

Location-Unaware Coverage in Wireless Sensor Networks^{*}

Ossama Younis^{*}

*Applied Research, Telcordia Technologies
One Telcordia Drive, Piscataway, NJ 08854*

Marwan Krunz

Srinivasan Ramasubramanian

*Department of Electrical & Computer Engineering
University of Arizona, Tucson, AZ 85721*

Abstract

In scenarios where sensors are placed randomly, redundant deployment is essential for ensuring adequate field coverage. This redundancy needs to be efficiently exploited by periodically selecting a subset of nodes (referred to as a “cover”) that actively monitor the field, and putting the remaining nodes to sleep. We consider networks in which sensors are not aware of their locations or the relative directions of their neighbors. We develop several geometric and density-based tests that enable a location-unaware sensor to intelligently determine whether it should turn itself off without degrading the quality of field coverage. These tests rely on distance measurements and exchanged two-hop neighborhood information. We design an algorithm (LUC) that exploits these tests for computing covers. Based on this algorithm, we propose two *distributed* protocols (LUC-I and LUC-P) that periodically select covers and switch between them so as to extend the network lifetime 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 the network lifetime and achieves comparable performance to location-aware protocols.

Key words: Sensor networks, field coverage, location-unawareness, distributed protocols, implementation.

^{*} This work was supported by NSF under grants ANI-0095626, ANI-0313234, ANI-0325979, and ANI-0435490. Any opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not reflect the views of NSF.

^{*} Corresponding author.

Email addresses: `oyounis@telcordia.com` (Ossama Younis), `krunz@ece.arizona.edu` (Marwan Krunz), `srini@ece.arizona.edu` (Srinivasan Ramasubramanian).

1 Introduction

Sensor networks are used to support various types of civilian and military applications. In some applications, such as environmental monitoring and wildlife tracking, sensors remain unattended after deployment until their batteries are depleted. Therefore, a primary design objective is to maximize the “network lifetime,” defined here as the time until the external observer (i.e., the sink) no longer receives any sensing reports. Under such a definition, one or more sensors may still be alive even when the observer no longer receives any sensed data¹.

To improve reliability, sensors are 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 network lifetime. This is because the sensor’s radio expends a significant portion of its energy in idle-listening to support data forwarding, and thus live sensors tend to die at roughly the same time. For example, the powers consumed by the radio of the MICA2 mote [1] during idle-listening and reception are almost the same [2]. It was also reported in [3] 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 mote is three orders of magnitude less than that of idle-listening.

To exploit redundancy and prolong the network lifetime, the network topology should be controlled by selecting a subset of sensor nodes to actively monitor the field and putting the remaining nodes to sleep. More specifically, if the set of sensor 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 be later activated to form new covers. Besides prolonging the network lifetime, reducing the number of active nodes also reduces channel-contention and collisions in the network. Note that an additional improvement in network lifetime can be gained by using an intelligent MAC protocol (e.g., SMAC [4]) that puts active nodes to sleep *during* their duty cycles.

Cover selection protocols proposed in the literature often assume that nodes can estimate their locations² (via localization techniques) or at least the relative directions of their neighbors (e.g., [5–9]). Equipping every node with a GPS is not cost effective, so localization is typically performed by estimating distances between neighboring nodes (e.g., [10]) and triangulating positions using a small set of location-aware *anchor* nodes (e.g., [11]). In this work, we focus on applications in which network-wide localization is unnecessary and possibly infeasible. Localization is unnecessary when the location of events is not important as in warfare scenarios where the detection of any radiation or chemical activity is enough to alert the troops to evacuate. Localization may also be infeasible in indoor environments, or even in outdoor deployments where

¹ This definition implies that coverage is valid as long as at least one node in the sensor network is connected to the observer. Other definitions of network lifetime can also be used. Our definition, however, has a practical appeal, i.e., it can be easily measured by the observer.

² We say that a node is “location-aware” if it can estimate its coordinates.

the anchor nodes have no direct line-of-sight with the satellite (e.g., forests).

A node that is not aware of the absolute or relative locations of its neighbors cannot tell how much of its sensing range is already covered by these neighbors. Although node localization can be performed based on distance estimates and arbitrarily selected anchors, localization of some nodes may fail due to inaccurate distance estimation, as shown in [12]. Network-wide localization may also suffer from significant inaccuracies, especially when no anchor nodes are present. In the presence of mobility, relative node locations have to be recomputed periodically, adding significant protocol overhead. For these reasons, we advocate a new approach for determining node redundancy, which does not require network-wide localization. Our approach relies on rough distance estimates between 1-hop neighbors along with advertised 2-hop neighborhood information.

Contributions. In this work, we develop several theoretically provable and heuristic tests for determining node redundancy, assuming that nodes are *not* aware of their locations or the relative directions of neighbors. To determine if a node v is redundant, our tests exploit two-hop neighborhood information provided by v 's one-hop neighbors, as well as estimated neighbor distances. The primary advantages of these tests are their low overhead and robustness against inaccurate distance estimation (no localization required). In addition, most of the computations that a node uses to determine redundancy can be reused when new covers are selected. We propose a location-unaware coverage (LUC) algorithm that incorporates our tests. Based on LUC, 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 [13] and evaluate it in the context of a multi-hop network application using the TOSSIM simulator [13]. To the best of our knowledge, our work is the first to address the coverage problem in location-unaware sensor networks.

The rest of the paper is organized as follows. Section 2 briefly surveys related work. Section 3 introduces the system model and our redundancy-check tests. Section 4 presents and analyzes the LUC algorithm and its associated protocols. In Section 5, we study the properties of our protocols via simulation. Section 6 describes our implementation of LUC-I in TinyOS and evaluates its performance. Section 7 gives concluding remarks and directions for future research. Finally, Appendix A provides insights for extending the LUC-I protocol to achieve k -coverage.

2 Related Work

Selecting an optimal (minimum) cover size is a well-known NP-hard problem [14]. Therefore, most proposed protocols apply greedy heuristics to compute covers. These protocols are either centralized (e.g., [15,16,8]) or distributed (e.g., [6,5,9]). They are targeted to either field coverage, where a whole area is to be monitored, or target coverage, where a set of targets (points) in the field are to be monitored (see Figure 1).

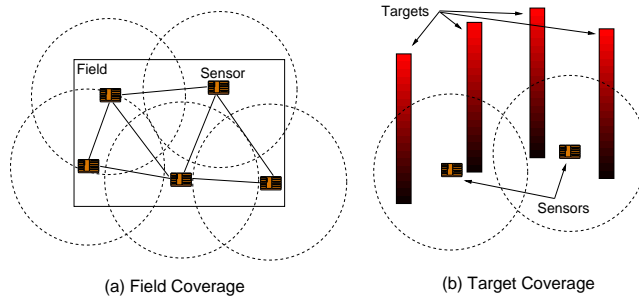


Fig. 1. Types of coverage required by applications.

Feige [14] proposed a centralized approximation algorithm that sequentially adds nodes to the cover based on the remaining uncovered area. Cardei et al. [16] computed a number of set covers to be used successively, where every cover is computed using a similar approach to [14]. They proposed two *centralized* heuristic techniques for target coverage; one uses linear programming and the other is a greedy approach. We later use the greedy approach in [16] as a baseline for comparison with our protocols. Meguerdichian et al. [15] proposed centralized algorithms for achieving both deterministic and statistical coverage. Their approach exploited Voronoi diagrams to determine the best sensor cover to achieve a required fraction of field coverage. Slijepcevic and Potkonjak [17] proposed a centralized heuristic to compute a disjoint maximal set of covers. Their “most-constrained least-constraining” algorithm favors non-redundant nodes.

