Solving the Wake-up Scattering Problem Optimally

Luigi Palopoli*, Roberto Passerone*, Amy L. Murphy‡, Gian Pietro Picco*, and Alessandro Giusti†
* Dip. di Ingegneria e Scienza dell'Informazione (DISI), University of Trento, Italy
{luigi.palopoli,roberto.passerone,gianpietro.picco}@unitn.it
‡ Fondazione Bruno Kessler—IRST, Trento, Italy, murphy@fbk.eu
† Dept. of Electronics and Information, Politecnico di Milano, Italy, giusti@elet.polimi.it

***Abstract*—Wireless sensor networks (WSNs) often rely on duty-cycling, alternating periods of low-power stand-by with others where sensing, computation, and communication are performed. While duty-cycling brings substantial energy savings, it may complicate the WSN design. The effectiveness of a node to perform its task (e.g., sensing events occurring in an area) is affected by its wake-up schedule. Random schedules lead to deployments that are either ineffective (e.g., insufficient sensing coverage) or inefficient (e.g., areas covered by multiple nodes simultaneously awake).**

In this paper, we focus on the problem of *scattering* the nodes' wake-up times *optimally*, to improve the coverage of a given area. In previous work [1], we presented a decentralized wake-up scattering protocol that performs significantly better than random schedules. Here, we complete the work in [1] by providing a (centralized) *optimal* solution that constitutes a theoretical upper bound for decentralized protocols. Simulation results show that our original decentralized algorithm comes within 4% and 11% of the optimum. Moreover, the modeling framework we present, based on integer programming techniques, is novel and general enough to encompass alternative formulations of the problem.

I. Introduction

The operation of a wireless sensor network (WSN) node is typically characterized by (long) periods of inactivity, spent in a low-power stand-by state, interleaved with (short) periods where the expected sensing, computation, and communication duties are carried out. This duty-cycling is frequently used to reduce energy consumption to a minimum, and therefore maximize the lifetime of the network at large.

Nevertheless, the performance of the WSN application critically depends on the quality of the duty-cycling schedule, which ensures that the right nodes are active at the right time. A random schedule may lead to highly inefficient deployments. For instance, consider Figure 1, where three nodes cover, with overlaps, a certain area. In this case, ensuring that nodes 2 and 3 are *not* active at the same time is beneficial, since they share a large portion of the area. A random schedule may miss considerable opportunities for optimization. In general, since a point in the target area may be monitored by multiple nodes, it is possible to switch off certain nodes without affecting the coverage, as long as the remaining active nodes cover the area of interest.

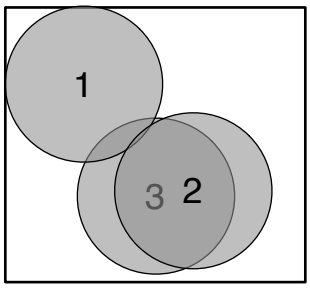


Figure 1. A topology where wake-up scattering matters.

In our recent work [1] we presented a decentralized protocol that leverages this observation by *scattering* the nodes' wake-up times, therefore taking advantage of the overlap among the nodes' sensing range. We demonstrated the effectiveness and practical relevance of our *wake-up scattering* protocol in several common WSN scenarios, including the coverage problem above. However, although we showed that our protocol yields significant improvements over random schedules, an open question remained about whether further improvements can be obtained.

This paper provides an answer to this question, by giving a means to compute the *optimal* schedule of wake-up times that maximizes the area covered by a set of sensor nodes. Our technique is inherently centralized, as it assumes global topology knowledge. Therefore, its most obvious application is in case of fixed and constrained sensor networks, for which the required information is available *before* deployment. In the more common case of a random deployment, the proposed technique provides a theoretical upper bound against which to evaluate distributed solutions. In the intermediate scenario of a fixed sensor network subject to dynamic changes (e.g., due to node failures), we envisage a combined utilization of the two techniques, the optimal off-line technique being used for the initial configuration of the system (upon its deployment) and the distributed algorithm being used online to adapt to dynamic changes. A complete description of the problem and of the assumptions is offered in Section II, followed by the description of our approach. Notably, this includes a proof that, under reasonable assumptions (essentially constant and periodically repeated awake intervals) the resulting optimization problem can be radically simplified, therefore enabling an efficient computation of the solution. In Section III we compare quantitatively the optimal solution we derive, using integer linear programming techniques, against the decentralized solution we put forth in [1]. The evaluation confirms that our original decentralized approach is very efficient, as it comes within 4% and 11% of the optimum.

By leveraging the mathematical formulation we are able to gain insights into the relationships among coverage, lifetime, and node density. Moreover, the modeling framework we use

to derive our optimal solution is actually general enough to represent problems other than coverage, similarly affected by duty-cycling. Both topics are discussed in Section IV. Finally, after a concise survey of related approaches in Section V, we end the paper with our concluding remarks in Section VI.

II. System Model and Solution Algorithm

Our approach assumes[1] a *target area* A to be monitored using a set of *sensor nodes* $\mathcal{N}$. For each node $n \in \mathcal{N}$, we know its *position* in the target area A, represented as a pair of coordinates (x_n, y_n), as well as its *sensing range*, i.e., the area that is directly monitored by n. We assume that the sensitivity of a node in its sensing range is constant, and that the boundaries of the sensing range are sharp. In other words, the complete topology of the problem is known in advance. An example was already shown in the introduction in Figure 1.

The system operates according to a periodic schedule. The period, called the *epoch*, is denoted by E. In our problem, the lifetime of the system is fixed and determined by the *awake* or *activation interval* of a node, i.e., the interval during which a node is awake in the epoch. We assume that the awake interval is the same across all nodes, and that each node is woken up only once per epoch. Alternative formulations with different awake intervals or multiple activations per epoch are possible at the expense of increased computational complexity.

The objective of our optimization is to maximize the coverage by scheduling the wake-up time of the nodes. At any time t, the set of nodes that are awake at that time defines a covered area $S(t)$, which is equal to the area of the union of the sensing areas of all awake nodes.[2] Our objective is therefore to maximize the integral of $S(t)$ over the epoch.

Our method to solve the optimization problem goes through two steps, which are described in detail in Section II-A and II-B, respectively.

1) We consider the problem of determining a *finite* number of regions in the target area that, without loss of accuracy, can be used to model the topology of the system and the area overlaps. We refer to this as the *spatial partitioning* problem.
2) We express the scheduling problem by setting up an integer linear program, with coverage constraints for each node and for each region found in the previous step.

The optimization problem is then solved using standard linear programming and branch and bound techniques, and yields as a result the optimal schedule for the given objective function and constraints. In the remainder of the section we discuss our solution in detail.

