7-29-2009

A Distributed Algorithmic Framework for Coverage Problems in Wireless Sensor Networks

Akshaye Dhawan
Ursinus College, adhawan@ursinus.edu

Sushil K. Prasad

Recommended Citation

Akshaye Dhawan and Sushil K. Prasad. A distributed algorithmic framework for coverage problems in wireless sensor networks. *The International Journal of Parallel, Emergent and Distributed Systems*, 24 (4): 331-348, 2009.

The International Journal of Parallel, Emergent and Distributed Systems
Vol. 00, No. 00, January 2008, 1–17

RESEARCH ARTICLE

A Distributed Algorithmic Framework for Coverage Problems in Wireless Sensor Networks

Akshaye Dhawan* and Sushil K. Prasad

Department of Computer Science, Georgia State University, Atlanta, Ga 30303

(*released June 2008*)

One of the key challenges in Wireless Sensor Networks (WSNs) is that of extending the lifetime of the network while meeting some coverage requirements. In this paper, we present a distributed algorithmic framework to enable sensors to determine their sleep-sense cycles based on specific coverage goals. The framework is based on our earlier work on the target coverage problem. We give a general version of the framework that can be used to solve network/graph optimization problems for which melding compatible neighboring local solutions directly yields globally feasible solutions such as the maximal independent set problem. We also apply this framework to several variations of the coverage problem, namely, target coverage, area coverage and k-coverage problems, to demonstrate its general applicability. Each sensor constructs minimal cover sets for its local coverage objective. The framework entails each sensor prioritizing these local cover sets and then negotiating with its neighbors for satisfying mutual constraints. We employ a dependency graph model that can capture the interdependencies among the cover sets. Detailed simulations are carried out to further demonstrate the resulting performance improvements and effectiveness of the framework. These show an improvement of between 10%-20% over existing algorithms.

Keywords: wireless sensor networks, scheduling, energy efficiency, maximum lifetime, distributed algorithms, localized algorithms, maximum independent set

1. Introduction

Wireless sensor networks (WSNs) have attracted a lot of recent research interest due to their applicability in security, monitoring, disaster relief and environmental applications. WSNs consist of a number of low-cost sensors scattered in a geographical area of interest and connected by a radio interface. Sensors gather information about the monitored area and send this information to gateway nodes [2].

In order to keep their cost low, the sensors are equipped with limited energy and computational resources. The energy supply is typically in the form of a battery and once the battery is exhausted, the sensor is considered to be dead. To ensure that the area or targets of interest can be covered, sensors are usually deployed in large numbers by randomly dropping them in a region [20]. Such a deployment scheme gives rise to an overlap in the monitoring regions of individual sensors.

A simplistic approach to meeting the coverage objective would be to turn on all sensors after deployment. But this needlessly reduces the *lifetime* of the network since the overlap between monitoring regions implies that not all sensors need to be on at the same time. In order to extend the lifetime of a sensor network while maintaining coverage, a minimal subset of the deployed sensors are kept active

*Corresponding author. Email: akshaye@cs.gsu.edu

ISSN: 1744-5760 print/ISSN 1744-5779 online

DOI: 10.1080/1744576YYxxxxxxxx
http://www.informaworld.com

while the other sensors can enter a low power sleep state. The ratio of the energy consumed in the active versus sleep states can be as much as several orders of magnitude [22]. In order to share the load, through some form of scheduling, this active subset changes over time until there are no more such subsets available to satisfy the coverage goal. In using such a scheme to extend the lifetime, the problem is two fold. First, we need to select these minimal subsets of sensors. Then there is the problem of *scheduling* them wherein, we need to determine how long to use a given set and which set to use next. For an arbitrarily large network, there are exponential number of possible subsets making the problem intractable and it has been shown to be NP-complete in [8, 13].

Existing centralized and distributed heuristics for arriving at a lifetime extending schedule are discussed in Section 2. Centralized solutions like those in [8, 21] are based on assuming that the entire network structure is known at one node (typically the gateway node), which then computes the schedule for the network. The schedule is often computed using *linear programming* based algorithms. Like any centralized scheme, it suffers from the problems of scalability, single point of failure and lack of robustness. The later is particularly relevant in the context of sensor networks since sensor nodes are deployed in hostile environments and are liable to failure. Existing distributed solutions in [4, 5, 23] work by having a sensor exchange information with its neighbors (limited to k-hops, where k is usually 1 or 2). These algorithms use information like targets covered and battery available at each sensor to greedily decide which sensors remain on. This happens in rounds so the set of active sensors is periodically reshuffled. The problem with these algorithms is that they use simple greedy criteria to make their decision and do not efficiently take into account the problem structure.

In [18] we focused on the target coverage problem and presented distributed algorithms for scheduling the sensors to extend the lifetime. We used the idea of constructing *local* cover sets consisting of sensors that can cover local targets. Since certain cover sets are *better* than others, we presented a *Lifetime Dependency Graph* model that enabled us to use some properties of this graph in order to prioritize these covers. We also showed how other existing algorithms like [4, 5] can be modeled by variations in the prioritizing scheme. We carried out simulations to show improvements of 10-20% over LBP [4] and similar performance to DEEPS [5] which is a 2-hop algorithm. A 2-hop version of our algorithm outperformed DEEPS [5].

Our Contributions [1]**:** We realize that the scheme of creating local solutions and modeling their dependencies applies to various other problems besides target coverage. In general, for certain graph and network problems where solving the problem locally implies that a globally feasible solution can be reached, our framework can be used (See Section 3). In this paper we use the solution in [18] to develop a framework and show its application to the area coverage and k-coverage problems besides showing how the target coverage solution fits into this generalized framework. We also illustrate the use of the framework in a more general context by applying it to the problem of finding an independent set in a distributed environment. The key points of the framework include construction of local solutions, modeling the dependency between the local solutions using a *dependency* graph, prioritizing the interdependent local solutions using a *priority* function that can utilize the dependency graph and negotiating with neighbors to arrive at a mutually satisfactory local solution.

[1]This paper is an extended version of [12]. We generalize the concepts introduced in [18].

