Spatial-Temporal Coverage Optimization in Wireless Sensor Networks

Changlei Liu, *Student Member*, *IEEE*, and Guohong Cao, *Fellow*, *IEEE*

Abstract—Mission-driven sensor networks usually have special lifetime requirements. However, the density of the sensors may not be large enough to satisfy the coverage requirement while meeting the lifetime constraint at the same time. Sometimes, coverage has to be traded for network lifetime. In this paper, we study how to schedule sensors to maximize their coverage during a specified network lifetime. Unlike sensor deployment, where the goal is to maximize the *spatial* coverage, our objective is to maximize the *spatial-temporal* coverage by scheduling sensors' activity after they have been deployed. Since the optimization problem is NP-hard, we first present a centralized heuristic whose approximation factor is proved to be $\frac{1}{2}$, and then, propose a distributed parallel optimization protocol (POP). In POP, nodes optimize their schedules on their own but converge to local optimality without conflict with one another. Theoretical and simulation results show that POP substantially outperforms other schemes in terms of network lifetime, coverage redundancy, convergence time, and event detection probability.

Index Terms—Wireless sensor network, coverage, sensor scheduling, distributed protocol, parallel algorithm.

---

1 INTRODUCTION

IN wireless sensor networks, there is a trade-off between network lifetime and sensor coverage. To achieve a better coverage, more sensors have to be active at the same time, then more energy would be consumed and the network lifetime is reduced. On the other hand, if more sensors are put into sleep to extend the network lifetime, the coverage will be adversely affected. The trade-off between network lifetime and sensor coverage cannot be simply solved at the deployment stage, because it is hard to predict the network lifetime requirement, which depends on the application and may change as the mission changes. For example, in a surveillance application, the initial mission is to monitor the battle field for 6 hours. As the battle goes on, the commander finds that the battle may have to last for 10 hours. Then, the mission of the sensor network is changed, which requires the network to last for 10 hours. Since it may not be possible to deploy more sensors, some sensors have to sleep longer during each duty cycle to extend the network lifetime. As a result, sensor coverage needs to be traded for network lifetime.

The coverage issue in sensor networks has been studied extensively [1], [2], [3], [4], [5], [6], where scheduling algorithms are proposed to maximize the network lifetime while maintaining some predefined coverage degree.[1] However, if the same coverage degree is maintained all the time, the lifetime requirements may not be satisfied as

1. The area is k-covered if every point in the area is covered by k sensors at the same time.

network condition and mission change. For example, the sensor density may drop over time and the coverage requirement may vary according to the application's demand. Different from existing works, we study how to schedule sensor nodes to maximize coverage under the constraint of network lifetime. This reverse formulation is especially useful when the number of nodes is not enough to maintain the required coverage degree for a specified time period, as shown in the above example.

In this paper, we aim to resolve the conflict between the static status of sensor deployment and the dynamic nature of mission requirements. As mission dynamically changes, the lifetime and coverage requirement may not be satisfied at the same time. Then, the coverage needs to be traded for the network lifetime. Our work is thus complementary to the existing work, which can be applied when the sensor density is sufficient to sustain both the lifetime and coverage requirement. To fulfill this goal, we have to consider the coverage in both spatial and temporal domain. In particular, we define a new *spatial-temporal coverage* metric, in contrast to the traditional area coverage. The spatial-temporal coverage of each small area is defined as the product of the area size and the length of the period during which the area is covered. Then, our objective becomes how to schedule the sensor's on-period to maximize the global spatial-temporal coverage, calculated as the sum of individual spatial-temporal coverage over all the areas. This new formulation arises naturally from the mission critical applications with the network lifetime constraint and differentiates itself from most existing works which only consider the spatial domain.

Consider a surveillance example shown in Fig. 1a. Three sensors monitor a rectangular area, where the overlap between sensor 1 and senor 2 is four units, and the overlap between sensor 2 and sensor 3 is one unit. Suppose the network is required to provide full coverage and operate for 10 hours. Since the battery lifetime is 6 hours for each sensor, the coverage and lifetime requirement cannot be
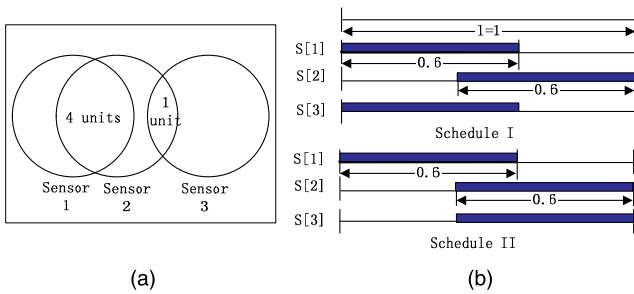
• *The authors are with the Department of Computer Science and Engineering, The Pennsylvania State University, Information Science and Technology Building, University Park, PA 16802.*
E-mail: {chaliu, gcao}@cse.psu.edu.

Fig. 1. A surveillance example with three sensors.

Fig. 2. An example to illustrate how to calculate the redundancy for k-redundant elementary regions.

satisfied. Most existing coverage-oriented algorithms in such a case would activate the three sensors simultaneously for 6 hours, without considering the network lifetime constraint. However, we trade the coverage for lifetime by dividing the mission duration of 10 hours into 10 cycles, and within each cycle, each sensor is active for $6/10 = 0.6$ hour. Then, the scheduling issue becomes how to place the 0.6 hour in each cycle so that the spatial-temporal coverage of the overlapping regions is maximized. Fig. 1b shows two solutions. From the spatial coverage point of view, the two schedules make no difference because each sensor covers the same area for the same length of time in different schedules. But from the spatial-temporal perspective, schedule I is better because the four-unit overlapping region is covered for a full cycle in schedule I but for only 0.6 cycle in schedule II. The spatial-temporal coverage of the overlapping regions in schedule I is $4 \times 1 + 1 \times 1 = 5$ and $4 \times 1 + 1 \times 0.6 = 4.6$ in schedule II.

The above example shows that different schedules may result in different coverage redundancy. Although the optimal solution for this example can be easily found (schedule I is actually the optimal solution) in a complex network setting where thousands of sensors are arbitrarily deployed, we need a systematic way to address the problem. Our contribution in this paper can be summarized as follows: First, we formalize the sensor scheduling problem in the spatial and temporal dimension with the objective to maximize the spatial-temporal coverage with network lifetime constraint. We further prove that it is equivalent to minimize the coverage redundancy under certain conditions. Second, we prove that the problem is NP-hard and propose a centralized greedy algorithm with an approximation factor of $\frac{1}{2}$. Third, we propose a distributed heuristic, the parallel optimization protocol (POP), where nodes not only optimize their schedules on their own but also converge to local optimality without conflict with one another.

The rest of the paper is organized as follows: Section 2 presents the problem formulation. Section 3 shows the problem complexity and proposes a centralized approximation algorithm. Section 4 presents the distributed heuristic. Performance evaluations are done in Section 5. Section 6 gives related work and Section 7 concludes the paper.

2   PROBLEM FORMULATION

When the sensor density is not sufficient to satisfy both the lifetime and coverage requirements, the coverage has to be traded for lifetime. In such a case, the sensors have to make
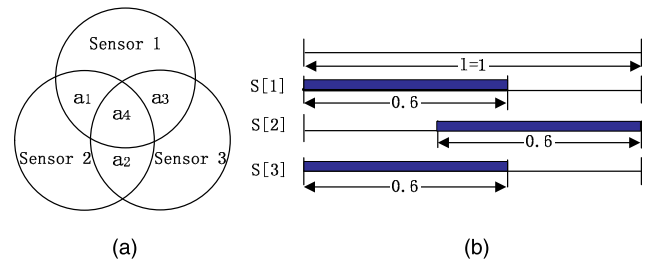
their best efforts to provide the coverage while meeting the lifetime constraint. To achieve this, we divide the network lifetime L into cycles and turn on each sensor within each cycle for a period proportional to its battery life. We further designate that the same schedule repeats in each cycle, such that the sleep schedule can be implemented, e.g., using the Power Saving Mode of 802.11. Then, the purpose of the scheduling is to place the on-periods within each cycle, such that the total spatial-temporal coverage can be maximized. We formalize it as a $maxCov$ problem in Section 2.1, and then, transform it to a $minRed$ problem in Section 2.2 whose objective is to minimize the overall coverage redundancy.

2.1   Maximize the Spatial-Temporal Coverage

Problem MaxCov. *Given a unit-disk graph $G(N, E)$ with n nodes, the battery life of each sensor $B_i, i = 1 \ldots n$, and a mission lifetime of L, where $B_i \leq L$, we want to calculate an "on" schedule per cycle for each sensor such that the overall spatial-temporal coverage is maximized.*

To quantify the overall spatial-temporal coverage (or coverage, for short), we first define *elementary region* as the minimum region formed by the intersection of a number of sensing disks. Notice that different points belonging to the same elementary region are covered for the same length of time. Therefore, the spatial-temporal coverage of each elementary region can be calculated as the product of its area size and the length of time during which the region is covered by *at least* one sensor. Note that for each elementary region, the area size is fixed after the sensors are deployed, but the coverage time varies depending on the different sensor schedule.

Further, define the *k-redundant elementary region* as the elementary region formed by the intersection of k sensors, where $k \geq 2$. For example, in Fig. 2a, there are seven elementary regions and four of them are redundant elementary regions, whose area sizes are $a_1 = a_2 = a_3 = a_4 = 1$. a_1, a_2, and a_3 refer to the two-redundant elementary regions and a_4 refers to the three-redundant elementary region. The *nonredundant elementary regions* are covered by only a single sensor, such as those elementary regions other than a_1, a_2, a_3, and a_4 in Fig. 2. Since the coverage time of the nonredundant elementary region is the same as the "on" period of that sensor, its spatial-temporal coverage is constant irrespective of the sensors' schedule. Therefore, to devise a better "on" schedule per cycle for each sensor, we only need to focus on the redundant elementary regions to maximize their total spatial-temporal coverage.

