (activation test) seconds in order to periodically check  $v$ 's eligibility to join  $V_A$ . Whenever  $T3$  expires during the coverage process,  $v$  checks whether it can go to sleep. If not,  $v$  uses its  $P_{on}$  to probabilistically set itself to the ACTIVE state. If the activation test is successful,  $v$  performs RTest-H1 before committing itself to  $V_A$ . If  $v$  does not pass the activation test, it doubles its  $P_{start}$  value and re-evaluates  $P_{on}$ . As in LUC-I, if  $v$  has decided to become active, it is still allowed to *prune* itself from  $V_A$  if it becomes redundant before  $T2$  expires. If  $T2$  expires and  $v$  is still in the UNDECIDED state,  $v$  decides to become active. Ideally, the  $t_{at}$  value of  $T3$  should be selected such that  $v$ 's  $P_{on}$  probability is allowed to grow to 1 before  $T2$  expires. Since  $P_{min}$  is constant, the maximum number of iterations until  $P_{on}$  reaches 1 ( $N_{max}$ ) is also constant (computed below). Therefore,  $t_{at}$  should be selected as  $t_{at} = t_{cp}/N_{max}$ .

#### D. Analysis

**Correctness.** Three observations can be made about our LUC protocols. First, when the protocols terminate, every node in the network will have joined  $V_A$  or  $V_S$ . Second, the area covered by the nodes in  $V_A$  is equal to that covered if all the nodes in  $V$  are active. Third, applying the LUC protocols improves coverage time and reliability in the network. The first observation stems from the fact that every node is forced to decide after a timer expires ( $T2$  in LUC-I or  $T3$  in LUC-P) to avoid waiting indefinitely for neighbors that may have failed. The second observation follows from the sequential construction of  $V_A$  according to node weights. A node whose sensing range is not completely covered will fail all the redundancy check tests and will have to be activated. Note that the size of the selected cover is governed by the quality of the redundancy check tests and not the operation of the protocols. The third observation is due to the refreshment of  $V_A$  every  $t_{cu}$  interval, which results in distributing energy consumption and extending the lifetime of every individual sensor.

**Time Complexity.** The time complexity of our LUC protocols is defined in terms of: (1) the average number of iterations until

convergence ( $N_{iter}$ ), and (2) the processing time at each node. An ‘‘iteration’’ is defined as one attempt by a node to decide whether to go to sleep or continue executing the LUC protocol. In LUC-I,  $N_{iter} \sim O(n)$  in the worst-case (very rare). To bound the convergence time of LUC-I, we limit the time of the coverage process by a timer (as described in Section IV-B). The worst-case  $N_{iter}$  for LUC-P is  $\lceil \log_2 \frac{1}{P_{min}} \rceil + 1$ . For example, for  $P_{min} = 0.01$ ,  $N_{iter} = 8$  iterations.

The processing complexity of the LUC algorithm is not significant. For a node  $v$ , RTest-D1 and RTest-D2 take  $O(n_a^3)$  in the worst case, where  $n_a$  is the number of active neighbors of  $v$ . RTest-H1 and RTest-H2 take  $O(n_b)$  time, where  $n_b$  is the number of neighbors within range  $R_s$ . ATest takes  $O(n_a)$  time for LUC-I and  $O(1)$  time for LUC-P.

**Message overhead.** In the LUC protocols, each node sends one message to announce its information (e.g., identifier and remaining energy), another message to announce its 1-hop neighbors and their proximities, and a third message to announce its decision (ACTIVE/ASLEEP), if any. Therefore,  $O(1)$  messages are required per node.

## V. SIMULATION EXPERIMENTS

We first focus on the construction of one cover only and assume that  $n$  nodes are randomly distributed in a  $50 \times 50$  meters<sup>2</sup> field ( $n = 1000$ , unless otherwise specified). We focus on a snapshot during network operation where the residual energy of each node is a uniformly distributed fraction between 0 and 1. We choose  $n_d = 10$  (other values did not have a noticeable effect on LUC's performance). Every point in our results is the average of 10 experiments of different random topologies. We focus on the following metrics: (1) size of the active set  $V_A$ , (2) coverage redundancy, (3) coverage quality, and (4) average residual energy in  $V_A$ . Coverage quality is defined as the fraction of the field area covered by  $V_A$ . Coverage redundancy is defined as the minimum number of sensors covering any point in the sensing range of any active sensor. Thus, the optimal value for coverage redundancy is 1.