Several distributed algorithms were recently proposed. Wang et al. proposed CCP [5], which probabilistically provides different degrees of coverage according to the application requirements. A node starts in the sleeping mode for a random interval of time, and then wakes up to check whether it should go to sleep according to an “eligibility rule.” Zhang and Hou proposed OGDC [6], which determines the minimum set of working nodes by reducing their overlap. They provided necessary conditions for 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 [7] 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. [8] and Iyengar et al. [18] proposed algorithms for selecting connected covers. These algorithms do not enforce constraints on the relation between sensing and transmission ranges. Zhou et al. [19] extended the approach in [8] to k -coverage. Huang and Tseng [20] provided the necessary conditions for achieving k -coverage. Carle and Simplot-Ryl [21] proposed a technique for determining redundancy that is based on dominating sets. They assumed equal sensing and communication ranges, and extended a dominating set algorithm to select nodes that cover the field instead of just covering their neighboring nodes.

Dasika et al. [22] proposed a technique for computing active covers based on binary decision diagrams. Cao et al. [23] proposed a sleeping schedule for sensors that guarantees a bounded-delay sensor coverage. Lu et al. [24] proposed a scheduling mechanism to minimize the delay on forwarding paths. Yan et al. [9] proposed a protocol for collaborative sleep and wakeup among neighboring nodes, assuming that the network is synchronized. Ye et al. [25] proposed the PEAS protocol for providing fault-tolerance and prolonging the network lifetime using randomized

sleep/wakeup. PEAS focuses on maintaining network connectivity by periodically awakening nodes to probe the active ones. PECAS [26] improved the network lifetime over PEAS by periodically selecting a new set of active nodes. Kumar et al. [27] provided theoretical bounds on the number of nodes required to achieve k -coverage under different models of node deployment. They also proposed a randomized sleep scheduler (RIS).

All the aforementioned protocols assumed that nodes can accurately estimate their locations and/or the relative directions of their neighbors, which may not be possible in practice, especially in large-scale networks [12]. In this work, we do not make such an assumption. We also study the quality of the selected covers under inaccurate estimation of distance.

3 Determining Node Redundancy

In this section, we first introduce our system model and assumptions, and then describe our proposed tests for determining node redundancy.

3.1 System Model

We consider sensor nodes for which R_t is the transmission range and R_s is the sensing range, defined as the distance from the sensor after which the observed event cannot not be detected. An example is a chemical sensor that can detect a chemical activity within a certain distance. We assume the following:

- (1) Nodes are randomly and redundantly deployed.
- (2) Nodes do not possess GPS, nor do they compute the coordinates of their locations (such computation is sensitive to distance-estimation errors, and it involves high messaging overhead). Furthermore, nodes are equipped with simple omni-directional antennae.
- (3) $R_t \geq 2R_s$. Under this condition, coverage implies connectivity [6]. An example where this assumption holds is the MICA2 mote [1], which has a maximum transmission range of about 1000 feet and a sensing range of about 100 feet [9].
- (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 [28,10]. If transmission ranges are short, these approaches can provide reasonably accurate estimates of one-hop distances. For example, the Cricket sensor [1] uses time-of-flight of packets for accurate ranging based on ultrasound and RF beacons. A node can use several measured signal strengths from each neighbor for calibrating and refining the estimated distances, as proposed in [29]. We assess the effect of distance inaccuracy on our protocols in Section 5.
- (5) Links can be asymmetric due to irregularity in radio ranges [30]. Assuming that every node sends “HELLO” messages for neighbor discovery, a node u does not consider itself a neighbor of another node v until: (1) u has received a HELLO message from v , and (2) v has confirmed the reception of u ’s HELLO message.

3.2 Redundancy Check Tests

Let $N(v, r)$ denote the set of neighbors of node v that lie within a range r . The discovery of such a set relies on an approach that is described later in Section 4. 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 . For example, $wgt(v)$ can be defined as $\frac{e(v)}{e(v) + \sum_{i=1}^{|N(v, R_s)|} e(i)}$, where $e(v)$ is the residual energy of node v and $|N(v, R_s)|$ is the number of v 's neighbors within sensing range R_s . We will use different definitions for $wgt(v)$ in our LUC-I and LUC-P protocols, described in Section 4.

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 $C(v)$. They provide sufficient conditions for redundancy. RTest-H1 and RTest-H2 decide that a node is redundant if certain conditions on node density and distribution are satisfied within a transmission range $R_p \leq R_s$, which we refer to as the ‘‘probing range.’’ They assume that a node’s sensing region is convex but not necessarily circular. Although in rare cases it is possible for the density-based tests to introduce false positives (i.e., erroneously putting a node to sleep), these tests are very effective in determining node redundancy. The effectiveness of the tests when applied individually or in combination is studied in Section 5.

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.

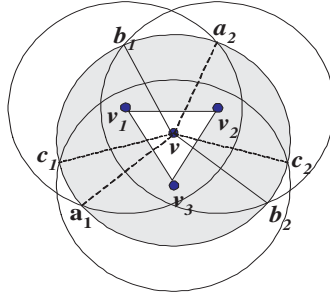


Fig. 2. Demonstrating RTest-D1 where v lies inside the triangle formed by three of its neighbors.

RTest-D1 can be explained in the context of Figure 2. 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 more 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 2, $C(v_1)$, $C(v_2)$ and $C(v_3)$ intersect $C(v)$ in sectors a_1va_2 , b_1vb_2 , and c_1vc_2 , respectively. It is clear that the union of these sectors

spans the entire $C(v)$. The challenge is how to determine these sectors in the absence of location information. We propose a simple approach to solve this problem. A rough estimate of the distance between any two neighbors of v can be determined (see Section 4.1.1), as well as two-hop neighborhood connectivity. 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. Note that knowing the relative directions of neighbors is not necessary for the applicability of this approach.

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

Proof. It is geometrically trivial to show that if the conditions stated 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. Therefore, 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 (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 .*

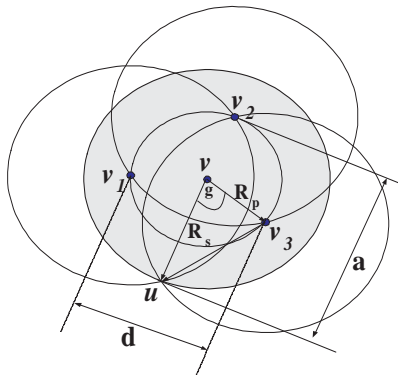


Fig. 3. Determining the probing range (R_p) in RTest-D2.

Proof. Geometrically, three circles completely cover a fourth one if their centers lie *sufficiently* close to the center of the fourth circle. 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 while remaining non-neighbors. Figure 3 shows an organization which results in the smallest R_p . This 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

$$a \stackrel{\text{def}}{=} R_s + R_p. \quad (1)$$

From basic geometry, we have:

$$\begin{aligned} a &= \frac{1}{d} \times \sqrt{(-d + R_s - R_p)(-d - R_s + R_p)(-d + R_s + R_p)(d + R_s + R_p)} \\ &= \frac{1}{d} \times \sqrt{d^2(-d + 2R_s)(d + 2R_s)} \\ &= \sqrt{4R_s^2 - d^2} \end{aligned}$$

where d is shown in Figure 3. To compute d , consider the angle g in Figure 3. Since $uv = uv_3 = R_s$, $g = \cos^{-1}(\frac{R_p}{2R_s})$. In addition, because $\cos^{-1}(x) = \sin^{-1}\sqrt{1-x^2}$, $g = \sin^{-1}\sqrt{1 - \frac{R_p^2}{4R_s^2}}$, 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: $R_p^4 - 5R_s^2 R_p^2 - 2R_s^3 R_p + 3R_s^4 = 0$. 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 RTest-D2 indicates a success ratio of about 3.2%. This test is useful in dense deployments because of three reasons: (1) the computational complexity of this test is the least among all the tests, as discussed in Section 4.4; (2) the test is useful if applied separately, as shown in Section 5; and (3) any node that decides to go to sleep based on this test is guaranteed to be completely covered by active neighbors (this is in contrast to density-based sleeping tests, which provide asymptotic bounds on coverage).

