역동적 이벤트 영역 탐색을 위한 에너지 절약형 분산 알고리즘

Nhu T.Anh, 나현숙
숭실대학교 컴퓨터학과 dkhtanh@gmail.com, hsnaa@ssu.ac.kr

Energy-Saving Distributed Algorithm For Dynamic Event Region Detection

T.Anh Nhu, Hyeon-Suk Na
Department of Computer Science, Soongsil University

Abstract

In this paper, we present a distributed algorithm for detecting dynamic event regions in wireless sensor network with the consideration on energy saving. Our model is that the sensing field is monitored by a large number of randomly distributed sensors with low-power battery and limited functionality, and that the event region is dynamic with motion or changing the shape.

At any time that the event happens, we need some sensors awake to detect it and to wake up its k-hop neighbors to detect further events. Scheduling for the network to save the total power-cost or to maximize the monitoring time has been studied extensively. Our scheme is that some predetermined sensors, called *critical sensors* are awake all the time and when the event is detected by a critical sensor the sensor broadcasts to the neighbors to check their sensing area. Then the neighbors check their area and decide whether they wake up or remain in sleeping mode with certain criteria.

Our algorithm uses only 2 bit of information in communication between sensors, thus the total communication cost is low, and the speed of detecting all event region is high. We adapt two kinds of measure for the wake-up decision. With suitable threshold values, our algorithm can be applied for many applications and for the trade-off between energy saving and the efficiency of event detection.

1. Introduction

Wireless sensor network (WSN) becomes an important area of many real world applications and research activities, e.g. continuous monitoring of environment or animals or event detection [2]. It offers the ability to monitor large scale areas in which wired networks maybe unavailable or infeasible to establish. The essential requirement for WSN is that the union of sensing area of all sensors covers the region of interest. In monitoring applications, sensor nodes regularly report sensory readings. Event detection applications, however, are concerned with detecting events and sensor nodes report data only when an event is detected. In this paper, we focus on event detection only in WSN. Many applications, such as re surveillance, gas leaking detection, pollution detecting and radiation prevention [1], concern only event detection.

An important issue in monitoring a region by sensors with low-power battery is energy saving. Scheduling for WSN to save the total power-cost or to maximize the monitoring time has been studied extensively. We only need some to be awake to detect the coming event while the others could reduce their function to recharge or save energy. But, inactivating sensors gives rise of less event detecting. There cannot be any perfect solution to achieve both the energy-saving and the efficiency of event detection. Many researchers [3, 4, 5, 6] have studied about event detection in WSN, focusing only on one aspect. In this paper we will provide a way to properly trade-off between them according to the situations.

In [6], the authors focused on energy-saving of the whole network. Each sensor node sleeps most of the time and wakes up for T_0 in every T time unit. While in active mode, sensor would detect if there is any event around. Their approach is good if we can choose a good time period, yet all sensors need to periodically wake up even if there is no event. In [4], the authors studied a good fault-tolerant distributed way of detecting event. The deviation of measuring values between event sensors and normal sensors is quite big, sensors could locally detect if they are neighbor of event sensors or not, therefore, an event boundary could be generated. Yet, in energy-efficiency, they require all the network to wake up all the time to keep track on the dynamic events. Zhu et al. [5] interests in creating a contour boundary of event regions which could dynamically change along with the events. Building this contour is a good way to study the topology of events, but this contour is easily broken if the boundary bandwidth is thin. Moreover, as in [4], in order to keep track of the event changing, they need all sensors to wake all

the time. And for us, with most of the sensors is sleeping, when an event is detected, there's a high chance that some other events occurs in nearby region, so, we want to expand the boundary more to nearby regions to detect these events. The most relevant research to ours is given in [3]. The authors select some special sensors, called tripwire sensors, to stay awake all the time to detect events, while the others are in sleeping. When an event is detected by a tripwire sensor, they will wake up other sensors and create connected component within the event regions. Our experiments for randomly distributed discrete events show that their approach catches only around 60% of all events.

