Solving the Wake-up Scattering Problem Optimally

Luigi Palopoli[1], Roberto Passerone[1], Amy L. Murphy[2], Gian Pietro Picco[1], and Alessandro Giusti[3]

[1] Dip. di Ingegneria e Scienza dell'Informazione (DISI), University of Trento, Italy
{luigi.palopoli,roberto.passerone,gianpietro.picco}@unitn.it
[2] Fondazione Bruno Kessler—IRST, Trento, Italy
murphy@fbk.eu
[3] Dept. of Electronics and Information, Politecnico di Milano, Italy
giusti@elet.polimi.it

Abstract. In their EWSN'07 paper [1], Giusti et al. proposed a decentralized *wake-up scattering* algorithm for temporally spreading the intervals in which the nodes of a wireless sensor network (WSN) are active, and showed that the resulting schedules significantly improve over the commonly-used random ones, e.g., by providing greater area coverage at less energy costs. However, an open question remained about whether further improvements are possible. Here, we complete the work in [1] by providing a (centralized) *optimal* solution that constitutes a theoretical upper bound for wake-up scattering protocols. Simulation results shows that the decentralized algorithm proposed in [1] comes within 4% to 11% of the optimum. Moreover, we show that the modeling framework we use to derive the solution, based on integer programming techniques, allows for a particularly efficient solution. The latter result discloses important opportunities for the practical utilization of the model. The model is also general enough to encompass alternative formulations of the problem.

1 Introduction

The operation of a wireless sensor network (WSN) node is usually 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 used to minimize energy consumption, therefore maximizing the lifetime of the network at large. Nevertheless, the performance of WSN applications critically depends on the quality of the duty-cycling schedule, ensuring that the right nodes are active at the right time. A random schedule may lead to an inefficient use of resources. Consider Figure 1, where three nodes cover, with overlaps, a given area. Ensuring that nodes 2 and 3 are *not* simultaneously active saves energy. Indeed, since a point in the target area may be monitored by multiple nodes, it is possible to switch some off as long as the remaining active nodes cover the target area.

In their EWSN'07 paper [1], Giusti et al. 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. They demonstrated the effectiveness and practical relevance of their *wake-up scattering* protocol in several common WSN scenarios, including the coverage problem above. However, although they showed that their protocol yields significant improvements over random schedules, they did not answer the question about whether further improvements can be obtained and, if so, how significant.

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, based on integer linear programming (ILP), is inherently centralized as it assumes global topology knowledge. However, the significant theoretical contribution of this paper is that, under the reasonable assumption of a constant and periodically-repeated awake interval, the resulting optimization problem can be radically simplified, enabling an efficient solution. The significant computational speed-up we gain allows us to deal with a large number of nodes in a reasonable time. Therefore, besides providing a theoretical upper bound against which to evaluate distributed solutions, our technique can be used as an effective design tool in cases where parameters such as the deployment topology are under control. Moreover, we envision a combination of the two techniques, where our optimal off-line solution determines the initial configuration of the system at deployment time, and the distributed algorithm in [1] is used online to adapt to topology changes.

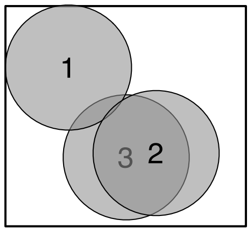


Fig. 1. A topology where wake-up scattering matters.

Section 2 presents a complete description of the problem, followed by the description of our approach and a proof that our formulation, which offers significant computational speed-ups, provides results equivalent to the general case. In Section 3 we compare quantitatively our optimal solution against the original decentralized one in [1]. The evaluation confirms that the latter algorithm, albeit very simple, is very efficient, as it comes within 4% to 11% of the optimum. Moreover, through our mathematical formulation we are able to gain insights into the relationships among coverage, lifetime, and node density.

Our modeling framework is actually general enough to represent problems other than coverage, similarly affected by duty-cycling: we discuss these issues in Section 4. Finally, after a concise survey of related approaches in Section 5, we end the paper with our concluding remarks in Section 6.