RTest-D1 and RTest-D2 provide sufficient conditions for determining node's redundancy. Next, we provide heuristic tests that exploit conditions on node density to improve the level of 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) $N(v, R_s) \subseteq \cup_{i=1}^M N(v_i, R_s)$.

RTest-H1 says that if v has at least four active neighbors within distance R_s and if every neighbor of v is also a neighbor of one or more of these 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 [31].

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 [31] for one-dimensional networks. We omit it here for brevity³.

Lemma 4 *If the density model provided in Lemma 3 is satisfied, then RTest-H1 correctly determines whether or not v should be put to sleep.*

Proof. Consider the scenario depicted in Figure 4, which conforms to the conditions in Lemma 3, i.e., each cell contains at least one node. Assume a worst-case organization in which all the active neighbors of node v (v_1, v_2, v_3 , and v_4) are placed in one cell (say 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 according to the lemma 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.

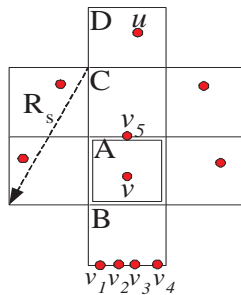


Fig. 4. Correctness of RTest-H1. $\{v, u, v_1, \dots, v_5\}$ identify nodes while $\{A, B, C, D\}$ identify square cells within the network area.

The above argument may only be violated at the cells on the four corners of the network area. This does not significantly affect the coverage of the field since we assume that the number of cells is $\gg 1$. Our simulation experiments (Section 5) indicate that RTest-H1 is very 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 γ is a constant, such that (1) $wgt(v) < wgt(v_i) \forall v_i \in S$, and (2) $N(v, R_s) \subseteq \bigcup_{i=1}^{\gamma} N(v_i, R_s)$.

³ 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 [31].

Unlike previous tests, RTest-H2 aims at turning off “weak” nodes in the network. It checks if node v has at least γ 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 γ neighbors. Node v is then put to sleep if its weight is the smallest among the γ 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 erroneously putting a node to sleep (we use $\gamma = 6$ in Section 5).

4 Location-Unaware Coverage

Based on the proposed redundancy tests, we now present the LUC algorithm for determining whether a node is to be added to V_A or V_S . We then present two distributed protocols that implement LUC in operational scenarios.

4.1 LUC Algorithm

4.1.1 Neighbor Discovery and Distance Estimation

The LUC algorithm at node v proceeds as follows. By exchanging HELLO messages over the transmission range R_t , node v discovers its neighbors and estimates their approximate distances. Such distances are estimated using the time of flight (e.g., Cricket [1]) and/or received signal strength (e.g., RADAR [10]). 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 set of n_d discrete values⁴ and every range of estimated distances is mapped to one of these values (similar to radio maps [28]). 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 5 show that imprecision in distance estimation does not affect the performance of our protocols due to the efficiency of RTest-H1 and RTest-H2.

4.1.2 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

⁴ We refer to n_d as the “discretization level.”

A_{Test} succeeds, the LUC algorithm (Fig. 5) gives another opportunity to a node to go to sleep (by executing R_{Test}-H1) before making the activation final.

A_{Test}: 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 $wgt(u) < wgt(v)$.

4.1.3 Order of Test Execution

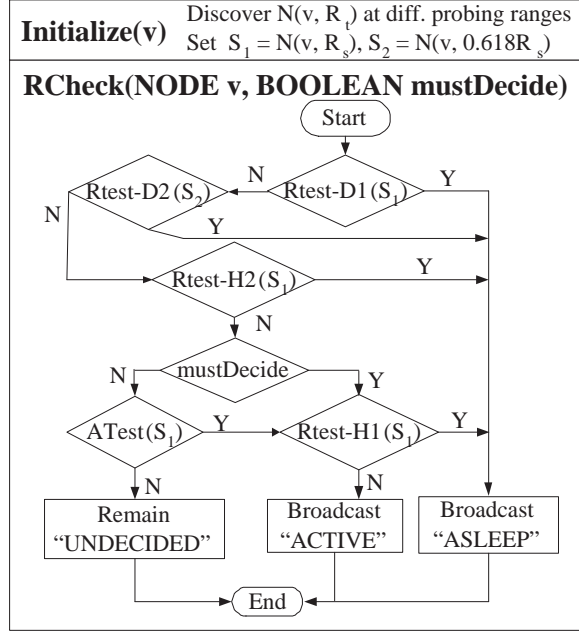


Fig. 5. The LUC algorithm.

The flowchart of the LUC algorithm is provided in Figure 5. The order of the tests is chosen as follows. First, the theoretically proven (geometric) tests are executed before the density-based (heuristic) ones because the geometric tests are guaranteed to not introduce any false positives. Among the two geometric tests, we execute R_{Test}-D1 first because it has a higher success ratio, as depicted in the results shown in Section 5. R_{Test}-H2 is then executed to put a “weak” node to sleep as early as possible. By conservatively setting its parameter to be ≥ 6 , R_{Test}-H2 does not result in any false positives. R_{Test}-H1 is executed only when a node must decide on a new state and all the above tests have failed. This reduces the chances of having false positives. In our experiments, we hardly faced any false-positive case.

The $RCheck(v, mustDecide)$ function in Figure 5 checks whether a node can make a decision (ACTIVE/ASLEEP) or remain in the UNDECIDED state. If $mustDecide=1$, then the node must decide to be active or go to sleep. This occurs when a node has the highest weight among its undecided neighbors. Our protocols (introduced in the next sections) force an undecided node to decide after a certain time elapses in order to avoid waiting indefinitely for neighbors’ decisions. Next, we discuss two protocols that realize the LUC algorithm.

4.2 Iterative LUC

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 decide to put itself to sleep if one or more of the redundancy-check tests are 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:

$$wgt(v) = \frac{e(v)}{e(v) + \sum_{i=1}^{|N(v, R_s)|} e(i)}. \quad (2)$$

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 Figure 5). 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 Figure 6.

LUC-I(v)

1. Initiate timer $T1$ for t_{nd} seconds (neighbor discovery)
2. After $T1$ expires:
 - 2.1. Compute S_1 and S_2 as in Figure 5
 - 2.2. Initiate timer $T2$ for t_{cp} seconds (coverage process)
 - 2.3. *RCheck*(v , 0)
 - 2.4. If ($v.state \neq UNDECIDED$), stop $T2$ and exit
3. While $T2$ has not expired
 - 3.1. If an update is received from u , $u \in S_1$
 - 3.1.1. If ($v.state = UNDECIDED$), *RCheck*(v , 0)
 - 3.1.2. Else *RCheck*(v , 1)
 - 3.1.3. If ($v.state \neq UNDECIDED$), stop $T2$ and exit
4. After $T2$ expires
 - 4.1. If ($v.state = UNDECIDED$), *RCheck*(v , 1)

Fig. 6. Pseudo-code for the LUC-I protocol executed at node v .