Our scheme is that some predetermined sensors, called critical sensors, are awake all the time and when the event is detected by a critical sensor the sensor broadcasts to the neighbors to check their sensing area. Then the neighbors check their area and decide whether they wake up or remain in sleeping mode with certain criteria. For such wake-up decision, each sensor calculates how many neighbors are seeing the event now and how many neighbors are awake. Both are the indicators for how close the event is from the sensor, but the former is more for the present and the latter for the future of dynamic event change.

The rest of the paper is organized as follows: In Section 2, we describe our model of the network and the distributed algorithm. In Section 3, we explain for how to adapt our algorithm for many applications by adjusting the threshold values in the wake-up criteria. In Section 4, we discuss about our simulation and the experimental results.

2. Distributed algorithm to detect event regions

2.1. The model of the network

We partition the region of interest into many sub-regions, each covered by randomly distributed unit disks centered at point-sensors with a given density. Our sensors are identical 0-1 sensors with low-powered battery and limited memory just enough to perform simple arithmetic operations, and the sensing (resp. communication) area of one sensor is a disk of radius one (resp. r_c). We can either say that each sensor contains a certain predicate that outputs 1 if sensor reading is within some specific range of interest to us and 0 otherwise. We assume that the communication network generated by all deployed sensors is connected, so that some base station(s) or center(s) can collect information on detected event regions, and that each sensor has knowledge of the number of its neighbors within the communication network, but neither of its global location nor of the relative position or distance from its neighbors. We assume that sensors can stay in one of the two modes: SLEEP or WAKE, and between them there's an intermission state where the sensor is awaken temporarily, only to check if there's event within the sensing area and then to perform simple arithmetic operations to decide which mode it will move into. In SLEEP mode, the only functionality that a sensor performs is message receiving. In WAKE mode, a sensor monitors its disk all the time and broadcasts some messages whenever needed.

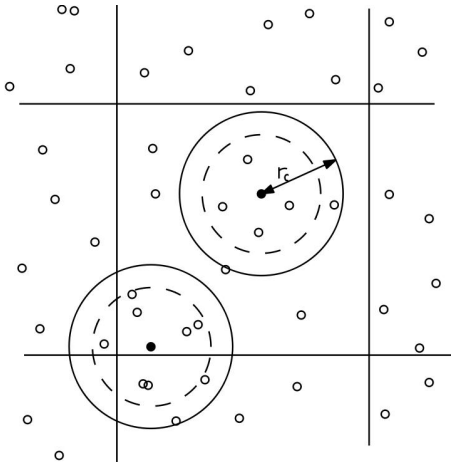


Figure 1: Illustration of our model of the network. The region of interest is divided into squares, each monitored by randomly distributed sensors (tiny circles). A circle in dashed line indicates the sensing area of radius 1 of a black sensor, and the circle in solid line indicates the communication area of radius r_c of that sensor.

Let S be the set of all deployed sensors in the region of our interest. We randomly choose fixed number of sensors in each sub-region, called critical sensors, and denote their set by S_c. To save the total energy-consumption, we make only the *critical sensors* monitoring their regions all the time, while the others in $S \setminus S_c$ are in sleeping mode which will be described in detail below. (We may cycle the monitoring role after grouping the critical sensors. Scheduling for the network to save the total power-cost or to maximize the monitoring time has been studied extensively.) Any sensor, say s_i , has limited memory only enough to maintain the following information and to perform basic arithmetic calculations:

