Available online at www.sciencedirect.com

ScienceDirect

Computer Communications 30 (2007) 2615–2626

computer communications

www.elsevier.com/locate/comcom

Sensor replacement using mobile robots

Yongguo Mei, Changjiu Xian, Saumitra Das, Y. Charlie Hu *, Yung-Hsiang Lu

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA

Available online 27 June 2007

Abstract

Sensor replacement is important for sensor networks to provide continuous sensing services. Upon sensor node failures, holes (uncovered areas) may appear in the sensing coverage. Existing approaches relocate redundant nodes to fill the holes and require all or most sensor nodes to have mobility. However, mobility equipment is expensive while technology trends are scaling sensors to be smaller and cheaper. In this paper, we propose to use a small number of mobile robots to replace failed sensors for a large-scale static sensor network. We study algorithms for detecting and reporting sensor failures and coordinating the movement of robots that minimize the motion energy of mobile robots and the messaging overhead incurred to the sensor network. A manager receives failure reports and determines which robot to handle a failure. We study three algorithms: a centralized manager algorithm, a fixed distributed manager algorithm, and a dynamic distributed manager algorithm. Our analysis and simulations show that: (a) the centralized and the dynamic distributed algorithms have lower motion overhead than the fixed distributed algorithm; (b) the centralized algorithm is less scalable than the two distributed manager algorithms, and (c) the two distributed algorithms have higher messaging cost than the centralized algorithm. Hence, the optimal choice of the coordination algorithm depends on the specific scenarios and objectives being optimized.

Keywords: Sensor networks; Mobile robots; Sensor replacement; Motion overhead; Wireless ad hoc networks

1. Introduction

Sensor networks have been intensively studied. Many studies focus on how to effectively collect and transfer data through energy-aware and/or fault-tolerant routing techniques [1,2,13,19,21,23]. Sensor networks are assumed unattended in various environments such as disaster areas, hazard fields, or battle fields. It is desirable to deploy and maintain the sensor network autonomously. There have been many studies on sensor deployment [8,10,18,27,33], for example, moving sensors to desired locations to provide coverage. Sensor nodes usually have simple designs, and their components are prone to failures. This can be especially serious in a hazardous environment, such as high temperatures or humidity. After a sensor network is deployed, some nodes may fail and leave holes in coverage. It is necessary to fill the holes and keep desired coverage. One way of maintaining coverage is to replace failed nodes with functional ones. This is called *sensor replacement*. Our paper focuses on sensor replacement upon failures.

Wang et al. [29] propose using some redundant mobile sensors relocating themselves to fill the holes. They present a cascading movement method to balance the energy cost and the response time of sensor replacement. In their study, sensor nodes are assumed to have mobility. However, mobility is an expensive feature, as mobile sensors need to have motors, motion control, and GPS modules. In large-scale sensor networks, the nodes are usually assumed to be small and cheap. Adding mobility to a large number of sensor nodes is expensive.

In this paper, we propose using only a few robots to assist sensor replacement. All robots are mobile and can pick, carry, and unload sensor nodes. When nodes fail, the robots move to the locations of the failed nodes and

* Corresponding author. Tel.: +1 765 494 9143.
E-mail addresses: ymei@purdue.edu (Y. Mei), cjx@purdue.edu (C. Xian), smdas@purdue.edu (S. Das), ychu@purdue.edu (Y.C. Hu), yunglu@purdue.edu (Y.-H. Lu).

0140-3664/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2007.05.047

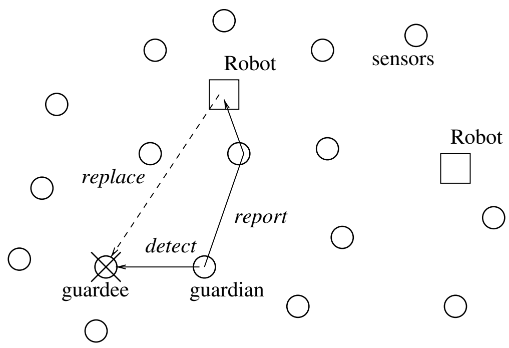

Fig. 1. An overview of sensor replacement using robots. The circles represent sensor nodes and squares represent robots. A guardian node detects a guardee's failure (shown by a cross over the circle) and then reports the failure to a robot. The robot moves to the failure location and replaces the failed node.

unload functional nodes. In our algorithms, a *manager* is a robot that receives failure reports and determines which robot to handle a specific failure. A *maintainer* is the robot that moves and replaces failed nodes. A robot can be both a manager and a maintainer. Senor nodes serve one another as *guardians* and *guardees* for failure detection.