4.3 Probabilistic LUC

Under the worst-case distribution of node weights, the convergence of LUC-I will be dependent on the network size (more details in Section 4.4). 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, the active neighbors of v do not directly affect v 's decision to become active. Figure 7 provides pseudo-code for the LUC-P protocol. LUC-P uses a modified version of ATest (ATest-P) that exploits P_{on} . The function $RCheck2()$ is the same as $RCheck()$ except that it uses ATest-P instead of ATest.

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 its activation probability as follows:

$$P_{on} = \frac{e(v) \times P_{start}}{e(v) + \sum_{i=1}^{|N(v, R_s)|} e(i)}, \text{ such that } P_{min} \leq P_{on} \leq 1 \quad (3)$$

where P_{start} is an initial probability that is periodically doubled in order to force the set of active nodes to grow gradually and 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 third timer $T3$ is also initialized to t_{at} (activation test) seconds in order to periodically check v 's eligibility to join V_A . When $T3$ 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 decides 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 $t_{at} = t_{cp}/N_{max}$.

4.4 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 network lifetime 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

ATest-P:****

1. $r = \text{Uniform}(0,1)$
2. If $(r < P_{on})$ return SUCCESS
3. Else return FAIL

LUC-P(v):

1. Initiate timer $T1$ for t_{nd} seconds (neighbor discovery)
2. Set P_{start} to P_{min}
3. After $T1$ expires:
 - 3.1. Compute S_1 and S_2 as in Figure 5
 - 3.2. Initiate timer $T2$ for t_{cp} seconds (coverage process)
 - 3.3. Initiate timer $T3$ for t_{at} seconds (activation test)
 - 3.4. Compute P_{on} according to (3)
 - 3.5. RCheck2(v , 0)
 - 3.6. If $(v.state \neq \text{UNDECIDED})$, stop $T2$ and $T3$, and exit
4. While $T2$ has not expired
 - 4.1. When $T3$ expires
 - 4.1.1. If $(v.state = \text{UNDECIDED})$, RCheck2(v , 0)
 - 4.1.2. If $(v.state \neq \text{UNDECIDED})$, stop $T2$ and $T3$, and exit
 - 4.1.3. ElseIf $(v.state = \text{ACTIVE})$, stop $T3$
 - 4.1.4. $P_{start} = 2P_{start}$
 - 4.1.5. Compute P_{on} according to (3)
 - 4.1.6. Initiate timer $T3$ for t_{at} seconds (activation test)
 - 4.2. If an update is received from u , $u \in S_1$
 - 4.2.1. If $(v.state = \text{UNDECIDED})$, RCheck2(v , 0)
 - 4.2.2. Else RCheck2(v , 1)
 - 4.2.3. If $(v.state \neq \text{UNDECIDED})$, stop $T2$ and $T3$, and exit
5. After $T2$ expires
 - 5.1. If $(v.state = \text{UNDECIDED})$, RCheck2(v , 1)

Fig. 7. Pseudo-code for the LUC-P protocol executed at node v .

sensing range is not completely covered will fail all the redundancy-check tests and will have to be activated. Since this applies to all the nodes in the network, the area covered by V_A will be no less than the area covered by V . Note that the size of the selected cover is governed by the quality of the redundancy-check tests and not by 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. It is also worth noting that our protocols operate correctly irrespective of the network density.

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, since an undecided node has to wait for all its neighbors with higher weights to decide before making a decision, $N_{iter} \sim \mathcal{O}(|V|)$ in the worst-case (a very rare case). To bound the convergence time of LUC-I, we limit the time of the coverage process by a timer (as described in Section 4.2). 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. Although this is an important advantage over LUC-I, LUC-P usually selects a larger set of active nodes than LUC-I, especially at low node densities.

The processing complexity of the LUC algorithm is not significant. For a node v , RTest-D1 and RTest-D2 take $\mathcal{O}(n_a^3)$ in the worst case, where n_a is the number of active neighbors of v . RTest-H1 and RTest-H2 take $\mathcal{O}(n_b)$ time, where n_b is the number of neighbors within range R_s . ATest takes $\mathcal{O}(n_a)$ time for LUC-I and $\mathcal{O}(1)$ time for LUC-P. In a practical application, the LUC algorithm will probably be invoked at a time granularity of hours, which reduces the impact of the processing overhead.

Message overhead. In the LUC protocols, each node sends one message to announce its information (e.g., identifier and remaining energy), one message to announce its 1-hop neighbors and their proximities, and a third message to announce its decision (ACTIVE/ASLEEP), if any. Therefore, $\mathcal{O}(1)$ messages are required per node. This indicates that our LUC protocols are very efficient in terms of message overhead. Note that in multihop networks, this cost can be completely subsumed by heartbeat messages (or routing updates) that are sent to compute paths to different destinations.

5 Simulation Experiments

We focus on the construction of one cover and assume that n nodes are uniformly distributed in a 50×50 meters² field. Unless otherwise specified, $n = 1000$. We use different values of n or R_s in some experiments to vary the density of the network and assess the effect of low densities that do not satisfy the model needed for RTest-H1. 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). To examine the properties of our protocols to the full extent, we assume a simplified scenario where no packets are lost (we use a more realistic MAC layer in the TOSSIM experiments in Section 6). 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 quality, (3) coverage redundancy, and (4) average residual energy in V_A . Coverage quality is defined as the fraction of the field that is covered by V_A . The coverage redundancy in the field (CoRe) depends on the coverage redundancy of every active node j (C_j), which is defined as the minimum number of active sensors that cover any point in the sensing range of j . That is, $C_j = \min\{S_i, \forall i \in P_j\}$, where S_i is the number of sensors in V_A that cover point i and P_j is the set of points (targets) in

the sensing range of node j ($j \in V_A$). CoRe is then defined as $\sum_{j=1}^{n_a} C_j/n_a$. The optimal value for coverage redundancy is 1.

We compare LUC-I and LUC-P with a centralized approach (Greedy-MSC) [16] as well as a distributed approach [7] that assume complete knowledge of node locations. Although Greedy-MSC was proposed for target coverage and not area coverage, we generalize it by discretizing the area into a large number of points. In our first set of experiments that does not include lifetime assessment, we only use the first cover that Greedy-MSC computes. Individual cover selection in Greedy-MSC follows the approximation algorithm in [14], which tries to minimize the size of V_A . Since the set-cover approximation algorithm in [14] is the best one known so far, we conjecture that the first set V_A that is selected by Greedy-MSC will be at most equal to that selected by any other protocol. The distributed approach in [7] uses a geometric test that assumes 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. We compare with SP-SECT because it can still be used if only the relative directions of neighbors are known and not their exact locations.

