Achieving Realistic Sensing Coverage in Wireless Sensor Networks

Joengmin Hwang, Tian He, Yongdae Kim
Department of Computer Science, University of Minnesota, Minneapolis

1 Introduction

Despite the well-known fact that sensing patterns in reality are highly irregular, researchers continue to develop protocols with simplifying assumptions about the sensing. For example, a circular 0/1 sensing model [2] is widely used in most existing simulators and analysis. While this model provides high-level guidelines, it could cause wrong estimation of system performance in the real world.

Our answer to this issue is a sensing area modeling technique, which obtains the coverage of sensor nodes through event training. The main idea is using *discrete controlled events* to identify the sensing coverage based on event detection results by individual sensor nodes. A key architectural advantage of this approach is a lightweight design in sensor node with minimal overhead. Besides communication, each sensor node only needs to support a simple detection function (with optional time synchronization requirement). We evaluate our model using a physical experiment on a testbed consisting of 40 MicaZ motes. Our evaluation results shows we can identify the sensing area with low error rate with simple training method.

2 Sensing Area Modeling

In this section, we introduce the design of our system at the architectural level.

2.1 Assumptions

We assume that we can generate controlled events with known time and location. This can be done through two approaches. First, events can be generated by using real targets. For example, one or multiple robots can move along predefined traces to activate PIR motion sensors in the field. Other events, such as vibration and sound, can be generated similarly. Second, the controlled events can be injected using special devices such as infrared radiation generator, Spotlight [1] and our prototype system described. Since the methods to generate controlled events are diversified, we intentionally describe our approach conceptually independent of the concrete method used. The targeted application scenarios are 1) to identify the coverage of motion sensors with a room, 2) to discover blind spots in a surveillance area, and 3) to compensate the irregularity of tiny proximity sensors used for paper-edge detection in printers.

2.2 Main Idea

The main idea of our training-based modeling approach is to generate *controlled events* in the space where the sensor nodes are deployed. An event could be, for example, the presence of an object in an area or a light spot projected on a plate of sensors. Formally, an event can be defined as a detectable phenomenon $e(t,p)$ that occurs at time t and at location $p \in A \subset \mathbb{R}^k$ $(k = 1,2,3)$. Without loss of generality, we use $k = 2$ in the rest of the paper. An event is said to be *controlled* if we can enforce a relationship between the time t and location p. In other words, a set of controlled events can be described as the event locations over the discrete time: $G : \mathbb{R} \rightarrow \mathbb{R}^2$, where $G(t) = p_t = (x_t, y_t)$ where $t \in \{t_1, t_2, ..., t_n\}$.

It consists of two major parts: an event generator G and a set of sensor nodes n_i $(i \in N)$. The event generator G could be a single device or multiple distributed devices that can generate a sequence of controlled events $e(t,p)$ with known spatiotemporal correlation $G(t) = p(x_t, y_t)$. We define $S_i(t,p)$ as the detection function of node n_i. If node n_i can detect event $e(t,p)$, $S_i(t,p) = 1$, otherwise $S_i(t,p) = 0$. In case of detection, sensor nodes store the timestamp t locally. By the end of training, a sensor computes the location of all events it detects by inputting the timestamps into $G(t)$. Therefore, a set of timestamps $T_i = \{t_1^i, t_2^i, \ldots, t_n^i\}$ stored in node n_i can be converted to a set of locations $P_i = \{p_1^i, p_2^i, \ldots, p_n^i\}$ within the sensing area. The location set P_i can be directly used to describe the sensing area of node n_i.

2.3 Design of Event Generator $G(t)$

Since the overhead and accuracy of the sensing modeling is largely determined by $G(t)$, it is important to consider several solutions to optimize $G(t)$ under different system configurations.