Given the schedule in Fig. 2b, the spatial-temporal coverage of the two-redundant elementary region can be

calculated similar to that of Fig. 1. For example, the spatial-temporal coverage for a_1 is the product of the area of a_1 and the time during which a_1 is covered by either s_1 or s_2, or both, i.e., $1 \times 1 = 1$. Similarly, the coverage for a_2 and a_3 is $1 \times 1 = 1$ and $1 \times 0.6 = 0.6$, respectively. For the three-redundant elementary region a_4, we need to find out the length of time during which it is covered by at least one of the three sensors, which is 1 time unit in Fig. 2. Therefore, the total spatial-temporal coverage over all the redundant elementary regions in Fig. 2 is $1 + 1 + 0.6 + 1 = 3.6$.

In general, we can formalize the problem in the form of mathematical programming. Before giving the formulation, we first define some notations that will be used throughout the paper.

- N, E: the node set and the edge set of the graph.
- $s_i, N(i)$: sensor i and the neighbor set of sensor i, i.e., $N(i) = \{j|\ \text{sensor } s_j \text{ is the neighbor of sensor } s_i\}$.
- I: the index set of the redundant elementary regions, i.e., $I = \{i|\ \text{region } i \text{ is a redundant elementary region}\}$.
- $I(m)$: the index set of the redundant elementary regions within the sensing disk of sensor m.
- $A(i)$: the index set of sensors whose intersection of sensing disks forms the ith redundant elementary region, with $|A(i)|$ denoting its cardinality.
- a_i: the area size of the ith redundant elementary region.
- T_i: the time during which the ith redundant elementary region is covered by at least one sensor.
- L, l: L is the network lifetime and l is the length of the cycle, so there are $\frac{L}{l}$ number of cycles, assuming $\frac{L}{l}$ is an integer. l should not be too small so that the switch overhead between the on/off state is negligible.
- B_i, b_i: B_i is the battery life of s_i and b_i is the length of s_i's on-period per cycle. Since there are $\frac{L}{l}$ number of cycles, we have $b_i \times \frac{L}{l} = B_i$.
- $s_i.start$, $s_i.end$: the start and end of s_i's on-period, respectively, where $s_i.end - s_i.start = b_i$.
- C: the total coverage of all the redundant elementary regions.

With these notations, we can calculate the total coverage as the sum of the product of area size and coverage time, over all the redundant elementary regions. Thus, our objective function and constraints are

$$\text{Max } C = \sum_{i \in I} a_i \times T_i, \tag{1}$$

$$\text{ST} : 0 \leq s_i.end \leq l, i \in N, \tag{2}$$

$$b_i = B_i \times \frac{l}{L}, i \in N, \tag{3}$$

$$s_i.end - s_i.start = b_i, i \in N. \tag{4}$$

The purpose of the above optimization is to determine the variables $s_i.end$ (or $s_i.start$) to maximize the spatial-temporal coverage subjecting to the constraints (2), (3), and (4). In the objective function, a_i is the area size of the ith redundant elementary region, which could be in arbitrary shape and T_i is the time during which the ith redundant elementary region is covered by at least one sensor, which depends on the schedules of all the neighboring sensors. Constraint (2) shows that the on-period may fall

on the boundary of the cycle, so $s_i.end$ ranges from 0 to l. Constraint (3) requires that each node's on-period within a cycle is proportional to its battery life. Constraint (4) establishes the relationship between $s_i.end$, $s_i.start$ and the length of its on-period.

2.2 Minimize the Coverage Redundancy

In this section, we consider the coverage maximization problem from another perspective and propose a new formulation. In the previous section, the objective is to maximize the total spatial-temporal coverage, which desires the total coverage time of each redundant elementary region to be as large as possible. Alternatively, we can achieve the same goal by minimizing the schedule overlap of the sensors that monitor the same redundant elementary region. Toward this direction, we propose another metric, *spatial-temporal coverage redundancy*, whose value depends on the area size, the overlapping "on" periods, and the number of sensors that monitor the area in each period. With the concept of spatial-temporal coverage redundancy (or *coverage redundancy*, for short), the problem of "maximizing coverage under the constraint of network lifetime" becomes "minimizing the coverage redundancy under the constraint of network lifetime" (called $minRed$ problem). We can prove that the two objectives are equivalent under certain conditions.

We first use Fig. 2 as an example to illustrate how to calculate the coverage redundancy of the redundant elementary regions. For instance, the coverage redundancy for a_1 is the area of a_1 times the schedule overlap of s_1 and s_2, i.e., $1 \times 0.2 = 0.2$. Similarly, the redundancy for a_2 and a_3 is 0.2 and 0.6, respectively. The coverage redundancy of a_4 consists of two parts, i.e., the part of time when a_4 is covered by *exactly* two sensors, and the part of time when it is covered by *exactly* three sensors. Intuitively, the two parts should have different contribution to the coverage redundancy, because more resources will be wasted as more sensors overlap in time. To reflect this, we assign different *weight* to different periods during which the same region is monitored by different number of sensors. In particular, a_4 is *solely* monitored by s_1 and s_2 for 0 unit of time, by s_1 and s_3 for 0.4 unit of time, by s_2 and s_3 for 0 unit of time, all of which are assigned weight 1. On the other hand, a_4 is solely monitored by s_1, s_2, and s_3 for 0.2 unit of time, and it is assigned weight 2. Then, the total coverage redundancy is the weighted sum of the product of area size and time overlap over all the redundant elementary regions. For example, in Fig. 2, the total redundancy is

$$(0.2) + (0.2) + (0.6) + (1 \times 1 \times 0 + 1 \times 1 \times 0.4 + 1 \times 1 \\ \times 0 + 3 \times 1 \times 0.2) = 2.$$

To study the problem from the perspective of coverage redundancy, we need to define more notations in addition to those used in the previous section.

- $t_i^j(S), S \subseteq A(i)$: the time during which the ith redundant elementary region is covered by *exactly* j sensors that include all the sensors in S. S can be an empty set $\emptyset$.
- $\overline{a_i a_j}$: the area overlap (i.e., the size of the overlapping area) between s_i and s_j.

- $\overline{s_i s_j}$: the time overlap (i.e., the length of the overlapping on-period) between s_i and s_j.
- R: the coverage redundancy of the whole network.

With these notations, the problem $minRed$ is formulated as follows, with the objective to minimize the total coverage redundancy:

$$\text{Min } R = \sum_{i \in I} \sum_{j=2}^{|A(i)|} w(j) \times a_i \times t_i^j(\emptyset), \qquad (5)$$

$$\text{ST} : 0 \le s_i.end \le l, i \in N, \qquad (6)$$

$$b_i = B_i \times \frac{l}{L}, i \in N, \qquad (7)$$

$$s_i.end - s_i.start = b_i, i \in N. \qquad (8)$$

In the objective function, the total coverage redundancy is calculated as the weighted sum of the product of area overlap and time overlap, first, over the possible coverage degrees within each region, and then, over all the redundant elementary regions. Specifically, $t_i^j(\emptyset)$ is the time during which the ith redundant elementary region is covered by *exactly* j sensors, which depends on the schedules of the j sensors. Intuitively, a larger j contributes more redundancy, so the weight factor $w(j)$ is used in the objective function to reflect this, which should be a monotonically increasing function of j. In the following, we can prove that this new objective function (5) is equivalent to the previous objective function (1) with a properly set weight factor:

Theorem 1. *With the same graph, network lifetime requirement, and battery constraints, the objective to maximize the total spatial-temporal coverage is equivalent to minimize the total spatial-temporal coverage redundancy when setting the weight factor to be $w(j) = j - 1$.*

Proof. We first rewrite the objective of total coverage, i.e., (1), by decomposing the coverage time T_i into a multitude of subperiods according to the different coverage degree.

$$C = \sum_{i \in I} a_i \times T_i = \sum_{i \in I} \sum_{j=1}^{|A(i)|} a_i \times t_i^j(\emptyset). \qquad (9)$$

Then, we add the two objective functions together, i.e., (5) and (9). Since $w(j) = j - 1$, we have

$$C + R = \sum_{i \in I} \sum_{j=1}^{|A(i)|} j \times a_i \times t_i^j(\emptyset). \qquad (10)$$

Note that a set of sensors $A(i)$ monitor the elementary region a_i, and each sensor $k \in A(i)$ is active for b_k period of time per cycle. So each active period b_k can be decomposed into subperiods according to the different coverage degrees of region a_i, i.e., $b_k = \sum_{j=1}^{|A(i)|} t_i^j(s_k)$. Therefore, if we sum up $b_k, k = 1, \ldots, |A(i)|$, each $t_i^j(\emptyset)$ would be counted exactly j times. Then, we have

$$\sum_{k \in A(i)} b_k = \sum_{j=1}^{|A(i)|} j \times t_i^j(\emptyset). \qquad (11)$$

Combining (10) and (11), we have

$$C + R = \sum_{i \in I} a_i \sum_{k \in A(i)} b_k,$$

which is a constant value. This implies that maximizing the total coverage C is equivalent to minimizing the total coverage redundancy R. □

3 CENTRALIZED ALGORITHM DESIGN

In this section, we first prove that the $maxCov$ problem is NP-hard, and then, propose a centralized greedy algorithm whose approximation factor is $\frac{1}{2}$.

Theorem 2. *The problem $maxCov$ is NP-hard.*

Proof. Theorem 1 tells that the problem $maxCov$ can be transformed to the problem $minRed$. Therefore, we only need to prove that problem $minRed$ is NP-hard.

The $minRed$ problem can be proved to be NP-hard via a reduction from the graph k-coloring problem, which asks whether a given graph G can be colored using k colors such that no two neighboring vertexes have the same colors [7]. Given an instance I of graph k-coloring problem, we can construct an equivalent instance I' of $minRed$ *decision* problem in polynomial time, such that instance I has a solution if and only if instance I' has a solution. The decision problem can be stated as follows: let the node battery life B and network life L satisfy $\frac{L}{B} = k$. In the same graph G of k-coloring problem, is there a node scheduling scheme such that the total coverage redundancy is 0 in the $minRed$ decision problem?