A. Spatial partitioning

The optimization problem can be set up as a linear program by enforcing a certain level of coverage on the target area A, through the use of appropriate constraints (see Section II-B). This approach is feasible if A is discretized into a finite number of regions, to avoid generating constraints for each of the infinitely many points in A. In our work we use the concept of *field* [2], i.e., regions of A that are invariant relative to node coverage. In other words, any two points within one such region are covered by exactly the same nodes. In this case, it is sufficient to consider only one point per region, or, equivalently, consider the region as a whole.

Several algorithms have been proposed for field computation. Slijepcevic and Potkonjak approximate the computation by a regular sampling of the area [2], while Tian and Georganas rely on geometric approximations to compute the required intersections [3]. Huang and Tseng propose an exact and efficient algorithm for deciding if an area is covered by at least k sensors by considering only the perimeter of the sensing area [4], and by reasoning on the angles of intersection. By doing this, the computational complexity is reduced to $O(nd \log n)$, where n is the number of nodes and d is the average number of nodes that overlap a given node. We also reason on the perimeter of the sensing area and develop an algorithm with the same computational complexity that computes the set of nodes that cover each field, and its area. However, we consider the case of rectangular rather than the more traditional circular sensing areas, and use coordinates instead of angles. The rationale behind our use of rectangles is that in both cases the model is only a rough approximation of the real sensing area, and rectangles greatly simplify the computation of fields. In addition, our algorithm works without change when the sensing area is represented by *unions of rectangles*, including the case of unconnected sensing areas. This feature can be used to approximate any arbitrary shape, albeit with increased computational complexity. We also note that rectangles have already been used in the context of localization with results which are sometimes superior to the use of circles [5], [6].

In the following sections, we will use $r : A \to 2^{\mathcal{N}}$ to denote the function that for each point $p \in A$ returns the set of nodes that cover p. The regions (or fields), denoted by ρ, are given by the partition of the points of A induced by the equivalence relation $\sim$ defined by:

$$p_1 \sim p_2 \iff p_1 \in \rho \wedge p_2 \in \rho \iff r(p_1) = r(p_2).$$

We denote the set of regions by the symbol $\mathcal{R}$. For each region $\rho \in \mathcal{R}$, the function $w : \mathcal{R} \to \mathbb{R}_+$ returns the corresponding area. Given that there is a one-to-one mapping, we use $r(\rho)$ to denote the set of nodes that cover a region ρ. Note also that each region need not necessarily be connected, even when the sensing areas of the nodes are connected and convex. Thus, this kind of spatial partitioning cannot be obtained using a simple regular discretization.

The partitioning algorithm works by scanning the target area horizontally, left to right, stopping at every vertical boundary of a node's range, as shown in Figure 2. At every stop, the algorithm performs a vertical scan, from bottom to top, that computes the regions found at that horizontal position, and

[1] For simplicity, we describe the problem in a two-dimensional space: the extension to a three-dimensional space is straightforward.

[2] Note that this is different from taking the sum of the areas of the awake nodes, since overlaps would be counted multiple times.

Algorithm 1 Spatial partitioning algorithm.

1: *h_scan* = list of vertical boundaries of the nodes ordered by their horizontal position
2: *v_scan* = $\emptyset$
3: $\mathcal{R} = \emptyset$
4: **for** i = *h_scan*.begin() to *h_scan*.end() **do**
5: **if** i is left boundary of node n **then**
6: Insert the horizontal boundary of n in *v_scan* ordered by their vertical position
7: **else** // i is the right boundary of node n
8: Erase the horizontal boundary of n from *v_scan*
9: **end if**
10: *h_extent* = horizontal extent to next stop
11: *node_set* = $\emptyset$
12: **for** j = *v_scan*.begin() to *v_scan*.end() **do**
13: **if** j is bottom boundary of node n **then**
14: Insert n in *node_set*
15: **else** // j is the top boundary of node n
16: Remove n from *node_set*
17: **end if**
18: *v_extent* = vertical extent to next stop
19: *area* = *h_extent* · *v_extent*
20: $\mathcal{R} = \mathcal{R} \cup \{$*node_set*$\}$
21: *node_set.area* = *node_set.area* + *area*
22: **end for**
23: **end for**

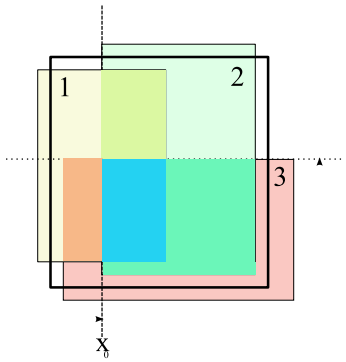


Figure 2. Rectangular topology with scanning sequence. The thick rectangle represents the target area A.

updates their area. This is sufficient, since the computation of the optimal schedule (Section II-B) does not require the shape of the regions, but only their area and corresponding set of covering nodes. Each region can conveniently be identified by its set of covering nodes, as discussed above. For example, in Figure 2, the vertical scan at x_0 finds a sequence of regions corresponding to the sets of nodes:

$$\{3\}, \{2,3\}, \{1,2,3\}, \{1,2\}, \{2\}. \tag{1}$$

Areas are accumulated incrementally, and computed up to the following stop in the scanning sequence.

The procedure is shown in detail in Algorithm 1. To perform the horizontal scan, we first build a sequence *h_scan* of all the vertical boundaries of the nodes, ordered by their horizontal position (line 1). We then loop through this sequence (line 4) and stop at every element. To perform the vertical scan we also build a sequence, denoted *v_scan*, by inserting the horizontal

Region	Nodes	Area	Region	Nodes	Area
ρ_0	$\emptyset$	56	ρ_4	{ 1, 3 }	96
ρ_1	{ 1 }	144	ρ_5	{ 2 }	244
ρ_2	{ 1, 2 }	140	ρ_6	{ 2, 3 }	272
ρ_3	{ 1, 2, 3 }	160	ρ_7	{ 3 }	112

Table I
REGIONS FOR THE TOPOLOGY OF FIGURE 2.

boundaries of the nodes that intersect the scan at the current horizontal position in the order of their vertical position. For efficiency, this sequence is constructed incrementally: it is initialized to empty (line 2) and at every step of the horizontal scan we insert or remove the horizontal boundaries of the node at that position, according to whether we are entering (from the left) or exiting (to the right) the sensing area of the node (lines 6 and 8). In the example of Figure 2, the vertical scan at x_0 would already have the ordered entries $(3_{bot}, 1_{bot}, 3_{top}, 1_{top})$. At x_0, since we are entering node 2, we add 2_{bot} and 2_{top} in the correct order, to obtain

$$(3_{bot}, 2_{bot}, 1_{bot}, 3_{top}, 1_{top}, 2_{top}). \tag{2}$$

The vertical scan starts with the loop at line 12. During this phase, we keep track of the set of nodes *node_set* for which the current point is in range. This set is updated at every step of the scan according to whether we are entering the node's range from the bottom (line 14) or exiting it from the top (line 16). For example, scanning sequence (2) we obtain the regions of sequence (1). After computing the area of the *node_set*, we add it to the set $\mathcal{R}$ (line 20), which was initialized to empty at the beginning of the procedure (line 3). If the region was already present in the set, we do not add a new element but simply update its area (line 21). Since regions can span several horizontal positions, we use for the set $\mathcal{R}$ an ordered data structure that provides efficient insertion and retrieval.