We compare LUC-I and LUC-P to a centralized approach (Greedy-MS) [6] and a distributed approach [16] that assume complete knowledge of node locations. Although Greedy-MS was proposed for target coverage and not area coverage, we generalize it by discretizing the area into a large number of points (targets). Individual cover selection in Greedy-MS follows the approximation algorithm in [8], which tries to minimize the size of every  $V_A$ . Thus, we conjecture that its selected  $V_A$  will be smaller than that selected by any distributed protocol. The distributed approach in [16] uses a geometric test assuming that relative node locations are known. Every node  $v$  is initially active and has a randomly set timer  $T$ . When  $T$  expires,  $v$  checks if the areas covered by its currently active neighbors (referred to as *sponsored sectors*) completely span its sensing range. If so, then  $v$  decides to go to sleep. We refer to this approach as SP-SECT.

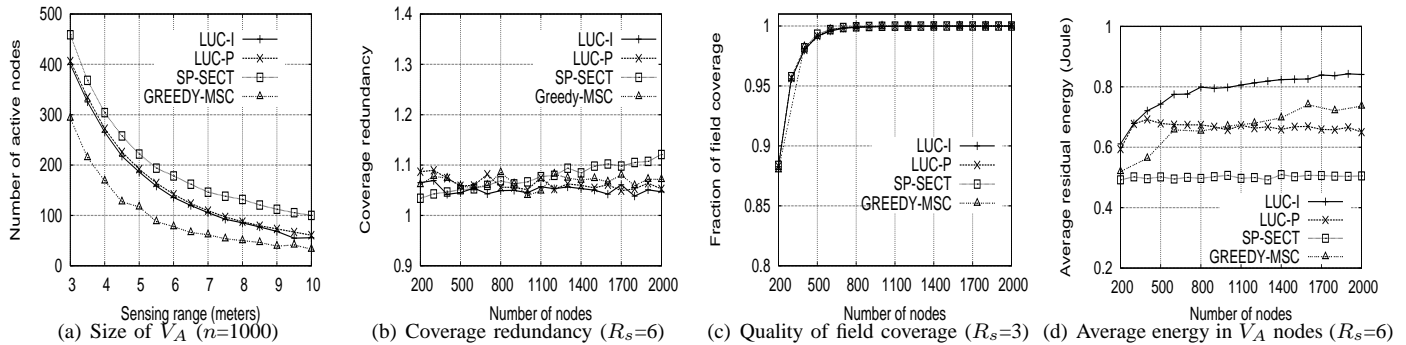


Fig. 6. Performance of LUC-I and LUC-P in contrast to SP-SECT and Greedy-MS-C.

Fig. 6(a) depicts the size of  $V_A$  ( $|V_A|$ ) as a function of the sensing range. As expected,  $|V_A|$  drops for all the compared protocols as  $R_s$  increases.  $|V_A|$  of LUC-I and LUC-P is about 20-50% larger than that generated by Greedy-MS-C and is about 10-35% smaller than that of SP-SECT. This is a very good result given that our protocols are distributed and location-unaware. LUC-P generates a set  $V_A$  that is about 2-10% larger than that of LUC-I.

Fig. 6(b) shows that all protocols have coverage redundancies that are about 3-6% from the optimal value (except for SP-SECT which goes to 10% redundancy for large densities). This indicates that our LUC protocols are able to compute near-minimal covers (a minimal cover is one in which every sensor is necessary for field coverage).

Fig. 6(c) shows that all the compared protocols have similar coverage quality, which is governed by the node density. We also study the quality of the selected  $V_A$  in terms of the average battery levels of nodes included in it. Selecting active nodes that are richer in energy than their peers is important for maximizing the lifetime of every individual sensor. Fig. 6(d) illustrates that LUC-I and LUC-P select nodes that have higher average residual energy compared to SP-SECT and Greedy-MS-C.