2 System Model and Solution Algorithm

Our approach assumes[4] 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* n_A, i.e., the area that is directly monitored by n. We assume that the sensitivity of

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

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.

Our objective is to maximize the coverage by scheduling the nodes' wake-up time. At any time t, the set of nodes awake at t defines a covered area $S(t)$, equal to the union[5] of the sensing areas of all awake nodes. Our objective is thus to maximize the integral of $S(t)$ over the epoch. Clearly, constraining the awake interval and the epoch is tantamount to fixing the network lifetime, and one could argue that we address the problem of maximizing the coverage *given* the lifetime. However, as shown in Section 4, solving the problem for different values of the lifetime allows us to explore the tradeoffs between lifetime and coverage.

Our method to solve the optimization problem goes through two steps, which are described in detail in Section 2.1 and 2.2, respectively.We first consider and solve the *spatial partitioning* problem, i.e., how to determine a *finite* number of regions in the target area that, without loss of accuracy, can be used to model the system topology and the area overlaps. Then, we set up an integer linear program expressing the scheduling problem, with coverage constraints for each node and region found in the previous step. The optimization problem is then solved using standard linear programming and branch-and-bound techniques, and yields the optimal schedule for the given objective function and constraints.

2.1 Spatial partitioning

The optimization problem can be setup as a linear program by enforcing a certain level of coverage on the target area A, using appropriate constraints discussed in Section 2.2. 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. Here 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 reasoning on the angles of intersection.

[5] Unlike the *sum* of these areas, overlaps are not counted multiple times.

This way, 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. Here, we present a simpler algorithm, with the same complexity, for the case of rectangular rather than the more traditional circular sensing areas, and use coordinates instead of angles. 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 arbitrary shapes, albeit with increased computational complexity. In any case, our optimization is *independent* of the shape of the sensing area and the way regions are computed.

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. A region (or field) ρ is the largest subset of points of A which are covered by the same set of nodes, We denote the set of regions by the symbol $\mathcal{R}$. For each region $\rho \in \mathcal{R}$, the corresponding area is returned by the function $w : \mathcal{R} \to \mathbb{R}_+$. Given that there is a one-to-one mapping, we use $r(\rho)$ to denote the set of nodes covering 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.

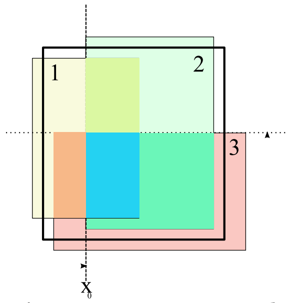


Fig. 2. Target area and scanning sequence.

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

Fig. 3. Regions for the topology of Figure 2.

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 updates their area. This is sufficient, since the computation of the optimal schedule does not require the shape of the regions, but only their area and corresponding set of covering nodes. The latter conveniently identifies the region, 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 next 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 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,

Algorithm 1 Spatial partitioning algorithm.

```
1: $h\_scan$ = list of vertical boundaries of 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 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:        $area = h\_extent \cdot v\_extent$
20:        $\mathcal{R} = \mathcal{R} \cup \{node\_set\}$
21:        $node\_set.area = node\_set.area + area$
22:    end for
23: end for
```

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, by 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). If the region was already present in the set, we do not add a new element but simply update its area (line 21). The result of applying Algorithm 1 to the topology of Figure 2 is shown in Figure 3, where each region ρ is associated with its covering nodes and its area. Note that the area covered outside the target area A is ignored.