- $w(i)$, called the *mode* of the sensor; 0 for SLEEP and 1 for WAKE

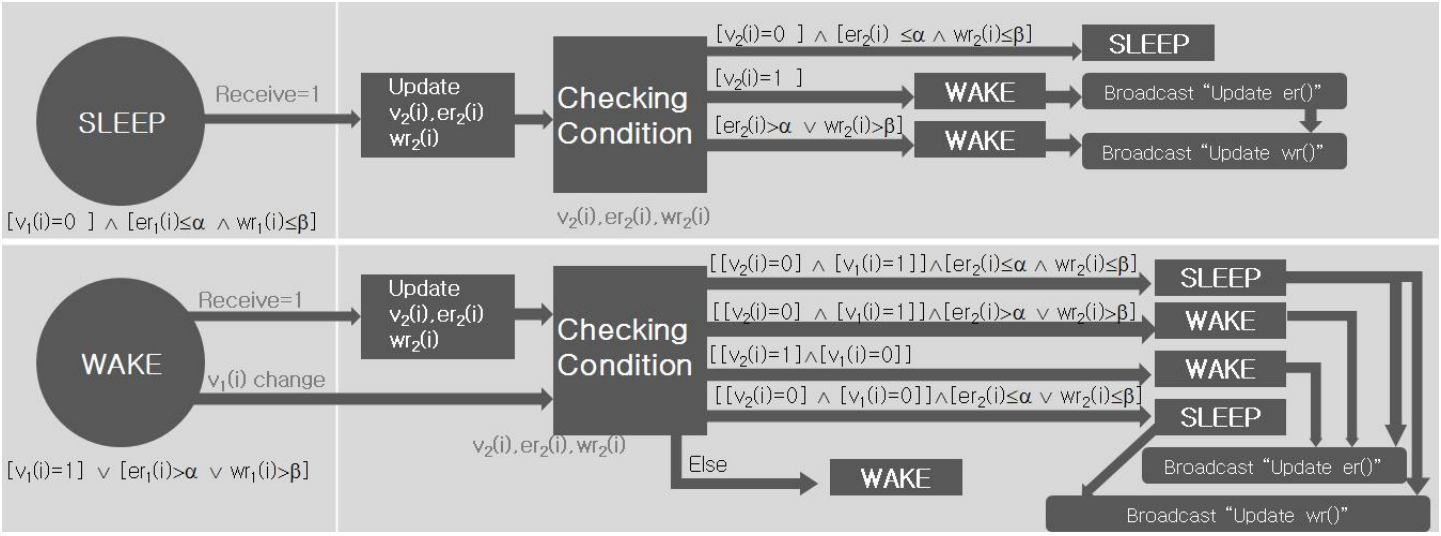


Figure 2: The algorithm run by sensors in the intermission state.

- $v(i)$, called the *event value* of the sensor; 1 if its reading is within the range of interest, and 0 otherwise. If $v(i)$= 1, we call this *event sensor.*
- $er(i)$, called the *event ratio*, which is defined as the ratio of the number of its neighboring event sensors over the total number of neighbors.
- $wr(i)$, called the *wake ratio*, which is defined as the ratio of the number of waking neighbors over the total number of neighbors.

2.2. The algorithm

Every sensors are either in SLEEP mode or in WAKE mode. As defined before, a sensor in SLEEP mode can only receive a message from the neighbors, and in WAKE mode it is in full functionality including monitoring its sensing area and broadcasting a message to the neighbors whenever needed. Whenever a sensor receives a message from their neighbor or it detects event in WAKE mode, the sensor enters into the intermission state and runs the algorithm shown in Figure 2. The algorithm runs in distributed way by individual sensors without any global information or centralized calculation.

Before describing our algorithm, we need to dene the condition for WAKE mode that every sensor checks in the intermission state.

Definition 1 (Condition) *Let* $0 \le \alpha, \beta \le 1$ *be the prescribed thresholds. The sensor si should be awake if one of the following conditions are satisfied:*

- *Event value v(i) is 1, which means that some event is detected around this sensor.*
- *Event ratio* $er(i) > \alpha$.
- *Wake ratio* $wr(i) > \beta$.

We denote by S the set of distributed sensors and by E the set of distributed events: $S = \{S_i\}$ and $E = \{E_j\}$. N_i is the set of neighbors of S_i, r_s is the radius of sensing, and r_c is the radius of