To see the equivalence of the two problems, we divide each cycle in instance I' into $k = \frac{L}{B}$ time slots, with each time slot mapped to a distinct color in instance I. In essence, the instance I concerns about the assignment of one of the k colors to each node, while the instance I' concerns about the allocation of one of the k time slots to each node. Then, it can be observed that if there exists a k-coloring scheme where each node is assigned a color different from that of its neighbors', there also exists a corresponding scheduling scheme where each node is allocated a different time slot from that of its neighbors, with the total coverage to be 0.

On the other hand, suppose there is a scheduling scheme for instance I' where the total coverage is 0, we can always find a solution to the instance I of graph k-coloring problem. Since the "on" period may not exactly occupy a time slot, we need to first preprocess the schedule by aligning the on-period of each node with its left time slot. By doing this, the total coverage remains 0 and the aligned schedule is still the solution for instance I. After that, to construct a solution to the instance I' simply becomes equivalent to "color" each vertex using one of the k time slots, such that no neighboring vertexes have the same colors. In this regard, the solution of instance I' readily produces a corresponding solution for instance I.

To sum up, G can be k-colored if and only if there is a zero coverage redundancy scheduling scheme for G, which means that the graph coloring problem can be reduced to $minRed$ decision problem in polynomial

time. As the graph k-coloring problem is NP-hard, the $minRed$ decision problem is NP-hard. On the other hand, the $minRed$ optimization problem is at least as hard as its decision problem, and thus, is also NP-hard. □

Despite the problem complexity, we propose the greedy heuristics for the problem $maxCov$, with the pseudocode listed in Algorithm 1. Assume that the area size a_i is known for each elementary region i. The algorithm is executed in iterations. During each iteration, it calculates the maximal additional spatial-temporal coverage ΔC_i that each sensor s_i can provide based on the existing schedule and picks the sensor with the largest ΔC_i. Each time a new sensor is picked, the existing schedule within a cycle is augmented by placing the active period of the selected sensor in the optimal position of the cycle (i.e., where the sensor can provide the maximum coverage). This process will be repeated iteration after iteration, until no more sensors can be picked to increase the total spatial-temporal coverage.

Algorithm 1. Greedy Algorithm.
  Input: graph $G(N, E)$, mission lifetime L, the battery life $B_i, i = 1 \ldots n$, the area of each redundant elementary region $a_i, i \in I$
  Output: a subset of sensors S_{gre} with their schedules determined, the overall spatial-temporal coverage C_{gre}
  Procedure:
  1: $S_0 = N, S_{gre} = \emptyset$
  2: **while** $S_0 \neq \emptyset$ and $\Delta C_k \neq 0$ **do**
  3:   **for** each $i \in S_0$ **do**
  4:     identify all the end points of the active periods among the neighbors whose schedule has been determined. Use S_e to denote the collection of these end points. $S_e = S_e + \{0\}$
  5:     calculate $\Delta C_i = \max_{s_i.start \in S_e} \sum_{j=2}^{|A(i)|} a_j \times \Delta T_j$, and the corresponding $s_i.start$ that achieve the optimal schedule
  6:   **end for**
  7:   find the sensor s_k that can provide the maximum coverage, i.e., $\Delta C_k = \max_{i \in S_0} \Delta C_i$
  8:   **if** $\Delta C_k \neq 0$ **then**
  9:     $S_0 = S_0 - \{s_k\}, S_{gre} = S_{gre} + \{s_k\}, C_{gre} += \Delta C_k$.
  10:    update the current schedule by adding the on-period of s_k
  11: **end if**
  12: **end while**
  13: Output S_{gre}, and C_{gre}.

Define $\Delta C_i = \sum_{j \in I(i)} a_j \times \Delta T_j$ as the sum of the product of area size a_j and additional coverage time ΔT_j over all the redundant regions covered by sensor s_i. The process of deriving an optimal schedule is to calculate the maximum ΔC_i for each sensor s_i by choosing the right $s_i.start$. Given an existing schedule among sensor i's neighbors whose on-periods are determined by its $s_i.start$ (or $s_i.end$), it turns out that it takes just $O(|N(i)|)$ computational effort to derive $\max\{\Delta C_i\}$, where $|N(i)|$ is the number of the neighbors of sensor s_i. The linear computational efficiency results from the observation that the maximum ΔC_i is achieved when $s_i.start$ is at one of the end points of the active periods of s_i's neighbors. This is because when $s_i.start$ moves between

two neighboring end points, the value of ΔC_i changes (i.e., increases or decreases) linearly. Therefore, we first identify the end points, i.e., $s_j.start$ and $s_j.end$, for each neighbor j of s_i, and then, select $s_i.start$ among those end points where ΔC_i can be maximized.

To analyze the performance of the algorithm, we cannot simply borrow the techniques from the traditional coverage model in the spatial domain. Due to the unique challenges of coverage issue in the spatial-temporal domain, the algorithm not only needs to select sensors but also needs to determine their corresponding schedules. In addition, the scheduling decision needs to be made in a continuous space, while there are infinite possibilities to place the on-period in a cycle. As a result, some traditional modeling approaches such as the set cover model [8] cannot be simply applied. Therefore, we need new techniques to analyze the algorithm complexity.

Theorem 3. *Algorithm 1 achieves an approximation factor of $\frac{1}{2}$ for the $maxCov$ problem.*

Proof. Suppose there are total n sensors picked in the greedy algorithm. Sensor s_i^g is picked in the ith iteration, which provides ΔC_i^g additional coverage with its active period placed at the optimal position of the cycle. Use the vector $V_i^g = (v_{i1}^g, v_{i2}^g \ldots, v_{i|I|}^g)$ to denote the coverage provided by s_i^g, with each component corresponding to each redundant elementary region. Since there are total $|I|$ redundant elementary regions, there are $|I|$ components for each vector. Each v_{ij}^g denotes the time periods during which the region j is covered by s_i^g. For example, $v_{ij}^g = \{[1, 3], [5, 6]\}$ means that region j is covered by sensor s_i^g during the periods $[1, 3]$ and $[5, 6]$. It can be seen that each period in v_{ij}^g is determined by a start time and an end time, with no overlap between the different periods. If region j is not covered by sensor s_i^g, set $v_{ij}^g = [0, 0]$. Further, use $|v_{ij}^g|$ to denote the total length of time during which region j is covered by s_i^g. For example, if $v_{ij}^g = \{[1, 3], [5, 6]\}$, then $|v_{ij}^g| = 3$.

Define the norm of the vector V_i^g in the form of a weighted sum, i.e., $|V_i^g| = \sum_{j=1}^{|I|} a_j \times |v_{ij}^g|$, where the weight is the area size a_j. For example, if $V_i^g = (\{[1, 3], [5, 6]\}, [0, 0], \{[2], [4]\})$, then $|V_i^g| = 3a_1 + 2a_3$. It can be seen that $|v_{ij}^g|$ denotes the total spatial-temporal coverage provided by s_i^g.

Further, suppose V_i^g and V_j^g are two different vectors, then define the vector addition/substraction as follows:

$$V_i^g + V_j^g = (v_{i1} \cup v_{j1}, v_{i2} \cup v_{j2}, \ldots, v_{i|I|} \cup v_{j|I|}), \quad (12)$$

$$V_i^g - V_j^g = (v_{i1} - v_{i1} \cap v_{j1}, \ldots, v_{i|I|} - v_{i|I|} \cap v_{j|I|}). \quad (13)$$

Thus, the total coverage vector generated by the greedy algorithm is $V^g = \sum_{i=1}^{n} V_i^g$ and the total coverage is

$$C_{gre} = \sum_{i=1}^{n} \Delta C_i^g = |V^g| = \left| \sum_{i=1}^{n} V_i^g \right|. \quad (14)$$

Suppose there are total m iterations in the optimal algorithm and during each iteration, one sensor is picked. Then, we rearrange the order of sensors selected in the optimal algorithm, such that if a sensor is also

chosen by the greedy algorithm, it is chosen in the same iteration in both the algorithms. Since the schedule of each sensor remains intact, changing the order of sensors selected will not affect the outcome of the optimal algorithm. Similar to the greedy algorithm, we also define the notations $s_i^o, V_i^o, |V_i^o|, v_{ij}^o, |v_{ij}^o|, \Delta C_i^o, V^o, C_{opt}$ for the optimal algorithm.

In the ith iteration, the greedy algorithm picks sensor s_i^g and enhances the total coverage by ΔC_i^g. Thus, we have $\Delta C_i^g = |V_i^g - \sum_{j=1}^{i-1} V_j^g|$. According to the definition of the greedy algorithm, during each iteration, it picks the sensor that can provide the maximum additional coverage, thus $|V_i^g - \sum_{j=1}^{i-1} V_j^g| \geq |V_i^o - \sum_{j=1}^{i-1} V_j^g|$, where V_i^o is the coverage vector of the sensor s_i^o picked in the same iteration by the optimal algorithm. In other words, the greedy algorithm picks s_i^g instead of s_i^o because s_i^g can provide more additional coverage than s_i^o. On the other hand, $V^g \supseteq \sum_{j=1}^{i} V_j^g$ for $\forall i = 1 \ldots n$. Therefore, we have

$$\Delta C_i^g \geq \left| V_i^o - \sum_{j=1}^{i-1} V_j^g \right| \geq \left| V_i^o - V^g \right|, \quad \forall i = 1 \ldots n. \quad (15)$$

Since n sensors are picked in the greedy algorithm and m sensors are picked in the optimal algorithm, to make the above equation also hold when $m < n$, let $V_i^o = ([0,0], \ldots, [0,0])$ for $\forall i \in (m, n]$. Then, adding all the inequalities denoted by (15) gives

$$\sum_{i=1}^{n} \Delta C_i^g = C_{gre} \geq \sum_{i=1}^{n} \left| V_i^o - V^g \right|. \quad (16)$$

Note that the spatial-temporal coverage of each sensor is denoted by a vector and each component of a vector is a collection of time periods. Then, based on the vector analysis and the addition/substraction defined in (12) and (13), it is not hard to see

$$\sum_{i=1}^{n} \left| V_i^o - V^g \right| \geq \left| \sum_{i=1}^{n} \left(V_i^o - V^g \right) \right| \geq \left| \sum_{i=1}^{n} V_i^o \right| - |V^g|. \quad (17)$$

As $C_{opt} = |\sum_{i=1}^{n} V_i^o|$ and $C_{gre} = |V^g|$, combining (16) and (17) gives

$$C_{gre} \geq C_{opt} - C_{gre}. \quad (18)$$

The above relationship shows that the approximation ratio of Algorithm 1 is $\frac{1}{2}$.   □

The centralized algorithm has theoretical favor, as it gives a constant factor performance bound. However, it is not practical as it is difficult to enumerate and compute each a_i and let each node have such global knowledge. Thus, in the next section, we propose the distributed heuristics based on the local coverage redundancy.

