SCISPACE
formerly Typeset

Open access • Proceedings Article • DOI:10.1109/ICDCS.2007.150

uSense: A Unified Asymmetric Sensing Coverage Architecture for Wireless Sensor Networks — Source link

Yu Gu, Joengmin Hwang, Tian He, David H. C. Du

Institutions: University of Minnesota

Published on: 25 Jun 2007 - International Conference on Distributed Computing Systems

Topics: Wireless sensor network, Scheduling (computing), Intelligent sensor and Algorithm design

Related papers:

- Integrated coverage and connectivity configuration in wireless sensor networks
- Achieving Asymmetric Sensing Coverage for Duty Cycled Wireless Sensor Networks
- Differentiated surveillance for sensor networks
- Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links
- A survey on sensor networks

Share this paper:

View more about this paper here: https://typeset.io/papers/usense-a-unified-asymmetric-sensing-coverage-architecture-xboasbj390

uSense: A Unified Asymmetric Sensing Coverage Architecture for Wireless Sensor Networks

Yu Gu, Joengmin Hwang, Tian He, David Hung-Chang Du
{yugu,jhwang,tianhe,du}@cs.umn.edu
Department of Computer Science and Engineering, University of Minnesota

***Abstract*— As a key approach to achieve energy efficiency in sensor networks, sensing coverage has been studied extensively. Researchers have designed many coverage protocols to provide various kinds of service guarantees on the network lifetime, coverage ratio and detection delay. While these protocols are effective, they are not flexible enough to meet multiple design goals simultaneously. In this paper, we propose a Unified Sensing Coverage Architecture, called uSense, which features three novel ideas: *Asymmetric Architecture*, *Generic Switching* and *Global Scheduling*. We propose asymmetric architecture based on the conceptual separation of switching from scheduling. Switching is efficiently supported in sensor nodes, while scheduling is done in a separated computational entity, where multiple scheduling algorithms are supported. As an instance, we propose a two-level global coverage algorithm, called uScan. At the first level, coverage is scheduled to activate different portions of an area. We propose an optimal scheduling algorithm to minimize area breach. At the second level, sets of nodes are selected to cover active portions. Importantly, we show the feasibility to obtain optimal set-cover results in linear time if the layout of areas satisfies certain conditions. We evaluate our architecture with a network of 30 MicaZ motes, an extensive simulation with 10,000 nodes, as well as theoretical analysis. The results indicate that uSense is a promising architecture to support flexible and efficient coverage in sensor networks.**

I. Introduction

Wireless Sensor Networks (WSNs), consisting of thousands of low-cost sensor nodes, have been used in many application domains such as military surveillance [1], habitat monitoring [2] and scientific exploration. Limited power supplies and difficulties in harvesting ambient energy make energy conservation a critical issue to address. Energy-efficient sensing coverage extends system lifetime by leveraging on the redundant deployment of sensor nodes. Within a couple of years, sensing coverage has become a well studied subject which provides either full coverage in both time and space [3], [4], [5], [6], [7], [8], coverage with guaranteed delay and connectivity [9], [10], [11], [12], or guaranteed target detection within a certain stealth distance [13], [14]. These algorithms are designed to be well distributed and localized, providing solid performance gains with a certain guarantee on service (e.g., a bounded delay in detection or a guaranteed lifttime). While the state-of-the-art is encouraging, we believe there are some aspects that need further investigation. First, currently different sensing coverage algorithms focus on different service guarantees (e.g., coverage vs. detection delay). Any single design is not general enough to meet a wide range of sensing requirements under different operating scenarios. Second, in most algorithms, extending system lifetime is achieved essentially through coordination among neighboring nodes. The local node density, therefore, imposes a theoretical upper bound on the system lifetime, if a continuous sensing coverage or a partial coverage is required. Such a bound can be surpassed through global scheduling. However, the overhead of global scheduling would increase significantly if the coordination among the nodes goes beyond the neighborhood.

To address these two issues simultaneously, in this paper, we introduce a new sensing architecture, called uSense, which features three novel ideas: *Asymmetric Architecture*, *Generic Switching* and *Global Scheduling*. The key concept of uSense is the decoupling of sensing coverage into two separated functions: *scheduling* and *switching*. The former calculates the parameters of a working schedule for individual nodes, while the latter turns on/off the sensors according to the scheduling parameters. We employ an asymmetric architecture to improve the flexibility and extensibility of the design. Switching is a lightweight generic algorithm, taking two parameters as inputs. With these parameters, it can faithfully execute the sleep/awake schedule of individual nodes decided by *any* existing coverage algorithm. Switching must be supported in the sensor nodes, since a node has to be on to provide coverage and has to be off to save energy. On the other hand, it is not absolutely necessary to implement the scheduling within constrained sensor nodes. We opt to support scheduling on a powerful computational entity, which can be implemented at the second-tier nodes (e.g., Intel Stargate used in TENET [15] and ExScale [1]), or on an outside server, or on a cluster of servers to avoid single point of failure.

Asymmetric design is now considered as a promising guiding principle for sensor networks. By decoupling the scheduling function and implementing it outside the network core, we can achieve *efficiency* and *performance* simultaneously. This is because firstly, with fewer functions sharing the limited resources on a node, we can build the switching functions, the ones that must be embedded into individual sensor node, in a less stringent design space. Secondly, we can design and implement the scheduling functions outside of sensor networks, free of resource constrains inherent in the sensor nodes, therefore, will be more sophisticated and powerful, leading to a significantly improved overall performance.

Before describing the uSense design in detail, we identify the objectives and intellectual contributions of this work as follows:

- **Asymmetric Sensing Architecture:** The asymmetric architecture enables us to design sophisticated coverage algorithms in an unconstrained design space and represent such intelligence with a lightweight algorithm implemented in sensor nodes.
- **Generic Switching Algorithm:** To the best of our knowledge, we propose here the first generic and lightweight switching algorithm that is suitable for sensor nodes with

limited resources. We demonstrate our lightweight design at the sensor side is very effective.

- **Global Scheduling Algorithms:** We design two new global scheduling algorithms, using the concept of set-cover. Different from all previous work, we demonstrate the feasibility to identify a minimum cover set in linear time when the coverage area is a continuous curve.
- **System Implementation and Extensive Evaluation:** We have invested significant amount of effort to evaluate our design. We have implemented and evaluated the design on the TinyOS/Mote platform, using 30 MicaZ motes. We also provide a 10,000-node large scale simulation to compare with state-of-art solutions, lower-bonds and upper-bounds.

The rest of this paper is organized as follows. Section II introduces the overarching architecture of uSense. Section III describes the design of uScan, a two-level global scheduling algorithm. Section IV provides an analytic study of the proposed scheduling algorithms. Section V describes our system implementation and provides evaluation on the TinyOS/Mote platform. The results of the 10,000-node simulation are presented in Section VI to compare the performance of uSense and uScan with state-of-the-art solutions. Section VII discusses the related work. Section VIII concludes the paper.

II. uSense Architecture

One of key new ideas of this work is the conceptual separation of *switching* from *scheduling*. In this section, we provide an overview of our asymmetric sensing architecture.