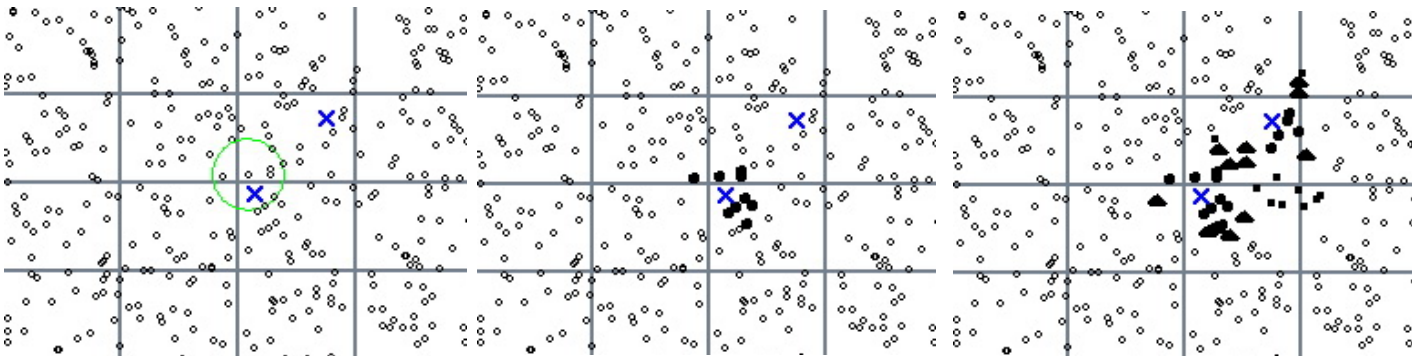


Figure 3: Illustration of the algorithm when $\alpha = 0.3$ and $\beta = 0.48$: Sensors are depicted by small circles, events by crosses, event sensors by black disks, sensors in WAKE mode by triangles (if its event ratio is greater than α) and by squares (if its wake ratio is greater than β).

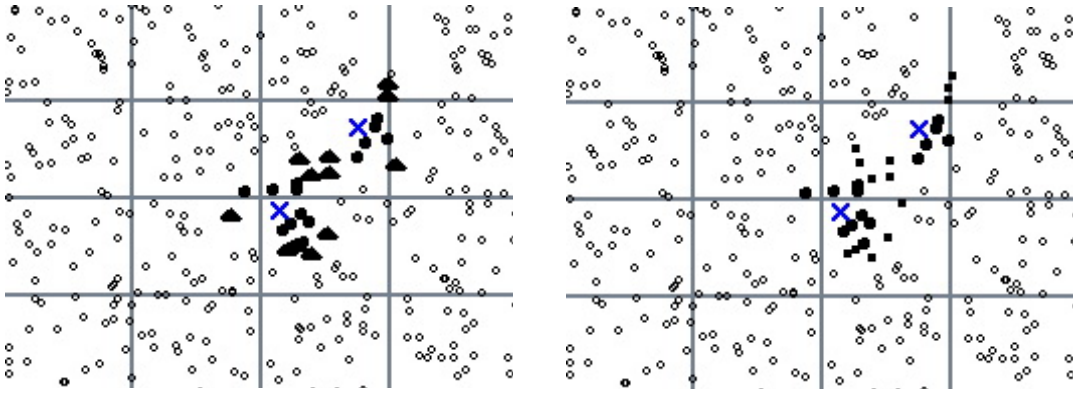


Figure 4: Illustration of adapting only $\alpha = 0.23$ by letting $\beta = 1$, or only $\beta = 0.44$ by letting $\alpha = 1$.

communication. We use notations $S_i(N_i, r_c, r_s)$ and $E_j(N_j, r_s)$.

The first condition implies that some event is being detected in the sensing area of this sensor, so it needs to be in WAKE mode. Q_E denotes event sensors : $Q_E = \{S_i \in S : d(S_i, E) \le r_s\}$

(1)The second condition tells us that more than α portion of the neighbors are detecting event now, so even if there is no event detected around this sensor, it needs to be awake since the event is nearby.

$$Q_{EN} = \left\{ S_i \in S \setminus Q_E : \left\{ \begin{pmatrix} d(S_i, E) > r_s \\ d(S_i, Q_E) \le r_c \end{pmatrix} \left\{ \left(1 \ge er(i) = \frac{\max\limits_{E_j \in E} (B(S_i, r_c) \cap B(E_j, r_s) \cap E)}{|N|} \ge \alpha \right) \right\} \right\} \right\}$$