Fig. 1 presents an overview of the robot assisted sensor replacement system. Since the number of robots is much smaller than the number of sensors, the cost is expected to be lower than requiring most of sensors to have mobility. Since sensors do not move, the routing algorithm can be more efficient for delivering sensing data than in mobile sensor networks. We present three different robot coordination algorithms: a centralized manager algorithm, a fixed distributed manager algorithm, and a dynamic distributed manager algorithm. The centralized algorithm has a central manager that receives failure reports from sensors and forwards them to individual robots. In the two distributed algorithms, the management responsibility is distributed over the robots, and each robot functions as both a manager and a maintainer. The three algorithms are compared in respect to motion overhead and messaging overhead. We perform simulations of the three algorithms using Glomosim [31], and the experimental results show that (a) the centralized or the dynamic algorithm can achieve lower motion overhead, (b) both the fixed and the dynamic algorithms are more scalable but also have higher messaging overhead.

The rest of the paper is organized as follows. Section 2 gives a formulation of the problem. Section 3 presents three different algorithms for solving this problem. Section 4 analytically compares the performance of the three algorithms. Section 5 presents experimental results. Finally, Section 6 discusses related work and Section 7 concludes the paper.

2. Problem statement

We make the following assumptions in this study:

(a) Sensors are randomly uniformly distributed into a 2-dimensional field. Their locations are known by themselves. This can be enabled in the initial deployment process. The lifetime of a node follows an exponential distribution.
(b) The robots travel at a constant speed and can localize themselves.
(c) All the sensors and the mobile robots communicate via wireless links and form a multi-hop wireless network.
(d) Replacement nodes are at the same locations as the corresponding failed nodes.

Using mobile robots for sensor replacement is essentially a coordination problem: how to coordinate multiple robots to maintain a large sensor network, in other words, to determine which robot should receive the failure report and which robot should replace the failed node. The problem can be divided into the following three stages.

(a) Initialization in three steps: (1) Setting up the roles of robots. A robot may be assigned to be a manager or a maintainer or both. (2) Setting up the initial relationship between the sensors and the robots. Every sensor node needs to know which robot is its manager and the location of the manager. (3) Setting up the guardian–guardee relationship among sensor nodes.
(b) Failure detection and reporting: when a guardian node detects a failure of its guardee node, the guardian reports the failure to a manager.
(c) Failure handling: once a failure is reported, the manager dispatches a maintainer to the failure location to replace the failed node with a functional one. When moving, the maintainer robot may need to update some other robots or some sensors with its new location.

The goal of this study is to minimize the motion and the messaging overhead. The motion overhead is measured as the energy and the time required for the robots' movement. The messaging overhead is measured as the number of wireless transmissions incurred for failure detection, reporting, and coordination. Replacement time is also one of the performance metrics, but it is directly related to the motion and messaging overhead: lowering the motion and messaging overhead reduces the replacement time.

3. Coordination algorithms

The coordination algorithms address two issues: how to report a failure, and which robot should handle the failure. We present three algorithms: (a) the centralized algorithm: a single robot serves as the manager, and all the other robots are maintainers; (b) the fixed algorithm: the sensor area is equally divided into fixed subareas and each robot independently handles one subarea; (c) the dynamic algorithm: the sensor area is dynamically divided into subareas based on the robots' current locations. In this section, we first describe the centralized algorithm in details. The description follows the three stages we introduced in the previous section. Then we present the fixed algorithm and the dynamic algorithm.

3.1. Centralized manager algorithm

The centralized algorithm has a robot functioning as the central manager. All failures are reported to the central manager. It then forwards failures to different maintenance robots. To simplify the design, we assume the manager does not move and is located at the center of the area to balance failure reports from all directions. Fig. 2 shows a typical scenario of this coordination algorithm.