2.2 Computation of the optimal schedule

As discussed, we consider periodic schedules of a fixed duration E, called the *epoch*. We assume that every node wakes up exactly once per epoch, and operates for a defined time I, 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. The original decentralized algorithm described in [1] approximates this idea by scattering the awake times of neighboring nodes, which are more likely to overlap. Finding the optimal solution to this problem is complex, and a naive formulation easily results in algorithms that are impractical even for small networks. Nonetheless, 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.*

Instead of considering arbitrary wake-up times distributed on the real line, we can discretize the problem by dividing the epoch E into $L = E/I$ equal *slots* of length I, where 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. Assuming that the epoch is an integer multiple of the awake interval may seem a severe limitation. However, high power savings can be achieved only at low duty-cycles, where our approach has sufficient granularity to provide accurate trade-offs.

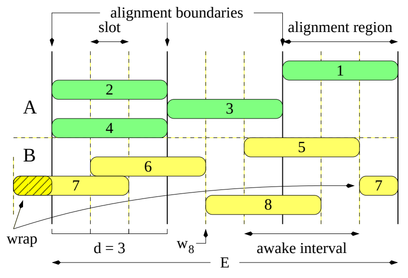


Fig. 4. Node alignment.

To prove the theorem (see [5] for an extended proof), we assume that the epoch E is initially finely discretized into s slots, $s \gg L$, and 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 4 depicts our notation. Slots are ordered and identified by their position $0 \leq k \leq s - 1$. In addition, all operations on slot indices are done modulo s. 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, i.e., 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 the set $\mathcal{A}$ of those that are already aligned, and the set $\mathcal{B}$ of those which are not. 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. Thus, for $k \bmod d = 0$, $\mathcal{B}_{k-1} = \mathcal{B}_k$. Similarly, the aligned nodes in slots that belong to the same alignment region are, of course, the same. Therefore, $\mathcal{A}_k = \mathcal{A}_{k'}$ for $dm \leq k, k' \leq d(m + 1) - 1$.

We compute the gain (positive or negative) in 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. The change in coverage depends on the area overlaps before and after: without overlaps all schedules are equivalent. More precisely, because we shift all non-aligned nodes together, the *gain* g_k^+ 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, 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 $A : \mathbb{R}^2 \to \mathbb{R}$ be the function that to a subset of $\mathbb{R}^2$ gives the corresponding area. Then,

$$g_k^+ = A(a_k \cap b_k) - A(a_{k+1} \cap b_k). \tag{3}$$

A slot k which is not near an alignment boundary, however, gives no gain, since in that case $a_k = a_{k+1}$. Recalling that $L = s/d$, and by similar arguments for left shifts, we can therefore express the total gains as

$$G^+ = \sum_{i=0}^{L-1} g_{di-1}^+, \qquad G^- = \sum_{i=0}^{L-1} g_{di}^-. \tag{4}$$

The key observation is that a right and a left shift give gains that are equal, but of opposite sign. In fact, let $di = k$ be a slot marking the beginning of an alignment region. As observed before, $b_{k-1} = b_k$. Therefore,

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

By matching corresponding terms in G^+ and G^-, it follows that $G^+ = -G^-$.

If a shift of the non-aligned nodes does not result in any new node being aligned, a further shift of the same non-aligned nodes in the same direction will give the exact same area gain. For instance, let G_1^+ be an initial right shift. Under the new configuration, if the aligned node do not change, it must be $G_1^+ = -G_2^-$. By the same argument, $-G_2^- = G_2^+$, and therefore $G_1^+ = G_2^+$. The same holds for a left shift that does not alter the partition.

The proof of Theorem 1 proceeds by induction. Given a non-aligned schedule, it is easy to construct an aligned schedule which has equal or better coverage. It is sufficient to shift the non-aligned nodes in the direction of zero or positive gain (since gains are opposite, they are either both zero or one of them is positive). Once a new node is aligned, we proceed by shifting the remaining non-aligned nodes, until the schedule is fully aligned. Since all moves had zero or positive gain, the new schedule has equal or better coverage. Thus, given an optimal schedule, we are always able to find an aligned schedule with the same coverage.