Our overall framework is a two phase distributed meta-algorithm. The first phase is the setup phase for each node to construct prioritized local solutions. The second phase is the rounds of negotiation phase during which each node chooses its best local solution compatible with its neighbors. Employing the framework entails the following:

- verifying the applicability of the framework,
- modeling the state space of local solutions and their interdependencies using a dependency graph structure,
- heuristically modeling the priority of local solutions based on the properties of the dependency graph structure, and
- determining the logistics of negotiating with neighbors to settle on mutually-compatible and high-priority local solutions.

The remainder of this paper is organized as follows. In section 3 we introduce the generic framework, its steps and the problems to which it applies. In section 4 we develop a coverage-problem specific framework. Section 5 discusses the solution of the three different coverage problems area, target and k-coverage within the developed framework for coverage problems. In Section 6 we evaluate our algorithms against those of [4, 5]. Section 2 surveys previous work on the lifetime for coverage problem. Finally, we conclude in Section 7.

2. Background Literature

2.1 Problem Statement

The lifetime problem can be stated as follows. Given a monitored region R, a set of sensors S and a set of targets T, find a monitoring schedule for these sensors such that

- the total time of the schedule is maximized,
- all targets are constantly monitored, and
- no sensor is in the schedule for longer than its initially battery.

2.2 Related Work

In this section, we briefly survey existing approaches to maximizing the lifetime of sensor networks, while meeting certain coverage objectives. [7] gives a more detailed survey on the various coverage problems and the scheduling mechanisms they use. [19] also surveys the coverage problem along with other algorithmic problems relevant to sensor networks. We end this section by focusing on two algorithms, LBP [4] and DEEPS [5], since we use them for comparisons against our algorithms in Section 6.

The maximum lifetime coverage problem has been shown to be NP-complete in [1, 8]. Initial approaches to the problem in [1, 8, 21] considered the problem of finding the maximum number of *disjoint* cover sets of sensors. This allowed each cover to be used independently of others. However, [3, 6] and others showed that using non-disjoint covers allows the lifetime to be extended further and this approach has been adopted since.

Broadly speaking, the existing work in this category can be classified into two parts - Centralized Algorithms and Distributed Algorithms. For centralized approaches, the assumption is that a single node (usually the base station) has access to the entire network information and can use this to compute a schedule that is

Table 1. Centralized Algorithms

Name	Area/Target	Disjoint	Main Idea
Abrams, Goel [1]	Area	Yes	Greedy: Max uncovered area
Meguerdichian [17]	Area	No	Integer Linear Program
Cardei [8]	Target	Yes	Mixed Integer Programming
Shah [3]	Area	Yes	LP, Garg Könemann
Cardei [6]	Target	No	Integer Linear Program

Table 2. Distributed Algorithms

Name	Area/Target	Disjoint	Main Idea
Sliepcivic [21]	Area	Yes	Greedy: Max uncovered fields
Tian [23]	Area	No	Geometric calculation of sponsored area
PEAS [25]	Area	No	Probing based determination of sponsored area
CCP [24]	Area	No	Random timers to evaluate coverage requirements
OGDC [26]	Area	No	Random back off node volunteering
Lu [16]	Area	No	Highest overall denomination sensor picks
Abrams [1]	Area	Yes	Randomized, Greedy picks max uncovered area
Cardei et al. [6]	Target	No	Sensor with highest contribution to bottleneck
LBP [4]	Target	No	Targets are covered by higher energy nodes
DEEPS [5]	Target	No	Minimize energy consumption for bottleneck target

then uploaded to individual nodes. Distributed Algorithms work on the premise that a sensor can exchange information with its neighbors within a fixed number of hops and use this to make scheduling decisions. We now look at the individual algorithms in both these areas.

A common approach taken with centralized algorithms is that of formulating the problem as an optimization problem and using linear programming (LP) to solve it [3, 8, 11, 17]. [17] formulates the area coverage problem using a Integer LP and relax it to obtain a solution. In order to solve the target coverage problem, [8] considers the disjoint cover set approach. Modeling their solution as a Mixed Integer Program shows an improvement over [21]. [3] formulates a packing LP for the coverage problem. Using the $(1 + \epsilon)$ Garg-Könemann approximation algorithm [14], they provide a $(1 + \epsilon)(1 + 2lnn)$ approximation of the problem. A similar problem is solved by us for sensors with *adjustable* ranges in [11]. A different algorithm to work with disjoint sets is given in [9]. Disjoint cover sets are constructed using a graph coloring based algorithm that has area coverage lapses of about 5%. [1] also gives a centralized greedy algorithm that picks sensors based the largest uncovered area.

The distributed algorithms in the literature can be further classified into greedy, randomized and other techniques. The greedy algorithms [1, 4–6, 16, 21] all share the common property of picking the set of active sensors greedily based on some criteria. [21] considers the area coverage problem and introduces the notion of a *field* as the set of points that are covered by the same set of sensors. The basic approach behind the picking of a sensor is to first pick the one that covers that largest number of previously uncovered fields and to then avoid including more than one sensor that covers a sparsely covered field. [1] builds on this work and presents three algorithms that solve variations of the set k-cover problem. The greedy heuristic they propose works by selecting the sensor that covers the largest uncovered area. [16] defines the sensing denomination of a sensor as its contribution, i.e., the area left uncovered when the sensor is removed. Sensors with higher sensing denomination have a higher probability of remaining active.

[3] gives a distributed algorithm based on using the faces of the graph. If all the

faces that a sensor covers are covered by other sensors with higher battery that are in an active or deciding state, then a sensor can switch off (sleep). Their work has been extended to target coverage in the load balancing protocol (LBP).

Some distributed algorithms use randomized techniques. Both OGDC [26] and CCP [24] deal with the problem of integrating coverage and connectivity. They show that if the communication range is at least twice the sensing range, a covered network is also connected. [26] uses a random back off for each node to make nodes volunteer to be the start node. [24] sets a random timer for each node following which a node evaluates its current state based on the coverage by its neighbors. [1] also present a randomized algorithm that assigns a sensor to a cover chosen uniformly at random.

A different approach has been taken in PEAS [23, 25]. PEAS is a distributed algorithm with a probing based off-duty rule is given in [25]. Here, every sensor broadcasts a probe PRB packet with a probing range γ. Any working node that hears this probe packet responds. If a sensor receives at least one reply, it can go to sleep. The range can be chosen based on several criteria. Note that this algorithm does not preserve coverage over the original area. In [23] the authors give a distributed and localized algorithm. Every sensor has an off-duty eligibility rule. They give an algorithm for a node to compute its sponsored area. To prevent the occurrence of blind-points by having two sensors switch off at the same time, a random back off is used. They show improved performance over PEAS.

