Sensor Relocation in Mobile Sensor Networks

Guiling Wang, Guohong Cao, Tom La Porta, and Wensheng Zhang
Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
Email: {guiwang,gcao,tlp, wezhang}@cse.psu.edu

***Abstract*— Recently there has been a great deal of research on using mobility in sensor networks to assist in the initial deployment of nodes. Mobile sensors are useful in this environment because they can move to locations that meet sensing coverage requirements. This paper explores the motion capability to relocate sensors to deal with sensor failure or respond to new events. We define the problem of sensor relocation and propose a two-phase sensor relocation solution: redundant sensors are first identified and then relocated to the target location. We propose a Grid-Quorum solution to quickly locate the closest redundant sensor with low message complexity, and propose to use cascaded movement to relocate the redundant sensor in a timely, efficient and balanced way. Simulation results verify that the proposed solution outperforms others in terms of relocation time, total energy consumption, and minimum remaining energy.**

I. Introduction

Due to many attractive characteristics of sensor nodes such as small size and low cost, sensor networks [10], [13], [15], [17], [1] have become adopted to many military and civil applications including military surveillance, smart homes [18], remote environment monitoring, and in-plant robotic control and guidance. In order to properly sense the phenomena of interest, sensor nodes must be deployed appropriately to reach an adequate coverage level for the successful completion of the issued sensing tasks [5], [14].

In many potential working environments, such as remote harsh fields or disaster areas sensor deployment cannot be performed manually or precisely. In addition, once deployed, sensor nodes may fail, requiring nodes to be moved to overcome the coverage hole created by the failed sensor. In these scenarios, it is necessary to make use of mobile sensors [20], [21], which can move to provide the required coverage. One example of a mobile sensor is the Robomote [16]. These sensors are smaller than $0.000047m^3$ and cost less than 150 dollars.

In this paper we address the problem of *sensor relocation*, i.e., moving previously deployed sensors to overcome the failure of other nodes, or to respond to an occurring event that requires that a sensor be moved to its location. This sensor relocation is different from existing work on mobile sensors which concentrate on sensor deployment; i.e., moving sensors to provide certain initial coverage [11], [12], [20], [21], [24]. Compared with sensor deployment, sensor relocation has many special difficulties. First, sensor relocation may have a strict response time requirement. For example, if the sensor monitoring a security-sensitive area dies, another sensor should move to replace it as soon as possible. Second, relocation should not affect the application currently using the sensor network, which means that the relocation should minimize its effect on the current sensing topology. Finally, since movement may be much more expensive in terms of energy than computation and communication, any algorithm must balance energy costs with response time. In particular, care must be taken to balance the energy costs of an individual node with the overall network energy cost to ensure maximum network lifetime.

In this paper, we propose a framework for relocating mobile sensors in a timely, efficient, and balanced manner, and at the same time, maintaining the original sensing topology as much as possible. In our framework, sensor relocation consists of two phases: the first is to find the redundant sensors in the sensor network; the other is to relocate them to the target location. For the first phase, we propose a *Grid-Quorum* based solution to quickly locate the redundant sensors with low message overhead. For the second phase, we propose efficient heuristics to achieve good balance between energy efficiency and relocation time when determining the sensor relocation path. Simulation results show that the proposed heuristics are very effective in reducing the relocation time and the energy consumption.

The rest of the paper is organized as follows. In section II, we introduce some related work. In section III, we define the sensor relocation problem and the grid-based system model. Section IV presents the Grid-Quorum solution and Section V presents our solution for relocating sensors. Performance evaluations are presented in Section VI. Section VII concludes this paper.

II. Related Work

There have been several research efforts on deploying mobile sensors. For example, the work in [24] assumes that a powerful cluster head is available to collect information and determine the target location of the mobile sensors. Sensor deployment has also been addressed in the field of robotics [11], [12], where sensors are deployed one by one, utilizing the location information of previously deployed sensors. This method is not suitable to the relocation problem because it will not meet response time requirements in many cases.

Recently, we proposed three mobility-assisted sensor deployment protocols where mobile sensors move from densely deployed areas to sparse areas to increase the coverage. The

0-7803-8968-9/05/$20.00 (C)2005 IEEE

protocols run iteratively [21]. In each round, sensors first detect coverage holes around them by utilizing the Voronoi diagram [2]. If coverage holes exist, sensors decide where to move to heal or reduce the holes by three different distributed algorithms called VOR, VEC and Minimax. To achieve a good balance between sensor cost and sensor coverage, we designed a bidding protocol for deploying mobile sensors in sensor networks composed of both mobile and static sensors [20]. Because all of these algorithms take potentially several iterations to terminate, they may not meet the response time requirements of the relocation problem.

III. Problem Statement

In theory, the two protocols we previously proposed [20], [21] can be used for sensor relocation. For example, after a sensor failure, the sensors neighboring the failed node can execute the algorithms. After several rounds, the neighbor sensors will move to cover the area initially covered by the failed sensor. However, moving neighbor sensors may create new holes in that area. To heal these new holes, more sensors must move. This process continues until some area having redundant sensors is reached and the sensors leaving this area do not create new holes. Using the method, sensors may move several times, wasting energy. In addition, since many sensors are involved, it may take a long time for the algorithm to terminate. Based on this observation, we propose to first find the locations of the redundant sensors, and then design an efficient route for them to move to the destination.

To determine which sensor(s) is redundant is a challenging problem. It is hard for a single sensor to independently decide whether its movement will generate a coverage hole. To make such a decision, the sensor requires information about whether its neighbors will move or not. More specifically, a number of sensors located closely must determine the redundant sensors among themselves.

A *grid-based* architecture is a natural solution for this problem. We can divide the target field into grids. The grid head is responsible for collecting the information of its members, and determining the existence of redundant sensors based on their locations. For redundant sensors located on the boundary of the grid, grid heads coordinate to make decisions. The grid head can also monitor its group members and initiate a relocation process in case of new event or sensor failure.

A grid-based architecture is feasible in a network in which nodes are relatively regularly deployed, for example as would be the case after the termination of previously proposed sensor deployment algorithms [20], [21]. This is because, unlike the case of a network in which nodes are randomly deployed, the cost of organizing sensors into grids is low. Further, this organization can facilitate data aggregation, routing, etc. [22], in addition to finding the redundant sensors. Since many existing techniques on grid (cluster) maintenance [6], [9], [19] can be directly applied, we will not address these issues in this paper.