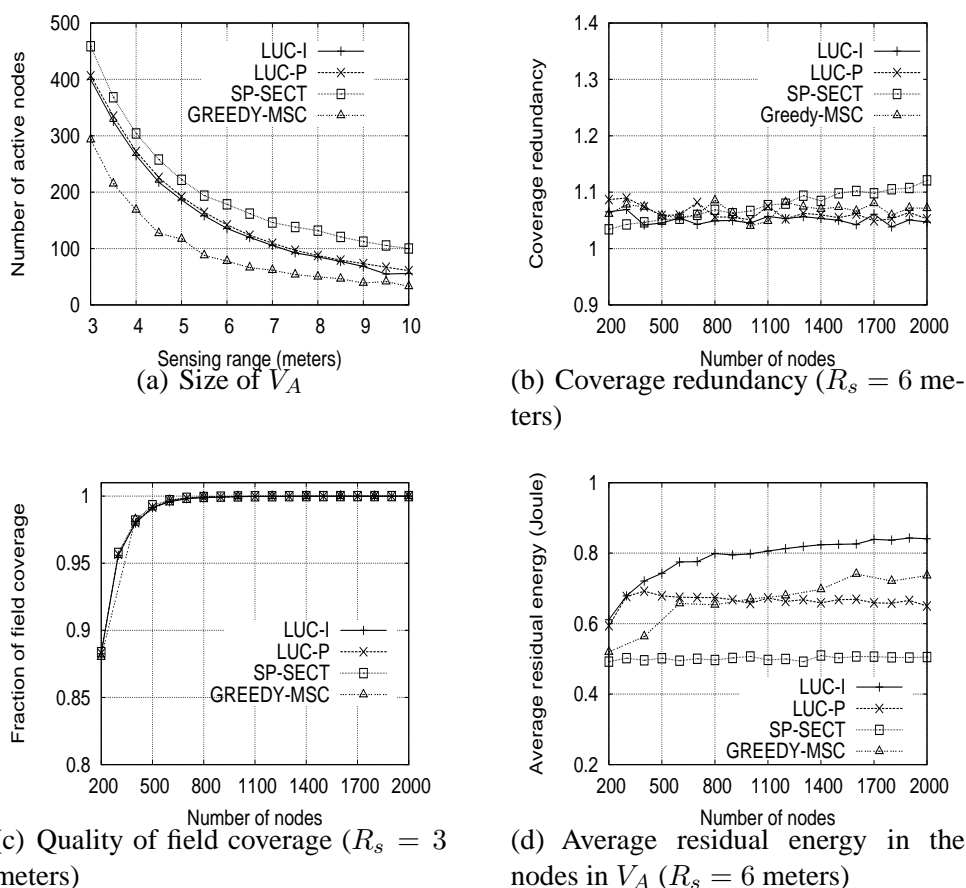


Fig. 8. Performance of LUC-I, LUC-P, SP-SECT, and Greedy-MSC.

Figure 8(a) depicts the size of V_A ($|V_A|$) as a function of the sensing range. As expected, $|V_A|$

decreases as R_s increases for all the studied protocols. $|V_A|$ of LUC-I and LUC-P is about 20-50% larger than that generated by Greedy-MSC 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.

Figure 8(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).

Figure 8(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 (residual energy) 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. Figure 8(d) illustrates that LUC-I and LUC-P select nodes that have higher average residual energy compared to SP-SECT and Greedy-MSC, especially at high node densities.

We also assess the performance of our proposed redundancy-check tests. We set $R_s = 6$ meters and we vary n . Figure 9(a) shows the percentage of nodes that are put to sleep by each test 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 9(b) where each test is applied individually. The figure indicates that the geometric tests are more effective than RTest-H2, while RTest-H1 is still the most effective in exploiting redundancy. Coverage quality is not compromised under any test.

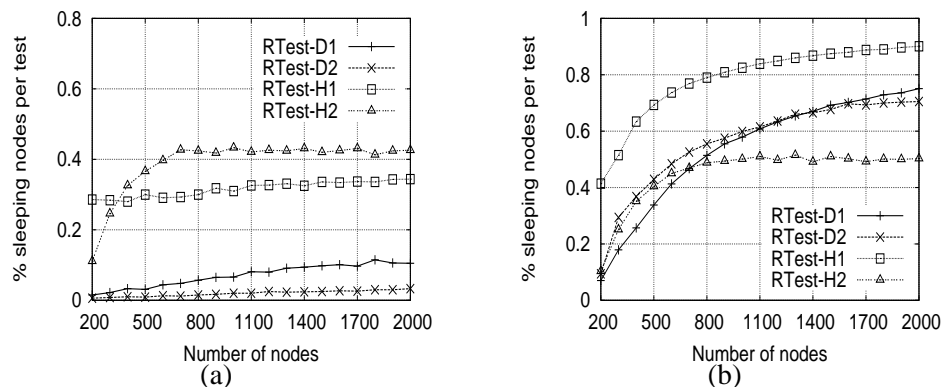


Fig. 9. Percentage of nodes that are put to sleep when the tests are applied: (a) in combination, or (b) individually.

We performed another set of experiments in which a random error is added to the real distance between every pair of nodes. The error can be positive or negative. We applied each test individually to the network and varied the error from 10% to 100% of the real distance. The experiment revealed a very interesting result: the quality of the generated covers was not affected by random errors. However, this came at the expense of having slightly larger cover sizes for most tests, as shown in Figure 10. This indicates that our tests tend to react conservatively when errors are

present. We performed another experiment on Greedy-MS-C where the locations of nodes were displaced by 10%-100% of the field size. In this case, the quality of the generated covers was mostly 100%, but fell to 95% in some cases of large errors. The sizes of the generated covers were also slightly increased with increasing error ratios.

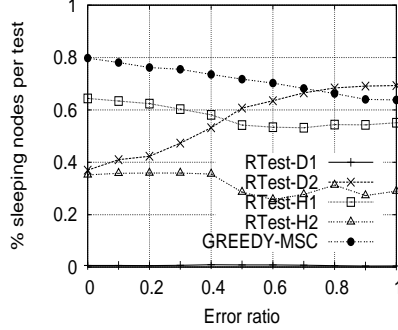


Fig. 10. Percentage of sleeping nodes under different LUC tests and under Greedy-MS-C.

Next, we assess the impact of our protocols on network lifetime. We 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 battery of 1 Joule, and consumes 10^{-4} Watts while active and 10^{-7} Watts while asleep. These values are consistent with those of typical sensor platforms, such as MICA2 [1]. We take $R_s = 6$ meters and update the network cover every 100 seconds. Figure 11(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 network lifetime 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. As expected, Greedy-MS-C shows the longest network lifetime. The difference in lifetime between LUC-P and LUC-I is attributed to the fact that the latter typically selects a smaller V_A .

Finally, we assess the effect of inaccurate estimation of distances on the performance of LUC-I. Such inaccuracy is due to the discretization of distances, as explained in Section 4.1.1. Figure 11(b) shows that the discretization level of the radio range (n_d) has an unnoticeable effect on network lifetime. We have also investigated the effect of random errors on the coverage quality of the active cover selected by LUC-I. Random errors also resulted in an unnoticeable effect on coverage quality. In both cases, this favorable behavior is attributed to the effectiveness of the density-based tests and their independence of distance measurements.

6 Implementation

Our platform is the MICA2 sensor motes [1] running TinyOS [13]. Each mote has a 7.38 MHz Atmel processor, 128 KB program memory, 4 KB RAM, and 512 KB non-volatile storage. The radio is a Chipcon SmartRF CC1000, with 916 MHz frequency, FSK modulation with data rate

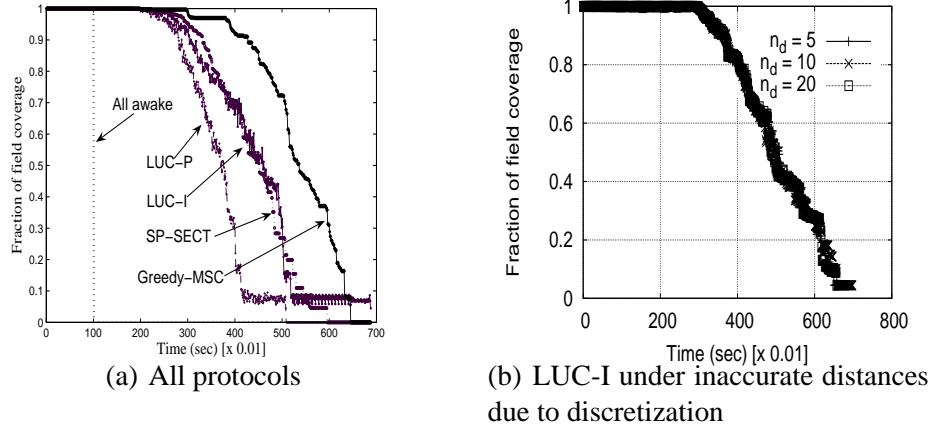


Fig. 11. Coverage quality with $n = 500$ and $R_s = 6$ meters.

of 38.4 kBaud (19.2 Kbps), Manchester encoding, and linear RSSI (received signal strength indicator). Its radio’s output power can be digitally programmed by setting the PA_POW register. In our experiments, we use the documented power consumption values from the Chipcon CC1000 data-sheet and the measurements reported in [32].