3.1.1. Initialization

In this stage, there are three types of messages. First, the manager broadcasts its location to all the sensor nodes and all the maintenance robots. Second, each maintenance robot sends a message with its current location to the manager, and broadcasts its location to their one-hop neighbor sensors, i.e., those sensors that are within transmission range of the robot. Third, the sensors broadcast their locations for establishing guardian–guardee relationship for failure detection. Every sensor receives one message from each of its neighbors, picks its nearest neighbor as its *guardian*, and then sends a confirmation message to the guardian to establish the relationship. After initialization, all the sensors and robots know the manager's location, the manager knows all robots' locations, and the guardian–guardee relationship is established.

We use geographic routing protocols in the routing layer because they utilize location information to reduce the routing overhead. The location service needed by the routing layer is provided as part of the coordination algorithms in the application layer. The location broadcast in the initialization stage is for this purpose. In the subsequent stages, when a robot moves, it needs to inform the manager and some sensors of its new location.

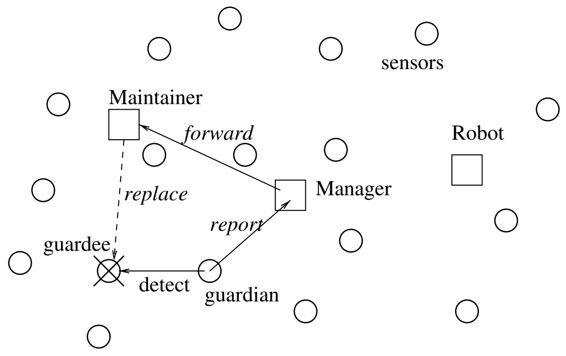

Fig. 2. An scenario in the centralized manager algorithm.

3.1.2. Failure detection and reporting

After initialization, each sensor node periodically sends beacon messages to its one-hop neighbor nodes. If a guardian has not received any beacon from a guardee for a certain amount of time (three beaconing periods in our study), the guardian conceives that the guardee has failed and sends a failure message with the failed node's location to the manager. We assume that the probability of both a guardian and a corresponding guardee fail close in time is small and negligible, and hence all the failures can be detected and reported to the manager. Similarly, if a guardee has not received any beacon from a guardian for a certain interval, it assumes the guardian has failed and selects a new guardian from its one-hop neighbors.

3.1.3. Failure handling

Upon a failure report, the manager selects a robot based on the *current location* of each robot, specifically, the manager selects the robot whose current location is the closest to the failure. To allow the manager to keep track of the current location of each robot, whenever a robot moves a certain threshold distance on its way to replace a failed sensor, the robot updates its location to the manager node, the robot, via geographic routing and to the one-hop neighbor sensors via broadcast. The threshold is decided based on the transmission range. If the threshold is too small, the update is too often and incurs high messaging overhead. If the threshold is too large, the robot may be unreachable sometimes. Upon receiving the request to replace a failed node, a robot moves to the failed node's location and replaces it by a functional node. When the robot is moving towards a failed sensor node, the robot still can receive replacement requests from the manager. The robot queues such requests and handles the failures in a first-come-first-serve fashion.

The pseudocodes of the centralized algorithm are shown in Figs. 3–5. They describe the behaviors of the sensors, the manager robot, and the maintainer robots, respectively. In the centralized algorithm, the manager has the global information of all the failures and all the robots. The algorithm can be effective for reducing the traveling distance and thus the motion energy of robots because the manager always forwards a failure to the closest robot. However, the central manager can become a performance bottleneck in a

```
SensorNode:
1   Receive the manager's initial location broadcast
2   Broadcast my location to all neighbor nodes
3   Receive broadcasting message from all neighbors
4   Setup guardian-guardee relationships
5   Set a timeout $\tau$
6   while (TRUE)
7         do if (timeout)
8               then Reset the timeout $\tau$
9                     Send a beacon to its guardian
10                    Check the last time receiving a beacon
11                    if the time duration is too long ($> 3\tau$)
12                       then Report failure to the manager
```

Fig. 3. A sensor node's behavior in the centralized algorithm.

```
MANAGER:
1  Broadcasting its location to all the sensors and robots
2  Receive locations from all maintenance robots
3  while (TRUE)
4        do if (Receive a failure message from a sensor)
5              then Retrieve the failure location
6                    Select a robot with closest location
7                    Forward the failure message
```

Fig. 4. The manager robot's behavior in the centralized algorithm.

```
MAINTAINER:
1  Receive the manager's initial location broadcast
2  Send its location to the manager
3  while (TRUE)
4        do if (Receive a failure message from the manager)
5              then Retrieve the failure location
6                    Move to the failure location
7                    Stop and replace the failed node
```

Fig. 5. A maintainer robot's behavior in the centralized algorithm.

large senor network where the distance traveled by the failure report and replacement request messages become too long.