The result of applying Algorithm 1 to the topology of Figure 2 is shown in Table I, where each region ρ is associated with its covering nodes and its area. Note that the area covered by the nodes outside the target area A is ignored.

B. Computation of the optimal schedule

As discussed, we consider periodic schedules of a fixed duration E, called the *epoch*. For each period, we assume that every node wakes up exactly once, and operates for a defined interval I of time, the *awake interval*. The optimization problem consists of finding the time within the epoch at which each node should wake up (the *wake-up time*) in order to maximize the total coverage. Intuitively, this can be achieved by scheduling the operation of overlapping nodes at different times, while nodes that do not overlap could be scheduled concurrently. To enable distribution, our previous work approximates this idea by scattering the awake times of neighboring nodes, which are more likely to overlap [1].

Finding the exact optimum for this problem is complex, and a naive formulation easily results in solution algorithms that are impractical even for a small number of nodes. Nonetheless,

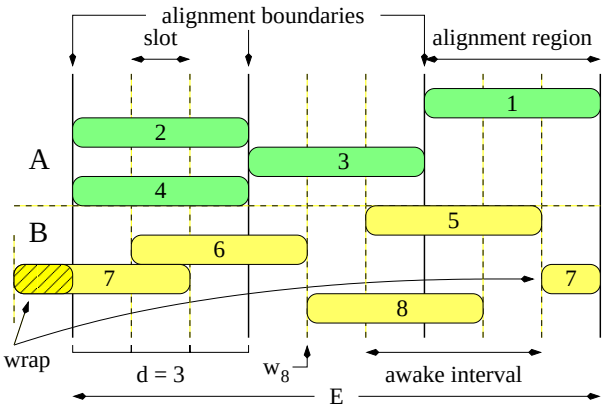


Figure 3. Key concepts in node alignment.

it is possible to simplify the problem by taking advantage of the following result.

Theorem 1: Let the duration E of the epoch be an integer multiple of the awake interval I. Then there exists a schedule such that every node wakes up at some integer multiple of I (within the epoch) which maximizes the average coverage.

Thus, instead of considering arbitrary wake-up times distributed on the real line, we can discretize the problem by dividing the period E into $L = E/I$ equal *slots* of length I. Theorem 1 tells us that the discretization does not affect our ability to find an optimal coverage, and guarantees that there exists an optimal schedule where the wake-up times are aligned at the slot boundaries, i.e., every node is awake in exactly one slot. Thus, to solve our problem, we do not need to look at schedules in which nodes overlap only partially in time, thereby pruning a large portion of the solution space.

The proof of Theorem 1 is by induction, and involves showing that given an arbitrary schedule one can always derive a new schedule with equal or better coverage, and in which more nodes have been aligned to slot boundaries. Since an optimal schedule exists, the procedure can be used to show that there is an equivalent *aligned* schedule that achieves the same coverage.

To prove the theorem (an extended version of the proof can be found in [7]), we assume that the epoch E is initially discretized into s slots (where $s \gg L$), and that the awake interval spans d slots (i.e., $I = d \cdot \frac{E}{s}$). Since, by hypothesis, E is an integer multiple of I, we have $s = L \cdot d$. The continuous case can be obtained when s tends to infinity.

Figure 3 depicts our notation. Slots are ordered and identified by their position $0 \leq k \leq s-1$. We use a subscript k to identify quantities related to slot k. We assume that operations on slot indices are done modulo s, so that index s is equal to 0, index $s+1$ is equal to 1, index -1 is equal to $s-1$, and so on. Node n has a wake-up time $0 \leq w_n \leq s-1$. We say that a node is *aligned* if it is scheduled at an integer multiple of the awake interval:

Definition 1 (Aligned nodes): A node n is *aligned* if and only if $w_n \bmod d = 0$.

We say that a *schedule* is aligned if *all nodes* in the schedule are aligned. The line that identifies the slots at which aligned nodes can be scheduled is called an *alignment boundary*. The slots between two consecutive alignment boundaries is called an *alignment region*. If the schedule is aligned, then all nodes are scheduled at the alignment boundaries, that is, they pairwise either completely overlap in time, or they do not overlap at all. Given an arbitrary schedule, we can partition the set $\mathcal{N}$ of nodes into those that are already aligned and those which are not:

$$\mathcal{A} = \{n \in \mathcal{N} \mid n \text{ is aligned } \},$$
$$\mathcal{B} = \{n \in \mathcal{N} \mid n \text{ is not aligned } \} = \mathcal{N} - \mathcal{A}$$

For every slot k, we denote by $\mathcal{A}_k$ the set of aligned nodes that are awake at slot k, and by $\mathcal{B}_k$ the set of non-aligned nodes that are awake at slot k. Because all non-aligned nodes span across an alignment boundary, the set of non-aligned nodes awake at the slots across an alignment boundary are the same.

Lemma 1: Let k be a slot at the beginning of an alignment boundary, i.e., $k \bmod d = 0$. Then, $\mathcal{B}_{k-1} = \mathcal{B}_k$.

The proof of Lemma 1, which can be done by contradiction, is left to the reader. Similarly, the aligned nodes in slots that belong to the same alignment region are, of course, the same.

Lemma 2: Let k and k' be two slots in the same alignment region, i.e., such that $dm \leq k, k' \leq d(m+1) - 1$, for some integer m. Then $\mathcal{A}_k = \mathcal{A}_{k'}$.

We now compute the gain (positive or negative) in the covered area that is obtained by shifting the schedule of *all* the non-aligned nodes together by one slot to the left or to the right. To do so, we must compute the coverage before and after the shift. Let $a_k \subseteq \mathbb{R}^2$ be the region covered by the nodes in $\mathcal{A}_k$, $b_k \subseteq \mathbb{R}^2$ the region covered by the nodes in $\mathcal{B}_k$, and let $A : \mathbb{R}^2 \to \mathbb{R}$ be the function that to a subset of $\mathbb{R}^2$ gives the corresponding area. The coverage for each slot can be computed as the area $A(a_k)$ covered by a_k, plus the area $A(b_k)$ covered by b_k, minus the area $A(a_k \cap b_k)$ covered by both. The total coverage S is simply the sum over all slots:

$$S = \sum_{k=0}^{s-1} (A(a_k) + A(b_k) - A(a_k \cap b_k)) \tag{3}$$

The gain due to a shift of the non-aligned nodes to the right can be computed by shifting the b_k and leaving the a_k fixed. The change in coverage will depend on the area overlaps before and after the shift (if there are no overlaps, then all schedules are equivalent). More precisely, because we shift all non-aligned nodes together, the *gain* of slot k for a right shift is given by the area overlap between the non-aligned and the aligned nodes before the shift, minus the area overlap of the same non-aligned nodes with the aligned nodes in the new slot, after the shift. Formally,

$$g_k^+ = A(a_k \cap b_k) - A(a_{k+1} \cap b_k). \tag{4}$$

The total gain can therefore be expressed as:

$$G^+ = \sum_{k=0}^{s-1} g_k^+. \tag{5}$$

Slots which are not near an alignment boundary give no gain, as shown next.