We also assess the performance of our proposed tests. We fix  $R_s$  at six meters and vary  $n$ . Figure 7(a) shows the fraction of the nodes that are put to sleep by each test (compared to the total number of deployed nodes) when the original LUC algorithm is applied. The figure indicates that the density-based tests are very effective in eliminating redundancy. This is because they can put a node to sleep faster than the geometric tests. The effectiveness of the geometric tests is demonstrated in Figure 7(b), where each test is applied individually (i.e., without the application of the other test). Coverage quality is not compromised under any test, as shown earlier.

Now, we assess the impact of our protocols on coverage time and consider a simple operational scenario in which the energy consumed by a node's radio is dominated by the wake state. We deduct a fixed amount of energy from the node's battery according to its state. Every node starts with a full

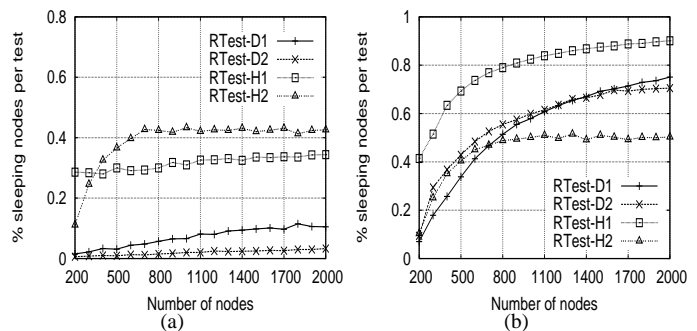


Fig. 7. Evaluating the percentage of nodes put to sleep by our tests when applied: (a) in combination, or (b) individually.

battery of 1 Joule, and consumes  $10^{-4}$  Watts while being active and  $10^{-7}$  Watts while being asleep. We take  $R_s = 6$  meters and update the network cover every 100 seconds. Fig. 8(a) shows the fraction of the field that is covered by active nodes when  $n = 500$  nodes. If none of the nodes is allowed to sleep, the network becomes completely uncovered after 100 time steps (a time step corresponds to 100 seconds). Redundancy elimination within a sleep/wakeup mechanism improves coverage time by a factor of 3 to 6. The figure shows that LUC-I and SP-SECT behave similarly. This is a very good result given that LUC-I is location-unaware. The difference in coverage time between LUC-P and LUC-I is attributed to the fact that the latter selects a smaller  $V_A$  at smaller node densities. Fig. 8(b) shows that the discretization level of the radio range ( $n_d$ ) has an unnoticeable effect on coverage time in LUC-I. This is attributed to the effectiveness of RTest-H1.

## VI. IMPLEMENTATION

We implement LUC-I within the TinyOS [17] operating system. Since in our protocols, the weight of a node depends on the remaining battery, we augment TinyOS with an energy dissipation model (explained below) that is based on the radio of Berkeley MICA2 mote [7]. We also integrate a TinyOS multi-hop network application with LUC-I to evaluate an operational scenario. In this application, reports are periodically transmitted to an observer (Surge) [1], [17]. While

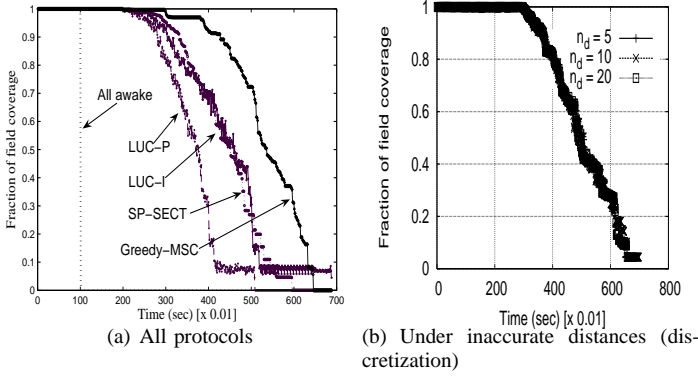


Fig. 8. Coverage quality in a scenario with  $n = 500$  and  $R_s = 6$  meters.

constructing the routing tree, nodes that can communicate directly with the sink are labeled “level-1” nodes. Nodes that cannot communicate with the observer but can communicate with level-1 nodes are labeled “level-2” nodes, and so on.