To our knowledge, [24] was the first work to consider the k-coverage problem. [15] also addresses the k-coverage problem from the perspective of choosing enough sensors to ensure coverage. Authors consider different deployments with sensors given a probability of being active and obtain bounds for deployment. [27] solves the problem of picking minimum size connected k-covers.

Now, we look at the two protocols that we compare our heuristics against. The load balancing protocol (LBP) [4] is a simple 1-hop protocol which works by attempting to balance the load between sensors. Sensors can be in one of three states sense/on, sleep/off or vulnerable/undecided. Initially all sensors are vulnerable and broadcast their battery levels along with information on which targets they cover. Based on this, a sensor decides to switch to off state if its targets are covered by a higher energy sensor in either on or vulnerable state. On the other hand, it remains on if it is the sole sensor covering a target. This is an extension of the work in [3]. LBP is simplistic and attempts to share the load evenly between sensors instead of balancing the energy for sensors covering a specific target.

The other protocol we consider is DEEPS [5]. The maximum duration that a target can be covered is the sum of the batteries of all its nearby sensors that can cover it and is known as the life of a target. The main intuition behind DEEPS is to try to minimize the energy consumption rate around those targets with smaller lives. A sensor thus has several targets with varying lives. A target is defined as a *sink* if it is the shortest-life target for at least one sensor covering that target. Otherwise, it is a *hill*. To guard against leaving a target uncovered during a shuffle, each target is assigned an in-charge sensor. For each sink, its in-charge sensor is the one with the largest battery for which this is the shortest-life target. For a hill target, its in-charge is that neighboring sensor whose shortest-life target has the longest life. An in-charge sensor does not switch off unless its targets are covered by someone. Apart from this, the rules are identical as those in LBP protocol. DEEPS relies on two-hop information to make these decisions.

In [18], we describe how [4] and [5] can be modeled using our target coverage solution (Section 5.1).

3. Our General Framework

Our framework applies to a class of graph and network problems wherein the locally compatible solutions can be melded in a distributed fashion to yield a globally feasible solution. For most sensor coverage problems, local covers when combined together yield global covers, since if all local targets are covered, this implies that globally, all targets are covered.

For such class of problems, even if it is intractable to find globally-optimal solutions, it may be tractable to find locally-optimal solutions, since the problem size is much smaller. For example, all possible covers of local targets of a sensor can be efficiently found for bounded sensing range. These local solutions often are interdependent, i.e., using one may impact a subsequent use of others. For example, in sensor coverage problems, using one cover set may reduce the lifetime of those covers which share sensors with the first. This is problematic if a series of solutions are needed.

In this section we provide an overview of the principles of the generic framework. The details as applied to the coverage problem are given in Section 4. Hence, in the discussion that follows, we describe the four steps in the order in which they are used.

(1) **Applicability of the framework:** The problem being solved must have the property that compatible local solutions of neighbors when combined give a globally feasible solution. Compatibility may be checked by communicating with the neighbors. For example, for the target coverage problem, if for each sensor, all local targets have been covered, this implies that all targets have been covered globally also. Hence, solving the problem locally for every node gives a globally valid solution. This step also establishes criteria for checking mutual compatibility of local solutions.

For an example of a problem that *cannot* be solved using this framework, consider constructing a spanning tree. Here, the local solution would be a spanning tree connecting a sensor to its neighborhood. However, these local trees when combined, may have cycles and do not imply that a global tree is formed. Of course, melding these subtree edges in a reduction-tree fashion rejecting those edges which cause a cycle will yield a global spanning tree [10]. However, that needs much more communication than what can be efficiently done distributively. Hence this property of all local solutions yield a globally feasible solution is imperative for our framework.

(2) **Modeling the local solutions and their interdependencies**: This step starts with modeling the local solutions for the given problem. In some cases it may possible to compute all solutions, whereas for others we would need a representative sample to model the state space. For many problems, these local solutions are not independent of each other. For example, in the target coverage problem, for a given sensor, there can be a number of different subsets of neighbors that cover the local targets being considered. Since these sets are not disjoint, using one set drains the lifetime of another set. To account for this, the framework envisions a graph model to capture these dependencies among the local solutions. Note the specifics of this graph would depend on the problem being considered. We call this a *Dependency Graph* (See Section 4.2). This is a key contribution of our framework. Instead of a simple greedy heuristic to choose the best solution, we consider a solution with relation to its impact on other possible solutions and use this as a criterion to assess a solutions

quality.

(3) **Prioritize the local solutions**: Each node needs to decide which of the possible local solutions to use. Based on the dependency structure of these local solutions and other problem specific metrics, a *priority* function can be defined that measures the quality of a local solution. Including the dependency information of the graph into this heuristic priority function gives us a way to account for the problem structure.

(4) **Negotiate with neighbors for mutually satisfying solutions**: This step involves deciding the details of communication related logistics for the 2-phase execution consisting of setup and negotiation phases. The setup phase is usually a round of information exchange with 1-hop or 2-hop neighbors followed by construction of the local solutions and the dependency graphs structure, and calculation of the priorities of the local solutions. In the negotiation phase, a node communicates with its neighbors and based on both its preference and those of its neighbors, picks a solution to use. Again, the nature of this step would be problem dependent and we explore this with respect to the target coverage problem in the next section. Through its negotiation phase, a node attempts to locally satisfy the twin goals of feasibility and local optimality. Also note that the problem may require several rounds of negotiation to arrive at a mutually acceptable solution. However, for effectiveness and scalability of the resulting distributed algorithm, the number of communication rounds with neighbors needs to be upper-bounded by a small constant.

An Example *Maximum Independent Sets:* To illustrate the use of this framework in a context outside that of the coverage problems for wireless sensor networks, we consider the problem of finding a valid independent set of vertices for a graph in a distributed manner. Given a graph $G = (V, E)$, an independent set of vertices V' is a subset $V' \subseteq V$ such that no two vertices in V' have an edge $e \in E$ that connects them. Let us now briefly develop the proposed framework for this problem and look at an example. We will go into much greater depth in developing the framework for various coverage problems in the following section.