Lemma 3: Let k be a slot that is not to the immediate left of an alignment boundary. Then, $g_k^+ = 0$.

Proof: Since k is not next to an alignment region, slot $k+1$ is in the same alignment region. Thus, according to Lemma 2, slots k and $k+1$ have the same aligned nodes, i.e., $\mathcal{A}_k = \mathcal{A}_{k+1}$. Therefore, also $a_k = a_{k+1}$. By Equation (4), $g_k^+ = 0$. ∎

By the previous lemma, the significant terms in Equation (5) are only those that correspond to slots to the immediate left of an alignment boundary. Recalling that $L = s/d$, we can rewrite the formula as:

$$G^+ = \sum_{i=0}^{L-1} g_{di-1}^+. \tag{6}$$

Similarly, we can define the gain for a left shift of the non-aligned nodes. We have:

$$g_k^- = A(a_k \cap b_k) - A(a_{k-1} \cap b_k). \tag{7}$$

By arguments similar to those above, one shows that $g_k^- = 0$ for all slots which are not to the immediate right of an alignment boundary. Hence,

$$G^- = \sum_{i=0}^{L-1} g_{di}^-. \tag{8}$$

We next show that shifting right or shifting left give gains that are equal, but of opposite sign. We first show it for gains of adjecent slots.

Lemma 4 (Local gains are opposite): Let $di = k$ be a slot marking the beginning of an alignment region. Then,

$$g_{di}^- = -g_{di-1}^+.$$

Proof: The proof consists of the following series of equalities:

$$g_{di}^- = g_k^- = A(a_k \cap b_k) - A(a_{k-1} \cap b_k)$$

Since $k = di$, $k \bmod d = 0$; therefore, by Lemma 1, $\mathcal{B}_{k-1} = \mathcal{B}_k$. Hence, $b_{k-1} = b_k$. Therefore,

$$\begin{aligned} &= A(a_k \cap b_{k-1}) - A(a_{k-1} \cap b_{k-1}) \\ &= -g_{k-1}^+ = -g_{di-1}^+ \end{aligned}$$

∎

Corollary 1 (Gains are opposite):

$$G^+ = -G^-.$$

Proof: The proof follows easily by matching corresponding terms in the expressions of G^+ and G^-. ∎

Our last result shows that by shifting the non-aligned nodes in one direction we obtain a change in the covered area which is equal, but of opposite sign, to the change obtained by shifting the non-aligned nodes in the opposite direction. The next result shows that if shifting does not result in any new node getting aligned, then an additional shift in the same direction will give the same gain as the previous shift in that direction.

Lemma 5 (Shift again): Let W_1 and W_2 be two schedules obtained by shifting the non-aligned nodes $\mathcal{B}$ to the right. Assume $\mathcal{A}_1 = \mathcal{A}_2$. Then, $G_1^+ = -G_2^- = G_2^+$.

Proof: Since $\mathcal{A}_1 = \mathcal{A}_2$, the node partition does not change after the shift. Thus, shifting back the non-aligned nodes must bring the coverage to its previous value. Hence, $G_1^+ = -G_2^-$. By Corollary 1, $-G_2^- = G_2^+$. ∎

The result is similar if we shift to the left, and the partition does not change. We can now prove the main theorem.

Proof of Theorem 1: Let a non-aligned schedule be given. It is easy to construct an aligned schedule which has equal or better coverage as follows. Compute G^+ and G^-, and shift the non-aligned nodes in the direction of positive gain. If $G^+ = G^- = 0$, then shift to the right. By Lemma 5, we can keep shifting in the same direction, with the same positive (or zero) gain, until at least one non-aligned node becomes aligned. At this point, repartition the nodes into aligned and non-aligned, and proceed recursively by shifting the remaining non-aligned node in the (now possibly different) direction of positive (or zero) gain. After all nodes are aligned, having moved only in the direction of positive or zero gain, the new schedule has equal or better coverage than the starting one.

Next, let W be an optimal schedule. If W is aligned then the theorem is proved. Otherwise, construct a new schedule W' using the procedure described above. By construction, W' is also optimal, and is aligned. Therefore, there exists an optimal aligned schedule. ∎

While Theorem 1 guarantees the existence of an aligned optimal schedule, it does not tell us how to find an optimal schedule in the first place, and is, therefore, not constructive. However, it guarantees that the coarsest possible discretization of the problem is also optimal, which allows us to set up a significantly simpler optimization problem than what it would be required to solve the general problem. In practice, we set up a boolean linear program (BLP), which can be solved by standard commercial and open source tools.

We compute the coverage starting from the partition of the target area in regions by introducing a set of binary *coverage* variables $C_{\rho,k}$ that say if region ρ is covered in slot k:

$$C_{\rho,k} = \begin{cases} 1 & \text{region } \rho \text{ is covered during slot } k \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Since regions are by definition disjoint, using the coverage variables $C_{\rho,k}$ and the A area function, the covered area can be computed as

$$S = \sum_{k=0}^{L-1} \sum_{\rho \in \mathcal{R}} C_{k,\rho} \cdot A(\rho). \tag{10}$$

Our objective is to maximize the total area S.

The value of $C_{\rho,k}$ depends on the schedule. To model the schedule, we introduce a set of binary *scheduling* variables

$x_{n,k}$ that say if node n is active in slot k. Formally,

$$x_{n,k} = \begin{cases} 1 & \text{if } n \text{ is awake in slot } k \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

We express the relation between the coverage and the scheduling variables as linear optimization constraints. This relation can be easily understood by observing that $C_{\rho,k} = 1$ if and only if at least one of the nodes that cover ρ is active in slot k, i.e., when $x_{n,k} = 1$ for some $n \in r(\rho)$. Likewise, $C_{\rho,k} = 0$ whenever all of the nodes that cover ρ are inactive in slot k, i.e., when $x_{n,k} = 0$ for all $n \in r(\rho)$. This translates into the following two sets of constraints:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1], C_{\rho,k} \leq \sum_{n \in r(\rho)} x_{n,k} \quad (12)$$

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1], \forall n \in r(\rho), C_{\rho,k} \geq x_{n,k} \quad (13)$$

Here, constraint (12) forces the condition $C_{\rho,k} = 0$ when no node covers a region in a slot, while constraint (13) ensures the condition $C_{\rho,k} = 1$ when at least one node covers a region.