The energy consumed due to the operation of a component c_i (e.g., the radio) is given by $e_i = V_i A_i t_i$, 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 radio transmission, the value of the PA_POW register determines the drawn current, and consequently the transmission range. For instance, assuming that the mote consumes 1 mW during transmission (which corresponds to a 16.8-mA current), the bit transmission time is $62.4 \mu\text{sec}$ (as measured in [32]), 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 of the various modes into integer values (points). For example, the radio consumes 3 points/sec in the sleep mode (which corresponds to $3 \mu\text{J}/\text{sec}$) and 24000 points/sec in the receive mode. We use this model to compute the energy consumed during network operation.

We consider a multi-hop network application in which reports from sensors are periodically transmitted to an observer (e.g., a base station). The application is implemented in TinyOS and is referred to as “Surge” [13]. Surge uses the multi-hop routing service in TinyOS [33]. While constructing the routing tree, nodes that can communicate directly with the observer 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. We assume that the observer is interested in all the reports sent by the sensors monitoring the field. Thus, no data aggregation is performed on the path from any node to the observer. We augment the multi-hop routing module with our LUC-I protocol to periodically select active covers. We then evaluate network lifetime for the extended Surge application.

6.1 System Design

We first discuss the design details of the multi-hop routing module in TinyOS [33] 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 code⁵. 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 code evaluation and is not a requirement of our design.

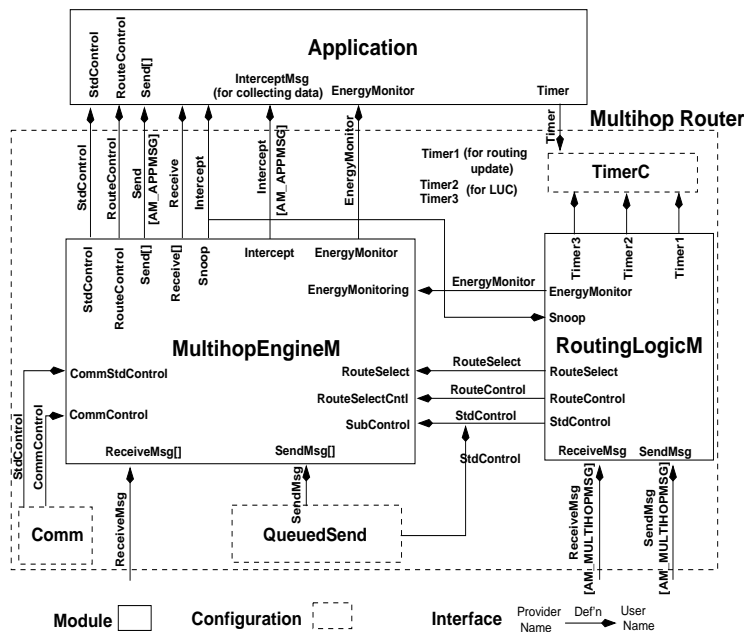


Fig. 12. Multi-hop routing in TinyOS augmented with LUC-I and energy control.

The schematic diagram for the extended multi-hop router is provided in Figure 12 (extension to that in [33]). RoutingLogicM is the module that performs 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. The quality of communication is determined by considering data losses and link symmetry [34].

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 because of the following: (1) the Surge application does not require any data processing, (2) the processing overhead of the LUC-I protocol is negligible because it is infrequently invoked in a real setting, and (3) the energy consumed in periodic sensing is typically much less than that consumed in an active radio. The “Multi-hop Router” uses the energy monitor to inform the application whether the battery is still

⁵ The complete implementation can be obtained by contacting the authors.

operational. The application uses this information to stop data transmission. The Comm interface, illustrated in Figure 12, 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 QueuedSend interface is responsible for buffering packets to be sent in sequence. Details of the Comm and QueuedSend interfaces can be found in [13].

We implemented new timers to support the operation of LUC-I, as was described in Figure 6. 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 6.2. A state diagram of a node executing LUC-I is depicted in Figure 13.

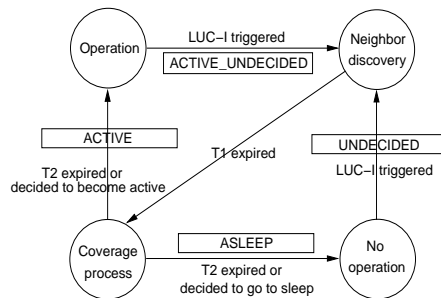


Fig. 13. State diagram for a node executing LUC-I. A circle indicates an action, while a rectangle indicates the new state of the node.

To asynchronously trigger LUC-I in the network, a node v whose timer $T2$ has expired *immediately* broadcasts a routing update packet to its immediate neighbors, indicating that v is starting LUC-I. Upon receiving this message, v 's neighbors who 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 $T2$ timers. Hence, the start of the coverage process diffuses throughout the entire network. Thus, triggering LUC-I only requires a routing update message.

6.2 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 comes with the TinyOS software [13]. 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.

Table 1
Simulation Parameters

| | |
|---------------------------------|------------------------------------|
| Number of nodes (n) | 75 |
| Field size | 50×50 meters ² |
| Observer location | (25, 50) |
| Sensing range (R_s) | 8 - 12.5 m |
| Maximum battery level | 6×10^7 points |
| Energy consumption (receive) | 24000 points/sec |
| Energy consumption (sleep) | 3 points/sec |
| Packet size | 100 bytes |
| Packet transmission time | 49.92 ms |
| Electric current (transmission) | 10.1 - 18.5 mA |
| Data rate | 1 packet/10 sec |
| Routing update | 1 packet/15 sec |
| t_{nd} (neighbor discovery) | 20 sec |
| t_{cp} (coverage process) | 20 sec |
| t_{cu} (cover update) | 300 sec |

The parameters used in our experiments are given in Table 1. 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 (which is about 30,000 Joules according to our computations). All the nodes start their operation randomly within an interval [0,5] seconds from the start of the simulation.

We define network lifetime as the time until the observer is completely disconnected from the sensors, i.e., it does not receive any more reports. This occurs when all level-1 nodes deplete their energy. Note that some nodes may still be alive but can not find parents in the routing tree to forward their reports. Figure 14(a) shows the network lifetime for LUC-I and Surge, where LUC-I refreshes V_A every $t_{cu} = 300$ seconds. LUC-I provides 100-200% improvement in network lifetime 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, facilitating longer periods of operation.

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. Figure 14(b) shows that most 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 often it is put to sleep during network operation.