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. However, it ensures that the coarsest possible discretization of the problem is also optimal, which allows us to set up a significantly simpler optimization problem. 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 $C_{\rho,k}$ which take value 1 whenever region ρ is covered during slot $k \in [0, L-1]$, and 0 otherwise. Since regions are by definition disjoint, using the coverage variables $C_{\rho,k}$ and the area function A, the covered area can be computed as

$$S = \sum_{k=0}^{L-1} \sum_{\rho \in \mathcal{R}} C_{k,\rho} \cdot A(\rho). \tag{5}$$

Our objective is to maximize the total area S.

The value of $C_{\rho,k}$ depends on the schedule. To model it, we introduce a set of binary *scheduling* variables $x_{n,k}$ which take the value 1 whenever node n is awake in slot k, and 0 otherwise. Then, 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 constraints:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1],\ C_{\rho,k} \leq \sum_{n \in r(\rho)} x_{n,k} \tag{6}$$

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1], \forall n \in r(\rho),\ C_{\rho,k} \geq x_{n,k} \tag{7}$$

$$\forall n \in \mathcal{N}, \sum_{k=0}^{L-1} x_{n,k} \leq 1 \tag{8}$$

Constraint (6) forces the condition $C_{\rho,k} = 0$ when no node covers a region in a slot, (7) ensures that $C_{\rho,k} = 1$ when at least one node covers a region, and finally (8) ensures that a node wakes up at most once per awake interval.

The optimization problem can be further simplified by observing that the coverage variables $C_{\rho,k}$ are constrained by Equations (6) and (7) 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:

$$\forall \rho \in \mathcal{R}, \forall k \in [0, L-1],\ \ 0 \leq C_{\rho,k} \leq 1 \tag{9}$$

More complex situations can also be handled, however at the expense of increased computational complexity. We describe some of these in Section 4, and compare them against related work in Section 5.

3 Optimal vs. Distributed: Evaluation

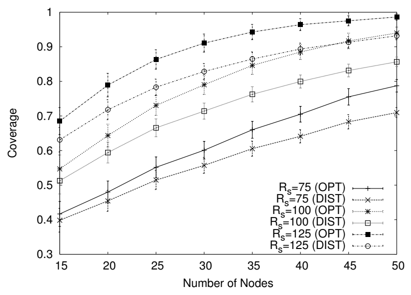
As stated previously, our motivation for this work was to find an optimal solution for the wake-up scattering problem, and determine how close the original distributed solution [1] comes to it. This section presents an evaluation by using covered area as the primary performance metric. Specifically, 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 [6], an open source linear programming kit[6], 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.

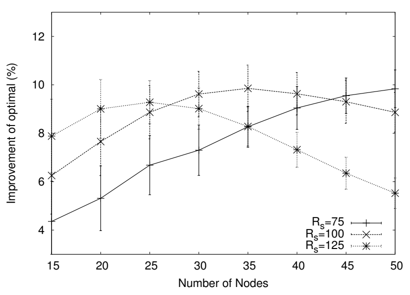
The majority of our experiments were performed in a 500×500 area. The number of nodes varied from 15 to 50, allowing us to consider increasing node densities. To achieve statistical significance, each point in our plots represents an average over 100 distinct topologies. Additionally, because the distributed 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 5(a) and 6(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. In general, the evaluation shows how the distributed wake-up scattering in [1], albeit very simple, is remarkably effective and yields schedules performing within 4% to 11% of the optimal ones computed with the technique presented here.

[6] Although the GLPK documentation mentions a 10-100 performance gap w.r.t. commercial tools, our computation times on a common workstation were in the order of a few hours even for a very complex topology with a hundreds of nodes.

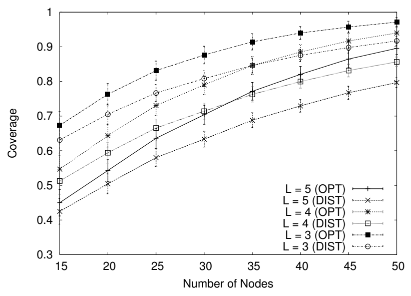


(a) Absolute coverage.