A. Motivation

Our work is aiming at flexibility and efficiency in sensing coverage. It is often the case that a sensor network needs to support multiple operating scenarios. For example, a military surveillance network could be required to provide full coverage during a red alert (a spatial coverage requirement), however, it may allow a certain detection delay (a temporal coverage requirement) on other occasions to aggressively conserve energy. Two algorithms ([7] and [9]) have been successfully designed to meet these two design goals. However, neither of them, unfortunately, is flexible enough to meet both requirements. Evidently, with these two algorithms, we can achieve both functionalities by downloading two separate program images and switching between them (as supported by Deluge [16]). Clearly, separate images introduce excessive overhead in terms of communication bandwidth, energy and storage, putting flexibility and efficiency at odds with each other.

We observe that the wakeup/sleep schedules of individual sensor node can be described by two parameters, which are independent of the methods to obtain them. Therefore, the conceptual separation of switching from scheduling becomes a natural approach to resolve the conflict between flexibility and efficiency. In the uSense architecture, changing scheduling algorithms only affects the values of the parameters, not the switching logic implemented at the nodes. Consequently, flexibility can be achieved by disseminating only a few parameters.

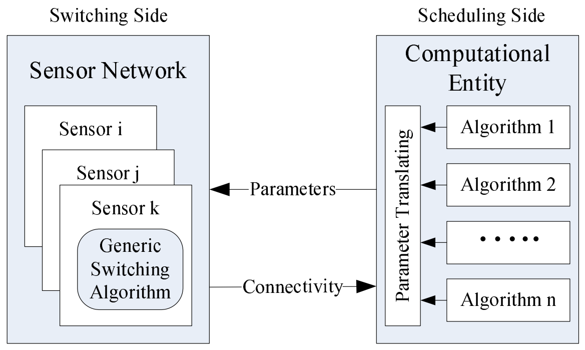

Fig. 1. Overview of uSense Architecture

B. uSense Design

The uSense architecture decouples switching from scheduling as shown in Figure 1. The switching algorithm is implemented in the sensor nodes, which takes two scheduling parameters as input. The scheduling algorithm, which is implemented separately, is responsible for generating these scheduling parameters. It takes the schedules decided by various sensing coverage algorithms and translates them into the two parameters to be used by the switching algorithm. The uSense architecture requires bi-directional communication, because that (i) most scheduling algorithms need the location information of the sensor nodes in order to create a wakeup/sleep schedule for individual node, and (ii)uSense needs to disseminate the scheduling results to the nodes within the network, where the actual switching happens. Obviously, in a static sensor network, these costs are very small, compared with the amount of sensing data transmitted by the sensor network. Furthermore, various existing protocols such as SPIN [17] and PSFQ [18] already can effectively perform energy efficient information dissemination and gathering in low power sensor networks. In the next two sections, we focus on the sensing coverage, describing the switching and scheduling algorithms, respectively.

C. Part I: Generic Switching Algorithm

The switching algorithm should be lightweight to run on resource constrained nodes and should be generic to accommodate various types of schedules. In our switching algorithm design, two parameters are used for each node, namely the schedule bits S and switching rate R.

- **Schedule bits** S is an infinite binary string in which 1 denotes the active state and 0 denotes the inactive state. The duty cycle of a node is the percentage of 1s in S.
- **Switching rate** R defines the rate of toggling between states. For example, a switching rate of 0.5HZ requires a node to read one bit from the schedule S every 2 seconds. When transient energy consumption is negligible, ideally a high switch rate R leads to a small detection delay. However, R cannot be arbitrarily small because: (i) a sensor has a warm-up delay before it can perform reliable samplings, (ii) in most detection algorithms, it is not robust enough to take just one sample to make a decision, so a sampling delay proportional to the number of samples is often introduced. These delays impose an upper-bound on the switching rate R.

Although our switching algorithm is simple and lightweight, it is powerful enough to support many types of coverage algorithms. Theoretically, when the switching rate R approaches infinity, schedule bits, as an infinite string, can precisely characterize on/off behavior generated by any coverage algorithm. As mentioned, the switching rate is finite in reality, therefore in the worst case, a node might need to extend a wake-up period by $\frac{1}{R}$ seconds to guarantee the coverage.

1) Regular Expression: A switching algorithm takes schedule bits S and switching rate R as inputs. Schedule bits S is formally defined as an infinite binary string. Obviously, it is practically impossible to disseminate an infinite binary string. Fortunately, the sensing coverage schedule is usually periodic, follows a certain pattern. Therefore, we can express S with a regular expression. For example, $(0010)^*$ can be used to denote a repeated off-off-active-off schedule. For a very dense network, after scheduling, the duty cycle of an individual node is usually low. In other words, the number of $1s$ in a schedule is comparatively small. In this case, we can use index values to represent the positions of active bits.

2) Timed Finite Automata: For a given schedule S described as a regular expression, a node builds a finite automata (FA). A straightforward method is to use a timer at the rate of R to drive the state transitions within the FA. However, this requires a node to wake-up the processor periodically, introducing excessive energy consumption. To address this issue, we therefore build a Timed Finite Automata (TFA). In a TFA, a state transition is triggered by 01 or 10 segments in the schedule S, and the delays of transitions are the gaps between 01 or 10 segments.

D. Part II: Scheduling Algorithms

As shown in Figure 1, a sensor scheduling algorithm is implemented separately (e.g., at the second tier). Free of resource constraints inherent in the sensor nodes, we can support a large number of coverage algorithms. In this section we focus on the generic scheduling framework, before proposing concrete scheduling algorithms in the next section.

To accommodate different sensing coverage algorithms, we need to convert the output of a coverage algorithm into two parameters understandable by the generic switching algorithm. To illustrate the idea, we use the algorithm presented in [8] as a case study.

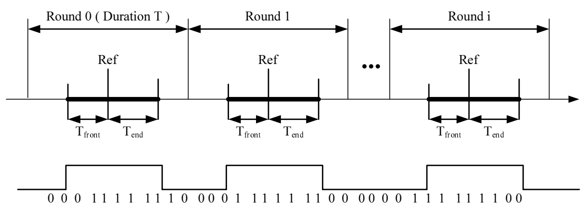

Fig. 2. Schedule Translation

In [8], the sensing phase of nodes is divided into rounds with equal duration *T*. The schedule for a node is determined by a tuple with four parameters: (*T*, *Ref*, T_{front}, T_{end}). As shown in Figure 2, *Ref* is a random time instance chosen by a node within [0, *T*). T_{front} is the duration of time prior to the reference point *Ref*, and T_{end} is the duration of time after reference point *Ref*. To build a schedule bits S, we check whether a time instance $\frac{k}{R}, k \in [0, T \cdot R]$ is located within the active periods. For example, the schedule shown in Figure 2 can be expressed as $(00111111100)^*$.

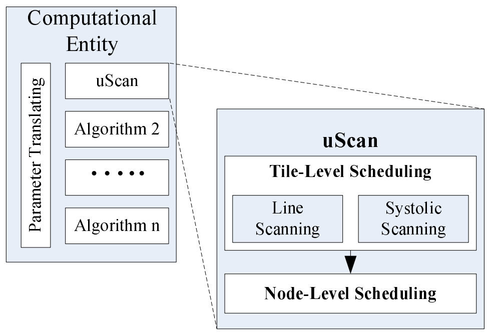

Fig. 3. The Design of uScan

III. Global Scheduling Algorithms

Conceptually, uSense can support many existing coverage algorithms. Due to its asymmetric architecture, it is especially friendly to the global scheduling algorithms. Since a global scheduling allows many more nodes to activate in turn rather than the localized ones that only schedule the nodes within neighborhood, it leads to a significant energy savings. In this section, we propose a global scheduling algorithm called uScan. Figure 3 shows the relation between uScan and uSense. Essentially, uScan is one of sensing coverage algorithms that are supported by the uSense architecture.