(2)The last condition shows that more than β portion of the neighbors have been awake, so with high probability the event is coming this way even if the event has not been detected yet. Recursively, we define:

$$Q_0 = (Q_E \cup Q_{EN})$$

$$\forall k \ge 0,$$

$$Q_{k+1} = \left\{ S_i \in S : \left\{ \begin{pmatrix} d(S_i, E) > r_s \\ d(S_i, Q_k) \le r_c \end{pmatrix} \left\{ \left(1 \ge wr(i,k) = \frac{\max\limits_{0 \le j \le k} |N_j \cap Q_j|}{|N_i|} \ge \beta \right) \right\} \right\} \right\}$$

Now our algorithm is as follows (see Figure 2): if a sensor receives a message from their neighbor or it detects event in WAKE mode, the sensor enters into the intermission state and checks first the above condition. Depending on the result of checking the above conditions, the sensor moves to SLEEP or WAKE mode and broadcast a update message to the neighbors once if necessary. Broadcasting a message happens only when the event value or the mode of the sensor are changed. Whenever the event value $v(i)$ is changed, the sensor broadcasts a message to the neighbors s_j , asking for the update of their $er(j)$ and $wr(j)$. If the mode is changed with no change of the event value $v(i)$, the sensor broadcasts a message to the neighbors s_j , asking for the update of their $wr(i)$ only. These kinds of messages can be simply encoded into 2 bits, so this process costs very little computation.

Figure 3 illustrates the results of our distributed algorithm by steps, where $\alpha = 0.3$ and $\beta = 0.3$ are prescribed in all sensors. In every figures, sensors are depicted by tiny circles and the events by crosses. The leftmost figure illustrates the beginning of the event detection. There is only one critical sensor in the square, monitoring the area within the big circle. As soon as the event happens, it is detected by the critical sensor being always awake since it is within the sensing area of the sensor. Then the event value of the critical sensor is changed from 0 to 1, as shown in Figure 2, it broadcasts a message to the neighbors s_j , asking for the update of their $er(j)$. This message makes all neighbors move to the intermission state and detect their sensing area to update their event values. Thus seven neighbors of this critical sensor (depicted by black disks in the middle figure) become event sensors with the event value as 1. Next, the situation becomes as in the rightmost figure; around the first detected event, some sensors become WAKE mode because their event ratio $er(j)$ is greater than α (black triangles) and some other sensors become WAKE mode because their wake ratio $wr(j)$ is greater than β (black squares). Finally, these sensors wake up the sensors nearby the undetected event, and the sensors around the unknown event become event sensors and wake up their neighbors.

3. Control of thresholds α and β

Our algorithm wakes up not only the sensors detecting events but also those nearby the events.

Figure 4 shows how we can extend the bandwidth of waking sensors by smaller α or β. Depending on the situation, one can control these thresholds to adapt for many real applications.

Threshold β : $wr(i)$ is the ratio of awake neighbors among all neighbors, so the high this is, the more likely the event is not yet detected but will come shortly to this position. Also, controling β provides a trade-off between the efficiency of detection and the communication cost; If $\beta = 0$, then all the sensors become wake-up, and if $\beta = 1$, then only event sensor become wake-up with no extending further (which is exactly the way of event detection by all previous research.) When we apply $\beta = 1$, the upper-right event has not been detected before reducing α to 0.23 and we obtain the left image of Figure 4,

According to the formulation (2), right now there is no way to give an geometrically estimation, so I suggest using β along with the limit that the expansion only within k hops of communication. So even in the worst case, the expansion would not go beyond this k hops of communication.

Threshold α :

When we apply $\alpha = 1$, the upper-right event has not been detected before reducing β to 0.44 and we obtain the right image of Figure 4, Another usage of this threshold is that instead of setting β too low (which ends up with waking too many sensors and high cost), we can apply very low α to wake up most of the sensor around the event region. This is because, unlike to β, extension of wake-up sensors by setting α very low cannot reach far from the boundary of the event region.

According to (1), $1 \geq er(i) \geq \alpha$, in the worst case when there's only 1 event, we could estimate $er(i)$ by calculate the intersection part of 2 circle areas in Figure 5. This is a basic geometry calculation with the result is a function depends on d_{ij}, which is the distance of the event location and sensor location. To make a summary, we have $1 \geq f(d_{ij}) \geq \alpha$, this could give the conclusion that changing α would limit sensors within certain range from the event location.