3.2. Fixed distributed manager algorithm

In the fixed algorithm, the area is partitioned into equal-size subareas. The number of subareas is the same as the number of robots, and each robot is assigned with an equal-size subarea. A robot is both the manager and the maintainer for its subarea. We consider two types of area partition methods: squares or hexagons. Fig. 6 shows two methods. In the initialization stage, the robots first move to the centers of their corresponding subareas, and then broadcast their locations to all the sensors within their subareas. Each sensor node then selects the closest robot *myrobot* for failure reports. The setup of guardian–guardee relationships is the same as in the centralized algorithm, with the restriction that they belong to the same subarea. In this way, each robot independently handles failures within its own subarea. Once a sensor detects a failure, it reports the failure to its "myrobot". In the failure handling stage, a robot moves to a failure location and replaces the failed node.

Similarly to the centralized algorithm, sensors broadcast their location information to their one-hop neighbors in initialization. However, the location update of a moving

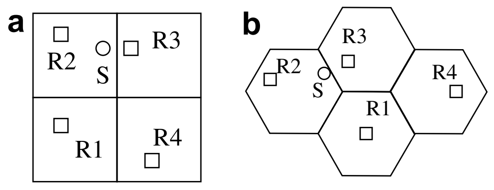

Fig. 6. Partition an area into (a) squares and (b) hexagons. The small squares represent the robots in each subarea and the small circles indicate failures.

```
SENSORNODE:
1  Receive all the robots' initial location broadcasts
2  Set the closest robot as myrobot
3  Broadcast my location to all neighbor nodes
4  Receive broadcasting message from all neighbors
5  Setup guardian-guardee relationships
6  Set a timeout $\tau$
7  while (TRUE)
8        do if (receive a new location update message)
9              then if (message.robot == myrobot)
10                   then update myrobot.location
11                        Relay the message
12             else  do nothing
13          if (timeout)
14             then Reset the timeout $\tau$
15                  Send a beacon to my guardian
16                  Check the last time receiving a beacon
17                  if the time duration is too long ($> 3\tau$)
18                     then Report the failure to myrobot
```

Fig. 7. A sensor node's behavior in the fixed algorithm.

```
ROBOT:
   /* The partitions are determined in advance*/
1  Move to the center of its fixed subarea
2  Broadcast its location to all sensors
3  while (TRUE)
4        do if (Receive a failure message from a sensor)
5           Move to the failure location
6           Stop moving and Replace the failed node
7           Broadcast its new location to its subarea
```

Fig. 8. A robot's behavior in the fixed algorithm.

robot is slightly different. Since all the sensors within the robot's subarea can potentially report a failure to the robot, all the sensors in a subarea need to receive the robot's location update. To achieve this, all the sensors relay the robot's location update. During this process, a sensor may receive the same update message multiple times, but it relays the message to its neighbors only once. This is achieved by remembering the sequence number of the robot location updates it has relayed before. The behaviors of the nodes and robots are shown by pseudo-codes in Figs. 7 and 8.

The fixed algorithm avoids the single manager bottleneck problem. However, the traveling distance tends to be longer as we will show by analysis and simulations in next two sections. For example in Fig. 6(a) and (b), robots are shown by small squares and nodes are shown by small circles. A node fails at location 'S', and it belongs to robot *R*2's subarea. For fixed partition algorithm, *R*2 moves to 'S' and replaces the failed node. However, *R*3 in a neighbor subarea is closer to 'S' than *R*2. The dynamic algorithm described in next subsection exploits this possibility by allowing dynamic area partition.