With the grid-based model, the sensor relocation problem can be reduced to two sub-problems: finding the redundant

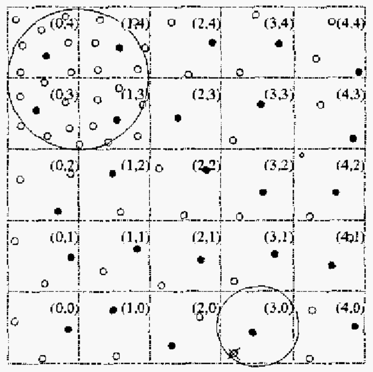


Fig. 1. The system model

sensors and then relocating them to the target location. Fig.1 illustrates the sensor relocation problem when grids are used; the black nodes are used to represent grid heads. Each grid is indexed by a tuple, whose first number is used to represent the column and the second number is used to represent the row. Grids (1,3), (0,3), (1,4) and (0,4) have redundant sensors. When a sensor at grid (3,0) dies, resulting in a coverage hole, its grid head first needs to locate the redundant sensor and then relocate some sensor to fix the coverage hole. For the first problem, we propose a Grid-Quorum solution to quickly identify the redundant sensors. For the second problem, we propose a cascaded movement solution to relocate sensors in a timely and energy efficient way.

IV. Finding the Redundant Sensors

In this section, we first give the background and motivation of the Grid-Quorum idea. Then, we present the detailed solution and illustrate its advantage in terms of message complexity and response time.

A. Background and Motivation

The problem of finding redundant sensors has some similarity to the publish/subscribe problem [7], [8], [4], where the publisher advertises some information and the subscriber requests the information. Mapping the terminology to our problem, the grids that need more sensors are the subscribers, and the grids that have redundant sensors are the publishers. In the publish/subscribe system, the matching of a request to an advertisement is called *matchmaking*.

Generally, there are three types of solutions for matchmaking. (1) Matchmaking occurs at the subscriber, which is referred as "broadcast advertisement" [7]. In our problem, this is similar to letting the grids having redundant sensors flood this information. Later, when some grid needs redundant sensors, it can get the information quickly. (2) Matchmaking occurs at the publisher, which is referred as "broadcast request" [7]. In our problem, this is similar to letting the grids that need sensors flood the request. The grid that has redundant sensors replies after receiving the request. (3) Matchmaking happens in the middle of the network [4], [8], [23]. In our problem, this is similar to letting the supply grid advertise the information to some intermediate grids from which the demand grid obtains the information.

Different from the traditional publish/subscribe problem, the information in our system is not reusable. The information about the redundant sensor can only be used once, since it may be changed after the redundant sensor moves to the requesting place. Due to this special property, the message complexity will be very high if we use the broadcast advertisement approach, which requires two network-wide broadcasts for each redundant sensor: one for advertisement and the other for data update after the redundant sensor moves.

For the broadcast request approach, the delay is relatively long since it is on-demand. Therefore, we prefer the third solution, which can achieve a balance between message complexity and response time. In this type of solution, a structure like that in [8], [23], can be used to facilitate the matchmaking between the advertisement and the request. Since the data may not be re-used, this structure should be simplified compared to that in [8], [23]; otherwise the benefit may not be worth the cost. Therefore, we need a simple and low-cost structure for matchmaking.

Our solution is motivated by the concept of quorum [3], which is defined as follows. Given a nonempty set $\mathcal{U}$, a coterie $\mathcal{C}$ is a set of $\mathcal{U}$'s subsets. Each subset $\mathcal{P}$ in $\mathcal{C}$ is called a quorum. The following condition must be held for quorums in a coterie $\mathcal{C}$ under $\mathcal{U}$:

- $(\forall \mathcal{P} \in \mathcal{C} :: \mathcal{P} \neq \emptyset \wedge \mathcal{P} \subseteq \mathcal{U})$
- Minimality Property: $(\forall \mathcal{P}, \mathcal{Q} \in \mathcal{C} :: \mathcal{P} \not\subset \mathcal{Q})$
- Intersection Property: $(\forall \mathcal{P}, \mathcal{Q} \in \mathcal{C} :: \mathcal{P} \bigcap \mathcal{Q} \neq \emptyset)$

By organizing grids as quorums, each advertisement and each request can be sent to a quorum of grids. Due to the intersection property of quorums, there must be a grid which is the intersection of the advertisement and the request. The grid head will be able to match the request to the advertisement. A simple quorum can be constructed by choosing the nodes in a row and a column. Instead of flooding the network with advertisements or requests, the request and the advertisement are only sent to nodes in a row or column. For example, as shown in Fig.1, suppose grid (0,3) has redundant sensors, it only sends the advertisement to grids in a row ((0,3), (1,3), (2,3), (3,3), (4,3)) and a column ((0,4), (0,3), (0,2), (0,1), (0,0)). When grid (3,0) is looking for redundant sensors, it only needs to send a request to grids in a row ((0,0), (1,0), (2,0), (3,0), (4,0)) and a column ((3,4), (3,3), (3,2), (3,1), (3,0)). The intersection node (0,0) will be able to match the request to the advertisement. Suppose N is the number of grids in the network. By using this quorum based system, the message overhead can be reduced from $O(N)$ to $O(\sqrt{N})$. Although the message overhead is very low compared to flooding, we can further reduce the message overhead by observing the specialty of our problem.

B. The Grid-Quorum Solution

In our Grid-Quorum system, we do not require the intersection of any two quorums. Instead, we deploy two coterie, called *supply coterie* and *demand coterie* separately, and only require that the quorum belong to the supply coterie intersects with all quorums in the demand coterie, and vice versa.

The formal definition is as follows. Given a nonempty set $\mathcal{U}$, there is a supply coterie $\mathcal{C}_s$ and a demand coterie $\mathcal{C}_d$, which are the sets of $\mathcal{U}$'s subsets. Each subset $\mathcal{P}_s$ in coterie $\mathcal{C}_s$ is called a supply quorum and each subset $\mathcal{P}_d$ in coterie $\mathcal{C}_d$ is called a demand quorum. Suppose coterie $\mathcal{C}_s$ has m quorums, and coterie $\mathcal{C}_d$ has n quorums. The following condition must be hold for quorums in coterie $\mathcal{C}_s$ and $\mathcal{C}_s$ under $\mathcal{U}$:

- $\bigcup_{i=1}^{m} \mathcal{P}s_i = \mathcal{U}$
  $\bigcup_{i=1}^{n} \mathcal{P}d_i = \mathcal{U}$
- Minimality Property:
  $(\forall \mathcal{P}_s, \mathcal{Q}_s \in \mathcal{C}_s :: \mathcal{P}_s \not\subset \mathcal{Q}_s)$
  $(\forall \mathcal{P}_d, \mathcal{Q}_d \in \mathcal{C}_d :: \mathcal{P}_d \not\subset \mathcal{Q}_d)$
- Intersection Property:
  $(\forall \mathcal{P}_s \in \mathcal{C}_s, \forall \mathcal{P}_d \in \mathcal{C}_d :: \mathcal{P}_s \bigcap \mathcal{P}_d \neq \emptyset)$
  $(\forall \mathcal{P}_d \in \mathcal{C}_d, \forall \mathcal{P}_s \in \mathcal{C}_s :: \mathcal{P}_d \bigcap \mathcal{P}_s \neq \emptyset)$

To construct a Grid-Quorum, the grid heads belong to the grids in one row are organized into one quorum, called *supply quorum* and the grid heads belong to the grids in a column are organized into one quorum, called *demand quorum*. All the supply quorums compose the supply coterie, and the demand quorums compose the demand coterie. In this way, the natural geographic relation ensures that every supply quorum has intersection with all the demand quorums and vice versa. When a grid has redundant sensors, the grid head propagates this information through the supply quorum to which it belongs. When any grid wants more sensors, the grid head needs only to search its demand quorum. Since every demand quorum has intersection with all supply quorums, the grid head can get all the information about redundant sensors. We can see that using the geographic information reduces the cost of building Grid-Quorum to almost zero.