4 DISTRIBUTED ALGORITHM DESIGN

From the above discussion, we know that in a complex network of large scale, it is computationally infeasible to enumerate each elementary area a_i and list each period $t_i^j(\emptyset)$ during which area i is covered by exactly j sensors. Therefore, in the distributed design, we focus on the pair-wise sensors

and let each node minimize its own *local coverage redundancy*, defined as the sum of pair-wise redundancy with its neighbors. Although the global optimal is computationally infeasible to achieve, we can design a class of algorithms in which each node is able to achieve the local optimal if certain conditions can be satisfied. The basic idea is to let each node first generate a random schedule independently. Then, each node adjusts its schedule individually to minimize the local coverage redundancy with its neighbors, until everyone converges to its local optimality. The seemingly simple idea has several challenges:

- How to do the local optimization? Does it have polynomial time algorithms to achieve the local optimal?
- If each sensor adjusts the schedule individually, is the algorithm able to converge?
- How to eliminate conflicts caused by simultaneous adjustments of the neighboring nodes?

The following sections will address these challenges one by one.

4.1 Local Optimization

Without loss of generality, suppose sensor s_0 has $|N(0)|$ neighbors. The local optimization problem at s_0 can be formalized as follows:

Given the area overlap between s_0 and its neighbors (i.e., $\overline{a_0 a_i}, i \in N(0)$), the individual schedule of its neighbors, we want to decide s_0's own schedule, such that the local sum of the coverage redundancy with its neighbors $R[0] = \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}$ can be minimized.

It can be seen that the local coverage redundancy (i.e., $R[i] = \sum_{j \in N(i)} \overline{a_i a_j} \times \overline{s_i s_j}$) is much easier to calculate than the global redundancy (5). Suppose two sensors s_i and s_j, whose sensing radius is r and the distance between them is d. Their area overlap and time overlap can be simply computed by

$$\begin{cases} \overline{a_i a_j} = 2r^2 \arccos(\frac{d}{2r}) - d\sqrt{r^2 - \frac{d^2}{4}}, & \text{if } d < 2r; \\ \overline{a_i a_j} = 0, & \text{if } d \geq 2r; \\ \overline{s_0 s_j} = \max\{\min(s[i].end, s[j].end) \\ \quad - \max(s[i].start, s[j].start), 0\}. \end{cases} \quad (19)$$

Each node has its own reference cycle. The cycles at different nodes are not required to be synchronized. Each node only needs to know the relative position of its neighbor's on-period. This can be easily achieved via exchange of hello packets with its neighbors.

Note that s_i's schedule per cycle is solely determined by the start of its on-period $s_i.start$ and the end of its on-period $s_i.end$, where $s_i.end - s_i.start = b_i$. Then, the objective of local optimization at s_0 is to decide $s_0.end$ (or $s_0.start$) within its own reference cycle such that $R[0]$ is minimized. However, because $s_0.end$ could be any value between 0 and l, it is not realistic to enumerate all the possibilities. In our solution, we only focus on some crucial points, which could jointly determine the redundancy $R[0]$ at every possible value of $s_0.end$.

Line traversal algorithm. s_0 first selects its own reference cycle and places each neighbor's schedule (i.e., on-period) in the cycle. Then, s_0's on-period traverses from the left of
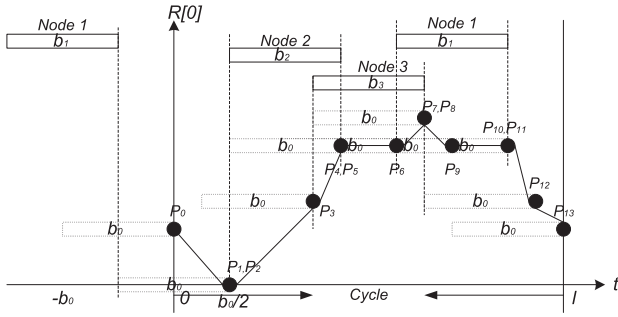
Fig. 3. An example to illustrate the line traversal algorithm. The piecewise curve depicts the relationship between $R[0]$ and $s[0].end$. There are total 14 crucial points, at which the slope k of the curve changes.

the cycle (i.e., $s_0.end = 0$) to the right of the cycle (i.e., $s_0.end = l$), during which the local redundancy $R[0]$ over the whole range can be recorded. In the end, the points corresponding to the minimum $R[0]$ are identified and selected as s_0's schedule. For example, in Fig. 3, s_0 has s_1, s_2, and s_3 as neighbors, whose schedules are given. For ease of illustration, assume $b_i = b_j, i, j = 0 \ldots 3$. However, the algorithm is not limited in the homogeneous case but allows the heterogeneous battery states at different nodes. Fig. 3 shows the relationship between $R[0]$ and $s_0.end$ by executing the line traversal algorithm.

The algorithm is based on the observation that the coverage redundancy $R[0]$ increases/decreases linearly as $s_0.end$ traverses from left to right, and the slope k shifts only at some *crucial points*, which corresponds to the following four cases:

- Case I: the end of s_0's on-period enters the start of s_i's on-period, i.e., $s_0.end = s_i.start$, then the slope increases by $\overline{a_0 a_i}$, i.e., $k = k + \overline{a_0 a_i}$.
- Case II: the end of s_0's on-period leaves the end of s_i's on-period, i.e., $s_0.end = s_i.end$, then the slope decreases by $\overline{a_0 a_i}$, i.e., $k = k - \overline{a_0 a_i}$.
- Case III: the start of s_0's on-period enters the start of s_i's on-period, i.e., $s_0.start = s_i.start$, then the slope decreases by $\overline{a_0 a_i}$, i.e., $k = k - \overline{a_0 a_i}$.
- Case IV: the start of s_0's on-period leaves the end of s_i's on-period, i.e., $s_0.start = s_i.end$, then the slope increases by $\overline{a_0 a_i}$, i.e., $k = k + \overline{a_0 a_i}$.

We use Case I as an example to illustrate why the slope k is updated in such a way. When the end of s_0's on-period enters the start of s_i's on-period (such as at point P_3 in Fig. 3), the time overlap between s_0 and s_i starts to increase as $s_0.end$ traverses to the right. Thus, the slope of $R[0]$ will increase by $\overline{a_0 a_i}$, i.e., $k = k + \overline{a_0 a_i}$, where $\overline{a_0 a_i}$ is the area overlap between s_0 and s_i.

Since the on-period may fall on the boundary of the cycle, we let $s_0.end$ traverse from 0 to l, and count the coverage redundancy $R[0]$ over the range $[-b_0, l]$. Because each crucial point corresponds to one of the above four cases and s_0 has $N(0)$ neighbors, there are $4 * N(0)$ crucial points, denoted as $P_j, j = 1 \ldots 4N[0]$. Adding two points $s_0.end = 0$ and $s_0.end = l$, denoted as P_0, $P_{4N[0]+1}$, there are total $4N[0] + 2$ crucial points. Since the slope k only changes at the crucial points, the relationship between $R[0]$ and $s_0.end$ can be

presented by a piecewise curve, as seen from Fig. 3. Note that some crucial points may overlap. For example, in Fig. 3, P_1 and P_2 overlap because $s_0.end$ enters b_2 and $s_0.start$ leaves b_1 at the same time; P_4 and P_5 overlap because $s_0.end$ leaves b_2 and $s_0.start$ enters b_2 at the same time. We use $R[0][j]$ to denote the coverage redundancy at P_j and use $k[j]$ to denote the slope between points P_j and P_{j+1}, then we have the following recursive relationship:

$$\begin{cases} R[0][j+1] = R[0][j] + k[j](P_{j+1} - P_j) \\ R[0][0] = \frac{1}{2} \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}. \end{cases} \quad (20)$$

The above recursive relationship shows that the value of $R[0]$ at the current crucial point can be determined by its value at the previous point and the slope in between. For example, in Fig. 3, initially at P_0, $s_0.end = 0$, $R[0] = \frac{1}{2} \overline{a_0 a_i} \times \overline{s_0 s_1}$, and $k[0] = -\overline{a_0 a_1}$. As $s_0.end$ moves right, $R[0]$ decreases linearly until it hits P_1. At P_1, $s_0.end$ enters b_2 and $s_0.start$ leaves b_2, so the slope k increases by $(\overline{a_0 a_1} + \overline{a_0 a_2})$. Then, $R[0]$ begins to increase linearly with $k[1] = -\overline{a_0 a_1} + \overline{a_0 a_1} + \overline{a_0 a_2} = \overline{a_0 a_2}$ until it reaches P_2. Similarly, as $s_0.end$ continues to move right, the value of k varies at the subsequent crucial points. When $s_0.end$ arrives at P_{13}, the value of $R[0]$ over the whole range of $[0, l]$ can be obtained, after which the same cycle is repeated.

With all the values of $R[0]$ at different points, the minimum $R[0]$ and its corresponding $s_0.end$ can be identified. In Fig. 3, at P_1, P_2, where $s_0.end = b_0/2$, $R[0] = 0$ is the minimum. In this case, $s_0.end = b_0/2$ is the only optimal schedule. However, in other cases, it is possible that the minimum $R[0]$ is achieved at multiple $s_0.end$. Then, we can break the tie arbitrarily and pick any optimal $s_0.end$.