3.3. Dynamic distributed manager algorithm

As the name indicates, there are no fixed boundaries between two robots in the dynamic algorithm. The boundaries between robots are constructed dynamically as Voro-

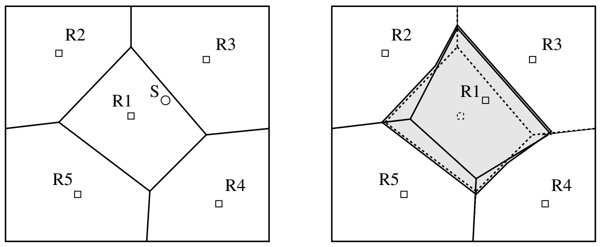

Fig. 9. Voronoi graphs. (a) Original Voronoi graph; a failure happens at ‘S’ inside R1’s subarea. (b) After R1 moves to ‘S’, the Voronoi graph changes; the original graph is shown by dashed lines. The shading area shows the area that the robot needs to update its location.

noi graphs. Voronoi graphs have been used in wireless multi-hop networks to study coverage-boundary detection by Carbunar et al. [5], and to study the coverage problem by Meguerdichian et al. [15]. Fig. 9 shows a Voronoi graph of five robots. The Voronoi graph partitions the area into five convex polygons with one robot in each subarea. Any sensor within each polygon is closer to the robot in that subarea than to any other robot. When a robot moves to replace failed nodes, the Voronoi graph needs to be adjusted dynamically.

The behaviors of sensors and robots are shown in Figs. 10 and 11. This algorithm dynamically maintains the Voronoi graph. This is not realized directly by exchanging locations between neighbor robots. On the contrary, robots do not know the border or the sensors they are responsible for. Instead, each sensor node decides which robot to report a failure to and sets the robot as its “myrobot”. This is achieved by robots’ dynamic broadcasting their locations to sensors (line 1 and line 6 in Fig. 11) and sensors’ dynamic updating their myrobots (lines from 8 to 15 in Fig. 10). Different from the fixed algorithm, the sensors

```
SensorNode:
1  Receive all the robots' initial location broadcasts
2  Set the closest robot as myrobot
3  Broadcast my location to neighbor nodes
4  Receive broadcasting message from all neighbors
5  Setup guardian-guardee relationships
6  Set a timeout $\tau$
7  while (TRUE)
8      do if (receive a new location update message)
9          then if (message.robot == myrobot)
10                 then Update the robot's location
11                      Update myrobot
12                      Relay the message
13         else if (the robot is closer than myrobot)
14                 then myrobot=message.robot
15                      Relay the message
16         else do nothing
17       if (timeout)
18         then Reset the timeout $\tau$
19              Send a beacon to my guardian
20              Check the last time receiving a beacon
21              if the time duration is too long ($> 3\tau$)
22                 then Detect a failure
23                      send the failure to myrobot
```

Fig. 10. A sensor node’s behavior in the dynamic algorithm.

```
Robot:
1  Broadcast its location to all sensors
2  while (TRUE)
3      do if (Receive a failure message from a sensor)
4          Move to the failure location
5          Stop and replace the failed node
6          Broadcast its new location
```

Fig. 11. A robot behavior in the dynamic algorithm.

that receive and relay the broadcasting message are not restricted to the nodes that are currently having the robot as their myrobot. Some nodes currently in the neighbor subareas may need to switch their myrobots to the moving robot, and such nodes may also need to relay the location update messages. In this way, the nodes update their myrobots dynamically to be the closest robot.

In the dynamic algorithm, since a sensor node reports a failure to the closest robot, the robot achieves similar traveling distance as in the centralized algorithm without suffering the scalability problem. Note, this is achieved at the cost of high messaging overhead, as the new location of a moving robot needs to be updated to many sensors. Compared with the fixed algorithm, the dynamic algorithm can have higher messaging overhead since some nodes in the neighbor subareas may also need to relay the location update messages. We will compare the messaging overhead in the next two sections.

4. Performance analysis

Our optimization goal is to reduce motion and messaging overhead. We use robot traveling distance to represent the motion overhead. For messaging overhead, we need to differentiate two different types of messages: unicast and multicast. In the centralized algorithm, failure report, failure forwarding, and robot location update are all unicast messages. In the distributed algorithms, failure report is unicast but robot location update is multicast. For unicast messages, we use message passing distance, the distance between the sender and the receiver, to represent the messaging overhead. For multicast messages, we use location update area, the size of the area that the multicast (robot location update) message reaches, to represent the messaging overhead. All the analysis is normalized relative to the number of failures.