Still using the example of Fig.1, Grids (0,4), (1,4), (0,3) and (1,3) have redundant sensors, while grid (3,0) needs more sensors. The grid head of (1,3) propagates its redundant sensor information through its supply quorum ((1,4), (1,3), (1,2), (1,1), (1,0)). The grid head in grid (3,0) searches its demand quorum ((0,0), (1,0), (2,0), (3,0), (4,0)). Grid (1,0) can reply the information about redundant sensors. Compared to using the quorum in the last example, using grid-quorum cuts the message by half.

Optimization: To further reduce the message complexity, we add a stopping criteria for propagating request messages through the demand quorum. Since we want the closest redundant sensor, the propagation should be stopped once we get the best solution. To implement the idea, the already visited grid head can piggyback the information in its supply quorum before forwarding the query message to the next grid in the same row. Then every grid head in this demand quorum can check whether a better result can be found for the demanding grid. If not, this grid head can reply to the query grid immediately without further propagating the request. For example, in Fig.2, sensor s_c dies and grid (3,2) has redundant sensor s_a. All sensors located outside the circle with center s_c

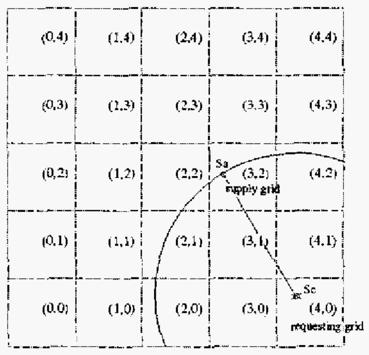


Fig. 2. Stopping criteria

and radius of the distance between s_a and s_c must be farther to s_c than s_a. Cluster head of grid (3,0) attaches the location of s_a in the requesting message before forwarding it. When the grid head in grid (2,0) receives the request message, it will not forward it further since no closer redundant sensor can be found.

Fig.3 compares the performance between the Grid-Quorum solution and the "Broadcast Request" method in a $10*10$ grid. The results were obtained from a simple ns2 simulation; more details on the simulation environment are given later. Cluster heads of neighbor grids communicate through a gateway sensor. Suppose there is no other traffic in the network. From the figures, we can see that the Grid-Quorum solution can significantly reduce the message complexity compared to the "Broadcast request" approach.

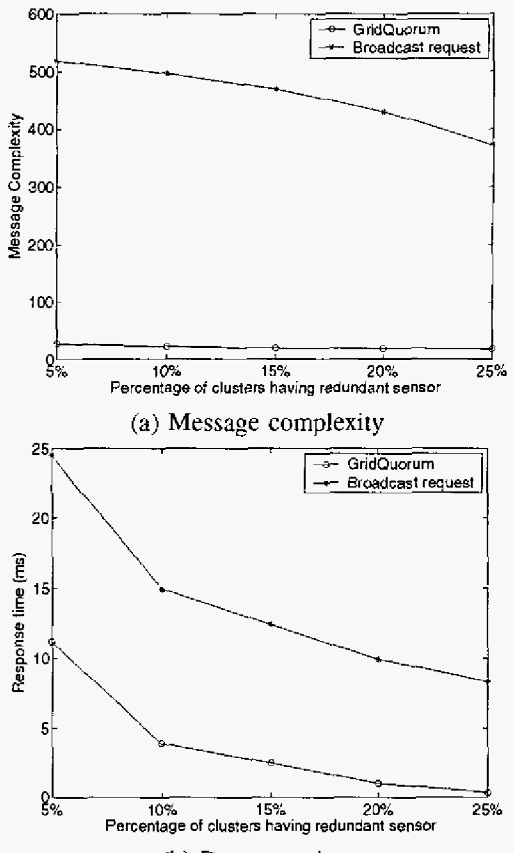


Fig. 3. Comparison between the Grid-Quorum solution and the "Broadcast Request" approach

V. Sensor Relocation

A. General Idea: Cascaded Movement

Having obtained the location of the redundant sensor, we need to determine how to move the sensor to the target location (destination). Moving it directly to the destination is a possible solution. However, it may take a longer time than the application requirement. For example, a sensor monitoring a strategic area dies and the application specifies that the maximum tolerable time for such a sensing hole is thirty seconds. If the redundant sensor is 100 meters away and it takes at least one minute for the sensor to reach its destination, the application requirement cannot be met. Moreover, moving a sensor for a long distance consumes too much energy. If the sensor dies shortly after it reaches the destination, this movement is wasted and another sensor has to be found and relocated.

We propose to use a *cascaded movement* to address the problem. The idea is to find some cascading (intermediate) nodes, and use them for relocation to reduce the delay and balance the power. As shown in Fig.4, instead of letting the redundant sensor s_3 move directly to the destination, s_1 and s_2 are chosen as cascading nodes. As a result, s_3 moves to replace s_2, s_2 moves to replace s_1, and s_1 moves to the destination. Since the sensors can first exchange communication messages (i.e., logically move), and ask all relevant sensors

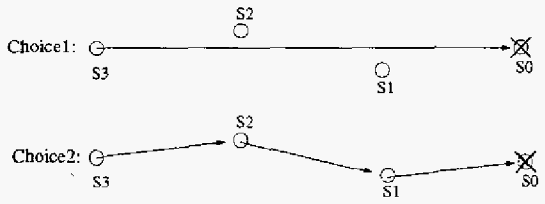


Fig. 4. Cascaded movement

to (physically) move at the same time, the relocation time is much shorter.

A node s_i which moves to replace another node s_j, is referred to as s_j's *successor*, and s_j is referred to as s_i's *predecessor*. In Fig.4, s_3 is s_2's successor and s_2 is s_3's predecessor. We also introduce a virtual node s_0, which is used to represent the target location. It may represent the failed sensor or the location where an extra sensor is needed to increase the sensing accuracy. In Fig.4, we say s_0 is s_1's predecessor and s_1 is s_0's successor.

Selecting cascading nodes is not easy since the sensor nodes may be used by some application and their movement may affect the sensing or communication tasks they are performing. To ensure that this effect is within application's requirement, each sensor s_i is associated with a *recovery delay* T_i. After

s_i's movement, its successor must take its place within T_i. T_i is determined by the application based on the critical level of s_i's sensing task, the size of the coverage hole generated by s_i's movement, and other application factors. We use T_0 to represent the recovery delay of the relocation event. It can be the maximum recovery delay of the failed sensor or the time limit for an additional sensor being placed at s_0.