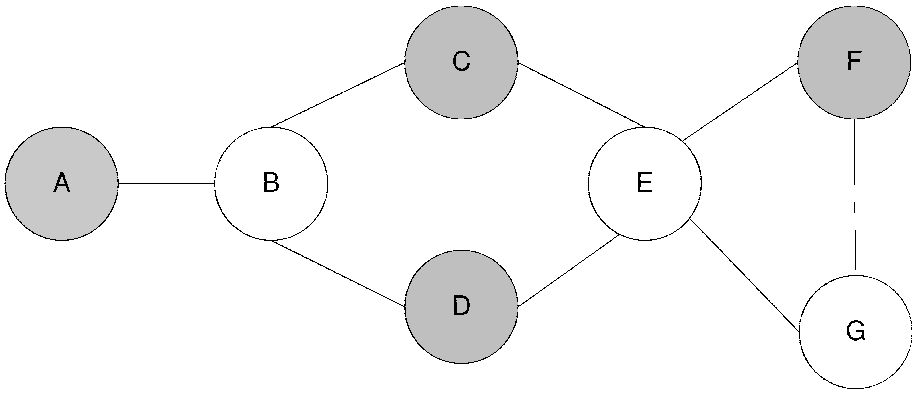

Figure 1. Example Graph for the Independent Set problem

1. Applicability: This problem of finding an independent set can be solved within our framework since every node and its neighbors can decide their local independent sets which collectively yields the global independent set. There will

never be any two nodes $x, y \in V$ that will both decide to be in the independent set if they have an edge $(x, y) \in E$, since they know of the existence of this edge.

2. Modeling the local solutions and their interdependencies: For the independent set problem, we define a local solution for a node to be given by a valid local independent set of all nodes in its closed one hop neighbor set. Hence, a node needs to construct all possible local independent sets to obtain all local solutions possible. For example, consider the graph shown in Figure 1. For the node B, the closed one hop neighbor set is given by $\{A, B, C, D\}$. All local solutions for this set are given by $\{A, C, D\}$ and $\{B\}$. For the node B, these two combinations are the only two possible local maximal independent sets. By maximal we mean that adding another vertex to the set would destroy the independence property. Similarly, for the node A, the set of local solutions is given by $\{A\}$ and $\{B\}$ (i.e., either A or B can be in the independent set) and so on.

3. Prioritize the local solutions: The objective of the prioritization phase is to have some criteria on the basis of which a node decides which local solution to use. For the independent set problem, one possible way to prioritize the local solutions may be given by ordering them in descending order of maximum cardinality. Hence, for the node B, $\{A, C, D\}$ would be the first choice local solution and $\{B\}$ would be the second. This is because the first would allow for a larger independent set. If cardinalities are equal, we can break ties by node id's.

4. Negotiate with neighbors for mutually satisfying solutions: A node must determine which of its solutions to use. Each solution is essentially a decision on which nodes in the one hop neighbor set are a part of the independent set. Since such a choice affects the solutions of other nodes in the neighborhood, a negotiation phase is required. For example, in the Figure 1, if the nodes $A, C,$ *and* D agree to the first choice of node B then this automatically renders the choice of $\{A\}$ invalid from the local solutions of node A, thereby making $\{B\}$ its local solution of choice. The nodes selected in the independent set are colored gray in the figure.

Let us now briefly sketch the implementation details of these phases. Initially every node broadcasts a message containing its *id* to discover who its neighbors are. Upon receiving this information, each node can then independently compute all local solutions and prioritize them. The negotiation phase would entail exchanging a message with its neighbors that describes the independent set it prefers to use. A neighbor can either accept or reject this proposed local partition. If a valid independent set exists, it has to be in the set of all local solutions and hence it will be satisfied. Else, none of the nodes will join the independent set. If the neighbors accept a node's choice, they have made a decision which now narrows their own choices. For the example in Figure 1, when the node B has negotiated its choice of $\{A, C, D\}$ with its neighbors and they have accepted this, they join the independent set. Node C can likewise negotiate with its neighbor E to select one of E's solutions in which E is not in the independent set. In case E has chose itself in the independent set, C can drop itself without impacting anyone else.

4. Developing the Framework for Coverage Problems

In this section we define the steps of our framework presented in Section 3 as applied to different coverage problems. The specifics for each of the three coverage problems - area, target and k-coverage are defined in the next section. We begin with some definitions. Section 4.2 then lays out the framework. As opposed to

the independent set example in the previous section, we will look at the coverage problems in much greater detail in this section.

4.1 Definitions

- *The Network Graph*: A common representation of a sensor network is that of using a graph to model the connectivity of the network. The network can be represented using a graph $G = (V, E)$ where, $V = \{s_1, s_2, ..., s_n\}$ is the set of sensors and an edge $e = (s_i, s_j) \in E$, if and only if sensor s_i is in communication range of sensor s_j. In a network with fixed communication ranges for each node, this model becomes a *Unit Disk Graph*.
- $b(s)$: the battery available at sensor s.
- *Cover Set* C: In general, by a cover set we are referring to a minimal subset of sensors that meets some coverage objective. The objective here depends on the coverage problem being solved. We use the terms cover and cover set interchangeably.
- $lt(C) = min_{s \in C} b(s)$, the maximum lifetime of a cover C. The sensor s with minimum battery is known as the *bottleneck* sensor of the cover C.
- *Local Cover Set*: For a given sensor s, there exists some local coverage objective. For example in the target coverage problem, any sensor s has some local targets (i.e., targets it can cover). We define the local cover set for a sensor s as a minimal set of sensors in the neighborhood of s that cover all local targets. Alternatively, all targets in the one-/two-hop neighborhood of s can be considered. See Section 5.1 for more details.

4.2 The Framework for Coverage Problems

4.2.1 Applicability of Framework

For coverage problems, if every sensor ensures that its local coverage objective (local targets/area) is satisfied then this implies that the global coverage objective has also been met. Also for the coverage problem, local solutions are always compatible with each other since a sensors local solution does not invalidate that of another sensor's solution.

4.2.2 Modeling the local solutions and their dependencies - The Lifetime Dependency (LD) Graph

We approach this problem by focusing on the local neighborhood of a sensor. Each sensor constructs all possible local covers that satisfy its local coverage objective. In the local 1-hop neighborhood, the number of local covers is usually small (where a cover could be for area or targets, see Section 4.1). This allows individual sensors to distributedly construct local minimal cover sets. A sensor can construct its local covers by considering one-hop neighbors it can communicate to while trying to meet its coverage objectives. For a better decision, it can also consider all neighbors up to two hops and their targets at a slightly increased communication cost.