4.1. Random variables

To compare the performance of the three algorithms, we assume all of them have the same number of maintenance robots, n. The whole area has the same size L^2 for all the three algorithm. Each subarea in the fixed algorithm has a size of $l^2 = \frac{L^2}{n}$, and thus $l = \frac{L}{\sqrt{n}}$. We use $\mathscr{A}$ and $a_1, a_2, \ldots$, and a_n to represent the whole area and the n subareas in the distributed algorithms. We use 2-dimensional variables $S_1, \ldots, S_k, \ldots$ to represent the failure locations in the order of their occurring times, where

$S_k = (S_k^x, S_k^y)$ represents two coordinates of a location. The maintenance robots are moving around to replace failed node. Their locations can be represented by random variables $R_1, R_2, \ldots, R_n$, and $R_m = (R_m^x, R_m^y)$, $m = 1, \ldots, n$.

In the centralized or the dynamic algorithms, since the nodes are evenly randomly distributed over the area and they are equally likely to fail, the failures $S_1, \ldots, S_k, \ldots$ are identical independent distributed (IID) random variables. The probability distribution function (PDF) of S_k is $f(S_k) = \frac{1}{L^2}$, if $S_k \in \mathscr{A}$. The robots' locations $R_1, \ldots, R_n$ have also the same PDFs as failures' PDFs, but they are not independent of each other. The reason is that when a new failure happens, the decision of which robot is selected to move to the failure location is not made randomly, but depending on the robots' current locations and the failure's location.

In the fixed algorithm, each robot is confined within its own subarea; events happened in one subarea are independent of the events in other subareas. We can focus only on one subarea to analyze the fixed algorithm. If S_k happens in the mth subarea, its PDF is $f(S_k) = \frac{1}{l^2}$, if $S_k \in a_m$. In the fixed algorithm, $R_1, R_2, \ldots, R_n$ are and the PDF of R_m is $f(R_m) = \frac{1}{l^2}$, if $R_m \in a_m$. In the rest of this section, we analyze the performance for the three algorithms using these random variables. The fixed algorithm is investigated first for its simplicity.

4.2. Fixed algorithm

Since all the subareas are independent, we only need to analyze one subarea.

4.2.1. Motion overhead

For motion overhead, we consider the robot's traveling distance from its current location to the failure's location $d = \sqrt{(S_k^x - R_m^x)^2 + (S_k^y - R_m^y)^2}$. The average value of traveling distance per failure is the expected value of d. Since we consider one subarea, S_k and R_m are evenly distributed inside the subarea. As indicated in last section, we partition the area with two methods, squares or hexagons, shown by Fig. 6. We run Monte Carlo simulations to get the expected values of d. For a square-shape subarea with a size of l^2, the expected value of d is $0.5214l$; for a hexagon-shape subarea with the same size1, the expected value of d is $0.5131l$. From the two values, we can see a hexagon partition has slightly (1.6%) lower average traveling distance than a square partition.

4.2.2. Messaging overhead

There are three types of messaging overhead: failure detection (beaconing), failure reporting, and robot location update. Since failure detection is the same for all the three algorithms, we only consider the latter two. A failure report is a unicast message. We investigate the report distance, the distance between a guardian that reports a failure and the robot who receives the report. For robot location broadcast, we investigate location update area. The guardian's location and the robot's location are two IID and evenly distributed random variables. Therefore, the failure report distance has the same distribution as the robot traveling distance d. The expected value of the failure report distance is $0.5214l$ for square-shape and $0.5131l$ for hexagon-shape. The robot location broadcast reaches all the sensors within the robot's subarea. Therefore, the robot location update area is l^2.

4.3. Centralized manager algorithm

4.3.1. Motion overhead

The n robots are located at $R_1, R_2, \ldots, R_n$. Upon a failure happening at S_k, the manager compute n distances $d_m = \sqrt{(S_k^x - R_m^x)^2 + (S_k^y - R_m^y)^2}$, $m = 1, \ldots, n$. Suppose $d_i = \min\{d_1, \ldots, d_n\}$, the robot R_i is selected to replace the failure S_k. The average traveling distance per failure is the expected value of the minimum distance $\min\{d_1, \ldots, d_n\}$.