The T value imposes restrictions on the spatial relationship and departure time of the cascading nodes. We use t_i to denote the departure time of s_i and d_{ji} to denote the distance between s_i and s_j. The following Inequality must be satisfied if s_j is s_i's successor.

$$d_{ji}/speed - (t_i - t_j) \leq T_i \qquad (1)$$

For simplicity, t_i is normalized to be the time period after the relocation request is sent and t_0 (for s_0) is set to be 0.

Based on Inequality (1), whether s_j can be the the successor of s_i is not determined solely by its distance to s_i, but also s_i's departure time. If s_i moves at t_0 (0), s_j must be within $speed * T_i$ from s_i; if s_i moves after another t minutes, s_j can be farther away from s_i as long as $d_{ji} \leq speed * (T_i + t)$. Whether s_i can stay at its place for this t minutes or must move immediately is determined by its own predecessor. For example, if s_i is the successor of s_0, and d_{i0} is shorter than $speed * T_0$, s_i can flexibly move between $(0, T_0 - d_{i0}/speed)$. In this case, we normally let s_i move at $T_0 - d_{i0}/speed$ (the upper limit) such that more sensors can be chosen as s_i's successor and we can choose the best one.

The set of cascading nodes for a relocation and their departure time together is defined by a *cascading schedule*. For example, in Fig.4, Choice 2, $s_3(t_3) \rightarrow s_2(t_2) \rightarrow s_1(t_1) \rightarrow s_0$ is a cascading schedule, which can be used to recover a sensor failure. Certainly, the cascading scheduling should make sure that the recovery delay is satisfied; i.e., Inequality (1) is satisfied. For example, Choice 1 is not a cascading schedule since the sensor failure cannot be recovered within the required time.

B. The Metrics to Choose Cascading Nodes

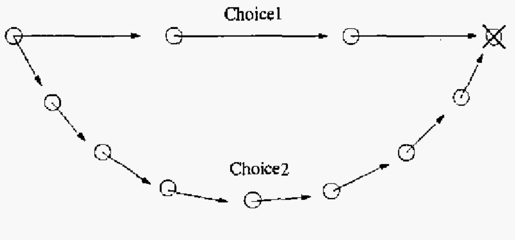


Fig. 5. Cascaded movement

The cascading schedule should minimize the total energy consumption and maximize the minimum remaining energy so that no individual sensor is penalized. However, in most cases, these two goals cannot be satisfied at the same time. As shown in Fig.5, suppose all sensors have the same amount of power. Choice1 consumes less energy, but the involved sensors will have lower remaining energy. Sensors in Choice2 have higher remaining energy, but the total energy consumption of Choice 2 is higher than that in choice1. There is a tradeoff between minimizing the total energy consumption and maximizing the minimum remaining energy, and we want to find a balance between them.

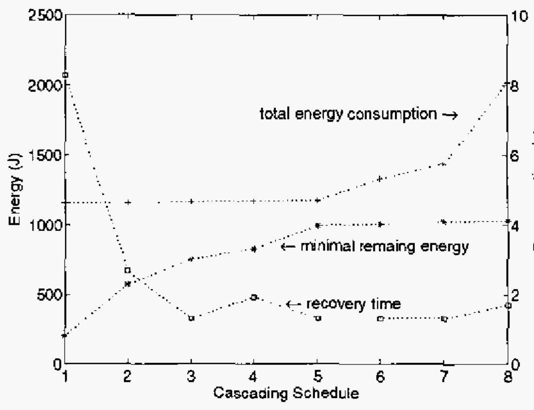


Fig. 6. Tradeoff

Before presenting our solution, we first show some observations. Based on the sensor deployment result generated by running VOR [21], we randomly choose some sensor and deplete its energy. Then, all cascading schedules to recover the failed sensor are enumerated and compared in terms of the total energy consumption and the minimum remaining energy. Here, the recovery delay ($T_i, i \neq 0$) is relaxed for better observation, but the relocation time is calculated for reference. The cascading schedules which are worse than some other schedule in both metrics (total energy consumption and minimum remaining energy) will be ignored; that is, we only keep the cascading schedules which perform better than others at least in terms of one metric.

Fig.6 shows the total energy consumption and the minimum remaining energy of these schedules in an increasing order. As shown in the figure, the total energy consumption is almost flat at the beginning and then significantly increased, whereas the minimum remaining energy has a steep increase at the beginning and then becomes flat. This observation motivates us to achieve a good balance between minimizing the total energy consumption and maximizing the minimum remaining energy.

From the observation, we can see that it is possible to continuously spend a little more energy for a much higher minimum remaining energy until a turning point after which the cost is higher but the gain is less. The cascading schedule just before this turning point should be the best schedule. In other words, the best schedule is the schedule with *the minimum difference between the total energy consumption and the minimum remaining power*. This new metric can be explained in a mathematical way. Suppose there are two cascading schedules with E_1 and E_2 as their total energy consumption, and $Emin_1$ and $Emin_2$ as their minimum remaining energy. Schedule 1 is chosen since $E_1 - Emin_1 \leq E_2 - Emin_2$. This inequality can also be expressed as $E_1 - E_2 \leq Emin1 - Emin_2$; i.e., the cascading schedule with more advantage and less disadvantage should be chosen.

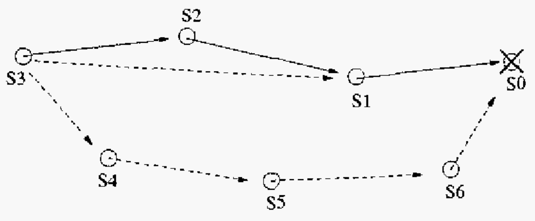


Fig. 7. An example

Fig.7 uses an example to further explain the reason. In Fig.7, moving s_3 directly to the target location is the most energy efficient solution. However, in this way, s_3 will be penalized, and its minimum remaining energy will be significantly reduced. If s_1 is added as a cascading node, the load of s_3 can be shared and the minimum remaining energy can be improved. Since the total length of the zigzag line $s_3s_1s_0$ is only a little bit longer than the length of s_3s_0, only a slightly more power is needed. If more sensors close to the line s_3s_0 are chosen as cascading nodes, the load can be further shared and the minimum remaining power can be further improved. Certainly, if some sensor close to this line has very low energy, it should not be selected for cascading. When all eligible sensors close to this line have been chosen as cascading nodes, a balanced and efficient schedule is obtained. Starting from this point, if we want to further improve the minimum remaining energy, faraway sensors such as s_4, s_5 and s_6, have to be chosen. However, in this way, the total energy consumption will be significantly increased, and then it may not be a good solution.

In the remaining of the paper, the cascading schedule with minimum difference between the total energy consumption and minimum remaining power is referred to as the *best* cascading schedule. The cascading schedule with the least total energy consumption is referred to as the *shortest* schedule.

C. The Algorithm

Before presenting the algorithm for calculating the best cascading schedule, we first introduce some notations, and describe how to modify Dijkstra's algorithm to calculate the shortest cascading schedule.