The outputs of uScan are the schedule bits S and switching rate R for individual nodes. uScan is a two-level schedule algorithm, which works as follows: Suppose we provide sensing coverage to a given area using uScan as shown in Figure 4. First, uScan divides the area into small regions, and decides the working schedules for these regions. This level of scheduling is conceptually independent of the deployment of the nodes. At the second-level, we assign nodes to cover the active regions at different time intervals, using a set-cover technique. By combining the first-level schedule and the set-cover assignment, we can decide the schedule bits S for individual nodes.

The biggest advantage of this two-level schedule algorithm is the separation of sensing pattern from the underlying node scheduling. The application only needs to specify the desired sensing behavior on the field in the first level of scheduling, and the second level of uScan can take various specified sensing patterns as input and produce the final working schedule of each individual sensor device. The two-level schedule of the uScan provides flexibility, reusability and efficiency to the sensing component of sensor applications by freeing various sensornet applications from designing their own scheduling protocols under different application requirements.

A. Assumptions

For the clarity of the protocol description in the rest of the paper, we assume that nodes are time-synchronized and their locations are precise. We refer the sensing area of a node as a circle with a nominal radius r centered at the location of the node. These are common assumptions for many sensor network applications [1], [2].

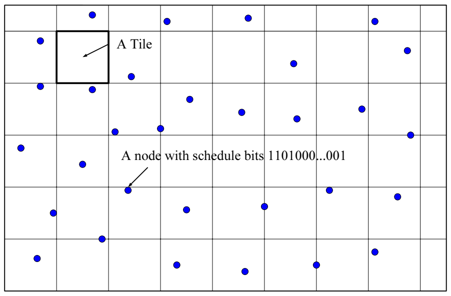

Fig. 4. Regular Tessellations

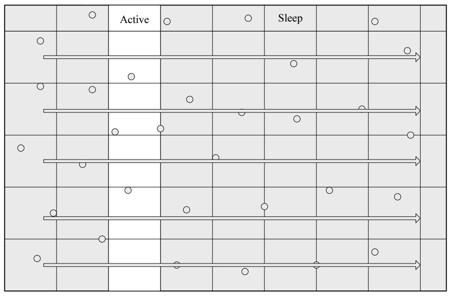

Fig. 5. Horizontal Scan

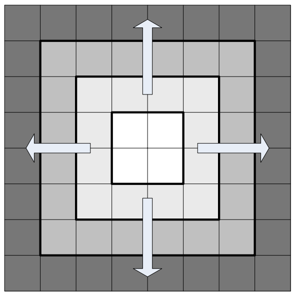

Fig. 6. Systolic Scan

B. Level I: Tile Scheduling

In uScan, we partition an area under surveillance into some small regions of the same shape, a process called tessellation. These small regions are called *tiles*, which can be regular triangles, rectangles or regular hexagons in a 2-D space. One simple example of tessellation is the rectangle-based partition, as shown in Figure 4. The size of tiles is set to be smaller than the minimum target size, so that a target is detected as long as a portion of a tile is covered. As a reminder, nodes within a sensor network only support a generic switching algorithm, which has neither the concept of tiles nor the partition information of the tiles. All the complex logic resides outside of sensor nodes. In this section, we describe two simple, yet effective methods for the tile-level scheduling. They differ in the energy consumption rate and the detection delay.

1) Line Scan: We start with a simple tile-level scheduling as shown in Figure 5. Instead of trying to cover all tiles, we only cover a column/row of tiles in a certain interval of time during one round of scan. The covered columns/rows are increasing or decreasing consecutively. Because only a small percentage of tiles are sensed at a specific point of time, line scan leads to a significant reduction in energy consumption, compared with full coverage [8].

Specifically, in the line-based global scanning, we introduce the concept of scanning speed *v*, which represents the speed of scan from one end to the other, horizontally or vertically. This scanning speed determines the maximum detection delay a network experiences. The scanning speed *v* can be transformed into the switch rate R. Suppose we have a rectangle-based tessellation, the length and width of a tile are L_l and L_w, respectively. For a given scan speed *v*, if we want to scan horizontally, the switch rate R is set to be $\frac{v}{L_l}$. Similarly, the switch rate is $\frac{v}{L_w}$, when we scan vertically. Starting from 0, we index the tiles in a row-major order. Therefore a tile with coordinates (row, col) is assigned the index of $row * col_{max} + col$ (col_{max} is the maximal column index). To cover a tile $t(i)$ with a coordinates (row, col) in a scanning round, schedule bits S for this tile is as follows:

$$\begin{aligned} S_h(i) &= (\underbrace{000..000}_{col-1}\mathbf{1}\ \underbrace{000..000}_{col_{max}-col})^* \qquad (Hscan) \\ S_v(i) &= (\underbrace{000..000}_{row-1}\mathbf{1}\ \underbrace{000..000}_{row_{max}-row})^* \qquad (Vscan) \end{aligned} \tag{1}$$

Moreover, we can perform horizontal and vertical scans simultaneously. Both directions share the same scanning speed *v*. We can obtain the schedule bits of a two-way scan by applying the bitwise OR operation on S_h and S_v obtained in Equation 1:

$$S(i) = S_h(i)|S_v(i) \tag{2}$$

2) Systolic Scan: Systolic Scan emulates the cardiac cycles of a beating heart. Figure 6 shows the design of systolic scan. The tiles are scanned from the inner layer to the outer layer, as denoted by different gray-levels in Figure 6. Without loss of generality, we describe the method with a simple case where $col_{max} = row_{max} = N$. Clearly, the length of schedule bits in a scanning round is $\lceil N/2 \rceil$

For the first time interval (represented by the first digit in the schedule bits), the tiles at the center of the area set their first digit of schedule bits to 1, and the schedule bits for these tiles are $(\mathbf{1}\underbrace{000..000}_{\lceil N/2\rceil-1})^*$.

Similarly, for the n^{th} time interval, the tiles whose coordinates meet one of follow four conditions:

$$\begin{array}{lllll} row == n & \& & col > n-1 & \& & col \le N-n \\ row == N-n & \& & col > n-1 & \& & col \le N-n \\ col == n & \& & row > n-1 & \& & row \le N-n \\ col == N-n & \& & row > n-1 & \& & row \le N-n \end{array} \tag{3}$$

set their schedule bits as follows:

$$\begin{aligned} &S(i) = (\ \underbrace{000..000}_{\lceil N/2\rceil-n-1}\ \mathbf{1}\underbrace{000..000}_{n})^* \\ &i = row * N + col \end{aligned} \tag{4}$$

where $n = 0, 1, 2..., \lceil N/2 \rceil - 1$ and i is the index of tiles that satisfies the requirements.

Both line scan and systolic scan specify only the set of tiles need to be activated (covered) at a given point of time. The task of covering each tile set is accomplished by the second-level node scheduling, which will be described in the next section.