The complexity of Line Traversal Algorithm is only $O(d)$, where d is the node degree. Suppose a node has d neighbors, then there are at most $4 * d + 2$ crucial points to be examined, and at each point, only linear algebraic operation is performed.

4.2 Convergence Property

In our distributed algorithm, each node locally optimizes its own schedule as long as its schedule does not remain locally optimal. Since altering a node's schedule can affect the redundancy of its neighbors, the schedule adjustment at different nodes may conflict with each other and the adjustment process may never end. For example, if two neighboring nodes adjust their own schedules at the same time, they may not be aware that their neighbor's schedule has been changed and cannot achieve local optimality. Next, we provide guidelines to guarantee that each node can converge to its local optimality.

Theorem 4. *Given a graph G and arbitrary schedules, a distributed algorithm will terminate in a finite number of steps and after termination, each node's schedule will converge to the local optimality, if*

- *no neighboring nodes optimize their schedules at the same time;*
- *each node's local adjustment continues as long as its local objective can be improved for at least a predefined threshold δ.*

Proof. First, we want to show that if the local objective improves (i.e., the redundancy at a node decreases), the global pair-wise redundancy will improve as well (i.e., the total redundancy decreases). Without loss of generality, suppose a node s_0 optimizes its local schedule. Because $R[0] = \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}$, we have $\Delta R[0] = \sum_{i \in N(0)} \Delta R[i]$. Further, because the sum of the local redundancy is $R_s = \sum_{i=1}^{n} R[i]$, we have $\Delta R_s = \Delta R[0] + \sum_{i \in N(0)} \Delta R[i]$. It is followed that $\Delta R_s = 2\Delta R[0]$, which shows that each time the local objective at s_0 is improved by $\Delta R[0]$, R_s is improved by $2\Delta R[0]$. Second, it can be shown that the global pair-wise redundancy is bounded, i.e., $R_s \leq \sum_{i=1}^{n} \sum_{j \in N(i)} \overline{a_i a_j} \times \min\{b_i, b_j\}$, where a_i, a_j and b_i, b_j are constant values. Therefore, the algorithm could terminate after a finite number of steps if the stated conditions are satisfied, where the threshold δ could be set arbitrarily small to approximate the local optimal point.   □

4.3 Distributed Protocol Design

Theorem 4 tells us that for a distributed protocol to converge, all three conditions have to be satisfied. Before presenting our distributed protocol, let's see two simple algorithms:

- *Random Algorithm*: each node generates a random schedule individually.
- *Serial Optimization Algorithm*: each node first generates a random schedule, based on which the schedule is locally optimized one by one. This serial optimization process is repeated until no improvement can be made beyond the predefined threshold δ.

Each of the above algorithms has its pros and cons. The random algorithm is simple, distributed, and has no message complexity. It can serve as a baseline for comparison. The serial optimization algorithm uses the Line Traversal Algorithm as a functional module to ensure that every node can achieve its local optimality, but it is centralized. In addition, for the serial algorithm to converge, many iterations are needed until no improvement can be made. Therefore, the serial algorithm takes a long time to terminate.

To retain the merit of the serial algorithm and remedy its weakness, we propose a POP. The basic idea of POP is to let many nodes locally optimize their schedules (using Line Traversal Algorithm) in parallel, so that it can converge much faster than the serial algorithm. According to Theorem 4, a set of nonneighboring nodes can adjust their own schedules simultaneously without causing any conflict. From the algorithmic point of view, to search for such a set of nonneighboring nodes is equivalent to finding an *Independent Set* [9], which is defined as a subset of nodes among which there is no edge between any two nodes. The set is a *maximal independent set (MIS)* if no more edges can be added to generate a bigger independent set. To find the MIS, each node independently determines whether it belongs to the set by comparing its weight with its neighbors. If it has the *best* weight in the neighborhood, it elects itself as belonging to the set, and then, no other neighbors can be chosen. In general, the algorithm can be denoted as $MIS(weight, criteria)$, where the weight can be *id, degree,*

energy, etc., and the criteria can be either *smallest* or *largest*. The criteria are used to interpret the meaning of best weight, i.e., the smallest or the largest.

Algorithm 2 lists the pseudocode of the POP protocol which can be implemented in a distributed manner. For clarity of presentation, we first introduce the protocol in a centralized manner, and then, give guidance to its distributed operation. Initially, all nodes are unlabeled. Then, each node individually determines whether it belongs to the MIS by comparing its weight with the neighbors. The labeled nodes locally optimize their schedules, after which the MIS algorithm will continue to run among the remaining unlabeled nodes. We term the time a *round* if during this period, an MIS is found and local optimization is executed in parallel at the nodes of the MIS. Several rounds comprise an *iteration* during which the coalition of the MIS elected can have all the nodes labeled. The MIS algorithm continues to run round after round and iteration after iteration until no improvements can be made to any node's schedule.

Algorithm 2. Line Traversal Algorithm.
  Input: Graph G, the schedules of s_0's neighbors
  $s_i, i \in N(0)$;
  Output: Node 0's schedule, $s_0.end$;
  Procedure:
  1: enumerate the set of crucial points in terms of the value
      of $s_0.end$, $\chi = \{x | x \in \sum_{i \in N(0)} \cup \{s_i.start, s_i.end,$
      $s_i.start + b_0, s_i.end + b_0\}, x \in [0, l]\}$
  2: sort the crucial set χ in increasing order
  3: $R[0][0] = \frac{1}{2} \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}, P_0 = 0, k[0]$
      $= \sum_{i \in A} \overline{a_0 a_i} - \sum_{i \in B} \overline{a_0 a_i}$
      /*the traversal starts at $s_0.end = 0$. A, B denote the set
      of neighbors whose on-period spans across 0 and $-b_0$
      (equivalently $l - b_0$), respectively.*/
  4: $j = 1$ /*the index of the crucial point*/
  5: **while** $\chi \neq \emptyset$ **do**
  6:    $R[0][j] = R[0][j-1] + k[j-1](P_j - P_{j-1})$
  7:    **if** $s_0.end == s_i.start$ **then**
  8:       $k[j] = k[j-1] + \overline{a_0 a_i}$
  9:    **end if**
  10:   **if** $s_0.end == s_i.start + b_0$ **then**
  11:      $k[j] = k[j-1] - \overline{a_0 a_i}$
  12:   **end if**
  13:   **if** $s_0.end == s_i.end$ **then**
  14:      $k[j] = k[j-1] - \overline{a_0 a_i}$
  15:   **end if**
  16:   **if** $s_0.end = s_i.end + b_0$ **then**
  17:      $k[j] = k[j-1] + \overline{a_0 a_i}$
  18:   **end if**
  19:   $\chi = \chi - \{P_j\}, j = j + 1$
  20: **end while**
  21: $R[0][i] = R[0][0]$ /*the traversal ends at $s_0.end = l$ */
  22: connect the neighboring points piecewise by lines
  23: select the optimal $s_0.end$ with the minimum $R[0]$

At the end of an iteration, all nodes' labels are removed and a new iteration starts with the criteria reversed, i.e., "smallest" becomes "largest" and vice versa. Therefore, the iterations alternate between the increasing and decreasing order of weight in executing the MIS algorithm. The criteria
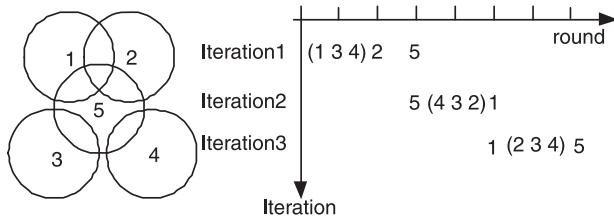
Fig. 4. An example to illustrate the POP protocol.

are reversed to facilitate the distributed operation, so that the nodes belonging to the MIS in the last round of previous iteration can start a new iteration.

Algorithm 3. Parallel Optimization Algorithm

Input: a graph $G(N, E)$

Output: the local optimal schedule of each node,
 i.e., $s_i.end, i = 1 \ldots n$

Procedure:

1: each node generates a random schedule independently
2: discover neighbors and exchange the schedule with each other
3: initialize $improve = Threshold, criteria = smallest,$
 $weight = id$
4: **while** $improve \geq Threshold$ **do**
5:    unlabel all the nodes /*start a new iteration*/
6:    **while** there are still nodes unlabeled **do**
7:      /*start a new round*/
8:      run distributed algorithm $MIS(weight, criteria)$
9:      run local optimization algorithm (i.e., line traversal algorithm) for each node of MIS, record $improve$
10:    **end while**
11:    **if** $criteria == smallest$ **then**
12:      $criteria = largest$
13:    **else**
14:      $criteria = smallest$
15:    **end if**
16: **end while**

Fig. 4 shows an example. In the first round, after $MIS(id, smallest)$ is executed, nodes s_1, s_3, s_4 which have the smallest id among its neighbors are selected to form an MIS and optimize their schedules simultaneously without conflict. Then, the MIS algorithm is executed for two more rounds among the remaining nodes, during which the MIS obtained in the second and third round consists of s_2 and s_5, respectively. So far all the nodes are labeled, so the first iteration ends. After that, nodes are unlabeled again and the second iteration starts. The algorithm $MIS(id, largest)$ is executed with the criteria reversed, so that s_5 (with the largest id among its neighbors) can initiate the second iteration. Similar to the first iteration, three rounds are needed in both the second and the third iteration. Note that the last round of the previous iteration coincides with the first round of the current iteration because their respective MIS is the same and there is no need to optimize the schedule of the same MIS twice. Overall seven rounds are needed for nodes to adjust their schedules in three iterations. This is much faster than the serial algorithm which needs $5 \times 3 = 15$ rounds.