2.3.1 Regular $G(t)$

To illustrate the basic functionality of an event generator, we start with a simple sensor system where the sensing area of a node is a line segment. We shall find out the portion of the line included in the sensing ranges of sensor node n_1 and n_2. To achieve this, the event generator creates discrete point events along this line $[0, L]$ with constant speed v with same interval D. Formally, $G(t) = t \cdot v$ where $t = kD/v$ and $0 \leq k \leq L/D$. For example, a sensor node n_1 collects a set of six timestamps $T_1 = \{t_1, t_2, \ldots, t_6\}$ at which the events are detected. From function G, the actual locations of events are converted to a set of locations $P_1 = \{t_1 v, t_2 v, \ldots, t_6 v\}$. The sensing coverage of sensor n_1 can be defined as the line segment that covers P_1. For the sensor n_2, it reports timestamps $T_2 = \{t_4, t_5, t_6, t_7\}$ and the sensing coverage of sensor n_2 is defined as the line segment that covers $P_2 = \{t_4 v, t_5 v, t_6 v, t_7 v\}$. The intersection of T_1 and T_2, $T_1 \cap T_2 = \{t_4, t_5, t_6\}$ indicates the coverage of two sensors is overlapped. The regular training can be generalized to the case when the events occur in a plane by dividing the plane into several lines with a certain interval.

In addition to the progressive scanning, the $G(t)$ function of the regular training can generate events with an arbitrary sequence as long as every point in the area is covered. Also, as long as we can match the event and its position we can use any training method for $G(t)$ which includes unsupervised event.

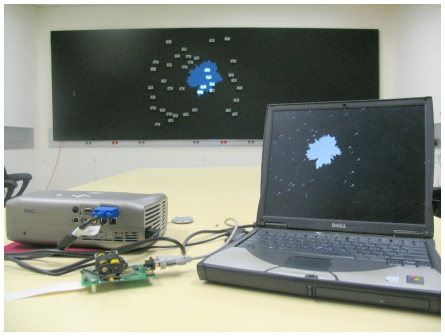

Figure 1. System Setup

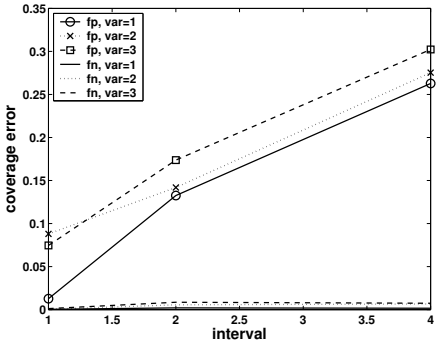

Figure 2. Errors in regular $G(t)$ with varying interval and irregularity

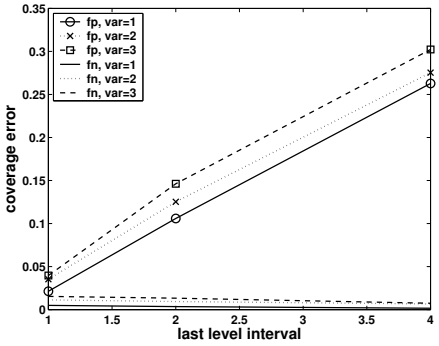

Figure 3. Errors in hierarchical $G(t)$ with varying interval and irregularity

Algorithm 1 Hierarchical G(t) process

Output: P_i: The sensing area of n_i.

1: $G(t)$ starts with level-1 events $e(t,p)$ (The number of level-1 event is decided by the minimum sensing area)
2: Node n_i reports $S_i(t,p)$ for all level-1 events
3: **repeat**
4: **for** all level-k adjacent pair $e(t_m,p_m)$ and $e(t_n,p_n)$ **do**
5: **if** any node detects only one of events && no event generated at position $\frac{p_m+p_n}{2}$ before **then**
6: Generate a level-$(k+1)$ event at position $\frac{p_m+p_n}{2}$
7: **end if**
8: **end for**
9: $k = k+1$
10: **until** (k = Maximum Level)
11: P_i is a set of positions p where $S_i(t,p) = 1$

2.3.2 *Hierarchical $G(t)$*

Hierarchical G(t) is motivated by the observation that the boundary area of a sensing coverage requires more detail than the area in the middle of coverage. With the hierarchical $G(t)$, we can reduce the number of events required to obtain the same accuracy as regular $G(t)$. A level-1 event divides the area into four sub-areas, and level-2 events divide the area into 16 sub-areas. In general, level-i events divide an area into 4^i sub-areas. If an event is a level-i event, it is also a level-j event, where $j \geq i$. Two events are said to be *adjacent*(or a pair) if they are neighboring each other vertically, horizontally or diagonally (e.g., an event could have maximal eight adjacent events). Two adjacent events are said to be a *boundary pair* if only one of two adjacent events is within a sensing range of some node. form a boundary pair). The event in the boundary pair is called *boundary event*. The main idea of Hierarchical $G(t)$ is to *recursively generate new events in the middle of boundary pairs.* It works in a way similar to the binary search within a two-dimensional space. We describe the step by step operation of Hierarchical $G(t)$ in Algorithm 1.