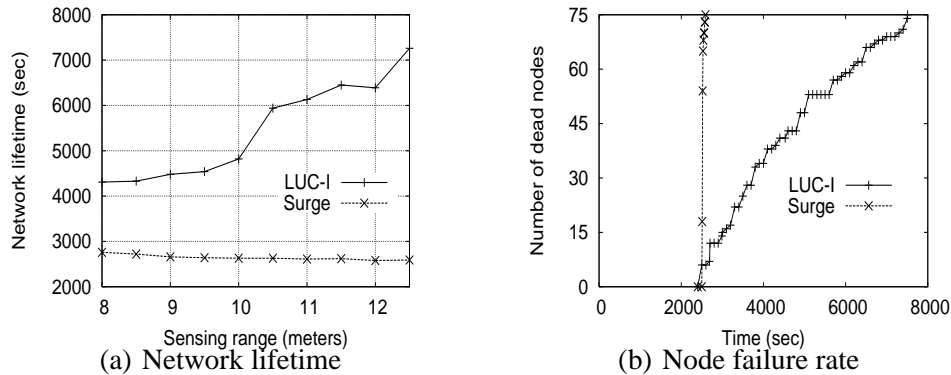


Fig. 14. Performance of LUC-I and Surge ($t_{cu} = 300$ sec).

To measure coverage quality, the observer keeps track of the nodes from which it has received reports within an interval $T_c = 20$ seconds. Figure 15 demonstrates how coverage quality evolves with time for different t_{cu} intervals. The figure includes the upper bound (UB) on the duration after which field coverage falls below 100%. This bound corresponds to the “depth” of field coverage, defined as the minimum number of sensors that cover any point in the field [35]. In our configuration, the depth was equal to two, and thus the UB duration corresponds to the lifetime of two sensors. Two observations can be made from the figure. First, LUC-I keeps the field completely covered even when the number of sleeping nodes is as large as $n/2$. Second, larger t_{cu} values improve network lifetime up to a certain limit. This is because LUC-I is less frequently triggered at larger t_{cu} . Figure 16 indicates that network lifetime drops for $t_{cu} > 500$ seconds, advocating the case for more frequent switching among covers. Choosing the appropriate value for t_{cu} to optimize network lifetime remains an open issue.

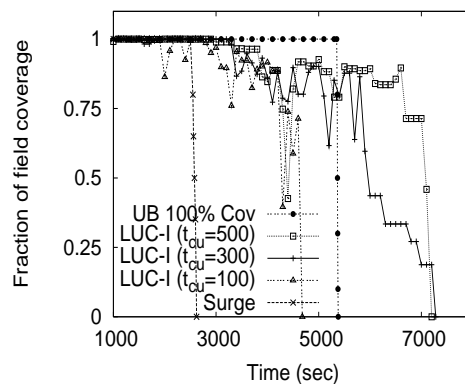


Fig. 15. Coverage quality for $R_s = 12.5$ meters and different t_{cu} values.

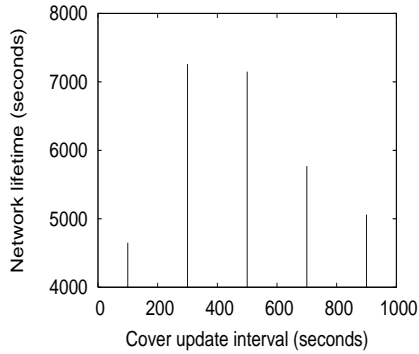


Fig. 16. Network lifetime for different values of t_{cu} ($R_s = 12.5$ meters).

7 Conclusions

In this paper, we proposed a novel distributed approach for exploiting redundant deployment in location-unaware sensor networks. Our redundancy-check tests rely only on exchanged neighborhood information and estimated neighbor distances. We incorporated our tests into a novel location-unaware coverage (LUC) algorithm, 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 network-lifetime extensions achieved by our protocols are comparable to those achieved by a location-aware distributed protocol (SP-SECT) and close to those achieved by another centralized location-aware protocol (Greedy-MSCT). 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 network lifetime.

We plan to examine other redundancy-check tests that exploit specific application requirements such as partial field coverage. We also plan to study how to select an active set that facilitates data aggregation. Efficient tuning of the protocol parameters (especially t_{cu}) is also needed to gain additional lifetime improvements.

References

- [1] Crossbow Technology Inc., <http://www.xbow.com/> (2007).
- [2] G. Anastasi, A. Falchi, A. Passarella, M. Conti, E. Gregori, Performance measurements of motes sensor networks, in: Proc. of MSWiM, 2004, pp. 174–181.
- [3] V. Raghunathan, C. Schurgers, S. Park, M. Srivastava, Energy-aware wireless microsensor networks, IEEE Signal Processing Magazine 19 (2002) 40–50.
- [4] W. Ye, J. Heidenmann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Proc. of the IEEE INFOCOM Conf., New York, 2002, pp. 1567–1576.

- [5] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, C. Gill, Integrated coverage and connectivity configuration for energy conservation in sensor networks, *ACM Transactions on Sensor Networks* 1 (1) (2005) 36–72.
- [6] H. Zhang, J. C. Hou, Maintaining sensing coverage and connectivity in large sensor networks, *Ad Hoc & Sensor Wireless Networks* 1 (2005) 89–124.
- [7] D. Tian, 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*, 2002.
- [8] H. Gupta, S. Das, 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)*, 2003.
- [9] T. Yan, T. He, J. Stankovic, Differentiated surveillance for sensor networks, in: *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (ACM SenSys)*, 2003.
- [10] P. Bahl, V. N. Padmanabhan, RADAR: An in-building RF-based user location tracking system, in: *Proc. of the IEEE INFOCOM Conf.*, 2000.
- [11] N. Bulusu, J. Heidemann, D. Estrin, GPS-less low-cost outdoor localization for very small devices, *IEEE Personal Communications Magazine* 7 (5) (2000) 28–34.
- [12] O. Younis, S. Ramasubramanian, M. Krunz, Location-unaware sensing range assignment in sensor networks, in: *Proceedings of IFIP Networking*, Atlanta, GA, 2007, pp. 120–131.
- [13] TinyOS, <http://www.tinyos.net> (2006).
- [14] U. Feige, A threshold of $\ln n$ for approximating set cover, *Journal of the ACM* 45 (4) (1998) 634–652.
- [15] S. Meguerdichian, F. Koushanfar, M. Potkonjak, M. Srivastava, Coverage problems in wireless ad-hoc sensor networks, in: *Proc. of the IEEE INFOCOM Conf.*, Anchorage, Alaska, 2001, pp. 1380–1387.
- [16] M. Cardei, M. T. Thai, Y. Li, W. Wu, Energy-efficient target coverage in wireless sensor networks, in: *Proc. of the IEEE INFOCOM Conf.*, 2005, pp. 1976–1985.
- [17] S. Slijepsevic, M. Potkonjak, Power efficient organization of wireless sensor networks, in: *Proc. of the IEEE International Conf. on Communications (ICC)*, 2001, pp. 472–476.
- [18] R. Iyengar, K. Kar, 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)*, 2005, pp. 332–342.
- [19] Z. Zhou, S. Das, H. Gupta, Connected K-coverage problem in sensor networks, in: *Proc. of the IEEE ICCCN Conf.*, 2004, pp. 373–378.
- [20] C.-F. Huang, Y.-C. Tseng, The coverage problem in a wireless sensor network, in: *Proc. of the ACM Workshop on Sensor Networks and Applications (ACM WSNA)*, 2003, pp. 115–121.
- [21] J. Carle, D. Simplot-Ryl, Energy-efficient area monitoring for sensor networks, *IEEE Computer* (2) (2004) 40–46.