The optimization problem can be further simplified by observing that the coverage variables $C_{\rho,k}$ are constrained by Equations (12) and (13) to never take values strictly between 0 and 1, regardless of whether they are defined as integer or real variables. Thus, we can relax $C_{\rho,k}$ to a continuous variable, which is more efficiently handled by optimization algorithms, by adding the constraints:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1], \qquad 0 \leq C_{\rho,k} \leq 1 \quad (14)$$

More complex situations can also be handled, however at the expense of increased computational complexity. We describe some of these in Section IV, and compare them against related work in Section V.

III. Optimal vs. Distributed: An Evaluation through Simulation

As stated previously, our motivation for this work was to find an optimal solution for the wake-up scattering problem, and determine how close our original distributed solution [1] comes to it. This section presents an evaluation by using covered area as the primary performance metric. Specifically, if $S(k)$ is the area covered during slot k, the covered area during an epoch is $\frac{1}{L}\sum_{k=0}^{L-1} S(k)$. Intuitively, this calculates the largest coverage possible given a network topology, ignoring regions of the field that are not covered by any sensors. Our evaluation uses GLPK [8], an open source linear programming kit[3], to compute the optimal schedule, and the simulator described in [1] to compute the distributed ones.

While the full details of the distributed solution are available in [1], for completeness we briefly describe its key functionality here. The distributed solution starts by assigning each node a random wake-up time. Through a simple sequence of message exchanges, each node learns which of its neighboring nodes wakes up immediately before and after it in the epoch. It also learns when these nodes wake up relative to its own wake-up time, then selects its new wake-up time to be approximately in the middle of these two points in time. This process repeats at all nodes until no significant changes in the wake-up time are made at a single step. Although the resulting schedule is not aligned on slot boundaries, by Theorem 1 it can be slotted without loss of coverage.

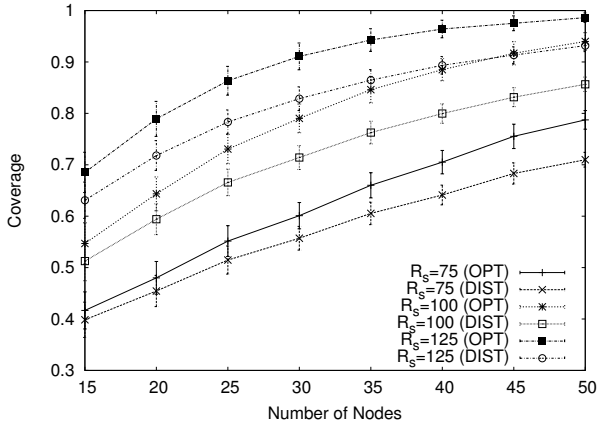
The majority of our experiments were performed in a 500×500 area. The number of nodes varied from 15 to 50, allowing us to consider increasing node densities. To achieve statistical significance, each point in our plots represents an average over 100 distinct topologies. Additionally, because the distributed scattering protocol does not change the wake-up order among nodes and is therefore affected by the initial random wake-up configuration, the results average 10 different initializations for each topology. This is sufficient as the variation with different initializations is small. We considered square sensing areas, the sensing range R_S being half of the edge length. For the distributed solution, we used a circular communication range $R_C = R_S\sqrt{2}$, creating a sensing square exactly inscribed in the communication circle. The resulting node densities for $R_S = 100$ range from 2.7 to 9.3. To verify that the results carry over to larger topologies, we also ran experiments with similar densities obtained with up to 200 nodes in a 1000×1000 area, as reported at the end of this section.

We studied two main configurations, corresponding to the two main dimensions of the problem. In the first setting, we fixed the number of slots per epoch and therefore the duty cycle, and varied the sensing range. In the second, we reversed roles, fixing the sensing range and varying the number of slots. In each case, a node is awake for only one slot. The results in Figures 4(a) and 5(a) clearly show that as either the sensing range or duty cycle increase, both solutions cover larger areas. The standard deviation is a few percent points and in general the distributed solution comes within 4% and 11% of the optimal.

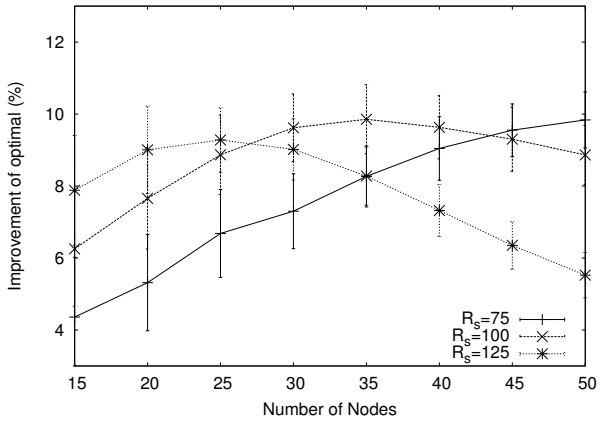
Interesting trends emerge when studying the percentage improvement of the optimal solution w.r.t. the distributed one, defined as $\frac{optimal - distributed}{optimal}$ and reported in Figures 4(b) and 5(b). In both plots, each line, representing either a different sensing range or number of slots, shows a general trend where the percent improvement increases to a certain point, then begins to decline. This indicates that at low and high densities the optimal and distributed solutions behave similarly, while at intermediate densities the optimal solution performs better.

Indeed, at low densities both solutions easily find schedules such that nodes with overlapping sensing areas have minimal overlap in time. As density increases, overlaps in space become inevitable and the system must scatter the wake up times to eliminate them as much as possible. In principle, nodes close to one another should be more scattered. Unfortunately, the distributed solution does not consider the distance between nodes and does not change the sequence of node wake-up times. Therefore, with more dense scenarios it is more likely

[3]Although GLPK documentation declares a performance gap from 10 to 100 times w.r.t. a known commercial tool in solving large LP problems, our computation times were acceptable.

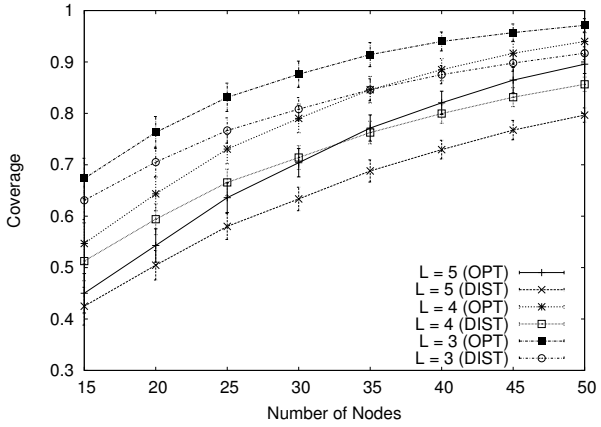


(a) Absolute coverage.