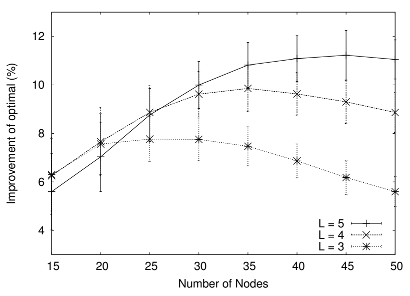


(b) Percent improvement of optimal.

Fig. 5. Fixed number of slots ($L = 4$), variable sensing range.



(a) Absolute coverage.

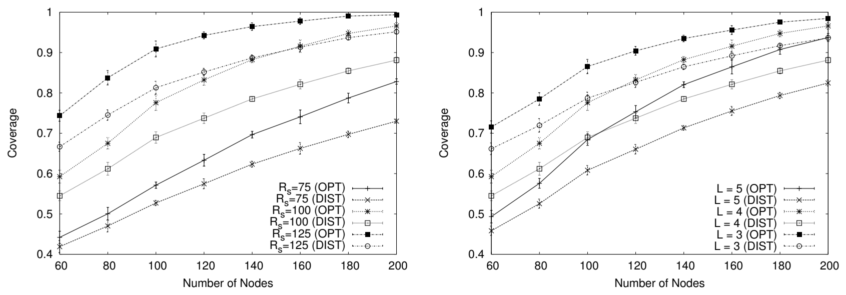


(b) Percent improvement of optimal.

Fig. 6. Fixed sensing range ($R_S = 100$), variable number of slots.

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 5(b) and 6(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 one performs better.

The initial similarity of the two solutions is due the fact that at low densities very few nodes overlap in space. In a sense, at low densities the system is far away from saturation and both solutions find schedules in which overlapping sensing areas overlap very little in time. As the density increases, overlaps become inevitable and the system must scatter the wake up times to eliminate them as much as possible. Unfortunately, to properly manage spatial overlaps, node locations must be considered and nodes close to one another should be more scattered. However, because the distributed solution does not consider the distance between nodes and does not change the sequence of node wake-up times, with more dense scenarios it is more likely that physically close nodes are next to one another in the wake-up sequence, and therefore less scattered. At low densities, physically close nodes are also close in the schedule, however, with fewer total neighbors, the interval between wake-up times is larger. In contrast, the



(a) Fixed number of slots ($L = 4$). (b) Fixed sensing range ($R_S = 100$).

Fig. 7. Experiments with 1000×1000 area.

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 5(b), larger sensing ranges saturate faster, while in Figure 6(b), in scenarios with fewer slots, when nodes are awake for longer, saturation occurs earlier. Interestingly, in Figure 5(b) the sensing range $R_S = 75$ does not reach saturation, which is however likely to 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 10 topologies for each point. The trends in Figure 7 clearly follow those in Figures 5(a) and 6(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.

4 Using and Extending the Model

Although our original motivation was to compare the optimal and distributed solutions, we explore here two additional research contributions: some practical applications of the optimal results and an extension to a new class of problems.

As shown in Section 3, the distributed solution closely approximates the optimal solution, coming within 4% to 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 the offline optimization tool, such as the one in Figure 8 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. In case of a controlled deployment, not only the centralized algorithm provide a quick evaluation of the results that can be achieved, but it can be also used as a full-fledged design tool to decide the initial schedule of nodes. In this case, the distributed algorithm can be used to adapt to changes (e.g., depletion of the battery on some nodes) affecting the system topology.

While the above provide an immediate, practical use for the optimal solution, we also explored how the formulation proposed in Section 2.2 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. If we take the common definition of lifetime based on failure of the first node, the goal 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, one 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 collection tree.