C. Level II: Node Scheduling

Tile-level scheduling determines the set of active tiles TS_i at the time interval i. For example, in a horizontal line scan, the i^{th} column is activated at time interval i. In this section, we describe how we can translate a known tile schedule into a corresponding node schedule bits S, which can be interpreted directly by a generic switching algorithm.

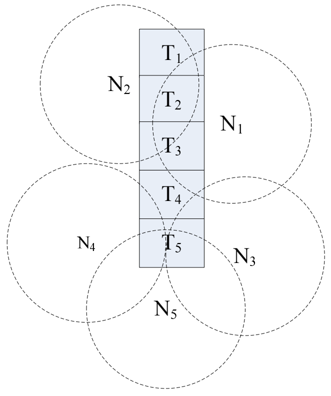

Fig. 7. Physical Coverage

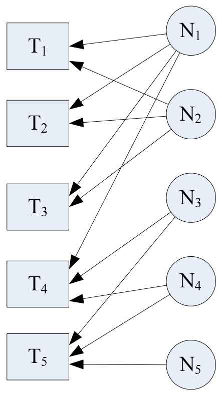

Fig. 8. Bipartite Graph

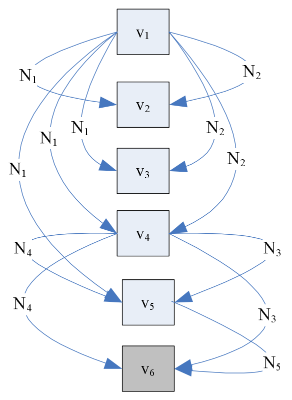

Fig. 9. MSC using DAG

1) Main Idea: Before we discuss the complete algorithm, we first illustrate our approach with a simple example. Figure 7 shows one column of tiles $TS = \{T_1, T_2, T_3, T_4, T_5\}$ that is covered by a set of nodes $NS = \{N_1, N_2, N_3, N_4, N_5\}$. Since we set the tile size smaller than the minimal target size, a tile is said to be covered as long as a portion of this tile is covered. Figure 7 can be mapped to the Coverage Bipartite Graph shown in Figure 8 according to the coverage relationship. Node scheduling consists of two steps. First, we keep identifying one-cover set with minimal number of nodes, until the size of one-cover set is above a certain threshold. For example, as shown in Figure 7, we identify three one-cover sets for TS: $CS^1 = \{N_1, N_5\}$, $CS^2 = \{N_2, N_3\}$ and $CS^3 = \{N_1, N_4\}$ to ensure that all nodes are used. Three sets CS^1, CS^2 and CS^3 can provide coverage to the tile set TS in a round-robin fashion. To do this, we create a node schedule that has three segments, each of which has a length of the tile schedule. If a node belongs to the CS^k set, the k^{th} segment has the same value as the tile schedule. Otherwise, the k^{th} segment has an all-zero value. For example, if the tile schedule of TS is 0010, the final schedules for the nodes shown in Figure 7 are:

$$\begin{aligned}
&S_1 = (\underbrace{0010}_{1}\underbrace{0000}_{2}\underbrace{0010}_{3})^* \quad S_2 = (\underbrace{0000}_{1}\underbrace{0010}_{2}\underbrace{0000}_{3})^* \\
&S_3 = (\underbrace{0000}_{1}\underbrace{0010}_{2}\underbrace{0000}_{3})^* \quad S_4 = (\underbrace{0000}_{1}\underbrace{0000}_{2}\underbrace{0010}_{3})^* \\
&S_5 = (\underbrace{0010}_{1}\underbrace{0000}_{2}\underbrace{0000}_{3})^*
\end{aligned} \tag{5}$$

2) Identifying Minimum Set-Cover within Linear Time: To save energy at each time interval, we need to identify a minimum set of nodes to cover an active tile set. This is a typical set-cover problem, which can be formally defined as:

Definition Given a collection C of subsets of a finite set T, find a set cover C' ($C' \subseteq C$) for T, such that every element in T belongs to at least one member of C'.

The generic Minimum Set Cover (MSC) problem has been proven NP-Hard and any polynomial algorithm can only find results of $1+ln|T|$ optimum [19]. Fortunately, line scan coverage is a special case of the generic set cover problem, because a node can cover only a continuous segment of tiles. The main idea of our polynomial algorithm is to map Coverage Bipartite Graph (figure 8) into a Directed Acyclic Graph (DAG) (figure 9). The one-to-one mapping rules are as follows:

1) We map N tiles in TS_i into N vertices $V = \{v_1, ..., v_N\}$ and add one extra vertex v_{N+1}.
2) If a node covers a set of tiles $\{T_i, ..., T_{i+n}\}$, we create n directional edges (v_i, v_j) where $v_j = v_{i+1}, ..., v_{i+n+1}$. Each edge has a unit cost.

Through this mapping, the tile set cover problem can be reduced to the problem of finding out the shortest paths from v_1 to v_{N+1}. The mapping process takes $O(|V|) + O(|E|)$ time. For an arbitrary graph, the shortest path algorithm finishes within $O(|V|^2)$ using the Dijkstra algorithm. Since the graph we create is DAG, we can find the shortest path in $O(|E|)$ time, using a fast reaching algorithm [20]. Therefore the whole algorithm finishes in $O(|V|) + O(|E|)$.

To illustrate the idea, Figure 9 shows a DAG which is mapped from Figure 8. To identify the minimal set cover, we need to find out the shortest path from v_1 to v_6. In this simple case, we can cover all the tiles using one of following node sets: $\{N_1, N_3\}$, $\{N_1, N_4\}$, $\{N_1, N_5\}$, $\{N_2, N_3\}$ or $\{N_2, N_4\}$, which are five corresponding shortest paths from v_1 to v_6.

We note that the proposed polynomial algorithm does not apply to generic tile scheduling. When a tile set does not form a continuous curve or a node can cover multiple segments of a tile set simultaneously, the polynomial algorithm can not guarantee the complete coverage of active tiles. In these cases, we adopt a greedy set-cover method by choosing the node that covers the most number of tiles first.

3) Selecting Cover Sets for Multiple TS: Up to now, node scheduling has been described using a simple example that assumes a node only needs to cover one tile set. Obviously, to support line scan or systolic scan in a 2-D space, we need to identify cover sets for the whole area (not just for a single column). Thus a node may need to cover multiple tile sets TS_i. The detailed process to cover the area is as follows:

1) Each node maintains a counter SC to record how many times it has been selected into final Cover Sets (for the purpose of energy balance).
2) For a tile set TS_i, the algorithm calculates the minimum cover set MCS_i among the nodes with minimum SC values. If the nodes with minimum SC values can not form a complete cover set, nodes with higher SC values

are used.

3) After we obtain all MCS_i, the smallest eligible MCS_i (SMCS) is selected and recorded for the purpose of node scheduling, and the SC values of nodes within this SMCS set are incremented.
4) Each TS_i has a coverage threshold, denoting the maximum number of nodes that can be used in a selected MCS_i. These thresholds are calculated based on the concept of redundancy. If the first MCS_i chosen for TS_i includes M nodes, the number of nodes in the following MCS_i for TS_i should not differ significantly, in order to reduce redundancy in coverage. We set the threshold for TS_i as $M \times \frac{2\pi}{\sqrt{27}}$, according to the redundancy in circle covering [21].
5) The SMCS selection process is repeated until the size of all MCS_i are larger than their thresholds.