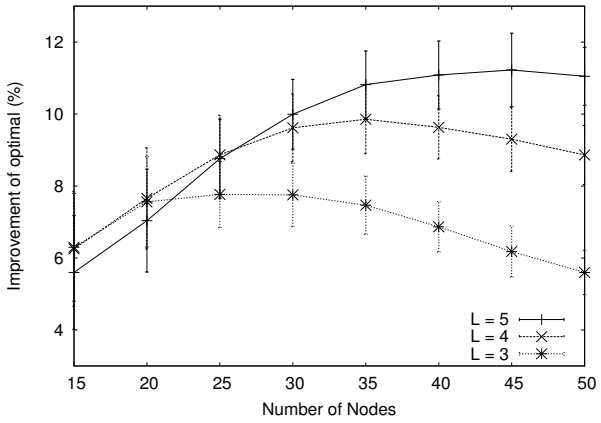


(b) Percent improvement of optimal.

Figure 4. Fixed number of slots ($L = 4$), variable sensing range.



(a) Absolute coverage.



(b) Percent improvement of optimal.

Figure 5. Fixed sensing range ($R_S = 100$), variable number of slots.

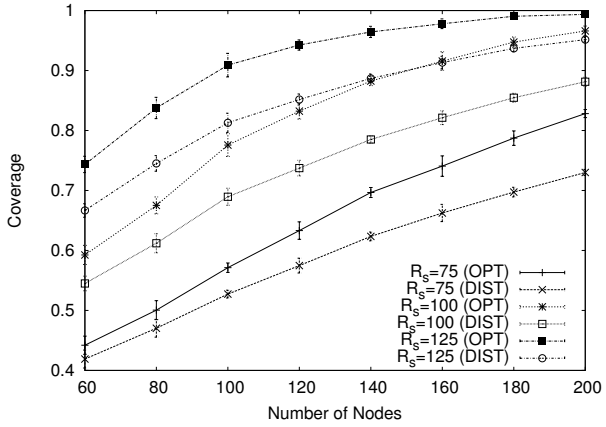
that physically close nodes are next to one another in the wake-up sequence, and therefore less scattered. In contrast, the optimal solution considers neither connectivity constraints nor initial wake-up times, and therefore does not suffer from the same problem.

The tail of the curves, where the two solutions again perform similarly, is due to the saturation of the covered area in either time or space. With a fixed number of slots, saturation is due to the increased number of nodes covering a single point in space. Instead, with a fixed sensing range, saturation arises from the higher number of nodes awake at a given time. In both cases, the distributed solution again approaches the optimal.

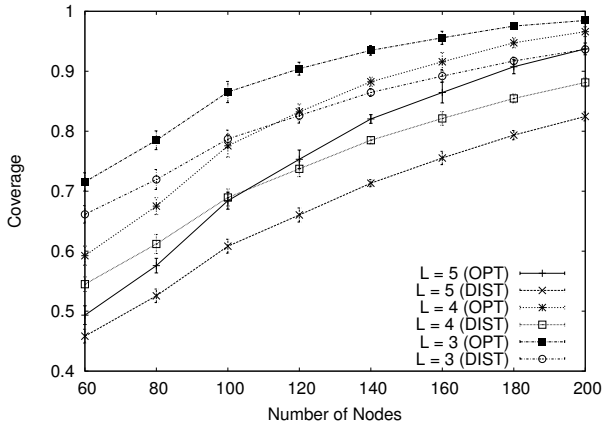
Because saturation depends on the sensing range and number of slots, the peak of each line appears at different network densities. In Figure 4(b), larger sensing ranges saturate faster, while in Figure 5(b), in scenarios with fewer slots, when nodes are awake for longer, saturation occurs earlier. Interestingly, in Figure 4(b) the sensing range $R_S = 75$ does not reach saturation, because the combination of neighbor density and sensing range is still far from saturation; however it would appear at higher densities.

Finally, to demonstrate that our results scale to larger topologies, we considered a 1000×1000 area containing from 60 to 200 nodes. This yields the same density as in the previous experiments. Because the computation of the optimal schedules in this case took two days to complete, we ran only ten topologies for each point. The trends in Figure 6 clearly follow those in Figures 4(a) and 5(a), suggesting that similar trends hold for large topologies as well. This is expected as both solutions work by considering overlapping sensing ranges, which by nature are localized. Put another way, a 1000×1000 topology behaves similarly to a scenario where four of the smaller 500×500 topologies are juxtaposed to form a 1000×1000 square area. The main difference is that, in the larger topologies, the *edges* of the four quadrants of the area are properly scattered, while in a solution which considers each quadrant independently, the edge nodes would not be properly scattered.

In summary, the evaluation shows that our original distributed wake-up scattering protocol [1], albeit very simple, is remarkably effective as it yields schedules whose performance is within 4% and 11% of the optimal ones computed with the technique presented in this paper.



(a) Fixed number of slots ($L = 4$).



(b) Fixed sensing range ($R_S = 100$).

Figure 6. Experiments with 1000×1000 area.

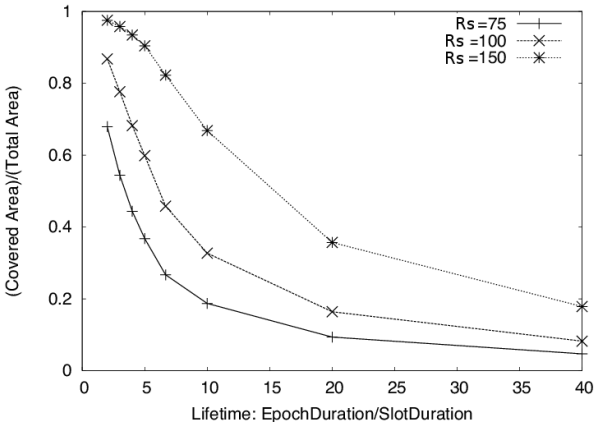


Figure 7. Lifetime vs. Covered area with a fixed number of slots ($L = 4$).

IV. Using and Extending the Model

Although our original motivation was to compare the optimal and distributed solutions, this section explores two additional research contributions, one a practical application of the optimal results and the other an extension to a new class of problems.

As shown in Section III, the distributed solution closely approximates the optimal solution, coming within 4% and 11% in all cases. Therefore, the optimal solution can be naturally applied as a pre-deployment tool to select system parameters that meet the application needs. For example, consider a system with given lifetime and coverage constraints. Plots generated by our offline optimization tool, such as the one in Figure 7 showing lifetime as the ratio of the epoch length and the slot length, clearly show the trade-off between lifetime and coverage and thus the system behavior with various settings. Although the distributed solution will not achieve equivalent results, such an evaluation still provides the developer with valuable insight into system behavior prior to deployment. Alternate topologies may also be considered, either fixing the locations of some nodes or changing the number of nodes. Because these simulations are relatively straightforward to perform, extensive pre-deployment evaluation can be performed at little cost.