When two cover sets have some sensors in common, they have some *dependency* on each other because using one cover set drains the battery of the sensors it shares with the other cover set. This is important because when we pick cover sets to use in a schedule, we should take into account this influence that they have on future cover sets in the schedule. To account for this we define the lifetime dependency graph as follows.

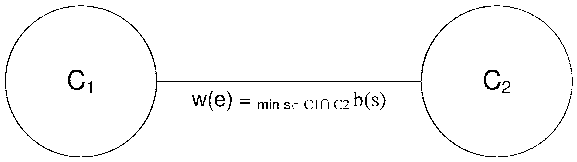

Figure 2. A Simple 2-node LD Graph

Every node in the LD graph represents a local cover and an edge between two nodes implies that the two covers share some sensors in common. The weight of an edge is given by the life of the *weakest* sensor in this set of common sensors that the edge represents. Finally, we define the *degree* of a cover C as the sum of the weights of all edges incident to that node. More formally, the local lifetime dependency graph can be defined as a weighted graph, $G' = (V', E')$ where, $V' = \{C_1, C_2, ..., C_k\}$ and each member $C_i \in V'$ represents a local cover meeting the local coverage objective, and an edge $e' = (C_i, C_j) \in E'$, if and only if $C_i \cap C_j \neq \emptyset$. We also define weights on the nodes and the edges as follows:

- $w(e')$: Let $e' = (C_i, C_j)$, and $I = C_i \cap C_j$. Then, $w(e') = min_{s \in I}\ b(s)$.
- $d(C)$=$\Sigma_{e' \in E' \text{ and incident to } C}\ w(e')$, the weight or *degree* of a cover C.

The reasoning behind this definition of the edge weight comes from considering a simple two node LD Graph (see Fig. 2) with two covers C_1 and C_2 sharing an edge e. The lifetime of the graph is given by,

$$L \leq min(lt(C_1) + lt(C_2), w(e))$$

This indicates that for the two node graph, the lifetime is bounded by either the sum of the individual lifetimes of the two covers C_1 and C_2 or by the lifetime of the smallest common sensor in $C_1 \cap C_2$ (given by $w(e)$). Note that if the two sets C_1 and C_2 were disjoint then the lifetime of the graph would have been $lt(C_1)+lt(C_2)$. The fact that they share some nodes in common implies that using one cover set also reduces the lifetime of the sensor it has in common with the other cover set. Hence, for the non-disjoint case, the bound on the graph could also be the smallest common sensor between these covers.

Similarly, the reasoning behind the definition of the *degree* of a cover C is that by summing the weights of all the edges incident on the cover C, we are getting a measure of the *impact* it would have on *all* other covers with which it shares an edge.

4.2.3 Prioritize the local solutions

Initially, each sensor s communicates with its neighbors and exchanges information on available battery $b(s)$ and the region (area or targets) it can cover. We will discuss the specific message exchanges in Section 5. Based on this information, sensor s can compute all the local covers for its local objective. Each sensor then constructs a local LD graph $G' = (V', E')$ for these covers, and calculates the degree $d(C)$ of each cover $C \in V'$ in G'.

A *priority function* can be defined to prioritize the local covers. We base the priority of cover C on its degree $d(C)$ in the lifetime dependency (LD) graph. A lower degree is better since this corresponds to a smaller impact on other covers. If the degree is the same for two or more covers, ties are broken by using (i) the cover with a longer lifetime, (ii) the cover with fewer sensors remaining to be turned on (See the next step), (iii) the cover with the smaller sensor id.

4.2.4 Negotiate solutions

The algorithm for a sensor to negotiate its solutions with that of its neighbors operates in rounds. Each round consists of two phases. Phase 1 is the setup phase as described above. In Phase 2, a sensor arrives at an on-off decision based on the messages it receives. The operation in rounds is very similar to other distributed

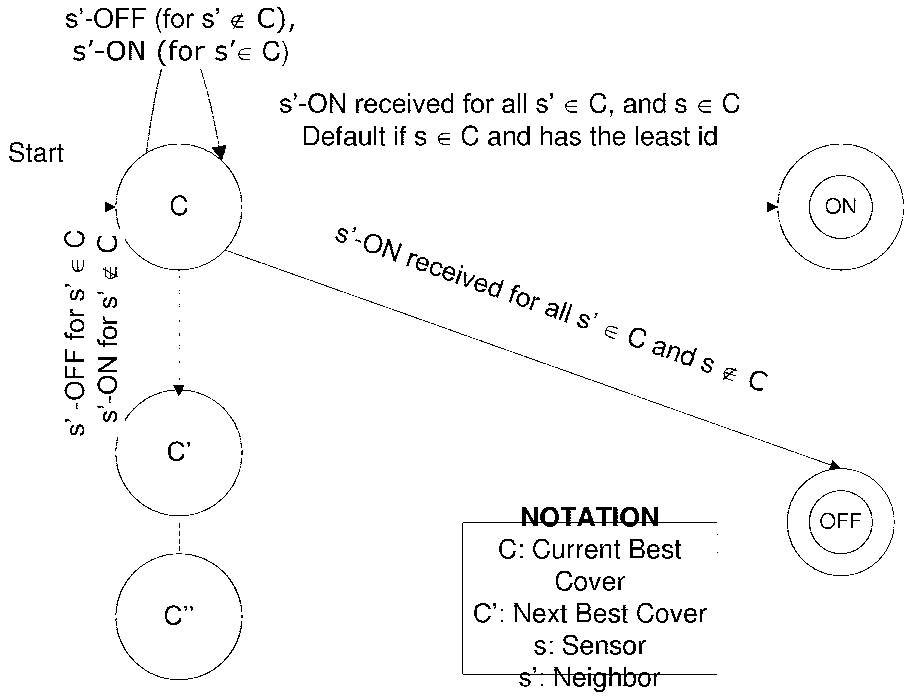

Figure 3. The state transitions to decide the On-Off Status [18]

After calculating the priority function, each sensor now has an ordering of its local cover sets in terms of preference. The goal is to try and satisfy the highest priority cover. However, a cover comprises of multiple sensors and if one of these switches off, this cover cannot be satisfied. Hence, each sensor now uses the automaton in Fig. 3 to decide whether it can switch off or if it needs to remain on.