4) Create Node Schedule Bits R : Suppose K one-cover sets are selected for a tile set TS_i with a tile schedule S_T (from Section III-B), we create a node schedule S_i for node N_i, which has K segments. The value of each segment is either S_T or zero. If a node belongs to the k^{th} one-cover set, the value of the k^{th} segments is S_T. Otherwise, the k^{th} segment has an all-zero value. Since in a 2-D space, a node might need to cover different tile sets in a single round. Supposing a node needs to cover M different tile sets, the final node schedule S is:

$$S = \underset{i=1}{\overset{M}{OR}}(S_i)^*. \tag{6}$$

5) Support Differentiated/Robust Surveillance: Differentiated surveillance [8] can be supported easily by uScan, due to its set-cover based approach. Instead of turning on one set of nodes to cover a column/row, uScan can turn on multiple disjoint set of nodes to increase the degree of coverage. This leads to a higher detection confidence, but at the cost of network lifetime. Similarly, fault tolerance can be achieved by turning multiple sets on. It is interesting to emphasize that nodes actually have no concept of set, which leads to a nice property for fault tolerance: To fix the failure of nodes, we only need to modify the schedule bits S of the nodes in the neighborhood of failed node and no coordination between nodes is needed.

IV. Design Analysis

Different from full coverage algorithms in [7], [22], uScan covers only a part of a network. On one hand, this approach significantly increases the network lifetime, but on the other hand, it introduces a certain delay in target detection. In this section, we provide analytic results on the performance of uScan. Here, we focus on the tile-level analysis instead of the node-level. Let's consider an area with N by N tiles.

A. Detection Delay for Static Targets

To evaluate the detection delay for static targets, we assume that a target is randomly located in an area and is detected after a neighboring node turns on for $\frac{1}{R}$ seconds. In line and systolic scan, in order to guarantee detection, a tile must be turned on once per round. The minimal detection delay happens when a target shows up in a tile right before this tile is turned on. In this case, the detection delay is $\frac{1}{R}$. The maximum detection delay happens when a target shows up in a tile right after this tile is turned on. In this case, the detection delay is $\frac{1+N}{R}$ for line scan and $\frac{1+\lceil N/2\rceil}{R}$ for systolic scan. Since the delay is uniformly distributed in a round, the expected delay is $\frac{2+N}{2R}$ for line scan and $\frac{2+\lceil N/2\rceil}{2R}$ for systolic scan. Under the same configuration, the detection delay for full coverage algorithms is zero. To reduce the detection delay in uScan, we can divide a network into sub-networks, where multiple line scans and systolic scans are executed in parallel.

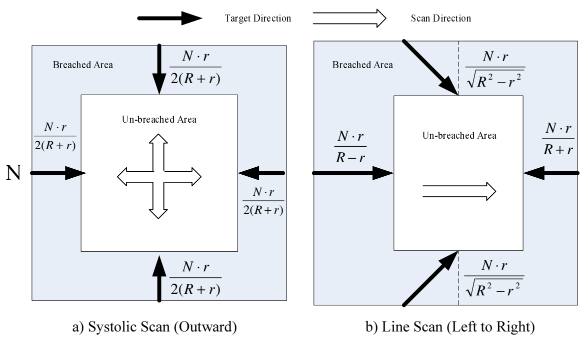
Target Direction
Scan Direction
Breached Area
Un-breached Area
N
$\frac{N \cdot r}{2(R+r)}$
$\frac{N \cdot r}{\sqrt{R^2-r^2}}$
$\frac{N \cdot r}{R-r}$
$\frac{N \cdot r}{R+r}$
a) Systolic Scan (Outward)
b) Line Scan (Left to Right)

Fig. 10. The Breached Area

B. Breached Area for Mobile Targets

In a full coverage scenario, the worst-case breach area is zero. A mobile target is detected once it enters into the area. In the scanning approach, a target would reach a certain portion of the area before it is detected. We define the largest percentage of the area that a target can reach without being detected as the Worst-Case Breach (WCB). A smaller WCB indicates a better performance in mobile target detection. To calculate WCB, we assume the following mobility model: A target can only enter from outside of the network, and the maximum speed of any target is r tiles per second.

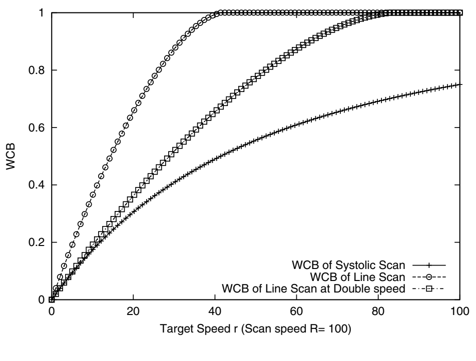

Fig. 11. Performance Comparison

The worst case breach scenario for systolic scan (outward direction) is shown in Figure 10(a). In this scenario, the worst-case breach happens when a target enters at the beginning of the scan round. The distance this target can breach without detection is $\frac{Nr}{2(R+r)}$. Therefore, the $WCB_s(r, R)$ for systolic scan under the target speed r and switching R is:

$$WCB_s(r, R) = \frac{(2R + r)r}{(R + r)^2}. \tag{7}$$

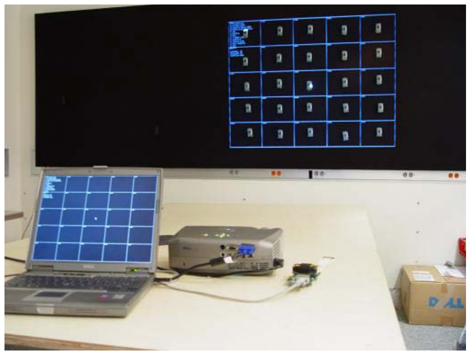
Fig. 12. uSense System Setup

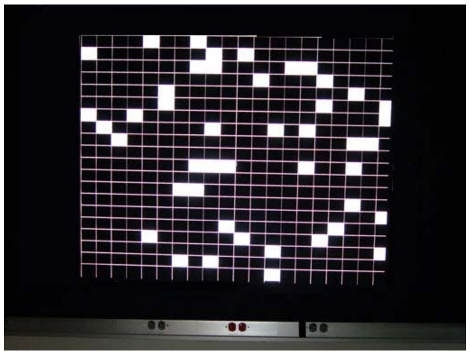
Fig. 13. Injecting Events

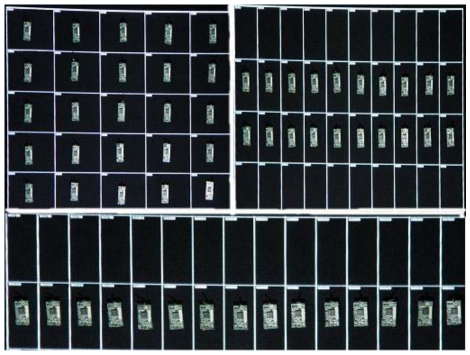
Fig. 14. 5×5, 2×10, 1×15 Layouts

The worst case breach scenario for line scan is shown in Figure 10(b), which has four more sophisticated cases:

1) If a target enters from the left edge (the same direction as the scan), the distance this target can breach is $\frac{Nr}{R-r}$.
2) If a target enters from the right edge (opposite direction as the scan), the distance this target can breach is $\frac{Nr}{R+r}$.
3) If a target enters from the top or bottom edge. In order to achieve the maximum breach, a target should enter with an angle $\alpha = \arctan(\frac{\sqrt{R^2-r^2}}{r})$. In this case, the maximum distance that this target can breach is $\frac{Nr}{\sqrt{R^2-r^2}}$.
4) If a target has a speed of $(\sqrt{2}-1)R$ or greater, it can enter the area from the left to reach at least $\frac{1}{\sqrt{2}}$ percent of the area and it can also enter the area from the right to reach at least $1-\frac{1}{\sqrt{2}}$ percent of the area. Therefore, the whole area is breached.