We assume the shape of the whole area $\mathscr{A}$ is a square, and we run Monte Carlo simulations to calculate average traveling distance with different numbers of robots. The results at different values of n are shown in Fig. 12. The size of the area is $L^2 = nl^2$, and the average traveling distance is in fact proportional to l, i.e., $\frac{L}{\sqrt{n}}$. When $n = 1$, there is only one robot and the average distance is the same as the average distance in the fixed algorithm. When n increases, the total area size L^2 increases proportionally, but the average distances decreases as shown in the figure. For example, the average distance decreases from $0.5124l$ to $0.4762l$, or by 8.67%, when n increases from 1 to 9. This shows the centralized manager algorithm can achieve lower robot traveling distance than the fixed algorithm when there are multiple maintenance robots.

4.3.2. Messaging overhead

Apart from failure detection, the centralized algorithm has three types of messages: failure report, failure forward-

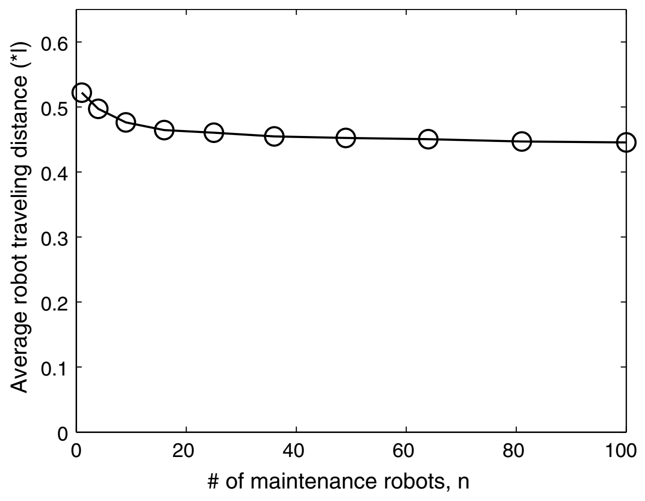

Fig. 12. Average robot traveling distance at different number of robots n.

ing, and robot location update. They are all unicast messages. The failure report is between the manager and a guardian; failure forwarding is between the manager and a selected robot; location update is between a robot and the manager. The location of the manager is fixed at the area's center. The message passing distance for all the three types of messages are a distance between an evenly distributed random variable and the center of the area. Monte Carlo simulations show that the average message passing distance is $0.3821L = 0.3821\sqrt{n}l$. The message passing distance increases as n increases.

4.4. Dynamic algorithm

4.4.1. Motion overhead

In the dynamic algorithm, a sensor reports a failure to the closest robot. Therefore, the traveling distance is the same as that in the centralized algorithm. The distance decreases as n increases, shown in Fig. 12.

4.4.2. Messaging overhead

There are two types of messages in the dynamic algorithm: failure report and robot location update, similar as in the fixed algorithm. The failure report is a unicast message between a guardian and a robot. Therefore, the failure report distance has the same distribution as the robot traveling distance. The average message passing distance is shown in Fig. 12. The location update is a multicast message that reaches both those sensors currently belonging to the robot, and some sensors in neighbor Voronoi cells that may change their *myrobot*s to this robot. The broadcast area is a union of the robot's previous Voronoi cell and the robot's Voronoi cell after the broadcast and adjustment, as shown by the shading area in Fig. 9 (b).

4.5. Overhead comparison

Table 1 is a summary of this section. We assume the area is a square and there are 9 maintenance robots, i.e., $L^2 = 9l^2$. From the table, we can draw the following conclusions.

4.5.1. Scalability

The fixed and the dynamic algorithms have better scalability than the centralized algorithm. We can see that the overhead in the fixed and dynamic algorithms does not increase as n increases, while the message passing distance in the fixed algorithm increases as n increases.