- [22] S. Dasika, S. Vrudhula, K. Chopra, S. Ramasubramanian, A framework for battery-aware sensor management, in: Proc. of the Conf. on Design, Automation, and Test in Europe (DATE), 2004, pp. 1–6.
- [23] Q. Cao, T. Abdelzaher, T. He, J. Stankovic, Towards optimal sleep scheduling in sensor networks for rare-event detection, in: Proc. of the International Symposium on Information Processing in Sensor Networks (IPSN), 2005, pp. 20–27.
- [24] G. Lu, N. Sandagopan, B. Krishnamachari, A. Goel, Delay efficient sleep scheduling in wireless sensor networks, in: Proc. of the IEEE INFOCOM Conf., 2005, pp. 2470–2481.
- [25] F. Ye, G. Zhong, S. Lu, 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, pp. 28–37.
- [26] C. Gui, P. Mohapatra, Power conservation and quality of surveillance in target tracking sensor networks, in: Proc. of the ACM MobiCom Conf., 2004, pp. 129–143.
- [27] S. Kumar, T. H. Lai, J. Balogh, On k-coverage in a mostly sleeping sensor network, in: Proc. of the ACM MobiCom Conf., 2004, pp. 144–158.
- [28] M. Youssef, A. Agrawala, The Horus WLAN location determination system, in: Proc. of the ACM International Conf. on Mobile Systems, Applications, and Services (ACM MobiSys), 2005, pp. 205–218.
- [29] K. Whitehouse, D. Culler, Calibration as parameter estimation in sensor networks, in: Proc. of the ACM Workshop on Sensor Networks and Applications (ACM WSNA), 2002, pp. 59–67.
- [30] G. Zhou, T. He, S. Krishnamurthy, 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), 2004, pp. 25–38.
- [31] D. M. Blough, 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, pp. 183–192.
- [32] V. Shnayder, M. Hempstead, B.-R. C. G. Werner, 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), 2004, pp. 188–200.
- [33] Multihop routing for TinyOS, http://www.tinyos.net/tinyos-1.x/doc/multihop/multihop_routing.html.
- [34] A. Woo, T. Tong, 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), 2003, pp. 14–27.
- [35] R. Balasubramanian, S. Ramasubramanian, A. Efrat, Coverage time characteristics in sensor networks, Tech. rep., University of Arizona, <http://www.ece.arizona.edu/~sriniftinyos/papers/Sensor-Coverage-TechReport.pdf> (Nov. 2006).
- [36] O. Younis, M. Krunz, S. Ramasubramanian, On maximizing coverage time in location-unaware wireless sensor networks, Tech. rep., University of Arizona (July 2006).
URL <http://www.ece.arizona.edu/~younis/papers/luc-tr.pdf>

Appendix A: Extending LUC-I to Achieve k -coverage

So far, we have focused on covering each point in a field with at least one node, i.e., 1-coverage. For reliability against unexpected failures and fidelity of the collected reports, the application might require that every point in the field be covered by at least k sensors, where $k > 1$. Below, we extend the LUC-I protocol to provide location-unaware k -coverage (referred to as “LUCK-I”). We assume the same system model provided in Section 3.1.

Achieving minimal k -coverage is not possible if the nodes are location-unaware. This is because a node v can not determine the *number* of nodes that cover every point in its sensing range unless it knows the locations of its neighbors. To cope with this problem, we construct k node-disjoint covers, which is a *sufficient*, but not necessary, condition for k -coverage. This means that the network might be k -covered although k node-disjoint covers could not be computed. In the case where node density does not support constructing k node-disjoint covers, our protocol (described below) gracefully degrades and selects k covers having nodes in common. The degree of redundancy achieved in this case is the best that the current node density can support.

One alternative for distributed construction of k node-disjoint covers is to *sequentially* trigger the construction of covers, i.e., one at a time. This alternative has two drawbacks: (1) it is roughly k times slower than constructing one cover, and (2) it requires node synchronization. Such synchronization is essential for starting the selection of nodes in each cover independently. To handle node asynchrony and to avoid the need for sequential cover construction, we interleave the selection of the k covers and do not allow a node to participate in more than one active cover. Our approach has some similarity with node-disjoint path selection techniques. In these techniques, a node that is selected on one path from the source to the destination does not participate in any other path discovery for the same source/destination pairs. In the LUCK-I protocol, every node applies the redundancy tests specified in Section 3 for each cover independently.

At a node v , LUCK-I operates as follows. Initially, v has a list of k potential covers to which it might belong. When a neighbor node u announces that it will become active, it also specifies which cover node u belongs to (say i , $i \leq k$). On receiving this message, v executes LUC and considers only active neighbors that belong to cover i . If v passes the redundancy-check tests, it considers itself ASLEEP in cover i . However, it does not actually put itself to sleep unless it is considered ASLEEP in all the k covers. If the redundancy-check tests indicate that v has to become active in cover i , it notifies its neighbors and remains active. The neighbors of v in this case consider v asleep for all covers but i . More details on LUCK-I and its pseudo-code can be found in [36].

LUCK-I has the same message overhead as LUC-I. The worst-case computational complexity of LUCK-I is also similar to that of LUC-I since a node v still needs to check its redundancy when receiving messages from any active neighbor.

We briefly study the properties of the LUCK-I protocol via simulation. We assume that $k = 4$ and $R_s = 3$ meters. We use the same network dimensions and topologies used in Section 5.

Figure 17(a) shows the number of active nodes selected by LUCK-I for each cover under different node densities. It is clear that the number of nodes in cover i is larger than that in cover j , $\forall i < j$. This is because nodes compete to fill the covers with smaller indexes first. This has a direct effect on the quality of the generated covers. Figure 17(b) shows that coverage quality improves for all the selected covers as the node density increases.

We also study the convergence time of LUCK-I, defined as the time (iterations) required until all the nodes in the network have decided to become ACTIVE or ASLEEP. Figure 17(c) shows that LUCK-I with $k = 4$ does not require significantly more time to terminate than the case with $k = 1$, especially at shorter sensing ranges and smaller densities. Note, however, that LUCK-I would have been k times slower than LUC-I if it had to construct one cover at a time before proceeding to the next cover.

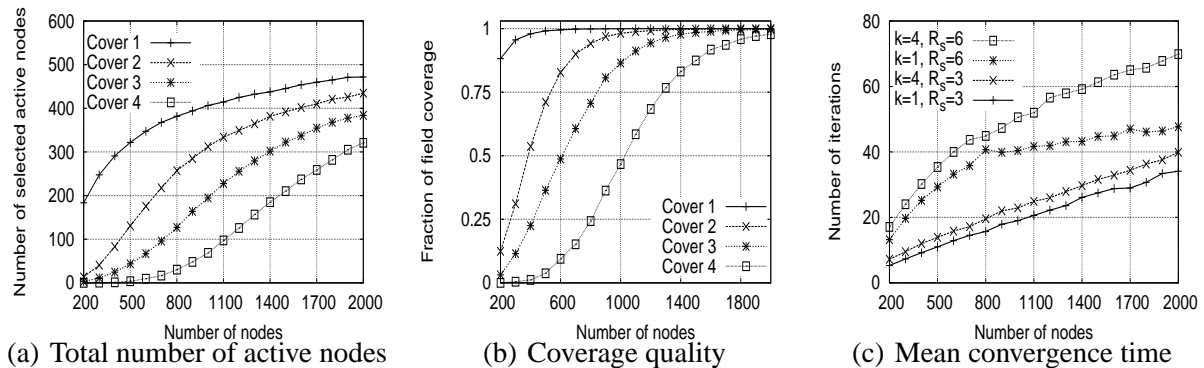


Fig. 17. Performance of LUCK-I for $k = 4$.