By combining these four cases, we get $WCB_l(r,R)$ for line scan , assuming target speed r and switching R:

$$WCB = \begin{cases} \frac{2Rr}{R^2-r^2} + \frac{2r}{\sqrt{R^2-r^2}}(1-\frac{2Rr}{R^2-r^2}) & r < (\sqrt{2}-1)R \\ 1 & r \geq (\sqrt{2}-1)R \end{cases} \tag{8}$$

Now we are ready to compare two global scheduling algorithms. As shown in Section III-B.2, for a given switching rate R, systolic scan consumes twice as much energy as line scan does. To obtain a fair comparison, we thus double the switching rate of line scan. By comparing WCB_l and WCB_s, it is easy to prove that $WCB_l(r,2R) \geq WCB_s(r,R)$ at all target speeds. In other words, systolic scan is better than line scan in terms of minimizing the breaching area. Actually we have proven that systolic scan is an optimal scanning algorithm in terms of preventing area breach when the target speed r is very fast. Due to the space constraint, we omit the detailed proof here. On the other hand, the line scan algorithm has its own advantages. As we have shown in Section III-C.2, we are able to obtain optimal set-cover results for line scan within a polynomial time. To illustrate the difference further, Figure 11 shows the WCB values under different target speeds r (0-100), when the switching rate R is 100. Clearly, the difference is significant. For example, when the target speed is half of scanning speed(r=50), systolic scan protects about half of the area, while line scan cannot protect any portion of the network.

V. Implementation and Evaluation

We have implemented a complete version of uSense (with uScan) as designed in Section II and III.

- **The generic switching algorithm** is written with the NesC language, running on the TinyOS/Mote platform. The compiled image of a full implementation occupies 21,040 bytes of code memory and 907 bytes of data memory. A simple timer-driven FA logic is implemented to turn a mote on/off according to the schedule bits. We use FTSP [23] for the purpose of time synchronization among motes and Deluge [16] for the purpose of wireless reprogramming. The synchronization accuracy is at tens of microseconds, which is sufficient for most sensing scheduling algorithms.
- **The scheduling algorithms** are written in Java, runs on a laptop. Since sensor nodes have no concept of scheduling, the global scheduling algorithm uScan and other coverage algorithms are written only in Java. The schedule bits S and the switching rate R are disseminated from the base, using a single packet. We also implement an evaluation engine using Java, which generates virtual targets using the light. To accurately measure the delay, we implement an NTP-like two-way handshaking synchronization protocol over a serial cable to synchronize the base mote and laptop. This synchronization protocol is not part of uSense and is only used for the evaluation purpose.

As shown in Figure 12 to evaluate uSense and uScan, we attach 25 MicaZ motes on a veltex board (4 feet by 12 feet) using velcro straps. We use a DELL 2300MP projector to generate light spots on the veltex board. These light spots are used to emulate static and mobile events. For example, the static events are randomly generated in the grid to trigger the detection, as shown in Figure 13. After the nodes detect the light events, they report timestamps to the laptop, where the delay is calculated.

As shown in Figure 14, three grid layouts are used in the experiments: 15 by 1, 10 by 2 and 5 by 5. In addition, we evaluate uScan with random placement as well. The locations of nodes in the random placement are obtained through a random generator. Each node is assigned an energy budget (the number of times a node can be turned on), which is used to evaluate the lifetime of the sensors. Events are repeated hundreds of times to obtain results with high statistical confidence.

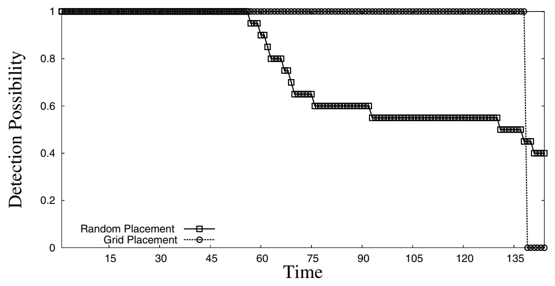

Fig. 15. Detection Over Time under Different Node Placements

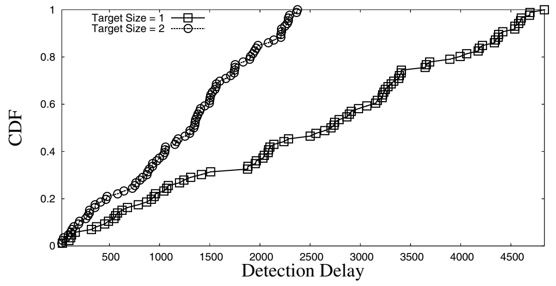

Fig. 16. Detection Delay of for Static Targets under Different Target Sizes

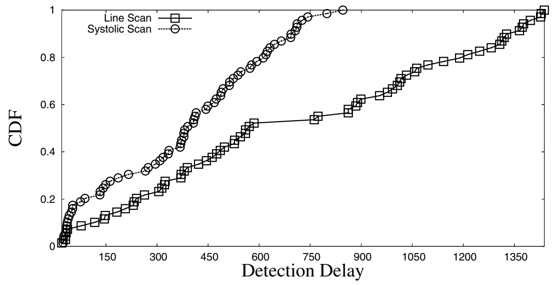

Fig. 17. Detection Delay of Line and Systolic Scan for Static Targets

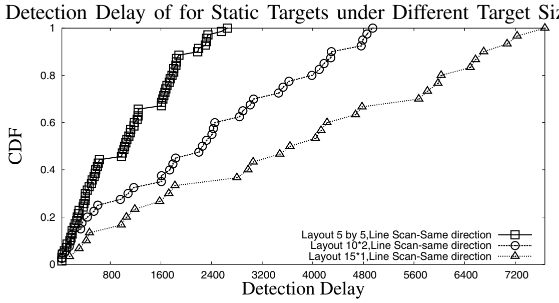

Fig. 18. Detection Delay of Line Scan for Mobile Targets

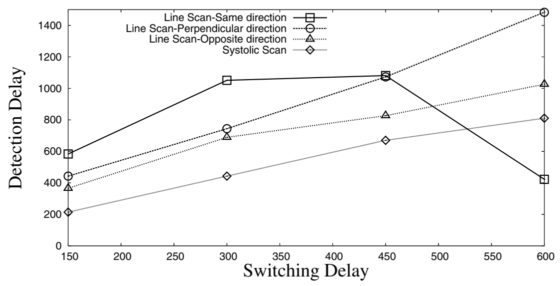

Fig. 19. Detection Delay Under Different Switching Delays

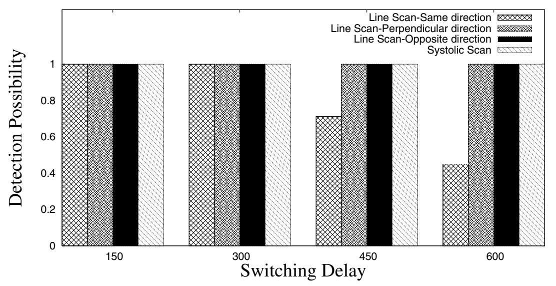

Fig. 20. Detection Probability Under Different Switching Delays

A. Testbed Evaluation

The system evaluation focuses on the detection delay for static and mobile targets, the detection probability for mobile targets and the node lifetime. All the experiment are repeated 10 times.