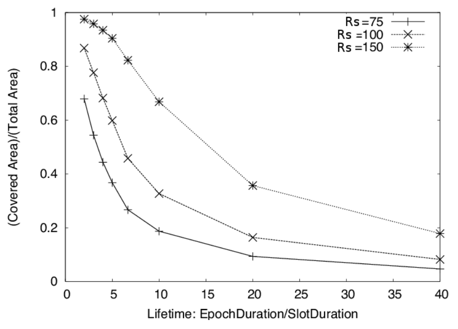


Fig. 8. Lifetime vs. covered area with $L = 4$ slots.

To give an example, we consider the problem of maximizing lifetime given a certain level 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:

$$\min \quad \max_{n \in \mathcal{N}} \alpha I(n) + \beta W_a(n), \tag{10}$$

subject to

$$< \text{the assigned area is covered} > \tag{11}$$

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:

$$\min \mu, \qquad \text{subject to} \qquad \forall n \in \mathcal{N},\, \mu \geq \alpha I(n) + \beta W_a(n)$$

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:

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

The above can be linearized using new variables $s_{n,k}$, and the constraints

$$s_{n,k} \geq x_{n,k} - x_{n,((k+1) \bmod L)}$$
$$s_{n,k} \geq x_{n,((k+1) \bmod L)} - x_{n,k}$$

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 (11) 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 are computed using (6) and (7).

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

5 Related Work

Several studies approached the problem of maximizing the WSN lifetime 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, making use of global information about the deployment, and distributed techniques, typically limited by network connectivity but more easily adapting to network dynamics.

Within centralized techniques, Slijepcevic and Potkonjak [2] are among the first to address the problem of maintaining *full coverage* while minimizing power consumption through active/sleep schedule. 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 leverages a centralized heuristic of quadratic complexity in the number of nodes, shown to significantly improve over a simulated annealing approach. However, no exact solution is derived in the paper. Cardei et al. propose a method where nodes are divided in sets that are not necessarily disjoint, achieving further improvements [7]. The goal is to maximize the network lifetime by scheduling the activity of nodes while maintaining *full coverage* over a *finite set of points*. The problem is formulated as an integer linear program that, due to its complexity, is only solved through heuristics. Similar formulations are proposed in [8,9]. In the first case, the authors use a two-step procedure to compute the maximal lifetime and to include communication costs. 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 yield solutions that may perform as much as 40% of the optimal.

Although our technique is also based on an mixed integer (boolean) linear program, unlike previous work we do not aim at maintaining full coverage of an area or of a set of points, 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, in accordance to the problem statement of [1]. By doing so, we are also able to compute the optimal trade-off between coverage and lifetime. An experimental comparison of our approach against previous work is thus difficult, since the optimization objectives differ. In our approach, each node is activated exactly once per epoch, with the latter 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 [7]. While this applies also to our problem, we have found that for random topologies, the possible increase in the *largest total coverage* is negligible—zero or 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).

As for distributed techniques, they 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: if so, they go back to sleep to conserve energy. Ye et al. [10] 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. More elaborate schemes take coverage information from neighboring nodes into account, 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 that is also covered by neighboring nodes, by exchanging position information. When a node determines that it is fully covered by others it goes to sleep for the rest of the round. Hsin and Liu [11] improve on this scheme with a coordinated sleep scheme 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. This way, they obtain a more graceful degradation of the overall coverage as nodes fail. Similarly to our distributed solution, Cao et al. [12] use a setup phase where nodes schedule their operation to overlap least in time with nearby nodes covering the same area. This is done by exchanging exact position information and incrementally adjusting the schedule until the procedure converges. More recently, Cărbunar et al. [13] proposed a distributed algorithm for preserving full coverage and computing the coverage boundary. They reduce 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 in [3].

In contrast, the distributed technique in [1] is extremely simple, and requires neither exact position information nor time synchronization—a significant asset when considering the overall energy budget of a node. Nonetheless, we have shown that the resulting schedules provide a degree of coverage that come very close to the optimum. This is possible since the goal is to maximize the largest total coverage, instead of maintaining full coverage that, again, makes the experimental comparison with previous work difficult. Moreover, we showed that our mathematical formulation is useful to determine the best trade-off between coverage and lifetime, a very valuable input for dimensioning real deployments.