While this provides an immediate, practical use for the optimal solution, we also explored how the formulation proposed in Section II-B can be applied to a wider class of problems other than coverage. For example, we can handle the inverse problem: given a required coverage, find a schedule that *maximizes the lifetime* of the network. Since we have defined lifetime with respect to first node failure, this can be achieved by minimizing the maximum energy consumed by any node in a single epoch. We can also relax several of the assumption made previously, yielding a more flexible solution space and perhaps additional efficiency. For example, we can allow nodes to activate/deactivate multiple times, effectively being awake for multiple non-contiguous slots each epoch. To ensure that the power consumption model remains accurate in this scenario, we can extend the power model such that a cost is incurred for each activation. Additionally, we can explore adding constraints to manage latency in delivery of information to a sink node. This is naturally expressed as a set of additional constraints on the awake times between parent and children nodes in a distribution tree.

To give an example, we consider the problem of maximizing lifetime given a certain level A_0 of required coverage. We also extend the model to consider multiple activations of nodes within the epoch, since in this case Theorem 1 no longer applies. For power consumption, we adopt a model in which a node incurs an energy cost proportional to its active interval, with an additional cost for each activation. A conceptual formulation of the problem is the following:

$$\min \quad \max_{n \in \mathcal{N}} \alpha I(n) + \beta W_a(n), \tag{15}$$

subject to

$$< \text{the assigned area is covered} > \tag{16}$$

The cost function to be minimized is the largest of the energy consumed by each node n. Here, α is the energy consumed

by a node during an active slot (i.e., for sensing, computation and communication); $I(n)$ is the total number of active slots for n; β is the energy spent for activation; and $W_a(n)$ is the number of times n is activated/deactivated during the epoch. The vector notation in the cost function can be reformulated in scalar form by rephrasing the minimization in terms of an additional decision variable μ constrained as follows:

$$\min \mu, \text{ subject to} \tag{17}$$

$$\forall n \in \mathcal{N},\ \mu \geq \alpha I(n) + \beta W_a(n) \tag{18}$$

Function $I(n)$ is simply given by $I(n) = \sum_{k=0}^{L-1} x_{n,k}$. Function W_a can be computed by accumulating the difference between adjacent scheduling variables:

$$W_a(n) = \sum_{k=0}^{L-1} |x_{n,k} - x_{n,(k+1) \bmod L}|, \tag{19}$$

This expression can be linearized by introducing new variables $s_{n,k}$, and using the constraints

$$\begin{aligned} s_{n,k} &\geq x_{n,k} - x_{n,((k+1) \bmod L)} \\ s_{n,k} &\geq x_{n,((k+1) \bmod L)} - x_{n,k} \end{aligned} \tag{20}$$

The minimization process ensures that $s_{n,k}$ will equal the absolute value, even if it is not declared as an integer variable. Finally, constraint (16) can be easily expressed by requiring that the area of the covered regions, for each slot, be greater than the given value A_0:

$$\forall k \in [0, L-1], \sum_{\rho \in \mathcal{R}} A(\rho) C_{\rho,k} \geq A_0$$

where the $C_{\rho,k}$ variables were introduced in Equation (9) and computed using constraints (12) and (13).

A full evaluation of such alternate scenarios is part of our future work.

V. Related Work

Several studies have approached the problem of maximizing the lifetime of a network by running sensor nodes on a low duty cycle, while maintaining a high level of performance. Regarding the coverage problem, we can classify existing approaches as centralized techniques, which make use of global information about the deployment, and distributed techniques, which are typically limited by network connectivity, but can more easily adapt to changes in the network.

Of the centralized techniques, Slijepcevic and Potkonjak are among the first to address the problem of maintaining *full coverage* while minimizing power consumption through active/sleep schedule [2]. The problem is solved by partitioning the nodes into disjoint sets, activated one at a time, where each set of nodes completely covers the monitored area. The solution is obtained through a centralized heuristic of quadratic complexity in the number of nodes, which is shown to significantly improve over a simulated annealing approach. However, no exact solution is derived in the paper. Cardei et al. improve on this technique, and propose a method where nodes are divided in sets which are not necessarily disjoint, thus achieving a higher degree of optimization [9]. The objective of the work is to maximize the network lifetime by scheduling the activity of the nodes while maintaining *full coverage* of a *finite set of points*. The optimization problem is formulated as an integer linear program, which, due to its complexity, is only solved through the use of heuristics. Similar formulations are proposed by Liu et al. [10], and by Alfieri et al. [11]. In the first case, the authors use a two step procedure to compute the maximal lifetime and to include the cost of communication between nodes. In the second approach, two coordinated optimization problems are solved to determine the subset partitions and their duration. The authors also propose a greedy distributed heuristic, which is however shown to reach solutions that may perform as much as 40% of the optimal.

Our centralized technique is also based on an mixed integer (boolean) linear program. Unlike the previous work, however, we do not aim at maintaining full coverage of an area or of a set of points, but rather at establishing a periodic schedule that guarantees the *largest total coverage* of an area over a scheduling period given a specified lifetime of the system. By doing so, we are also able to compute the optimal trade-off between coverage and lifetime. This makes it difficult to give an experimental comparison of our approach with previous work, since the objectives of the optimization differ.

In our approach, each node is activated exactly once per period, and the epoch is constrained to be an integer multiple of the awake interval. Cardei et al. show that this choice, compared to allowing nodes to wake-up multiple times in an epoch, is suboptimal in the case of *full coverage* of a set of points [9]. While this applies also to our problem, we have found that for random topologies, the possible increase in the *largest total coverage* is either zero or limited to fractions of a percent. On the other hand, the restriction allows us to greatly improve the performance of the *exact* optimization program, and handle a much larger number of nodes (see Theorem 1).