In our experiments, we use the documented power consumption values from the Chipcon CC1000 data-sheet and measurements reported in [14]. The MICA2 mote consists of several components, including a processor, a radio, and a sensing device. The operational energy consumption of a component  $c_i$  is given by  $e_i = V_i \times A_i \times t_i$ , where  $V_i$  is the voltage across  $c_i$ ,  $A_i$  is the current drawn by  $c_i$ 's circuits, and  $t_i$  is the time taken by  $c_i$  to complete its operation. For example, the radio of the MICA2 sensor draws 8 mA while receiving. Assuming that  $V_i = 3$  Volts and the radio is in the receive mode for 1 second, then  $e_i = 24$  mJ. While the radio is in the sleep mode, and assuming no other activities in the mote during radio sleep,  $A_i = 1 \mu\text{A}$ , and therefore,  $e_i = 3 \mu\text{J}$ . For transmission, the value of the PA.POW register determines the drawn current and the transmission range. For instance, assuming that the mote consumes 1 mW during transmission (which corresponds to 16.8 mA), the bit transmission time is  $62.4 \mu\text{sec}$  (as measured in [14]), and the packet size is 36 bytes,  $e_i$  for one packet transmission is 0.9 mJ. In our evaluation, we transform the energy cost into integer values.

### A. System Design

We discuss the design details of the multi-hop routing module in TinyOS [1] when augmented with our LUC-I protocol. LUC-I extends the existing multi-hop router by adding the cover selection logic, which is executed prior to parent selection in the routing tree. This adds about 1100 lines of code to the TinyOS code<sup>3</sup>. Most of this code is added to the module responsible for parent selection. We used a packet size of 100 bytes in the Surge application to accommodate larger routing tables (the default packet size in TinyOS is 36 bytes). Enlarging the packet size was needed to facilitate evaluation of the code, and is not a requirement of our design.

<sup>3</sup>The complete implementation can be obtained by contacting the authors.

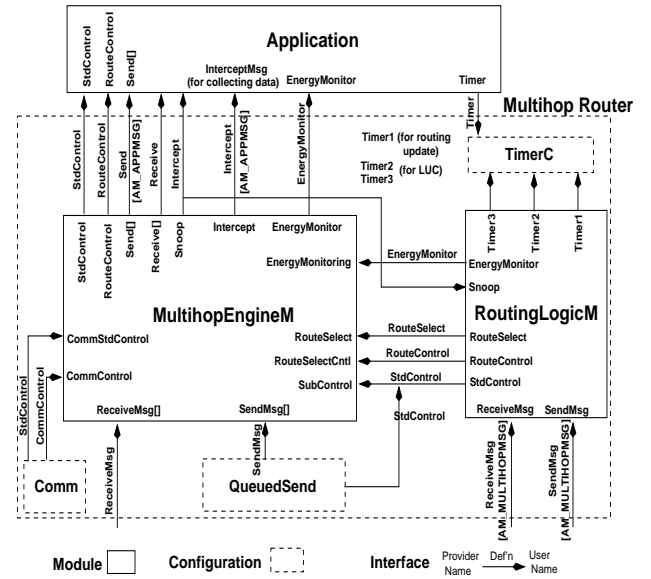


Fig. 9. Multi-Hop routing in TinyOS augmented with LUC-I and energy control. Arrows show interface provider/user relationships.

The schematic diagram for the extended multi-hop router is provided in Fig. 9 (extension to that in [1]). RoutingLogicM is the module that performs the cover selection. It also executes the link estimation and parent selection (LEPS) algorithms. Parent selection is responsible for estimating the link cost for each neighbor based on the “quality” of communications and its proximity to the observer [18].