The automaton starts with every sensor s in its highest priority cover C. The sensor s keeps trying to satisfy this cover C and eventually if the cover C is satisfied, then s switches on if $s \in C$ or, s switches off if $s \notin C$. Note that a cover C is considered satisfied if every sensor $s \in C$ has switched on. If a cover C cannot be satisfied, then the sensor s transitions to its next best priority cover C', C'' and so on, until a cover is satisfied.

The transitions of the automaton are outlined below.

- Continue with the best cover C: Sensor s continues with its current best cover C if its neighbor $s' \notin C$ goes off (since s' does not affect C) or if neighbor $s' \in C$ becomes on (thus improving chances for C).
- To on/sense status: If all the neighboring sensors in cover C except s become on, and $s \in C$, s switches itself on satisfying the cover C for its neighborhood, and sends its on-status to its neighbors.
- To off/sleep status: If all the neighboring sensors in cover C become on thus satisfying C, and $s \notin C$, s switches itself off, and sends its off-status to its neighbors.
- Transition to the next best cover C': Sensor s transitions to the next best priority

cover C', if (i) C becomes infeasible because a neighboring sensor $s' \in C$ has turned off, or (ii) priority of C is now lower because a sensor $s' \notin C$ has turned on causing another cover C', with same degree and lifetime as C, with fewer sensors remaining to be turned on.

Note that the automata for negotiation is simple because for coverage problems, we are not negotiating for mutual compatibility/feasibility. The only concern of the negotiation phase here is that of local optimality.

5. Applying the Coverage Framework

In this section we employ our framework to show how it can be applied to the Target, Area, and k-coverage problems.

5.1 Target Coverage

In the target coverage problem, we are given a set of targets $T = \{t_1, t_2, ..., t_m\}$ that are scattered around a region R. These targets are considered to be stationary. The objective of the problem is to monitor all targets in T while maximizing the lifetime of the network. In addition to the previous definitions, we define:

- $T(s)$: The set of targets that sensor s can sense,
- $N(s, k)$: The set of neighbors of sensor s at no more than k hops from s. This set is *closed* i.e. it includes the sensor s.
- *Local Cover Set*: A local cover set for a sensor s is a minimal set of sensors in $N(s, 1)$ that cover all targets in $T(s)$. For better performance, all targets in $T(s'), s' \in N(s, 1)$ $or N(s, 2)$ can be considered.

Using the framework defined in Section 4, the phases of the algorithm can be specified as follows. Note that we consider a 1-hop neighbor set i.e. $N(s, 1)$. This can easily be extended to more hops at a larger communication cost.

Compute, Weight and prioritize local solutions: Each sensor s communicates with each of its neighbor $s' \in N(s, 1)$ exchanging locations, battery levels $b(s)$ and $b(s')$, and the targets covered $T(s)$ and $T(s')$. Then it finds all the local covers using the sensors in $N(s, 1)$ for the target set being considered. The latter can be solely $T(s)$ or could also include $T(s')$ for all $s' \in N(s, 1)$. It then constructs the local LD graph $G = (V, E)$ over those covers, and calculates the degree $d(C)$ of each cover $C \in V$ in the graph G.

Negotiate solutions: This round essentially remains the same with each sensor using the automata of Fig. 3. Each cover set C in the automata is a set of sensors that can cover the target set being considered.

Correctness: In [18] we present a proof to show that the proposed heuristic is correct, terminates and is deadlock free.

Message and Time Complexity: If the maximum degree Δ of the network graph is assumed to be a constant, the message complexity is given by $O(\Delta)$ and can also be considered to be constant. If τ is the maximum number of targets in any sensors neighborhood, it can be shown that the time complexity is given by $O(\Delta^\tau)$. Even though this is exponential, since τ is usually small, the performance is not affected. See [18] for more details.

Self-organization and self-repair: The algorithm is self-organizing since each sensor s can run its own automaton independently to arrive at a decision to switch on/off. By exchanging messages, the sensor does however take into account its neighbors choices in making a decision. Also, because the distributed algorithm is

organized in rounds, it is self-repairing. If a sensor fails, its neighbors can account for that in the following round as they will not receive a ON message for that sensor. This limits the loss of coverage to a maximum of one round.

5.2 Area Coverage

For the area coverage problem, we are given a region R and the goal is to have this region completely covered at all times by active sensors.

In order to formulate the area coverage problem in our framework, consider any individual sensor s. The area covered by this sensor can be represented as a disk with the sensor at the center. The coverage objective of this sensor is to ensure that the area within its coverage disk is completely covered at all times. Now, certain parts of this disk are covered by different sensors in $N(s, 1)$. Hence, a cover set is any minimal set of sensors in $s' \in N(s, 1)$ that together cover this entire area.

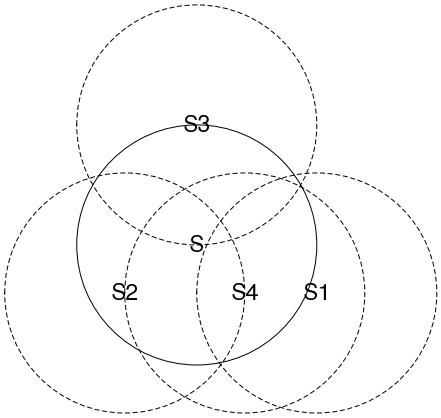

Figure 4. Example for area coverage

An example is shown in Figure 4. For the sensor s, the following set covers are possible for its area, $\{s\}, \{s_1, s_2, s_3\}$ and $\{s_2, s_3, s_4\}$. In order to compute what part of its area is covered by its neighbors, each sensor s exchanges its location and its battery information with its neighbors in $N(s, 1)$. To determine what part of its area is covered by its neighbors, we use the method described in [23] to determine sponsored coverage. Once sponsoring information has been determined, a sensor can compute all cover sets for its area. The weighting, prioritizing and negotiating phases of the framework then follow as in the target coverage problem.

Alternatively, [21] defines a field as a set of points that are covered by the same set of sensors. They then discretized the area into a grid. Once points have been grouped into fields, all that is needed is to ensure that all fields are covered. Hence, each field corresponds to a *virtual target* and the problem can effectively be reduced to that of target coverage. We experiment with both the field based method and the direct formulation described above in Section 6. A different approach is taken in [3], where the authors use the idea of covering all the faces of a graph. This avoids having to define the granularity of a grid.