The sensor network can be modeled as a complete weighted graph $G(V, E)$, where vertices correspond to the sensor nodes. There are edges between any pair of nodes, and the weight of edge s_is_j is the distance between s_i and s_j. The remaining power of s_i before relocation and after relocation is denoted by P_i and P_i' separately. We represent the energy as the distance that the sensor can move with this energy. In this way, if s_i moves to s_j in the relocation, $P_i' = P_i - d_{ij}$.

To calculate the shortest cascading schedule, we cannot simply apply Dijkstra's algorithm due to the constraint of the recovery delay. Different from the shortest path problem, not all the edges with a finite weight in the graph can be selected as a segment of the path. Also, we cannot set the weight of an edge to be infinity because its length solely can not determine whether or not these two nodes can be successor and predecessor of each other. To solve the problem, we add a *DeleteEdge* operation to Dijkstra's algorithm to guarantee the delay constraint. *DeleteEdge(s_i, t_i)* deletes edge s_js_i if s_j can not become the successor of s_i; i.e., $d_{ji} > (T_i + t_i) * speed$. The new algorithm, as shown in Fig.8, is referred to as the *Modified Dijkstra's algorithm*.

```
ModifiedDijkstra( Graph G(V,E), Vertex $s_0$)
Initialization: $S = \{s_0\}$, $Q = V$
DeleteEdge($s_0$,0)
while not Empty ($Q$)
  1.  Let $\mathcal{F} = \{\langle s_k, s_l\rangle \mid \langle s_k, s_l\rangle \in S \times Q,$
                        $d_{lk} \leq (T_k + t_k) * speed\}$
  2.  Find $\langle s_i, s_j\rangle \in \mathcal{F}$ such that $\forall\langle s_k, s_l\rangle \in \mathcal{F}, d_{ji} \leq d_{lk}$
  3.  $s_j$.predecessor = $s_i$
  4.  $t_j = T_i + t_i - d_{ji}/speed$
  5.  $P_j' = P_j - d_{ji}$
  6.  Add $s_j$ to $S$
  7.  DeletedEdge($s_j, t_j$)
end
```

Fig. 8. Modified Dijkstra's algorithm

```
Initialization: $E = 0, Emin = -2, E' = 0, Emin' = -1$
while (1)
  1.  find the shortest cascading schedule using
      the Modified Dijkstra's algorithm
  2.  record the minimum remaining power as $Emin'$
  3.  delete all edges $s_is_j$ if $P_i - d_{ij} \leq Emin'$
  4   if $E' - Emin' < E - Emin$ then
          $E = E', Emin = Emin'$
      else
          return the previously calculated schedule
```

Fig. 9. Algorithm to calculate the best cascading schedule

To calculate the best cascading schedule, we first calculate the shortest cascading schedule and record its total energy consumption E and its minimum remaining energy $Emin$. Then, we delete all the edges s_is_j if $P_i - d_{ij} \leq Emin$ and a new graph is generated. This process continues and a new shortest cascading schedule is calculated as long as the difference between the total energy consumption and the minimum remaining energy is increased compared to the previously calculated cascading schedule. When the process terminates, the schedule calculated before the last schedule is the best schedule, i.e., the schedule with the smallest difference between the last two schedules. As shown in Fig.6, schedule 1 is calculated first, and then 2, ..., 5 and 6. The energy difference of schedule 6 is larger than schedule 5. Thus, the algorithm stops and schedule 5 is chosen as the best cascading schedule. Fig.9 shows the formal description of the algorithm.

D. Distributed Protocol

In this section, we describe how to implement the algorithm presented above in a distributed way. We first present a distributed protocol to calculate the shortest cascading schedule

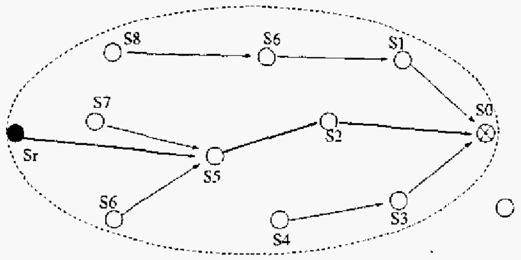


Fig. 10. The distributed protocol

and then describe how to use it to get the best cascading schedule.

To calculate the shortest cascading schedule, the grid head of s_0 initiates a dynamic programming computation by broadcasting a request message, which includes T_0, t_0, the redundant sensor s_r, E_0, and $Emin_0$, where E_0 and t_0 are set to 0, and $Emin_0$ is set to infinity. A node s_i receiving the request first determines if it can be the successor of the sender s_j. If the answer is yes, it sets $E_i = d_{ij} + E_j$, $Emin_i = min(P_i - d_{ij}, Emin_j)$, and $t_i = T_j + t_j - d_{ij}/speed$. Then, it rebroadcasts T_i, t_i, s_r, E_i and $Emin_i$, and remembers its predecessor s_j. If a node s_j receives several such messages, it will choose the one from s_k which can minimize E_j, which is the energy consumption of the shortest cascading schedule from s_j to s_0. Then, s_j calculates the other fields of the message, broadcasts the message and sets its predecessor to be s_k. Finally, when the request arrives at the redundant sensor s_r, the distributed calculation terminates.

To make the broadcast-based protocol work, we have to address one issue: how can a node determine it has received the message which can minimize the total energy consumption before broadcasting it? There are two intuitive solutions. In the first method, when a node receives a request, it calculates those fields. If E is lower than that calculated from the previously received message, it rebroadcasts the updated version. This method has relatively high message complexity since each nodes may broadcast several times. In another method, each node waits for a period of time before broadcasting the message. However, if the time threshold is low, there may not be enough information to decide the lowest E value; if the time threshold is high, the delay may be increased.

We propose to use the geographic information to ensure that nodes make correct decisions before broadcasting with a high probability. As a result, in most cases, each sensor only broadcasts once. Our solution needs two data structures: the *primary search area* and the *waiting list*. The primary search area is determined based on the location of the redundant sensor and the event, and it should have a high probability to encompass all the cascading nodes. The primary search area in Fig.10 is elliptic. It can also be other shapes like rectangle. Recall that the cascading nodes should be close to the line connecting the redundant sensor and the event location. Therefore, it is not necessary to involve nodes faraway. Each sensor node determines a *waiting list* after it receives the relocation request for the first time. Only after receiving all the messages from the nodes in the waiting list, a sensor can broadcast the message. The waiting list of s_i includes all the neighbors of s_i which are within the primary search area and are further away from the redundant sensor than s_i. As shown in Fig.10, s_2's waiting list includes s_1 and s_3. It only broadcasts the message after receiving the message of s_1 and s_3.