3 System Implementation

We design and implement a complete version of our system which includes regular and hierarchical training on the TinyOS/Mote platform. The NesC language is used to program the motes and Java is used to build the regular and hierarchical generators. The compiled image of a full mote implementation occupies 14,500 bytes of code memory and 605 bytes of data memory. For each event we generate, we assign a unique ID. By using these IDs, we eliminate the need for time synchronization. We use an *oracle algorithm* that assumes the knowledge of the sensing area of the nodes. Basically, this algorithm activates a sensor node (e.g., through projecting light to a sensor), if the controlled event $e(t,p)$ is within the sensing area of the node. We want to emphasize that the oracle algorithm and generated ground truth is used *only for the purpose of evaluation*. This knowledge is not used in any part of our proposed algorithm. Figure 1 shows the implementation setting. After each run, the training results are visualized on the board and compared with the ground truth.

In experiments, we investigate the impact of training interval (resolution) and sensing irregularity in both training methods. We divide error into two types: (1) **false positive** fp**:** the area measured as a part of sensing coverage but is not a part of real sensing coverage, or (2) **false negative** fn**:** the area which is measured as not in the sensing coverage but is a part of real sensing coverage. Figure 2 and 3 shows the coverage error in regular training and hierarchical training respectively. The degree of irregularity of sensing coverage is denoted by *var* where higher value means more severe irregularity. Our result shows that with a small training interval, we can achieve very precise coverage modeling, fp is almost 0% and fn is at 1% to 8% and even if we increase the interval, fp is still almost 0%. The performance of hierarchical training is almost similar to the regular training, which means we can do an efficient training while saving the cost of event generation.

4 Conclusion

This paper intends to draw attention to the sensing irregularity issue known but largely ignored by many designers. We contribute to this area by designing and implementing two training-based methods that accurately identify the sensing patterns of nodes using controlled events. Our design has been fully implemented and evaluated in a test-bed consisting of 40 MicaZ motes. We hope this work motivates our community to seriously consider the reality issues existed in the sensor networks.

5 References

[1] R. Stoleru, T. He, J. A. Stankovic, and D. Luebke. High-Accuarcy, Low-Cost Localization System for Wireless Sensor Networks. In *Sensys'05*, November 2005.

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

Achieving Realistic Sensing Coverage in Wireless Sensor Networks

Joengmin Hwang, Tian He, Yongdae Kim
Department of Computer Science and Engineering, University of Minnesota
{jhwang, tianhe, kyd}@cs.umn.edu

Research Issue:

Despite the well-known fact that sensing patterns in reality are highly irregular, researchers continue to develop protocols with simplifying assumptions about the sensing. For example, a circular 0/1 sensing model is widely used in most existing simulators and analysis. While this model provides high-level guidelines, it could cause wrong estimation of system performance in the real world.

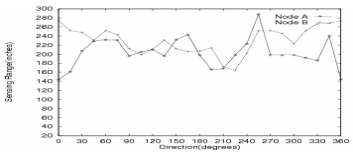

Fig 1. Directional range

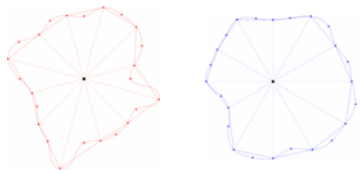

Fig 2. (1) Sensing area of node A (2) Sensing Area of node B

Project Overview

Our answer to this issue is a sensing area modeling technique, which obtains the coverage of sensor nodes through event training. The main idea is using discrete controlled events to identify the sensing coverage based on event detection results by individual sensor nodes. A key architectural advantage of this approach is a lightweight design in sensor node with minimal overhead. Besides communication, each sensor node only needs to support a simple detection function (with optional time synchronization requirement). We evaluate our model using a physical experiment on a testbed consisting of 40 MicaZ motes. Our evaluation results shows we can identify the sensing area with low error rate with simple training method.