1) Detection Probability Over Time: In this experiment, we inject static targets through the DELL projector into the network to evaluate the detection probability over time. A target is missed only if a tile is not covered (i.e., a node runs out of power). We test uScan under the random placement and grid placement using 10 MicaZ motes. The results are shown in Figure 15. Since nodes in the grid placement have well balanced duty-cycles, they provide full coverage until all of them run out of energy simultaneously. In the random placement, the sparser areas become uncovered first, and coverage degrades gradually over time. For example, random placement still keeps about 40% coverage when coverage reduces to zero in the grid placement.

2) Detection Delay for Static Targets: In this experiment, we investigate the detection delay for static targets under different minimum target sizes. Static targets are injected at random time intervals into the area. Since the tile size is determined by the minimum target size, the number of columns needed to cover the same area reduces when the minimum target size increases, therefore the detection delay reduces as well. Figure 16 shows the Cumulative Density Function (CDF) curves of the delays for two target sizes. Clearly, a larger target size leads to smaller delays. For example, when the target size is one, the maximum delay on detection is $4834ms$, while the maximum delay is $2378ms$ with a target size of two.

3) Comparison of Detection Delay: In this experiment, we use 25 MicaZ to form a 5 by 5 grid and compare the detection delay of static targets for line and systolic scan, again using CDF curves. From Figure 17, we can see that under the same switching rate, the detection delay of systolic scan is about one-half of line scan, which is consistent with the length of schedule bits shown in Section III-B. In the 5 by 5 grid layout, the average detection delays for systolic and line scan are $380ms$ and $706ms$.

4) Impact of the Network Size and Scan Direction: In this experiment, we study the impact of the network size using three network layouts. The target moves in the same direction as line scan. As shown in Figure 18, as the network size reduces, the detection delay decreases accordingly. For example, the average detection delays for the 1 by 15, 2 by 10 and 5 by 5 layouts are $3758ms$, $2284ms$ and $1051ms$, respectively. This indicates that to guarantee a certain detection delay, we should partition a large area and perform scans within the sub-areas.

5) Impact of the Switching Delay and Scan Direction: Figure 19 studies the detection delay under different switching delays (the reciprocal of the switching rate R) and different target directions. This investigation is intended to reveal the importance of designing fast hardware and detection algorithms. We use a 5 by 5 layout, generate mobile targets from three different directions, and measure the delays before the mobile targets are detected. Figure 19 shows four curves, representing three target moving directions on line scan and one direction on systolic scan. Systolic scan has the smallest detection delay at all switching rates. Line scan in the opposite direction of a target moving direction provides the second smallest detection delay. The longest delay happens when we scan at the same direction as the target moving direction.

In addition, Figure 19 shows that, generally, when the switching delay increases, the detection delay increases linearly. Interestingly, there are two data points that do not follow this trend. It is because that line scan misses the targets when the scan speed is below twice the target moving speed (when both move in the same direction). In our setup, this happens when the switching delay is longer than 300ms. Under these slow scanning speeds, uScan may miss targets and record only the short detection delays, leading to a small average detection delay. This is also confirmed by the detection probability results in Figure 20, which indicate when targets move oppositely to the direction of line scan, we can ensure the 100% detection of mobile targets. However, if the scanning direction is the same as the target moving direction, the detection probability drops to 45% at the long switching delay of $600ms$.

VI. Large Scale Simulation: Comparing with State-of-the-Art

Experiments on the test-bed indicate that uSense can be efficiently implemented on the resource constrained devices and reveal its nice features. In this section, we compare the performance of uScan with several state-of-the-art sensing coverage algorithms integrated into uSense architecture, validate the benefit of asymmetric sensing architecture and the two-level scheduling approach.

In this simulation, up to 10,000 sensor nodes are randomly distributed in a 300m$\times$300m square field. The sensing range is 10m. The sensor nodes are deployed with a random distribution into the square field. To avoid performance distortion due to the edge effect, we set the coverage area as the 290m$\times$290m square in the center of the square field and do statistics on the central 275m$\times$275m field. The following baselines and bounds are adopted for purposes of comparison:

1) **Baseline-I: All-Working:** The full coverage mode with all nodes on.
2) **Baseline-II: DiffSurv:** Differentiated Surveillance for sensor networks proposed in [8] integrated into the uSense architecture.
3) **Baseline-III: Virtual Patrol:** Coverage-oriented patrols using wireless sensor networks proposed in [24] integrated into the uSense architecture.
4) **Upper Bound-I:** Optimal full coverage using $2/\sqrt{27}r^2$ circles with a honeycomb layout.
5) **Upper Bound-II:** Ideal upper bound for line scan, which assumes that nodes are placed optimally such that $|MCS_i|$ for each time interval is minimized.

All the experiments are repeated 100 times with different random seeds and node deployments. The 95% confidence intervals are within 1$\sim$15% of the mean, which is not plotted for the sake of legibility. The following metric is used to evaluated the performance of uScan.

- **Network Half-life:** We define the half-life of a sensor network as the time from the beginning of the deployment until exactly half of the nodes are still alive. This metric indicates the energy efficiency of the network as a whole.

A. Performance under Full Coverage Mode

If the tile set TS to be covered includes all the tiles in the network, uScan essentially provides a full coverage. In this experiment, we compare uScan with other solutions under Full Coverage Mode to evaluate the effectiveness of the set-cover approach in uScan. Figure 21 shows the network half life of four different solutions: uSense, DiffSurv, all-working lower bound and theoretical upper bound (by assuming a honeycomb layout). From Figure 21, we can see that the half lives for all cases are increasing linearly when the node density increases. The slope of uSense is larger than DiffSurv, which implies that as the node density increases, the difference in half-life between uSense and DiffSurv increases as well. For example, at the node density 1, the half life for DiffSurv and uSense are 1.19 and 1.35, respectively. And when the node density reaches 4, the corresponding two half lives are 4.91 and 6.99 and the half-life difference increases from 13.4% to 42%.

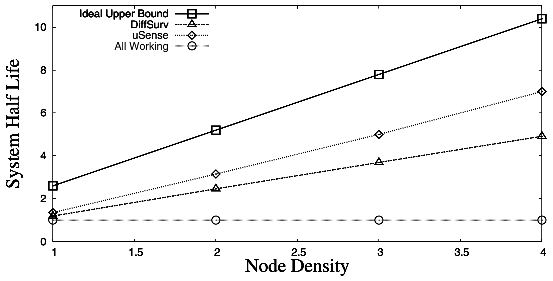

Fig. 21. System Half Life vs. Node Densities

B. Performance under Scanning Mode

In [24], Gui and Mohapatra propose a virtual patrol solution similar to our line scan scheme. The main difference between virtual patrol and uScan is that virtual patrol only provides single-level of scheduling. At each time, if a node's distance to the patroller is within its sensing range, the node is active; otherwise it is in sleep state. Figure 22 shows the half life of the line scan for uSense, virtual patrol and ideal upper bound. From the Figure we can see that as the node density increases, the system half life of the uSense increases almost linearly. On the contrary, the half life of the virtual patrol remains steady and cannot take the advantage of the increased node density. When the node density reaches 10, the half life of the uSense is 379.88, while the virtual patrol has a half life of 13.99, which is about 27 times energy inefficiency than the uSense. The major reason for such large performance gap is that virtual patrol activates all the nodes can sense the patroller, while the uSense only use a minimized subset of the eligible nodes to provide the desired coverage.