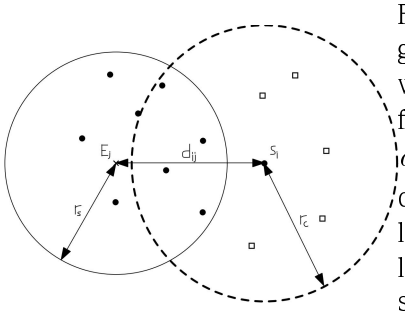
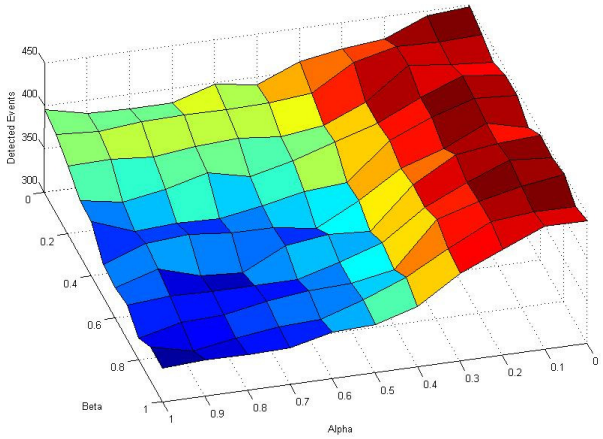
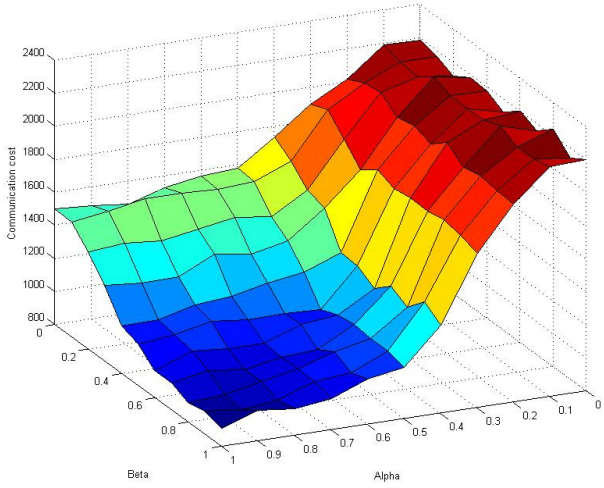


Figure 5: Determining event sensors neighbor

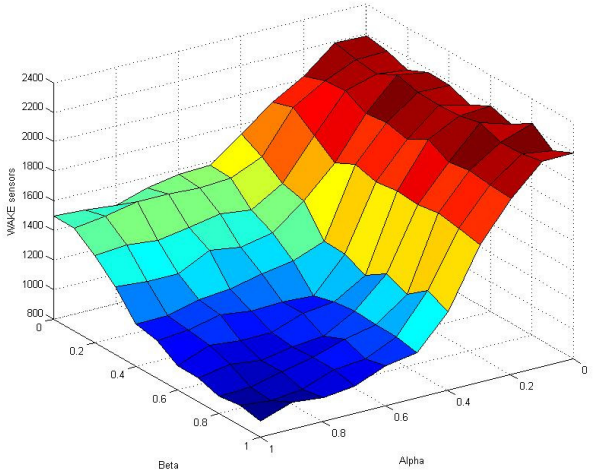
All previously studied boundary detection is based on the event sensors. We use two thresholds and combining and adjusting them provide a method to trade-off between the efficiency of detection and the communication cost.