It is straightforward to make Algorithm 2 distributed. Since both the MIS algorithm and the local optimization algorithm are distributed, the issue here is to let each node know when to elect itself to the MIS, when to start a new iteration with the reversed criteria, and when to terminate. To achieve this, we define a control packet in the format of $msg(id, criteria, schedule)$. After a node elects itself as belonging to the MIS and adjusts its schedule, it broadcasts $msg(id, criteria, schedule)$ to its neighbors. If the criteria are the "smallest" (or "largest"), the neighbors with the larger (or smaller) id will have the sender labeled, and check the sender's schedule to see whether it has changed. After an MIS is elected, all the nodes in the MIS will be labeled by their neighbors. Therefore, at least one unlabeled node's id will become the smallest (or largest) among the remaining unlabeled neighbors, and thus, eligible to adjust its own schedule.

To start a new iteration, the criteria need to be reversed. If an elected node finds itself to be the last node among its neighbors to be elected, it will realize that it is its responsibility to reverse the criteria, and start a new iteration by broadcasting an updated message. When other nodes receive the message with the reversed criteria for the first time, they will realize that a new iteration starts, so the labels of their neighbors are reset. A timer is set to control the termination of the algorithm at each node. If the node cannot improve its schedule beyond the predefined threshold δ after a few more iterations, it will exist and start using the calculated schedule.

The message complexity of the POP protocol is $O(n)$, which grows linearly with the number of nodes. This is because each node in each iteration broadcasts two messages: one is to exchange the id, *criteria*, and *schedule* in the beginning, while the other is to announce its labeled status after being selected to the MIS. Therefore, the message complexity of each node is $O(2T)$, where T is the number of iterations required for the protocol to terminate. According to our experiments in Section 5, T is a small constant with the typical parameter setting, e.g., $T \leq 5$ when $\delta = 1, n \leq 500$ and battery/network lifetime ratio is $\frac{1}{5}$.

4.4 Discussions and Future Work

In this paper, we assume that the disk sensing model is used, where the sensing range is modeled by a disk and a point is covered if and only if it falls within the sensing disk of one of the sensors. While the disk model provides valuable high-level guidelines, it may not accurately reflect the performance in reality. Recently, some researchers have started to investigate the impact of link irregularity and the corresponding nondisk model on the performance of the sensor networks [10], [11], [12], [13]. For example, the work in [12] employs an empirical approach to estimate the sensing range. A probability model is used in [13] to depict the coverage property of the sensor network, where the coverage probability of a point depends on the distance from the monitoring sensors.

To adapt the POP protocol to the nondisk model, we can leave the big framework intact but change the method to calculate the local coverage redundancy. The algorithm still executes in iterations, but during each iteration, each node calculates the pair-wise coverage redundancy based on the
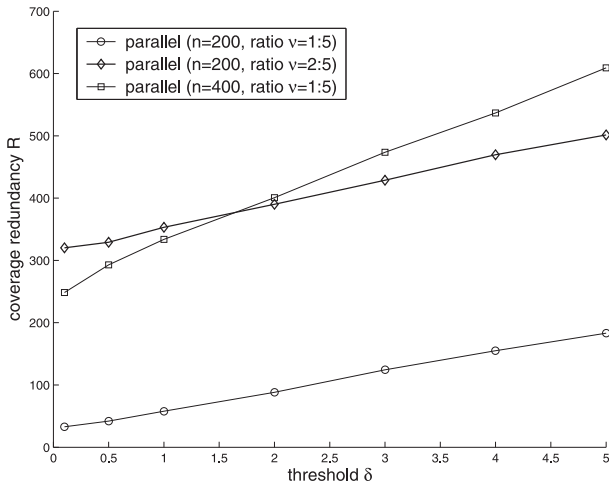
Fig. 5. Relationship between the coverage redundancy and δ (homogeneous).

Fig. 6. Relationship between the convergence time and δ (homogeneous).

specific nondisk model. Taking the probability model as an example, the local coverage redundancy of node s_0 can be calculated by $R[0] = \sum_{i \in N(0)} \int_{\theta=0}^{2\pi} \int_{\rho=0}^{w} P(\rho, \theta) \times \overline{s_0 s_i}$, as compared with $R[0] = \sum_{i \in N(0)} \overline{a_0 a_i} \times \overline{s_0 s_i}$ in the disk model. The new calculation is based on the polar coordinate system, with the middle point of the line connecting the pair-wise neighboring sensors as the pole. In particular, $P(\rho, \theta)$ is the coverage probability calculated based on the specific model,

$$w = \frac{\sqrt{4r^2 - d^2 \sin \theta^2} - d \cos \theta}{2}$$

and $\overline{s_0 s_i}$ follows (19). In general, extension of the coverage property to the nondisk model is still an open issue in many situations. We leave the complete design and evaluation to the future work.

Another issue worth of further investigation is the connectivity property of the sensor network. Although in this paper, we consider network lifetime as a constraint and connectivity is not our focus, achieving continuous connectivity is still valuable for the data delivery. It has been proved that when the communication range is at least twice the sensing range, the full coverage implies the connectivity of the sensor network [4]. However, in our paper, we study the scenario where the sensors may not be sufficient enough to sustain both coverage and lifetime, so sometimes, coverage has to be traded for lifetime, resulting in the partial coverage. As far as we know, the condition under which the connectivity can be achieved in the partially covered sensor network is still an open issue. Although we did not solve it in this paper, we point out that this is an interesting issue for the future research and have proposed a remedy solution in our previous work [14]. In [14], we design a new set of routing protocols for the data delivery over the intermittently connected network. In an intermittently connected network, the network may not be physically connected at all instants, but the data can still be delivered to the destination in a store-and-forward fashion.

5 PERFORMANCE EVALUATIONS

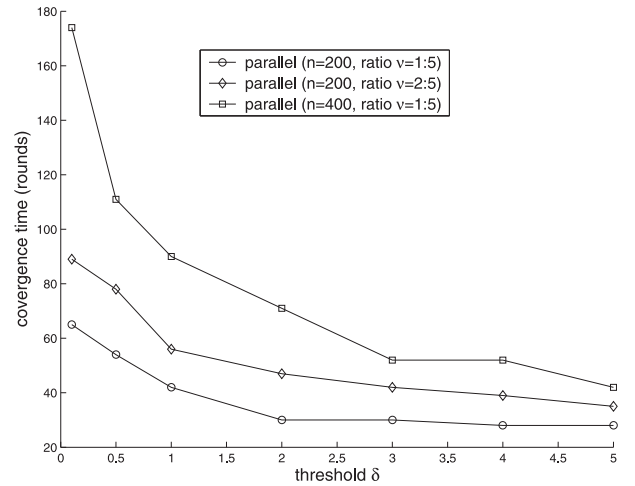In this section, we evaluate the performance of the proposed POP protocol. In the simulation, n sensors are randomly deployed in a 10×10 square area, with n varying from 100 to 500. The sensing range is 1 unless, otherwise, specified. We specifically examine the scenario where the coverage and lifetime requirement cannot be satisfied at the same time. For example, when $n = 500$ and the battery/network lifetime ratio is $\frac{2}{5}$, the full coverage cannot be maintained throughout the network lifetime using the algorithm in [4]. Both homogeneous and heterogeneous battery states are considered. In the homogeneous case, every node has the same battery/network lifetime ratio ν, but in the heterogeneous case, ν_i is a random variable uniformly distributed in $[\nu/2, 3\nu/2]$ with ν as the average ratio. The experiments are done over a customized C++ simulator.

Three schemes are evaluated, namely, random, serial, and POP, in terms of coverage redundancy, convergence time, and event detection probability. As the global coverage/coverage redundancy is infeasible to compute, we use the sum of local coverage redundancy as an approximation. The randomized event is considered whose location of occurrence is uniformly distributed in time and space, and whose length of occurrence e is normalized as the event/cycle ratio, i.e., $\frac{e}{l}$. The event detection probability is calculated by simulating 1,000 randomized events.

To compare with the existing schemes, we implement an extended version of the Coverage Configuration Protocol (CCP), which is shown to outperform other schemes in most of the scenarios [4]. While the objective of the original CCP is to select the minimum number of sensors to provide the full coverage, we extended it to a continuously operational case where the sensor node may die of limited battery. After a sensor dies, each sleeping sensor needs to decide whether it should be activated to remedy the coverage hole based on the eligibility rule in [4]. We evaluate CCP in terms of coverage redundancy and network lifetime. The network lifetime is defined as the period during which half of the nodes fail.

5.1 Determine the Optimization Threshold δ

δ is the threshold of improvement made at each step. It determines how accurate the algorithm can approach the local optimality and how fast the algorithm can converge.
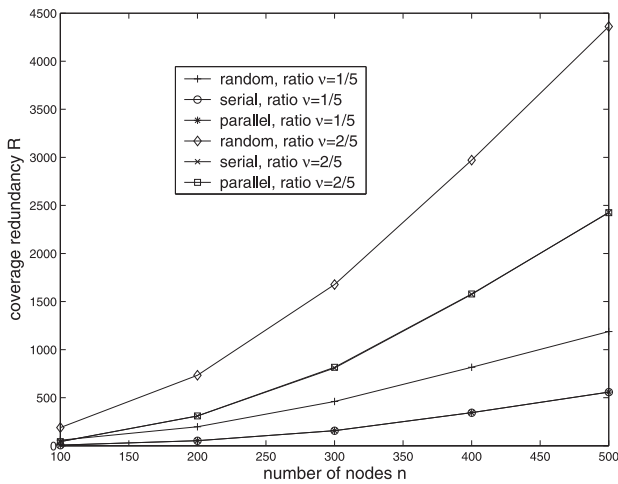
Fig. 7. Comparison of coverage redundancy (homogeneous).

Fig. 9. Comparison of event detection probability (homogeneous).

From Figs. 5 and 6, it can be seen that δ affects the coverage redundancy and the convergence time in different ways. As δ increases, the redundancy will rise but the convergence time goes down. In other words, the objectives of redundancy and convergence time conflict with each other from the perspective of δ. To make the coverage redundancy better, a smaller δ should be used, but to improve the convergence time, a larger δ should be employed. To balance coverage redundancy and convergence time, we set δ to be 1 in the following experiments.