Another issue of this broadcast-based protocol is that the potential successor node may be out of the communication range of the sender. Due to the limitation of the communication range, the successor may not be a communication neighbor. For example, in Fig.10, the redundant sensor s_r should be the successor of s_5, but they are not within each other's communication range. s_r may not know the existence of s_5 and can not get the shortest schedule. To address this problem, s_7 piggybacks s_5's message when broadcasting its own message if it finds that s_r may use it. In general, if s_i receives some message from s_k, it will check whether other node may need it. If so, it piggybacks T_k, t_k, E_k, and $Emin_k$ in its message. The formal description of the distributed calculation of the shortest cascading schedule is shown in Fig.11.

To calculate the best cascading schedule, we only need to execute the distributed calculation of the shortest schedule iteratively similar to the algorithm shown in Fig.9 after the following modifications: (1) The minimum remaining energy calculated in the previous iteration, $Emin'$ is attached. When s_i receives message from s_j, it will check whether $P_i - d_{ij} \leq Emin'$. If yes, s_i and s_j cannot be the successor and predecessor of each other. This change is similar to step 3 in Fig.9. (2) In addition to the current predecessor, each node also needs to record the predecessor in the previous iteration.

E. Optimization: Prepositioning

In some situations, the recovery delay constraint may result in a bad relocation schedule in terms of energy efficiency. Fig.5 shows such an example. If the recovery delay is too strict, we have to use choice2. Also, letting each cascading node move synchronously may result in temporary coverage holes though these holes may be within the application's tolerance. These problems can be addressed if the relocation request can be predicted. For example, if the grid head detects that a group member has very low remaining energy, it can initiate a relocation request to the redundant sensor. In this way, a best relocation schedule in terms of energy efficiency can be found, and the recovery delay constraint can be relaxed. If time permits, the redundant sensor can first move to its predecessor node. After it arrives there, its predecessor starts to move. In this way, no temporary coverage hole will be introduced.

VI. Performance Evaluation

The evaluation includes three parts: First, the effectiveness of the proposed solution is compared with the VOR scheme [21]. Second, the effectiveness of the cascaded movement is evaluated. Finally, the effectiveness of the metric for choosing the cascading schedule is evaluated.

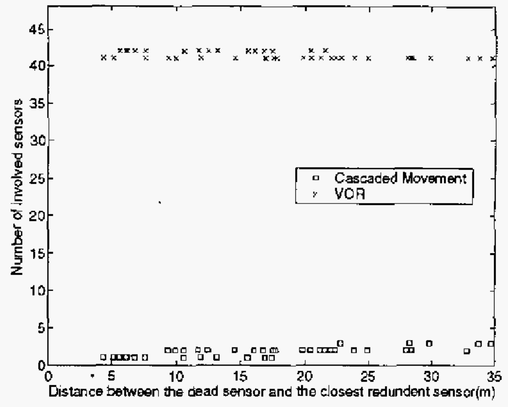
```
Notations:
  $listw_i$: waiting list of $s_i$
  $listo_i$: list of nodes from which $s_i$ has received messages
  $loc_i$: location of $s_i$
  $M_j$: $\langle t_j, T_j, loc_j, E_j, Emin_j \rangle$
  $listm_i$: list of $M$
At cluster head
Upon receiving the information of redundant sensor $s_r$
  1.  add $M_0\langle t_0, T_0, loc_0, 0, 0\rangle$ to $listm_0$
  2.  broadcast $request\langle listm_0, s_r, loc_r, loc_0\rangle$
At node $s_i$
Upon receiving $request\langle listm_j, s_r, loc_r, loc_0\rangle$ from $s_j$
  1.  if receive such message for the first time then
          determine $listw_i$, $E_i = \infty$
  2.  add $s_j$ to $listo_i$ if it is in $listw_i$
  3.  for each $M_k\langle t_k, T_k, loc_k, E_k, Emin_k\rangle$ in $listm_j$
      3.1  $d_{ik} = distance(loc_i, loc_k)$
      3.2  if $(d_{ik} \leq (T_k + t_k) * speed)$ then
               add $M_k$ to $listm_i$
               if $(E_i \geq E_k + d_{ik})$ then
                   $E_i = E_k + d_{ik}$
                   $Emin_i = min\{Emin_k, P_i - d_{ik}\}$
                   $t_i = T_k + t_k - d_{ik}/speed$
                   $s_i$.predecessor = $s_k$
  4.  if $(listw_i = listo_i)$ then
      4.1  add $M_i\langle t_i, T_i, loc_i, E_i, Emin_i\rangle$ to $listm_i$
      4.2  broadcast $request\langle listm_i, s_r, loc_r, loc_0\rangle$
At redundant sensor $s_r$
Upon receiving $request\langle listm_j, s_r, loc_r, loc_0\rangle$ from $s_j$
  1.  if receive such message for the first time then
          determine $listw_r$, $E = \infty$
  2.  add $s_j$ to $listo_r$ if it is in $listw_r$
  3.  for each $M_k\langle t_k, T_k, loc_k, E_k, Emin_k\rangle$ in $listm_j$
      3.1  $d_{rk} = distance(loc_r, loc_k)$
      3.2  if $(d_{rk} \leq (T_k + t_k) * speed \wedge E \geq E_k + d_{rk})$
               $E = E_k + d_{rk}$
               $Emin = min\{Emin_k, P_r - d_{rk}\}$
               $s_r$.predecessor = $s_k$
  4.  if $(listw_r = listo_r)$ then
          return the shortest cascading schedule
```

Fig. 11. Distributed calculation of the shortest schedule

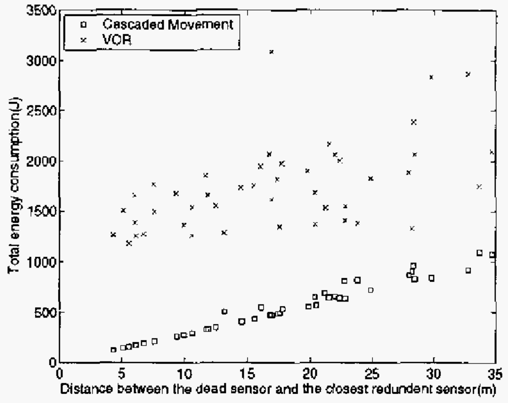
A. *Effectiveness of our sensor relocation solution*

The sensor relocation solution presented in this paper first searches the closest redundant sensor and then relocates it to the target location. Other solutions can be based on the idea of relocating the nearby sensor to the target location. The representative scheme is VOR [21]. VOR runs round by round. In each round, it detects the coverage hole and moves nearby sensors to heal it. This process continues until the target field is well covered.