Assumptions:

We assume that we can generate controlled events with known time and location. This can be done through two approaches. First, events can be generated by using real targets. For example, one or multiple robots can move along predefined traces to activate PIR motion sensors in the field. Other events, such as vibration and sound, can be generated similarly. Second, the controlled events can be injected using special devices such as infrared radiation generator, Spotlight and our prototype system described. Since the methods to generate controlled events are diversified, we intentionally describe our approach conceptually independent of the concrete method used. The targeted application scenarios are 1) to identify the coverage of motion sensors with a room, 2) to discover blind spots in a surveillance area, and 3) to compensate the irregularity of tiny proximity sensors used for paper-edge detection in printers.

Architecture:

We employ an architecture to model sensing coverage.

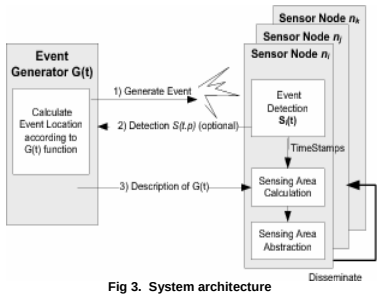

Fig 3. System architecture

Regular Training:

The sensing area of a node is a line segment. We shall find out the portion of the line included in the sensing ranges of sensor node n1 and n2. To achieve this, the event generator creates discrete point events along this line [0, L] with constant speed v with same interval D. Formally, G(t)=tv where t = kD/v and 0 < k < L/D. The sensing coverage of sensor n1 can be defined as the line segment that covers P1={t1 v, t2 v, … , t6v}.

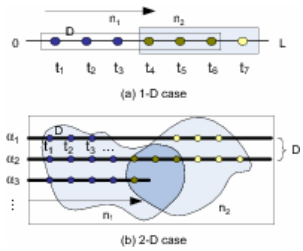

Fig 4. Regular training

Hierarchical Training:

With the hierarchical G(t), we can reduce the number of events required to obtain the same accuracy as regular G(t).

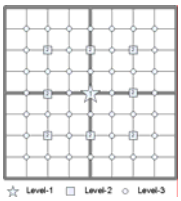

Fig5. Hierarchical partition

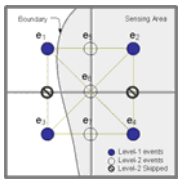

Fig 6. Level of detail

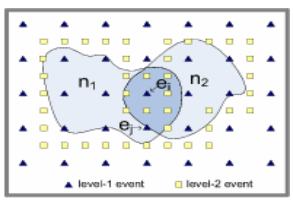

Fig 7. Hierarchical training

Implementation:

We have implemented a complete version of Regular Training and Hierarchical Training on Berkeley TinyOS/Mote platform, using 40 MicaZ motes as shown below.The compiled image of a full mote implementation occupies 14,500 bytes of code memory and 605 bytes of data memory.

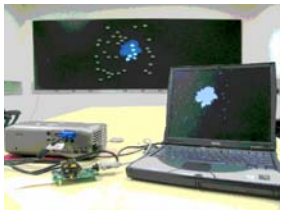

Fig 8. System Implementation

Evaluation:

We evaluate our architecture with physical system as well as an extensive simulation with 1,000 nodes.

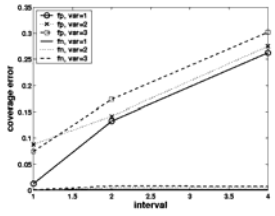

Fig 9. Error in Regular G(t) with varying interval and irregularity

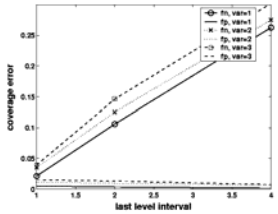

Fig 10. Error in Hierarchical G(t) with varying interval and irregularity

Conclusion:

This paper intends to draw attention to the sensing irregularity issue known but largely ignored by many designers. We contribute to this area by designing and implementing two training-based methods that accurately identify the sensing patterns of nodes using controlled events. Our design has been fully implemented and evaluated in a test-bed consisting of 40 MicaZ motes.

Acknowledgements:

This work is supported by NSF Nets NOSS Program
More information can be found at
Minnesota Embedded Sensor System Group
Http://mess.cs.umn.edu