The distributed techniques typically use information from neighboring nodes to locally compute a schedule. In the simplest form, nodes wake up regularly and check whether a neighboring node is awake. In that case, they go back to sleep to conserve energy. Ye et al. complement this simple scheme with an adaptive sleeping scheme which dynamically determines the duration of the sleep time to optimally maintain a certain degree of coverage [12]. More elaborate schemes take coverage information from neighboring nodes into account in order to preserve the *full coverage* of an area. Tian and Georganas propose a technique that proceeds in rounds [3]. At the beginning of a round, each node computes the fraction of its sensed area which is also covered by neighboring nodes, by exchanging position information. When a node determines that it is fully covered by other nodes,it goes to sleep for the rest of the round. Hsin and Liu improve on Tian and Georganas and propose a coordinated sleep technique where the duration of the sleep state is computed as a function of the residual energy of the node and of the neighboring nodes, instead of being fixed by the length of the round [13]. By doing so, they obtain a more graceful degradation of the overall coverage as nodes

fail. Similarly to our distributed solution, Cao et al. use a set up phase where nodes schedule their operation in order to overlap least in time with nearby nodes covering the same area [14]. This is done by exchanging exact position information, and by incrementally adjusting the schedule until the procedure converges. More recently, Cărbunar et al. developed an efficient distributed algorithm for preserving full coverage and for computing the coverage boundary [15]. One of their main results is the reduction of the problem of finding redundant nodes to that of checking coverage of certain Voronoi diagrams associated to the topology. This result is used to efficiently compute the coverage due to the neighbors, while the diagrams are updated dynamically as nodes fail. Their experiments show a significant improvement over the method proposed by Tian and Georganas. In addition, they study the effect of their technique on connectivity and on the network load.

In contrast to previous work, our distributed technique is extremely simple, and does not require exact position information nor time synchronization. This is important when considering the overall energy budget of a node. Nonetheless, the result of scattering the wake-up time of the nodes gives a degree of coverage that compares favorably to the exact optimum. This is possible since our objective is to maximize the largest total coverage, instead of maintaining full coverage, which, again, makes the experimental comparison with previous work difficult. On the other hand, we can use our exact formulation to determine the best trade-off between coverage and lifetime, and therefore correctly size the deployment in terms of number of nodes.

A related problem is how to achieve the best sensor *placement* to cover an area [16]. Solution techniques include integer programming [17], greedy heuristics [18], [19], [20] and virtual force methods [21]. These works complement our temporal distribution and can be applied in parallel to achieve further coverage improvements.

VI. Conclusions and Future Work

In this paper we presented a way to compute *optimal* schedules for scattering node wake-up times in a WSN. Notably, the efficient computation of the optimal solution is enabled by a mathematical result that allows us to drastically simplify the optimization problem. The ability to compute such optimal schedules allowed us to compare against our original decentralized algorithm [1]. The evaluation shows that the latter, albeit very simple, is remarkably efficient in generating schedules whose performance is within 4% and 11% of the optimum. Moreover, we also showed that optimal schedules can be used to evaluate the trade-off between coverage and lifetime and, given the small difference between the optimal and distributed schedules, guide engineering decisions in practical deployments. Finally, we discussed how the modeling framework presented here is amenable to extension and adaptation towards similar problems. We are currently investigating such extensions, beginning with the inverse problem of determining the lifetime of the system given a desired coverage.

References

[1] A. Giusti, A. Murphy, and G. Picco, "Decentralized Scattering of Wake-up Times in Wireless Sensor Networks," in *Proc. of the 4^{th} European Conf. on Wireless Sensor Networks (EWSN)*, ser. LNCS 4373. Springer, January 2007, pp. 245–260.

[2] S. Slijepcevic and M. Potkonjak, "Power efficient organization of wireless sensor networks," in *Proc. of the IEEE Int. Conf. on Communications (ICC)*, June 2001.

[3] D. Tian and N. Georganas, "A coverage-preserving node scheduling scheme for large wireless sensor networks," in *First ACM Int. Wkshp. on Wireless Sensor networks and Applications (WSNA)*, 2002.

[4] C. Huang and Y. Tseng, "The coverage problem in a wireless sensor network," in *Proc. of the 2^{nd} ACM Int. Conf. on Wireless Sensor Networks and Applications (WSNA03)*, September 2003.

[5] K. Langendoen and N. Reijers, "Distributed localization in wireless sensor networks: a quantitative comparison," *Computer Networks*, vol. 43, no. 4, pp. 499–518, 2003.

[6] A. Savvides, H. Park, and M. Srivastava, "The bits and flops of the N-hop multilateration primitive for node localization problems," in *First ACM Int. Wkshp. on Wireless Sensor Networks and Application (WSNA)*, Atlanta, GA, 2002, pp. 112–121.

[7] R. Passerone and L. Palopoli, "Aligned schedules are optimal," University of Trento, Technical Report, 2008, http://www.disi.unitn.it/~roby/Passerone_alignment_theorem.pdf.

[8] F. S. Foundation, "The Gnu Linear Programming Kit," http://www.gnu.org/software/glpk/.

[9] M. Cardei, M. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in *Proc. of INFOCOM*, 2005.

[10] H. Liu, X. Jia, P. Wan, C. Yi, S. Makki, and N. Pissinou, "Maximizing lifetime of sensor surveillance systems," *IEEE/ACM Trans. on Networking*, vol. 15, no. 2, pp. 334–345, 2007.

[11] A. Alfieri, A. Bianco, P. Brandimarte, and C. F. Chiasserini, "Maximizing system lifetime in wireless sensor networks," *European Journal of Operational Research*, vol. 127, no. 1, pp. 390–402, August 2007.

[12] F. Ye, G. Zhong, J. Cheng, S. Lu, and L. Zhang, "PEAS: A robust energy conserving protocol foo long-lived sensor networks," in *3^{rd} Int. Conf. on Distributed Computing Systems (ICDCS'03)*, May 2003.

[13] C. Hsin and M. Liu, "Network coverage using low duty-cycled sensors: Random & coordinated sleep algorithms," in *Proc. of the 3^{th} Int. Symp. on Information Processing in Sensor Networks (IPSN)*, 2004.

[14] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *Proc. of the 4^{th} Int. Symp. on Information Processing in Sensor Networks (IPSN)*, April 2005.

[15] B. C. arbunar, A. Grama, J. Vitek, and O. C. arbunar, "Redundancy and coverage detection in sensor networks," *ACM Transaction on Sensor Networks*, vol. 2, no. 1, pp. 94–128, February 2006.

[16] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *Proc. of 20^{th} IEEE INFOCOM*, Anchorage, Alaska, USA, April 2001, pp. 1380–1387.

[17] K. Chakrabarty, S. S. Iyengar, H. Qi, and E. Cho, "Grid coverage for surveillance and target location in distributed sensor networks," *IEEE Trans. on Computers*, vol. 51, no. 12, pp. 1448–1453, 2002.

[18] N. Bulusu, D. Estrin, and J. Heidemann, "Adaptive beacon placement," in *Proc. of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS)*, Phoenix, AZ, USA, April 2001, pp. 489–498.

[19] A. Howard, M. Mataric, and G. Sukhatme, "An incremental self-deployment algorithm for mobile sensor networks," *Auton. Robots*, vol. 13, no. 2, pp. 113–126, 2002.

[20] ——, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem." in *Proc. of the 7^{th} Int. Symp. on Distributed Autonomous Robotic Systems (DARS)*, June 2002.

[21] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization in distributed sensor networks," *IEEE Trans. on Embedded Computing Systems*, vol. 3, no. 1, pp. 61–91, 2004.