Our extended design introduces an energy monitor interface (*EnergyMonitor*), which is used to deduct energy during transmission, reception, and sleep cycles. In our evaluation, we ignore the energy consumed in processing and sensing. The “Multihop Router” uses the energy monitor to inform the application whether the battery is still operational. The application uses this information to stop data transmission. The Comm interface, illustrated in Fig. 9, is responsible for packet capture and transmission. The Message ID is used to identify whether the packet is an application packet (AM\_SURGEMSG) that is sent through the MultihopEngineM module or a routing update packet (AM\_MULTIHOPMSG) that is sent through the RoutingLogicM module. The Queued-Send interface is responsible for buffering packets to be sent in sequence. Details of the Comm and QueuedSend interfaces can be found in [17].

We implemented new timers to support the operation of LUC-I, as was described in Fig. 5. When the LUC algorithm is triggered, a node that is in the ASLEEP state moves to the UNDECIDED state and starts the neighbor discovery phase. On the other hand, a node that is initially in the ACTIVE state moves to another state called ACTIVE\_UNDECIDED. In this state, a node continues to send data packets as usual while executing LUC-I so as not to interrupt the network operation. The benefit of this approach will become apparent in the results shown in Section VI-B. A state diagram of a

node executing LUC-I is depicted in Fig. 10.

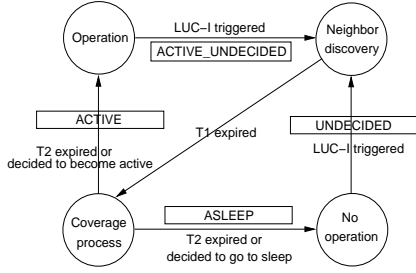


Fig. 10. State diagram for a node executing LUC-I. A circle indicates an action, while a rectangle indicates the new state of the node.

To asynchronously trigger LUC-I in the network, node  $v$  whose timer  $T_2$  has expired *immediately* broadcasts a routing update packet to its immediate neighbors, indicating that  $v$  is starting LUC-I. Upon receiving this message, the neighbors of  $v$  that are close to completing their cycle (i.e., more than  $0.5 t_{cu}$  has passed since the cycle started) trigger the execution of the LUC-I algorithm and re-initialize their  $T_2$  timers. Hence, the start of the coverage process diffuses throughout the entire network. Thus, triggering LUC-I only requires a routing update message. For a realistic scenario where the battery lifetime is in the range of months, LUC-I will be triggered at a coarse granularity, e.g., hours or days.

### B. Performance Evaluation

We conducted experiments to compare the performance of our extended Surge/LUC-I implementation (which we refer to as “LUC-I” for brevity) and the original Surge application that is included in the TinyOS software [17]. We used the TOSSIM discrete-event simulator, which is included with the TinyOS release, to evaluate our implementation. TOSSIM has several advantages: (1) it runs actual TinyOS implementations, (2) it allows experimentation with a large number of nodes, (3) it accurately captures the TinyOS behavior at a low level (e.g., timer interrupts), and (4) it models the CSMA/CA MAC layer of the node. Therefore, imperfections, such as interference and packet collisions, are accounted for.

The parameters used in our experiments are as follows:  $n = 75$ , field size  $L = 50 \times 50$  meters<sup>2</sup> (from (0,0) to (50,50)), the observer is at (25,50), maximum battery =  $6 \times 10^7$  points, energy consumption (receive) = 24000 points/sec, energy consumption (sleep) = 3 points/sec, packet size = 100 bytes, packet transmission time = 49.92 ms, current (transmission) = 10.1-18.5 mA, data rate = 1 packet/10 sec, routing update = 1 packet/15 sec,  $n_d = 10$ ,  $t_{nd} = 20$  sec,  $t_{cp} = 20$  sec, and  $t_{cu} = 300$  sec.  $t_{cu}$  is the time until LUC-I is re-triggered. Every node has to wait for  $t_{nd}$  to collect information about its neighbors. However, a node can terminate LUC-I any time during the ensuing  $t_{cp}$  interval if a *sleep* decision has been made. The values of  $t_{nd}$  and  $t_{cp}$  are selected in a way to allow for at least one routing update to arrive from every neighboring node. The energy consumed during transmission depends on

the PA.POW value, provided in the data sheets of the MICA2 radio. For simplicity, we assume that  $R_t = 16$  meters, which corresponds to a drawn current of 10.1 mA (from the MICA2 CC1000 radio datasheet), and every one-meter increase in  $R_t$  corresponds to the next current value reported in the data sheet. The maximum battery lifetime is selected to be a fraction of the maximum possible for 2 AA batteries of a MICA2 sensor. All the nodes start their operation randomly within an interval [0,5] seconds from the start of the simulation. In our experiments, the depth of the routing tree varies from 2 to 5 according to the specified transmission range and link quality.