5.2 Comparing POP with Other Schemes

Figs. 7 and 8 show that the number of nodes n and the on-period/network lifetime ratio ν affect the system performance in a similar way. Both coverage redundancy and convergence time increase as more sensors are deployed or as larger on-periods are used. In terms of redundancy, serial and POP have similar performance, and both outperform the random algorithm substantially. The improvement gradually decreases as the number of nodes increases. For instance, the performance improvement is more than 100 percent when $n = 200$ but reduces to about 80 percent when $n = 400$. This is because as more sensors are deployed, it is more likely
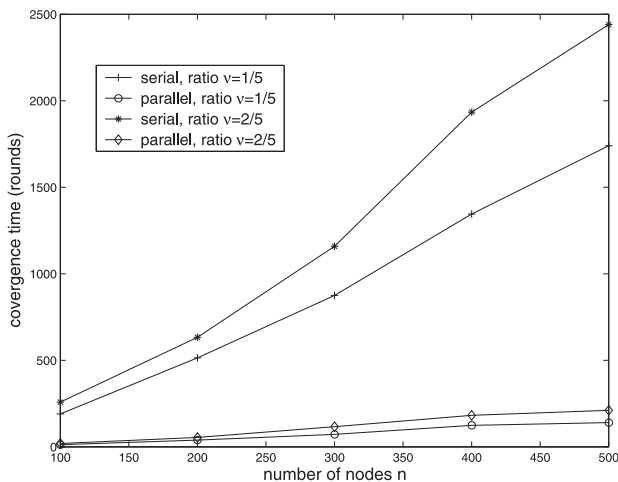
that the random algorithm can produce a relatively good schedule. In terms of convergence time, POP is much faster than the serial algorithm because parallel optimizations can take place at the same time. As shown in Fig. 8, irrespective of the number of nodes, the convergence time of POP is only $\frac{1}{10}$ of the serial algorithm.

Figs. 9 and 10 compare the different schemes in terms of event detection probability. In Fig. 9, the x-axis corresponds to the number of nodes, and in Fig. 10, the x-axis corresponds to the event/cycle ratio. It can be observed that for all the schemes, the detection probability increases as the number of nodes increases. The initial detection probability is below 50 percent when $n = 100$ but gradually approaches 1 as n increases to 500. Another observation is that the detection probability also increases as the length of event increases. This can be understood since the longer the event persists, the easier it should be detected. Among the different schemes, the serial algorithm and POP have almost the same performance, both of which outperform the
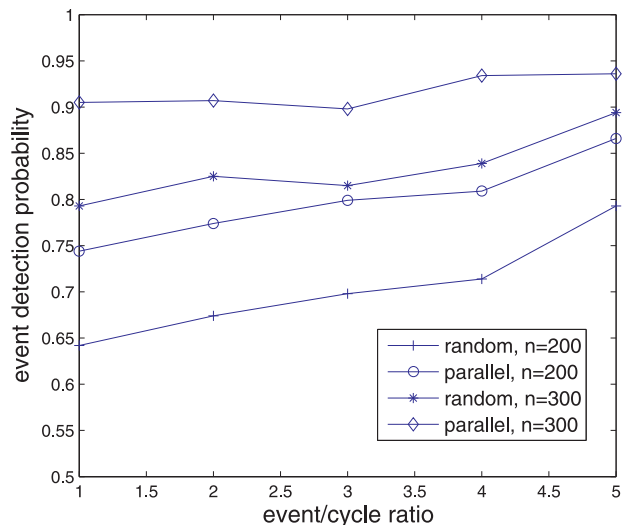

Fig. 8. Comparison of convergence time (homogeneous, $\nu = \frac{1}{5}$).
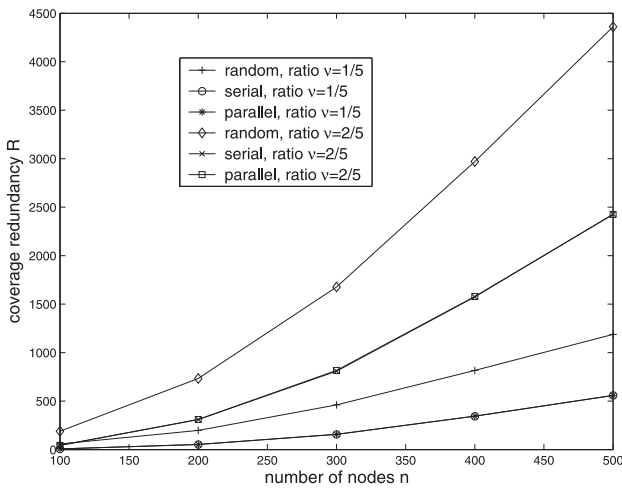

Fig. 10. Comparison of event detection probability (homogeneous).

Fig. 11. Comparison of coverage redundancy (heterogeneous).

Fig. 13. Comparison of network lifetime with CCP (heterogeneous, $\nu = \frac{2}{5}$).

random algorithm. The improvement is about 15 percent in terms of event detection probability.

Figs. 7 and 8 study the performance of homogeneous cases. The same trend exists in the heterogeneous case as shown in Figs. 11 and 12.

Figs. 13 and 14 compare the proposed schemes with CCP. In Fig. 13, it is shown that CCP is not able to meet the lifetime requirement in the scenarios simulated. However, as more sensors are deployed, it can support a longer network lifetime. For example, when $n = 600$, it maintains almost 90 percent of the required lifetime with the full coverage. By contrast, the randomized algorithm and POP divide network lifetime into cycles and within each cycle, the locations of their "on" periods are different. Therefore, both the algorithms bear the lifetime constraint in mind and can satisfy the lifetime requirement regardless of the node density, as seen in Fig. 13. In Fig. 14, it is shown that CCP's coverage redundancy is consistently larger than that of POP. Compared with the randomized algorithm, the performance of POP is close to the randomized algorithm when the number of sensors is small, but substantially improves as more sensors are deployed. This is because when sensor node density becomes larger, the network lifetime will
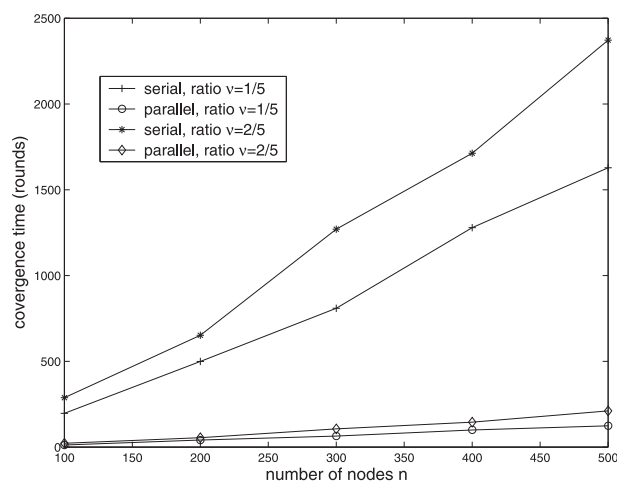
increase (as shown in Fig. 13), then the coverage redundancy will reduce as a result.

6   RELATED WORK

The sensor coverage problem has been extensively studied in the literature. Depending on the subject covered, most existing works can be classified into area coverage, point coverage, and barrier coverage [2]. In terms of area coverage, many works focus on how to select the minimum number of sensors to preserve the coverage degree (e.g., 1-degree or k-degree) [3], [4], [5], [6], [15], [16], [17], but they provide no network lifetime guarantee. In [18], a centralized scheduling algorithm is proposed to sequentially activate the sensor cover and guarantees a $O(log n)$ factor of the maximum network lifetime, where n is the total number of nodes. Further, in [19], a distributed scheduling algorithm is proposed which achieves a $O(log n * log n B)$ performance factor, where B is the upper bound of the initial battery. Besides coverage requirement, the connectivity property also attracts lots of research attention. For example, when the coverage requirement can be satisfied, the conditions to

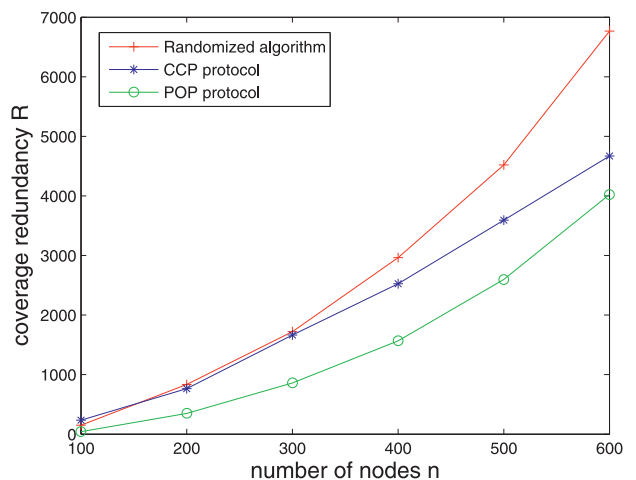Fig. 12. Comparison of convergence time (heterogeneous).

Fig. 14. Comparison of coverage redundancy with CCP (heterogeneous, $\nu = \frac{2}{5}$).

achieve the communication connectivity have been derived in [4], [20]. When the coverage requirement cannot be satisfied all the time, e.g., in presence of partial coverage, there are routing protocols proposed in [14] to ensure the delivery of the data to the sink in the store-and-forward fashion over the intermittent link.

The model of point coverage is studied in [21], [1], with the objective to maximize the network lifetime when only a given set of targets needs to be covered. The model is NP-hard, in general, so some greedy heuristics are proposed based on the linear programming relaxation [21]. But with the assumption that one sensor covers only one target a time, the optimal schedule can be derived based on the technique of matrix decomposition [1]. In the special case, each target to be covered is the sensor node itself, e.g., in the scenario of network monitoring. In [22], [23], distributed algorithms are proposed to construct the monitoring architecture for sensor networks where sensor nodes can monitor each other within a predefined communication range.