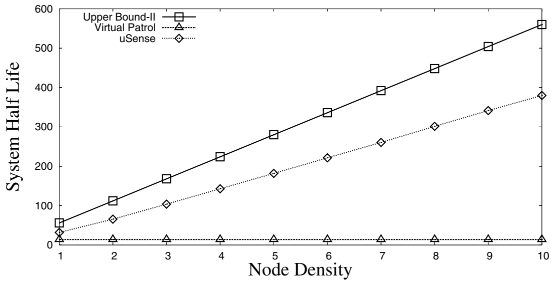

Fig. 22. System Half Life vs. Node Densities

VII. Related work

Physical sensing coverage is the research focus in the beginning. In [7], authors support full surveillance coverage based on an off-duty eligibility rule. DiffSurv [8] provides differentiated surveillance to an area with a certain degree of coverage. In [22], surveillance coverage is achieved through probing. Several other works focus more on the theoretical results of sensing coverage. Kumar et al. [5] identify a critical bound for k-coverage in a network, assuming a node is randomly turned on with a certain probability. In [25], Kumar et al. investigate the k-barrier coverage problem, identifying the critical condition for weak k-barrier coverage. Several algorithms are designed based on the concept of set cover. In [4], Cardei et al. propose two heuristic algorithms to identify a maximum number of set covers to monitor a set of static targets at known locations. In [3], Abrams et al. propose three approximation algorithms for a relaxed version of the previously defined SET K-COVER problem [6].

To achieve a higher energy efficiency, several recent works focus on the partial coverage within a fixed time delay. In [9], nodes coordinate to guarantee the worst-case detection delay and to minimize the average detection delay. Another type of temporal guarantee is to analyze the detection delay in the context of tracking. In [13], [14], [24], they assume the network is partially covered and provide a theoretical analysis and simulation on the delay before a target is detected.

Our work is unique in the following aspects: (i) uSense is a unified architecture instead of an individual solution. We are the first to propose the concept of generic switching. (ii) uScan demonstrates a novel two-level global scheduling method that can significantly reduce energy consumption. (iii) Our set-cover is uniquely implemented at the second level, allowing the first-level optimization. For example, we demonstrate optimal cover sets can be obtained in linear time when the coverage area is a continuous curve. In contrast, single-level set-cover approaches [4], [3] could be inefficient under non-uniform node distribution.

VIII. Conclusion

In this work, we propose a unified sensing architecture called uSense. It features an asymmetric design, which supports various kinds of coverage algorithms with a simple generic switching algorithm in sensor nodes. It allows us to flexibly change coverage algorithms with only two parameters. Another major contribution of this work is a two-level global scheduling algorithm called uScan, which is seamlessly supported by the uSense architecture. In the first level, we propose an optimal scheduling algorithm in terms of minimizing area breach. In the second level, we propose a linear algorithm to address the set-cover problem when the layout of tiles satisfies certain conditions. We have invested significant effort to evaluate our design, which includes a network of 30 MicaZ motes, an extensive simulation with 10,000 nodes, as well as theoretical analysis. With three novel ideas: *Asymmetric Architecture*, *Generic Switching* and *Global Scheduling*, our work has successfully achieved flexibility and efficiency for the sensor network coverage problem.

Acknowledgement

This work was supported in part by NSF grant CNS-0626614 and General Dynamics Research Fellowship.

References

[1] A. Arora and et al., " A Wireless Sensor Network for Target Detection, Classification, and Tracking," *Computer Networks (Elsevier)*, 2004.

[2] R. Szewczyk, A. Mainwaring, J. Anderson, and D. Culler, "An Analysis of a Large Scale Habit Monitoring Application," in *SenSys'04*, 2004.

[3] Z. Abrams, A. Goel, and S. Plotkin, "Set K-Cover Algorithms for Energy Efficient Monitoring in Wireless Sensor Networks," in *IEEE IPSN*, 2004.

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

[5] S. Kumar, T. H. Lai, and J. Balogh., "On k-coverage in a Mostly Sleeping Sensor Network," in *Mobicom*, 2004.

[6] S. Slijepcevic and M. Potkonjak, "Power Efficient Organization of Wireless Sensor Networks," in *IEEE ICC*, 2001.

[7] D. Tian and N. Georganas, "A Node Scheduling Scheme for Energy Conservation in Large Wireless Sensor Networks," *Wireless Communications and Mobile Computing Journal*, 2003.

[8] T. Yan, T. He, and J. A. Stankovic, "Differentiated Surveillance Service for Sensor Networks," in *SenSys'03*, November 2003.

[9] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards Optimal Sleep Scheduling in Sensor Networks for Rare Event Detection ," in *IPSN 05*, 2005.

[10] C.-F. Chiasserini and M. Garetto, "Modeling the performance of wireless sensor networks," in *IEEE INFOCOM*, 2004.

[11] C.-F. Hsin and M. Liu, "Network Coverage Using Low Duty-Cycle Sensors: Random & Coordinated Sleep Algorithms," in *IPSN'04*, 2004.

[12] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks," in *Sensys'03*, November 2003.

[13] C. Gui and P. Mohapatra, "Power Conservation and Quality of Surveillance in Target Tracking Sensor Networks," in *MobiCom'04*, 2004.

[14] S. Ren, Q. Li, H. Wang, X. Chen, and X. Zhang, "Analyzing Object Tracking Quality under Probabilistic Coverage in Sensor Networks," *ACM MC2R*, vol. 9, no. 1, 2005.

[15] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler, "The tenet architecture for tiered sensor networks," in *SENSYS*, 2006.

[16] J. Hui and D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," in *SenSys*, November 2004.

[17] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," in *Proc. of MOBICOM*, August 1999.

[18] C. Y. Wan, A. T. Campbell, and L. Krishnamurthy, "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks," in *WSNA '02*, 2002.

[19] V. T. Paschos, "A Survey of Approximately Optimal Solutions to Some Covering and Packing Problems," in *ACM Computing Surveys*, June 1997.

[20] *Kolmogorov-smirnov test.*, E.W. Weisstein. MathWorld, 2004, http://mathworld.wolfram.com/Kolmogorov-SmirnovTest.html.

[21] R. Williams, *Geometrical Foundation of Natural Structure:A Source Book of Design*. Dover Publications Inc, New York, 1979.

[22] F. Ye, G. Zhong, S. Lu, and L. Zhang, "PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks," in *ICDCS*, May 2003.

[23] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The Flooding Time Synchronization Protocol," in *SenSys'04*, 2004.

[24] C. Gui and P. Mohapatra, "Virtual patrol: a new power conservation design for surveillance using sensor networks," in *IPSN*, 2005.

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