We define coverage time here as the time until the observer cannot receive any reports. This occurs when all level-1 nodes deplete their energy. Fig. 11 shows the coverage times for LUC-I and Surge, where LUC-I refreshes  $V_A$  every  $t_{cu} = 300$  seconds. LUC-I provides 100-200% improvement in coverage time over Surge. The amount of gain is affected by two factors: (1) the size of  $V_S$ , and (2) the frequency of updating  $V_A$  ( $t_{cu}$ ). For  $n = 75$  nodes and  $R_s = 12.5$  meters, LUC-I achieves a  $V_S$  of size 20-40 nodes, i.e., about 30-60% of  $n$ . The effect of reducing the size of the active set is apparent only when more level-1 nodes are put to sleep.

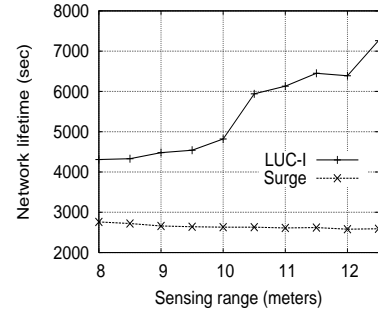


Fig. 11. Coverage time for LUC-I and Surge ( $t_{cu} = 300$  sec).

We also report the number of nodes that fail during operation due to energy depletion with  $R_s = 12.5$  meters and  $t_{cu} = 300$  seconds. Fig. 12 shows that most of the nodes die quickly in the original Surge application. This detrimental effect is due to the continuous listening of all nodes. As for LUC-I, nodes fail gradually because of their periodic sleep/wake-up, which reduces energy consumption among redundant nodes. The time at which a node dies depends on how frequently it was put to sleep during the network operation.

To measure the quality of coverage, the observer keeps track of the nodes from which it has received reports within  $T_c$  seconds, where  $T_c = 20$  sec. Fig. 13 demonstrates the coverage quality as time evolves for different  $t_{cu}$  intervals. An important observation to be made is that LUC-I keeps the field completely covered even when the number of sleeping nodes is as large as  $n/2$ . Other results (shown in [22]), indicate that the best coverage time is achieved at  $t_{cu} = 300$  seconds. Choosing the appropriate value for  $t_{cu}$  to optimize coverage time remains an open issue.

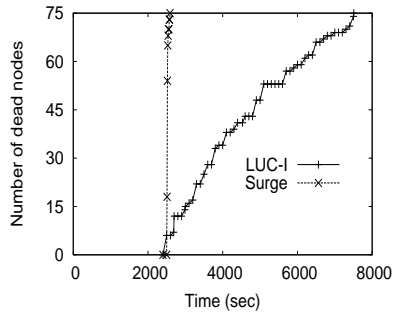


Fig. 12. Node failure rate.

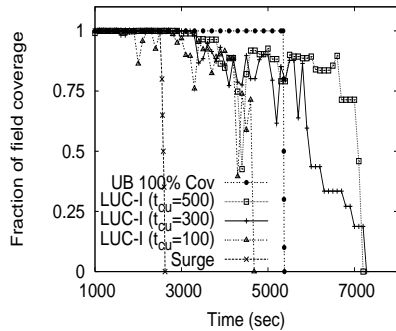


Fig. 13. Coverage quality for  $R_s = 12.5$  meters and different  $t_{cu}$  values.