The concept of barrier coverage is first proposed by [24], with the objective to minimize the probability of undetected penetration through the barrier. In [25], a global algorithm is proposed to determine whether a region is k-barrier covered. Although it has been proved that given a sensor deployment, sensors cannot locally determine whether the deployment provides global barrier coverage, a distributed algorithm is proposed based on the concept of local barrier coverage [26] assuming that the intruders move along restricted crossing paths in rectangular areas. Later, the restriction on crossing paths is removed in [27], [28], where the barrier construction algorithm is proposed when the sensors are deployed according to a Poisson point process [27] or along a line [28].

While all of the above works treat the lifetime as objective, we consider the network lifetime as a constraint and aim to schedule each sensor's on-period to minimize the total spatial-temporal coverage redundancy. This reverse formulation is especially useful for mission-driven sensor networks, where the network lifetime may have higher priority over coverage and the predeployed resources may not meet the changing mission requirements all the time. Thus, our work is complementary to the existing works, which can be applied when the sensor node density is sufficient to provide the preferred coverage degree for a specified length of time.

There are other application-driven scheduling algorithms, e.g., for minimum latency routing [29], [30], [31], target tracking [32], [33], [34], [35], event detection [36], and throughput optimization [37]. All these works support only a single mission and do not treat network lifetime as the objective or constraint. By contrast, our objective is to maximize the spatial-temporal coverage with the network lifetime as the constraint. Our model is different from the traditional maximum coverage problem in the spatial domain, which is known to have a $(1 - 1/e)$-approximation bound [8]. This is because in the spatial-temporal domain, we not only need to select sensors but also need to determine their corresponding schedules in a continuous cycle. Therefore, the $(1 - 1/e)$ ratio cannot be applied here. By contrast, we model the spatial-temporal coverage as a vector and propose a $\frac{1}{2}$-approximation algorithm for our problem.

7 CONCLUSIONS

As mission-driven sensor networks usually have stringent lifetime requirement, sometimes coverage has to be traded for network lifetime. In this paper, we studied how to schedule sensor active time to maximize the spatial-temporal coverage while meeting the lifetime constraint. While the optimization of the global objective is NP-hard, we have proposed both centralized and distributed algorithms with low complexity. It was proved that the centralized algorithm has an approximation ratio of $\frac{1}{2}$, and the distributed parallel optimization protocol can ensure each node to converge to local optimality without conflict with each other. The computational complexity of POP is only $O(d)$ per node, where d is the maximum node degree, and its message complexity is $O(n)$, which is linear with the number of nodes. Theoretical and simulation results showed that POP substantially outperforms other schemes in terms of coverage redundancy, convergence time, network lifetime, and event detection probability.

REFERENCES

[1] H. Liu, P. Wan, C.-W. Yi, X. Jia, S. Makki, and P. Niki, "Maximal Lifetime Scheduling in Sensor Surveillance Networks," *Proc. IEEE INFOCOM,* Mar. 2005.

[2] M. Cardei and J. Wu, "Energy-Efficient Coverage Problems in Wireless Ad Hoc Sensor Networks," *J. Computer Comm. Sensor Networks,* vol. 29, pp. 413-420, 2005.

[3] D. Tian and N. Georganas, "A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks," *Proc. ACM Int'l Workshop Wireless Sensor Networks and Applications,* 2002.

[4] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys),* 2003.

[5] T. Yan, T. He, and J.A. Stankovic, "Differentiated Surveillance for Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys),* 2003.

[6] H. Zhang and J.C. Hou, "Maintaining Sensing Coverage and Connectivity in Large Sensor Networks," *Wireless Ad Hoc and Sensor Networks,* vol. 1, pp. 89-124, 2005.

[7] M.H. Alsuwaiyel, *Algorithms Design Techniques and Analysis.* World Scientific, 1999.

[8] S. Khuller, A. Moss, and J. Naor, "The Budgeted Maximum Coverage Problem," *Information Processing Letters,* vol. 70, no. 1, pp. 39-45, 1999.

[9] S. Basagni, "A Distributed Algorithm for Finding a Maximal Weighted Independent Set in Wireless Networks," *Proc. 11th Int'l Conf. Parallel and Distributed Computing and Systems (PDCS),* 1999.

[10] G. Xing, R. Tan, B. Liu, J. Wang, X. Jia, and C.-W. Yi, "Data Fusion Improves the Coverage of Wireless Sensor Networks," *Proc. ACM MobiCom,* 2009.

[11] C. Liu and G. Cao, "Minimizing the Cost of Mine Selection via Sensor Networks," *Proc. IEEE INFOCOM,* 2009.

[12] J. Hwang, T. He, and Y. Kim, "Exploring In-Situ Sensing Irregularity in Wireless Sensor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 21, no. 4, pp. 547-561, Apr. 2010.

[13] M. Hefeeda and H. Ahmadi, "An Integrated Protocol for Maintaining Connectivity and Coverage under Probabilistic Models for Wireless Sensor Networks," *Ad Hoc and Sensor Wireless Networks,* vol. 7, pp. 295-323, 2009.

[14] L. Su, C. Liu, H. Song, and G. Cao, "Routing in Intermittently Connected Sensor Networks," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* 2008.

[15] Y. Zou and K. Chakrabarty, "A Distributed Coverage and Connectivity Centric Technique for Selecting Active Nodes in Wireless Sensor Networks," *IEEE Trans. Computers,* vol. 54, no. 8, pp. 978-991, Aug. 2005.

[16] S. Kumar, T.H. Lai, and J. Balogh, "On K-Coverage in a Mostly Sleeping Sensor Network," *Proc. ACM MobiCom,* 2004.

[17] S. Funkey, A. Kesselman, F. Kuhn, and Z. Lotker, "Improved Approximation Algorithms for Connected Sensor Cover," *Wireless Networks,* vol. 13, pp. 153-164, 2007.

[18] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsly, "Efficient Energy Management in Sensor Networks," *Ad Hoc and Sensor Networks,* 2005.

[19] G.S. Kasbekar, Y. Bejerano, and S. Sarkar, "Lifetime and Coverage Guarantees through Distributed Coordinate-Free Sensor Activation," *Proc. ACM MobiCom,* 2009.

[20] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T.H. Lai, "Deploying Wireless Sensors to Achieve both Coverage and Connectivity," *Proc. ACM MobiHoc,* 2006.

[21] M. Cardei, M. Thai, Y. Li, and J. Wu, "Energy-Efficient Target Coverage in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2005.

[22] C. Liu and G. Cao, "An Multi-Poller Based Energy-Efficient Monitoring Scheme for Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2009.

[23] C. Liu and G. Cao, "Distributed Monitoring and Aggregation in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2010.

[24] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M.B. Srivastava, "Coverage Problems in Wireless Ad-Hoc Sensor Networks," *Proc. IEEE INFOCOM,* 2001.

[25] S. Kumar, T.H. Lai, and A. Arora, "Barrier Coverage with Wireless Sensors," *Proc. ACM MobiCom,* 2005.

[26] A. Chen, S. Kumar, and T.H. Lai, "Designing Localized Algorithms for Barrier Coverage," *Proc. ACM MobiCom,* 2007.

[27] B. Liu, O. Dousse, J. Wang, and A. Saipulla, "Strong Barrier Coverage of Wireless Sensor Networks," *Proc. ACM MobiHoc,* 2008.

[28] A. Saipulla, C. Westphal, B. Liu, and J. Wang, "Barrier Coverage of Line-Based Deployed Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2009.

[29] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay Efficient Sleep Scheduling in Wireless Sensor Networks," *Proc. IEEE INFOCOM,* 2005.

[30] R. Cohen and B. Kapchits, "An Optimal Algorithm for Minimizing Energy Consumption while Limiting Maximum Delay in a Mesh Sensor Network," *Proc. IEEE INFOCOM,* 2007.

[31] A. Keshavarzian, H. Lee, and L. Venkatraman, "Wakeup Scheduling in Wireless Sensor Networks," *Proc. ACM MobiHoc,* 2006.

[32] S. Ren, Q. Li, H. Wang, X. Chen, and X. Zhang, "Design and Analysis of Sensing Scheduling Algorithms under Partial Coverage for Object Detection in Sensor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 3, pp. 334-350, Mar. 2007.

[33] C. Gui and P. Mohapatra, "Power Conservation and Quality of Surveillance in Target Tracking Sensor Networks," *Proc. ACM MobiCom,* 2004.

[34] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Trans. Wireless Comm.,* vol. 3, no. 5, pp. 1689-1701, Sept. 2004.

[35] W. Zhang and G. Cao, "Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks," *Proc. IEEE INFOCOM,* 2004.

[36] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards Optimal Sleep Scheduling in Sensor Networks for Rare-Event Detection," *Proc. ACM/IEEE Int'l Symp. Information Processing in Sensor Networks (IPSN),* 2005.

[37] C. Joo, "A Local Greedy Scheduling Scheme with Provable Performance Guarantee," *Proc. ACM MobiHoc,* 2008.
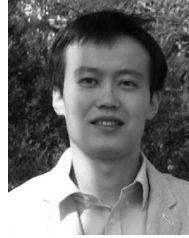
Changlei Liu received the BE degree from the University of Science and Technology of China and the MPhil degree from the University of Hong Kong, both in electronic engineering. He is working toward the PhD degree in the Department of Computer Science and Engineering, Pennsylvania State University. His research interests include the areas of wireless sensor networks, mobile computing, and distributed systems. He is a student member of the IEEE.

Guohong Cao received the BS degree from Xian Jiaotong University, China, and the MS and PhD degrees in computer science from Ohio State University in 1997 and 1999, respectively. Since then, he has been with the Department of Computer Science and Engineering at Pennsylvania State University, where he is currently a professor. His research interests include wireless networks and mobile computing. He has published more than 100 papers in the areas of sensor networks, wireless network security, data dissemination, resource management, and distributed fault-tolerant computing. He has served on the editorial board of the *IEEE Transactions on Mobile Computing* and the *IEEE Transactions on Wireless Communications*, as well as on the program committees of many conferences. He was the recipient of the US National Science Foundation CAREER Award in 2001. He is a fellow of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.