5.3 k-Coverage

The k-Coverage problem can be defined for either target or area coverage. Here, we discuss it in the context of target coverage. Let $T = \{t_1, t_2, ..., t_m\}$ be the set of targets scattered around a region R. The goal is to ensure that for every $t \in T$

at least k sensors cover t at all times. Note that $k \leq \delta$, where δ is the minimum number of sensors covering any target $t \in T$. See [24] [15] [27] for more details.

The extension of our framework to the k-Coverage problem is straightforward. All we need to do is to ensure that every local cover C constructed during the initial setup phase is a k-Cover. To construct local covers that are k-covering, each sensor picks k neighbors for every target in $T(s)$. By imposing this restriction on the local covers, we can ensure that globally, every target t is k-Covered.

6. Simulation Results

In this section, we evaluate the performance of our coverage algorithms as compared with LBP [4] and DEEPS [5]. The simulations were programmed using C^{++}. A static network of sensors is used. For the target coverage problem, the targets are considered to be static also. For the area coverage problem, we use both our direct formulation and the concept of *fields* (Section 5.2) to transform the area coverage problem into the target coverage problem. Then the same algorithms are applied with these *virtual* targets. The k-coverage problem is also simulated with respect to target coverage. However, since LBP and DEEPS are not extended to solve this problem, we cannot compare our results to theirs for k-coverage.

We consider sensors scattered randomly in a 100m x 100m area. It is also assumed that the communication range of each sensor is twice the sensing range. We consider two different energy models. In the *linear* model, the energy required to sense a target at a distance d is proportional to d. In the *non-linear* model, the energy needed to sense the same target is a function of d^p, where p varies typically from $2-6$. For our experiments, we fix the value of d to 2 (quadratic model).

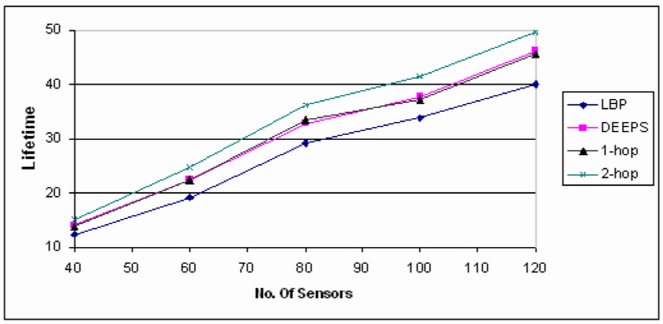

Figure 5. Lifetime with 25 Targets, Linear Energy Model

In order to compute all local covers, each sensor maintains a list of all local targets and the neighbors that cover them. This is stored in a matrix where the rows represent neighbor sensors in $N(s,1)$ and the columns targets in $T(s)$. This matrix is bounded since, by considering only one or two hop neighbors, we have restricted its size. Iterating over the different row combinations that cover all columns of this matrix gives us all possible covers. Since the number of local targets is limited, the number of columns of this matrix are also limited.

For the target coverage problem, we consider 25 and 50 targets, randomly deployed in the area. This is the same as considered in [4] and [5]. We vary the number of sensors in steps of 20. The results for 25 targets using a linear energy model are shown in Fig. 5. We also consider a 2-hop version of our heuristic. Here, the set of neighbors is defined as $N(s,2)$. Also, for each sensor the target set $T(s)$, is defined as all the targets of its one-hop neighbors and itself. As we can see from the results in Fig. 5, our 1-hop heuristic is on an average about 10% better than

LBP and almost similar to the 2-hop DEEPS in performance. If we compare the 2-hop version of our heuristic against DEEPS, it is superior. The same experiments are also carried out using the quadratic energy model. A snapshot of the results are shown in Fig. 6. Here, we consider a network with 60 sensors and compare the performance of both the energy models. We also consider the linear case with 50 targets. As can be seen from these results, the relative trends are similar to Fig. 5 with the basic 1-hop heuristic outperforming LBP and being very similar to DEEPS in performance.

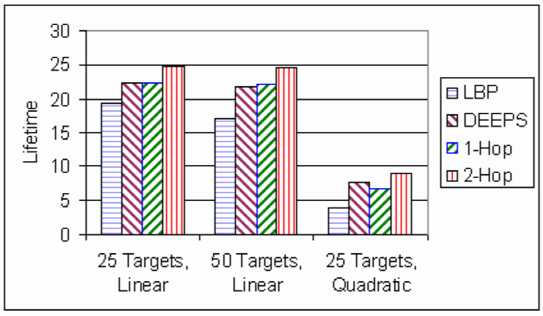

Figure 6. Lifetime with 60 sensors, energy model and number of targets varying

For the area coverage problem, once again sensors are scattered randomly in a 100m x 100m area. However, now the objective is to continuously monitor the area. We apply the same decomposition of a given graph into fields for the field based formulation. A virtual target corresponding to each field is considered. By covering each of these virtual targets, we ensure that all the fields and hence, the whole area is covered. We also use a direct formulation as explained in Section 5.2. The results are shown in Figure 7. Once again, a similar trend to the target coverage problem is observed with the 1-hop version outperforming LBP and being very similar to DEEPS. The field based decomposition yields slightly better results because of the loss in sponsored area calculation of the direct method as explained in [23].

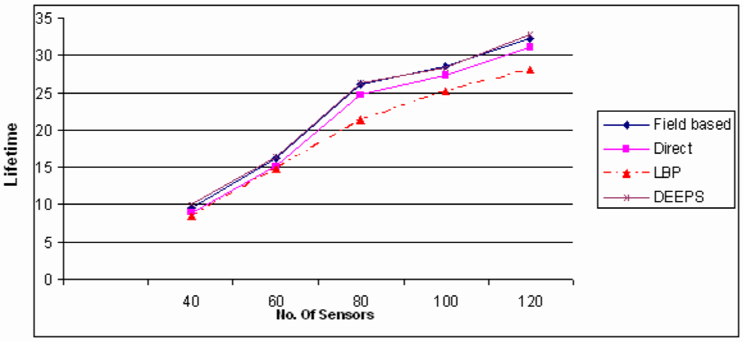

Figure 7. Area coverage with Number of Sensors varying, Linear Energy Model

Finally, we implemented the k-coverage for $k = 2$, i.e., the two covered case where every target is covered by at least two sensors. Since both LBP and DEEPS have not be extended to the k-coverage problem, we are unable to compare our performance relative to them. Instead, we compared the lifetime for the 2-covered case with the 1-covered case for our basic 1-hop algorithm. Using a network with 25 targets, we vary the number of sensors. The energy model we consider is linear. The results are summarized in Table 3. On an average we see a reduction of lifetime of 35-40% for the 2-covered case as compared to the 1-covered case.