A related problem is how to achieve the best sensor *placement* to cover an area [14]. Solutions rely on integer programming [15], greedy heuristics [16,17,18] and virtual force methods [19]. These complement the temporal spreading investigated here, and can be applied in parallel to achieve further improvements.

6 Conclusions and Future Work

In this paper we presented a way to compute *optimal* schedules for scattering node wake-up times in a WSN. Although here we focused primarily on area coverage, the wake-up scattering problem has practical relevance in many settings, as discussed in the original wake-up scattering paper [1]. Our evaluation through simulation shows that the decentralized algorithm there presented, albeit very simple, is remarkably efficient, generating schedules whose performance is within 4% to 11% of the optimal ones. The formulation presented in this paper 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, the modeling framework we 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. Giusti, A., Murphy, A., Picco, G.: Decentralized Scattering of Wake-up Times in Wireless Sensor Networks. In: Proc. of the 4^{th} European Conf. on Wireless Sensor Networks (EWSN). LNCS 4373, Springer (January 2007) 245–260
2. Slijepcevic, S., Potkonjak, M.: Power efficient organization of wireless sensor networks. In: Proc. of the IEEE Int. Conf. on Communications (ICC). (June 2001)
3. Tian, D., Georganas, N.: 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. Huang, C., Tseng, Y.: 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. Passerone, R., Palopoli, L.: Aligned schedules are optimal. Technical report, University of Trento (2008)
6. The Gnu Linear Programming Kit. `http://www.gnu.org/software/glpk/`
7. Cardei, M., Thai, M., Li, Y., Wu, W.: Energy-efficient target coverage in wireless sensor networks. In: Proc. of INFOCOM. (2005)
8. Liu, H., et al.: Maximizing lifetime of sensor surveillance systems. IEEE/ACM Trans. on Networking **15**(2) (2007) 334–345
9. Alfieri, A., Bianco, A., Brandimarte, P., Chiasserini, C.F.: Maximizing system lifetime in wireless sensor networks. European Journal of Operational Research **127**(1) (August 2007) 390–402
10. Ye, F., Zhong, G., Cheng, J., Lu, S., Zhang, L.: PEAS: A robust energy conserving protocol foo long-lived sensor networks. In: 3^{rd} Int. Conf. on Distributed Computing Systems (ICDCS'03). (May 2003)
11. Hsin, C., Liu, M.: 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)
12. Cao, Q., Abdelzaher, T., He, T., Stankovic, J.: 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)
13. Carbunar, B., Grama, A., Vitek, J., Carbunar, O.: Redundancy and coverage detection in sensor networks. ACM Trans. on Sensor Networks **2**(1) (2006) 94–128
14. Meguerdichian, S., Koushanfar, F., Potkonjak, M., Srivastava, M.: Coverage problems in wireless ad-hoc sensor networks. In: Proc. of 20^{th} IEEE INFOCOM. (April 2001) 1380–1387
15. Chakrabarty, K., Iyengar, S.S., Qi, H., Cho, E.: Grid coverage for surveillance and target location in distributed sensor networks. IEEE Trans. on Computers **51**(12) (2002) 1448–1453
16. Bulusu, N., Estrin, D., Heidemann, J.: Adaptive beacon placement. In: Proc. of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS). (April 2001) 489–498
17. Howard, A., Mataric, M., Sukhatme, G.: An incremental self-deployment algorithm for mobile sensor networks. Auton. Robots **13**(2) (2002) 113–126
18. Howard, A., Mataric, M., Sukhatme, G.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proc. of 7^{th} Int. Symp. on Distributed Autonomous Robotic Systems. (June 2002)
19. Zou, Y., Chakrabarty, K.: Sensor deployment and target localization in distributed sensor networks. IEEE Trans. on Embedded Computing Systems **3**(1) (2004) 61–91