## VII. CONCLUSIONS

We proposed a novel distributed approach for cover selection in the absence of location information. Our redundancy check tests rely on locally advertised neighborhood information and estimated neighbor distances. We incorporated our tests into a novel coverage algorithm (LUC), and designed two distributed protocols (LUC-I and LUC-P) that realize LUC in multi-hop sensor networks. Our LUC protocols incur low overhead and can significantly reduce the set of active nodes. Simulations showed that the coverage-time extensions achieved by our protocols are comparable to those achieved by a typical distributed protocol and close to those achieved by another centralized protocol, both assuming knowledge of node locations (unlike our protocols). We implemented the LUC-I protocol in TinyOS and incorporated it in a network application used for data aggregation. Experimental results show that LUC-I significantly improves coverage time.

## VIII. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grants CNS-0627118, CNS-0313234, CNS-0325979, and CNS-0435490.

## REFERENCES

[1] Multihop routing for TinyOS. [http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop\\_routing.html](http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html).

[2] G. Anastasi, A. Falchi, A. Passarella, M. Conti, and E. Gregori. Performance measurements of motes sensor networks. In *Proc. of MSWiM*, Oct. 2004.

[3] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location tracking system. In *Proc. of the IEEE INFOCOM Conf.*, Mar. 2000.

[4] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *Proc. of the ACM MobiCom Conf.*, 2002.

[5] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, Oct. 2000.

[6] M. Cardei, M. T. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. In *Proc. of the IEEE INFOCOM Conf.*, Mar. 2005.

[7] Crossbow Technology Inc., <http://www.xbow.com/>, 2007.

[8] U. Feige. A threshold of  $\ln n$  for approximating set cover. *Journal of the ACM*, 45(4):634–652, July 1998.

[9] H. Gupta, S. Das, and Q. Gu. Connected sensor cover: Self organization of sensor networks for efficient query execution. In *Proc. of the ACM International Symposium on Mobile and Ad-Hoc Networking and Computing (MobiHoc)*, June 2003.

[10] R. Iyengar, K. Kar, and S. Banerjee. Low-coordination topologies for redundancy in sensor networks. In *Proc. of the ACM International Symposium on Mobile and Ad-Hoc Networking and Computing (MobiHoc)*, May 2005.

[11] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In *Proc. of the ACM MobiCom Conf.*, Sep. 2004.

[12] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *Proc. of the IEEE INFOCOM Conf.*, Anchorage, Alaska, Apr. 2001.

[13] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19:40–50, 2002.

[14] V. Shnayder, M. Hempstead, B.-R. C. G. Werner, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (ACM SenSys)*, Nov. 2004.

[15] S. Slijepsevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Proc. of the IEEE International Conf. on Communications (ICC)*, June 2001.

[16] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proc. of the First ACM Workshop on Wireless Sensor Networks and Applications*, Sep. 2002.

[17] TinyOS. <http://www.tinyos.net>, 2006.

[18] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (ACM SenSys)*, Nov. 2003.

[19] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Transactions on Sensor Networks*, 1(1):36–72, Aug. 2005.

[20] T. Yan, T. He, and J. Stankovic. Differentiated surveillance for sensor networks. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (ACM SenSys)*, Nov. 2003.

[21] F. Ye, G. Zhong, S. Lu, and L. Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *Proc. of the IEEE Int'l Conf. on Distributed Computing Systems*, 2003.

[22] O. Younis, M. Krunz, and S. Ramasubramanian. On maximizing coverage time in location-unaware wireless sensor networks. Technical report, University of Arizona, July 2006.

[23] O. Younis, S. Ramasubramanian, and M. Krunz. Location-unaware sensing range assignment in sensor networks. In *Proceedings of IFIP Networking*, Atlanta, GA, May 2007.

[24] M. Youssef and A. Agrawala. The Horus WLAN location determination system. In *Proc. of the ACM International Conf. on Mobile Systems, Applications, and Services (ACM MobiSys)*, June 2005.

[25] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc & Sensor Wireless Networks*, 1:89–124, Jan. 2005.

[26] G. Zhou, T. He, S. Krishnamurthy, and J. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proc. of the ACM International Conf. on Mobile Systems, Applications, and Services (ACM MobiSys)*, June 2004.