Table 3. Lifetime of a 2-covered vs. 1-covered network with 25 targets

No. of Sensors	1-covered	2-covered
40	13.8	8.7
60	22.4	14.1
80	33.4	19.8
100	37.2	23.2
120	45.6	27.1

7. Conclusion

In this paper, we present a general algorithmic framework for solving certain graph and network problems. The framework applies to graph and network optimization problems that exhibit the property of local compatible solutions yielding globally feasible solutions when combined. We utilize the framework to develop heuristics for solving the different coverage problems for sensor networks. Experimental evaluation of these heuristics shows performance gains when compared to existing approaches in [4, 5].

A highlight of our framework is in defining a *Dependency Graph* to capture the interactions between local solutions. This enables us to not only model these interactions but also factor them into the process of heuristically picking better solutions. While previous algorithms are based on greedily picking solutions, our heuristics can use this graph to gain an insight into the problem structure. This is combined with a negotiating phase, where each node communicates with its neighbors to ensure compatibility of their mutual solutions and to determine which solution to use.

Overall, our framework presents a new way of looking at these problems. An initial version has been presented here and several variations of the Dependency Graph and the weight functions are currently being explored. We also illustrated how the framework can be applied to the maximum independent set problem. Other problems that seem good candidates include vertex cover, maximum triangle packing, and maximum channel assignment in cellular networks.

Acknowledgment

The authors would like to thank Dr. Yingshu Li for fruitful discussions on different coverage problems.

References

[1] Z. Abrams, A. Goel, and S. Plotkin. Set k-cover algorithms for energy efficient monitoring in wireless sensor networks. *Third International Symposium on Information Processing in Sensor Networks*, pages 424–432, 2004.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Commun. Mag.*, pages 102–114, 2002.

[3] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Power efficient monitoring management in sensor networks. *Wireless Communications and Networking Conference (WCNC)*, 4:2329–2334 Vol.4, 2004.

[4] P. Berman, G. Calinescu, C. Shah, and A. Zelikovsky. Efficient energy management in sensor networks. *In Ad Hoc and Sensor Networks, Wireless Networks and Mobile Computing*, 2005.

[5] Dumitru Brinza and Alexander Zelikovsky. Deeps: Deterministic energy-efficient protocol for sensor networks. *Proceedings of the International Workshop on Self-Assembling Wireless Networks (SAWN)*, pages 261–266, 2006.

[6] M. Cardei, M.T. Thai, Yingshu Li, and Weili Wu. Energy-efficient target coverage in wireless sensor networks. *INFOCOM 2005*, 3, March 2005.

[7] M. Cardei and J. Wu. Energy-efficient coverage problems in wireless ad hoc sensor networks. *Computer Communications*, 29(4):413–420, 2006.

[8] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless Networks*, 11:333–340(8), 2005.

[9] Mihaela Cardei, David MacCallum, Maggie Xiaoyan Cheng, Manki Min, Xiaohua Jia, Deying Li, and Ding-Zhu Du. Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks*, 3(3-4):213–229, 2002.

[10] S.K Das, N. Deo, and S. K. Prasad. Two minimum spanning forest algorithms for fixed-size hypercube computers. *Parallel Computing*, 15:179–187, 1990.

[11] A. Dhawan, C. T. Vu, A. Zelikovsky, Y. Li, and S. K. Prasad. Maximum lifetime of sensor networks with adjustable sensing range. *Proceedings of the International Workshop on Self-Assembling Wireless Networks (SAWN)*, pages 285–289, 2006.

[12] Akshaye Dhawan and Sushil K. Prasad. A distributed algorithmic framework for coverage problems in wireless sensor networks. *Procs. Intl. Parallel and Dist. Processing Symp. Workshops (IPDPS), Workshop on Advances in Parallel and Distributed Computational Models (APDCM)*, pages 1–8, 2008.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[14] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, page 300, Washington, DC, USA, 1998. IEEE Computer Society.

[15] Santosh Kumar, Ten H. Lai, and József Balogh. On k-coverage in a mostly sleeping sensor network. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 144–158, New York, NY, USA, 2004. ACM.

[16] Jun Lu and T. Suda. Coverage-aware self-scheduling in sensor networks. *18th Annual Workshop on Computer Communications (CCW)*, pages 117–123, 2003.

[17] Seapahn Meguerdichian Miodrag. Low power 0/1 coverage and scheduling techniques in sensor networks. *UCLA Technical Reports 030001*, 2003.

[18] Sushil K. Prasad and Akshaye Dhawan. Distributed algorithms for lifetime of wireless sensor networks based on dependencies among cover sets. In *HiPC: 14th International Conference on High Performance Computing, LNCS 4873*, pages 381–392, 2007.

[19] S. Sahni and X. Xu. Algorithms for wireless sensor networks. *Intl. Jr. on Distr. Sensor Networks*, 1, 2004.

[20] Eugene Shih, Seong-Hwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 272–287, New York, NY, USA, 2001. ACM.

[21] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. *IEEE International Conference on Communications (ICC)*, pages 472–476 vol.2, 2001.

[22] John A. Stine and Gustavo De Veciana. Improving energy efficiency of centrally controlled wireless data networks. *Wirel. Netw.*, 8(6):681–700, 2002.

[23] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, New York, NY, USA, 2002. ACM.

[24] Guoliang Xing, Xiaorui Wang, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Trans. Sen. Netw.*, 1(1):36–72, 2005.

[25] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. Peas: A robust energy conserving protocol for long-lived sensor networks. *IEEE International Conference on Network Protocols (ICNP)*, 00:200, 2002.

[26] Honghai Zhang and Jennifer Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Ad Hoc and Sensor Wireless Networks (AHSWN)*, 2005.

[27] H. Zongheng Zhou; Das, S.; Gupta. Connected k-coverage problem in sensor networks. *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, pages 373–378, 11-13 Oct. 2004.