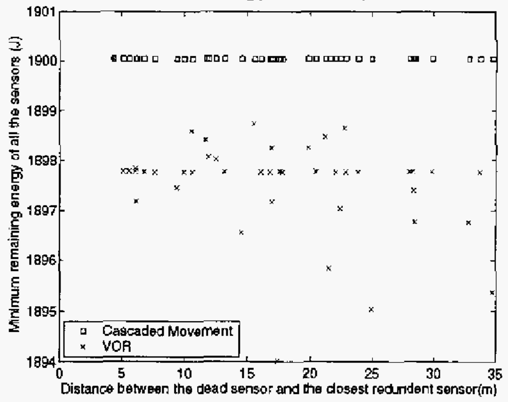
We use sensor failure recovery as an example to evaluate the proposed solution and the VOR scheme. The simulation environment is set as follows. 48 sensors are randomly deployed in a $60m * 60m$ target field. VOR is used to deploy them into a well-covered sensor network. The speed of the mobile sensor is $2m/s$ and the recovery delay is $10s$. The energy consumption per meter is $27.96J$, which is calculated based on a mobile sensor prototype [16]. Each sensor is randomly



(a) Number of sensors moved



(b) Total energy consumption



(c) Minimum remaining energy

Fig. 12. Comparison between our solution and VOR

assigned a remaining energy between $1900J$ and $2000J$. We randomly choose a sensor, deplete its energy, and generate a coverage hole. Then, the proposed sensor relocation and VOR are executed to recover the sensor failure. We measure the performance of both approaches by three metrics: the number of sensors moved, the total energy consumption, and the minimum remaining energy.

Simulation results are shown in Fig.12. Since the distance between the failed sensor and the redundant sensor directly affect the energy consumption, we use the distance as the x-axis. From the figure, we can see that our solution outperforms

VOR in all three metrics. In VOR, nearby sensors move to heal the coverage hole. Since their movement results in new holes, more and more sensors are involved. The propagation of the sensor movement is not directed to the redundant sensor, but to all directions, resulting in moving oscillation. From Fig.12(a), we can see that many sensors are involved in the relocation although the redundant sensor is nearby. As a result, the total energy consumption is very high (see Fig.12(b)) since a lot of movement is wasted. Also, in VOR, when a coverage hole is detected, nearby sensors are moved even when their remaining energy is low. This results in a much lower minimum remaining energy as shown in Fig.12(c).

The simulation results verified the advantage of our solution, where only a minimum number of nodes are involved and many other sensors are not affected. The energy consumption is low and the remaining energy is high. In summary, although VOR is effective in deploying mobile sensors, first finding the redundant sensor and then relocating it to the target location is much better for sensor relocation.

B. Cascaded Movement vs Direct Movement

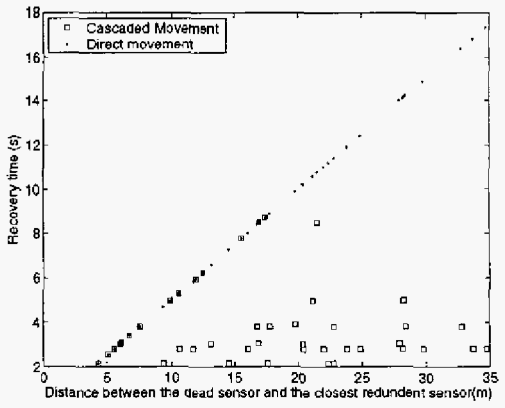
In this section, with the same simulation setup, we compare the cascaded movement approach with the direct movement approach, which moves the redundant sensor directly to the target location.

Simulation results are shown in Fig.13. As can be seen (Fig.13(a)), the relocation time can be significantly reduced in the cascaded movement approach. As for energy consumption, direct movement is better, but its advantage over cascaded movement is very limited (Fig.13(b)). This proves that cascaded movement is energy efficient. On the other hand, the minimum remaining energy of using cascaded movement is much better than that of direct movement. If the redundant sensor has relatively high power, moving it directly to the target location may not affect the minimum remaining power; otherwise, it may significantly reduce the minimum remaining power, especially when the moving distance is long. This explains why the minimum remaining energy drops proportionally as the distance increases in the direct movement approach (see Fig.13(c)).

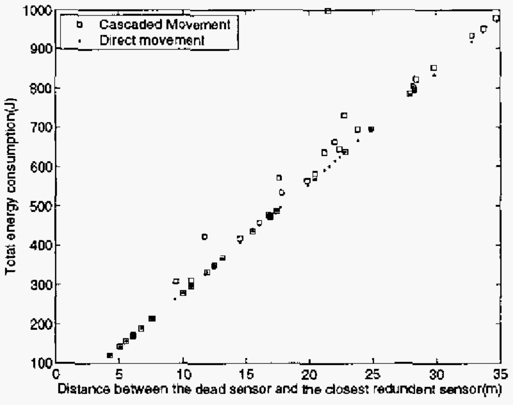
C. The metric to choose the cascading schedule

In our solution, the metric used to get the best cascading schedule is to minimize the difference between the total energy consumption and the minimum remaining power. Since there are other objective metrics existing, we compare our solution to other alternatives. Since we have shown (see Fig.13(b)) that the total energy consumption of our approach is similar to the direct movement approach, which is optimal, we only compare our approach with another alternative that maximizes the minimum remaining power.

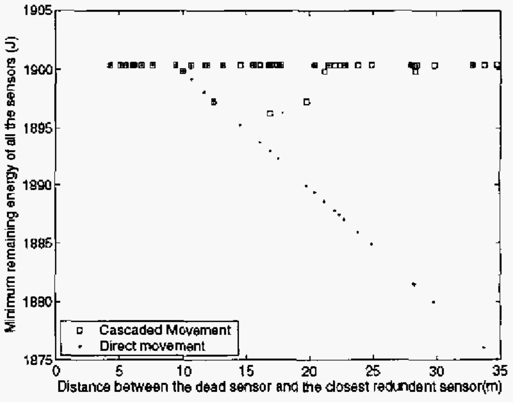
The simulation setup is the same as before except for the energy level before relocation. Two cases are considered. One is that sensors have similar remaining power, which is randomly distributed between $1990J$ and $2000J$. The maximum difference of the remaining power among sensors is



(a) Relocation time



(b) Total energy consumption

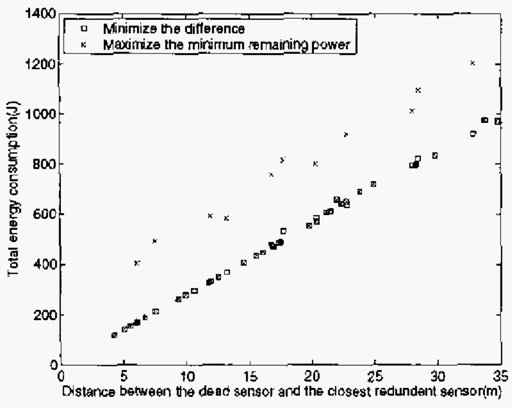


(c) Minimum remaining energy

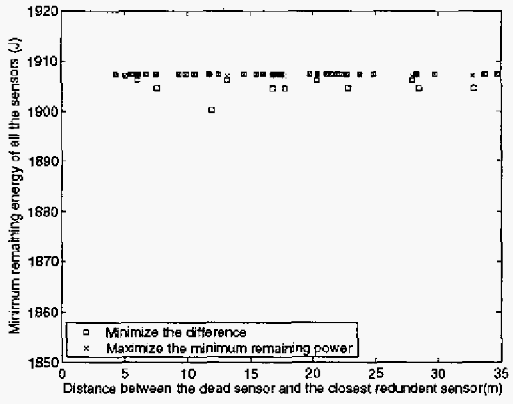
Fig. 13. Comparison between cascaded movement and direct movement

$10J$ and the variance is about $11.8J^2$. In the other case, the remaining power is very different. The sensor remaining power is randomly distributed between $1900J$ and $2000J$. The maximum difference of the remaining power among sensors is $100J$ and and the variance is about $855J^2$.