(a) Detected events



(b) Communication cost



(c) WAKE sensors

Figure 6: Illustrate the trade-off between events detecting efficient, communication cost and WAKE sensors

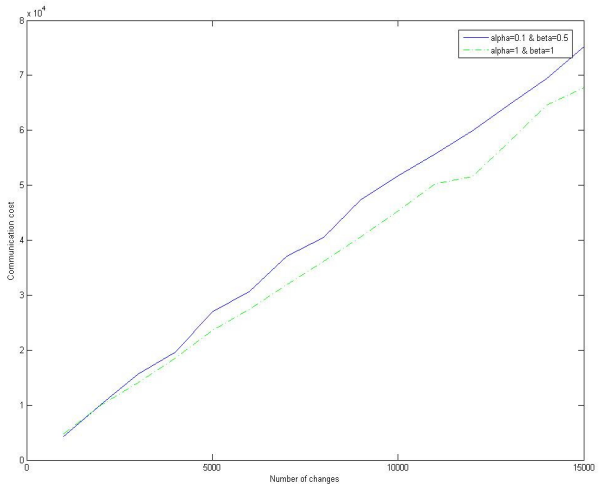


Figure 7: Illustration of the communication cost when event region changes

4. Simulations

In this section, we report our simulation results. We implemented the event detection algorithm of Section 2.2 in C++. We simulated on a network with more than 2500 sensors distributed on a field of size 1024 x 768 units. Each node has a communication radius as well as sensing radius as 20 units. Each value is the average over more than 100 runs. Based on the ratio of the detected events and the communication cost, one usually decides whether or not the detection algorithm will work well in practice. However, in this paper, we do not make any effort in trying to determine optimal values of communication cost and the ratio of detected events. A related issue will be addressed in our future work. We do not investigate the dependency of the communication cost on the density of sensors, since many previous works has already proved that with the increase of either the density sensors or events, the rate of events being catch is getting higher. Figure 6 shows the number of detected events with the variation of α and β. We could clearly see that the lower those threshold are, the more events would be caught.

As noted above, one WAKE sensor can send only 2 bits of message at most two times in the current WAKE mode, so the communication cost of building up event regions at any specific time is $O(4 * N)$ where N is the number of WAKE sensors. This implies that in dynamic situations, the communication cost for updating the event regions within some time interval is linear in the total number of sensors that have changed the mode during the time interval. To prove this, in each $k * k$ square, we randomly selected an event there and then change its coordinate. We calculated the total communication cost after some such change are made. We fixed to $\alpha = 0.1$ and $\beta = 0.5$. Then we compare the results with using $\alpha = 1$ and $\beta = 1$, which use only the event sensors. Figure 7 is the average of experiments.

5. Conclusions and Future works

In this paper we study the problem of event detecting with wireless sensor and propose an approach that is distributed and most likely energy-saving. We also provide 2 threshold to be applied in a more flexible situation to balance the trade-off between energy saving and effectiveness of event region detection.

For future works, we plan to take in to account the failure-tolerance property of WSN. Moreover, despite the fact that the 2 given thresholds provide the flexibility, we need to know, better about how to control them to get a good estimation of the event region detection.

References

[1] S.Brennan, A.Mielke, and D.Torney, “Radioactive Source Detection by Sensor Networks “ IEEE Transactions on Nuclear Science, vol. 52, 2005.

[2] P.Dutta, M.Grimmer, A.Arora, S.Bibyk, D.Culler, “Design of a wireless sensor network platform for detecting rare, random, and ephemeral events“, Information Processing In Sensor Networks , Proceedings of the 4th international symposium on Information processing in sensor networks, 2005

[3] F.Martincic, L.Schwiebert, “Distributed Event Detection in Sensor Networks,“ Systems and Networks Communication, International Conference on, p. 43, International Conference on Systems and Networks Communication (ICSNC’06), 2006

[4] W.Wu, X.Cheng, M.Ding, K.Xing, F.Liu, P.Deng. “Localized Outlying and Boundary Data Detection in Sensor Networks,“ IEEE Transactions on Knowledge and Data Engineering, pp. 1145-1157, August, 2007

[5] X.Zhu, R.Sarkar, J.Gao, J.Mitchell, “Light-weight Contour Tracking in Wireless Sensor Networks,“ Proceedings of 17th Fall Workshop on Computational and Combinatorial Geometry, 2007.

[6] Y.Zhu, Y.Liu, L.M.Ni, Z.Zhang, “Low-Power Distributed Event Detection in Wireless Sensor Networks“, IEEE INFOCOM 2007