As shown in Fig.14 and Fig.15, for both settings, our solution has much lower total energy consumption compared to the approach of maximizing the minimum remaining energy. In the extreme case, our solution saves about $2000J$. Meanwhile, the minimum remaining energy is at most $10J$ lower than its alternative.

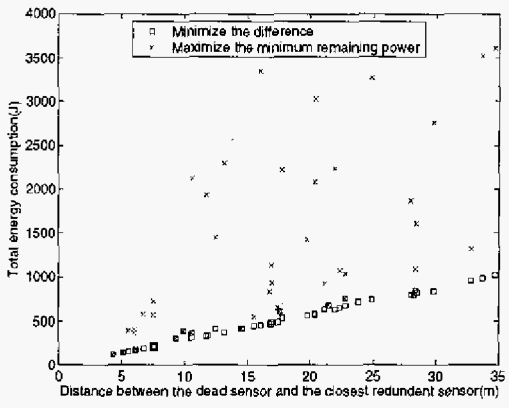


(a) Total energy consumption

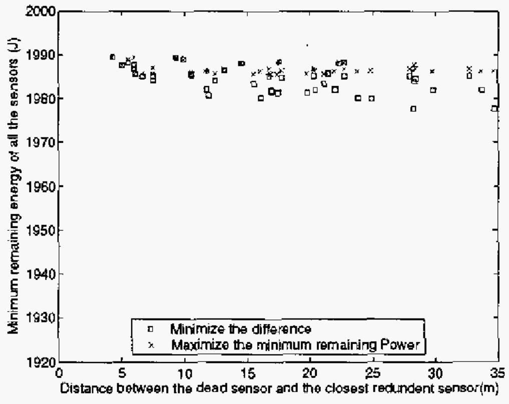


(b) Minimum remaining energy

Fig. 14. Comparisons when the remaining energy is very different



(a) Total energy consumption



(b) Minimum remaining energy

Fig. 15. Comparisons when the remaining energy is similar

Between these two settings, our approach saves more energy when the remaining energy is similar. The reason is as follows. Sensors with relatively more energy must be involved to maximize the minimum remaining energy. When the remaining energy is similar, it is not likely to find nearby sensors with high energy. Then, faraway sensors are more likely to be involved, and more energy will be consumed compared to our solution. On the other hand, when the remaining energy is similar, the disadvantage of our solution is a little bit larger since only nearby sensors are involved in the relocation. These sensors may become the sensors with minimum remaining energy after relocation and the minimum remaining energy among all the sensors is reduced consequently.

When the remaining energy is very different, both approaches have similar minimum remaining energy since a sensor with minimum remaining energy is more likely not involved in the relocation and the minimum remaining energy of the network does not change after the relocation.

VII. Conclusions

In this paper, we defined the problem of sensor relocation, which can be used to deal with sensor failure or response to new events. To effectively relocate sensors and minimize the effect on the application, we proposed a two-phase sensor relocation solution: redundant sensors are first identified and then relocated to the target location. We proposed a Grid-Quorum solution to quickly locate the closest redundant sensor with low message complexity, and proposed to use cascaded movement to relocate the redundant sensor. Since the sensors can first exchange communication messages (i.e., logically move), and ask all relevant sensors to (physically) move at the same time, the cascaded movement solution can significantly reduce the relocation time. Further, a distributed protocol has been proposed to find the best cascading schedule to minimize the difference between the total energy consumption and the minimum remaining power. Simulation results verified that the proposed solution outperforms others in terms of relocation time, total energy consumption, and minimum remaining energy.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, pp. 102–114, August 2002.

[2] F. Aurenhammer, "Voronoi diagrams — a survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, pp. 345–406, 1991.

[3] G. Cao and M. Singhal, "A Delay-Optimal Quorum-Based Mutual Exclusion Algorithm for Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 12, pp. 1256–1268, 2001.

[4] A. Carzaniga, D. Rosenblum and A. Wolf, "Design and Evaluation of a Wide-area Event Notification Service," *ACM Transactions on Computer Systems*, vol. 19, August 2001.

[5] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan and K. k. Saluja, "Sensor Deployment Strategy for Target Detection," *First ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[6] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," *ACM Mobicom*, August 1999.

[7] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, June 2003.

[8] Z. Ge, P. Ji, J. Kurose and D. Towsley, "Matchmaker: Signaling for Dynamic Publish/Subscribe Applications," *11th IEEE International Conference on Network Protocols(ICNP)*, November 2003.

[9] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, October 2002.

[10] W. R. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Network," *ACM Mobicom*, August 1999.

[11] A. Howard, M. J. Mataric and G. S. Sukhatme, "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks," *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, September 2002.

[12] A. Howard, M. J. Mataric and G. S. Sukhatme, "Mobile Sensor Networks Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem," *the 6th International Symposium on Distributed Autonomous Robotics Systems*, June 2002.

[13] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication," *Mobicom*, August 2000.

[14] S. Meguerdichian, F. Koushanfar, M. Potkonjak and M. B. Srivastava, "Coverage Problems in Wireless Ad-hoc Sensor Network," *INFOCOM*, April 2001.

[15] G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors," *Communications of the ACM*, May 2000.

[16] G. T. Sibley, M. H. Rahimi and G. S. Sukhatme, "Robomote: A Tiny Mobile Robot Platform for Large-Scale Sensor Networks," *Proceedings of The IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[17] K. Sohrabi, J. Gao, V. Ailawadhi and G. J. Pottie, "Protocols for Self-Organization of A Wireless Sensor Network," *IEEE Personal Communication*, vol. 7, no. 5, pp. 16–27, October 2000.

[18] M. Srivastava, R. Muntz and M. Potkonjak, "Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments," *Mobicom*, 2001.

[19] M. Steenstrup, B. Beranek and Newman, *Ad hoc Networking. chapter Cluster-Based Networks*, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 2001.

[20] G. Wang, G. Cao and T. La Porta, "A Bidding Protocol for Deploying Mobile Sensors," *The 11th IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[21] G. Wang, G. Cao and T. La Porta, "Movement-Assisted Sensor Deployment," *INFOCOM*, March 2004.

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

[23] W. Zhang, G. Cao and T. La Porta, "Data Dissemination with Ring-Based Index for Wireless Sensor Networks," *The 11th IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[24] Y. Zou and K. Chakrabarty, "Sensor Deployment and Target Localization Based on Virtual Forces," *INFOCOM*, 2003.