Table 1
Overhead comparison of the three coordination algorithms with 9 maintenance robots

Algorithm	Motion overhead	Message overhead (distance/area)
Centralized	$0.48l$	$0.38\sqrt{n}l/(N/A)$
Fixed (hexagons)	$0.51l$	$0.51l/l^2$
Fixed (squares)	$0.52l$	$0.52l/l^2$
Dynamic	$0.48l$	$0.48l/(>l^2)$

Motion overhead is represented by the average traveling distance per failure. Message overhead includes the massage passing distance and the location update area. The value of l is the square root of the size of average area per robot.

4.5.2. Motion overhead

The dynamic algorithm has the same motion overhead as that in the centralized algorithm, while the fixed algorithm has higher motion overhead. This shows that robots cooperating with each other can reduce the traveling distance and the motion overhead.

4.5.3. Messaging overhead

In general, the centralized algorithm has lower messaging overhead than the two distributed algorithms. The reason is that all the messages in the centralized algorithm are unicast, while the location update in the distributed algorithms is multicast. The dynamic algorithm has higher messaging overhead than the fixed algorithm, because the former has larger location update areas; in other words, the robot updates its location to more sensors.

5. Experiments

In this section, we compare the relative performance of the three algorithms using simulations.

5.1. Experimental setup

We use Glomosim to simulate using robots to replace failed nodes. Glomosim [31] is a packet-level simulator for ad hoc networks with an accurate radio model. The link layer uses IEEE 802.11, and the radio model has a nominal bit-rate of 11 Mbp. We differentiate three types of nodes: sensors, robots, and manager (for the centralized algorithm). The manager and the maintenance robots have the same transmission range of 250m, while the sensors have a transmission range of 63m to save the sensor's power.

In our simulations, the sensors are static, and robots move around to replace sensors. We have implemented an on-demand mobility model in which robots move on demand after receiving a failure report. The failure detection, report and replacement request are implemented in the application level. We implement a geographic routing protocol based on face-routing [4] whose implementation parameters are same as GPSR [13]. The protocol is described in the next subsection.

In realistic scenarios, the failure happening rate is expected to be low and robots spend most of the time waiting for failure replacement requests. This conserves the robots' energy consumption.

We selected the following simulation parameters: (1) The average area per robot is $200 \times 200m^2$. (2) The robots' speed is 1 m/s that is based on the specification of Pioneer 3DX robots [16]. (3) The sensor node density is $\frac{50}{200\times 200m^2}$, i.e., each robot is in charge of 50 nodes on average. (4) The number of maintenance robots varies from 1 to 16.

The sensor area and the number of sensors change with different number of robots. For example, with 16 robots, the sensor area is $800 \times 800m^2$ with total 800 sensors. (5) In the fixed algorithm, the area is partitioned into squares. (6) The sensors' expected lifetime is 16,000 s. (7) The simulation time is 64,000 s. (8) The sensors' beaconing period for failure detection is 10 s. We choose a relatively short expected lifetime so that the simulation can finish in a reasonable amount of time, while still showing the relative maintenance overhead for different algorithms.

5.2. Geographic routing and location service

Routing in our system is implemented by geographic forwarding. As we discussed in Section 2, each node knows its own geographic location and each robot can localize itself. During initialization, all the sensors broadcast their locations to their one-hop neighbors (within the radio transmission range) and the robots also broadcast their locations to some sensors and some other robots depending on the algorithms. Each node maintains the node ID and location information of its neighbors. Subsequently, a packet can be routed to a destination node using the local state at each node. Each packet contains the destination address in the IP header and the destination's location (x- and y-coordinates) in an IP option header. To forward a packet, a node searches its neighbor table and forwards the packet to its neighbor closest in geographic distance to the destination's location. Under the above greedy geographic forwarding, forwarded packets may potentially reach a node that does not know any other node closer to the destination than itself. This indicates a hole in the geographical distribution of nodes. Recovering from holes is possible using approaches such as GFG [4], GPSR [13], which use planar subgraphs to route around holes.

After initialization, sensors periodically send out beacons to their one-hop neighbors for failure detection. In addition, when one of the following two events occurs, extra updates of location information are required. (a) When a node detects a neighbor sensor node's failure, it deletes the failed neighbor from its neighbor table. After a failed node is replaced, the new node broadcasts its location to its one-hop neighbors. The neighbors then add the new node into their neighbor tables, and also send beacons containing their own locations. This enables the new node to set up its own neighbor table. (b) When a robot moves, the robot needs to send updates containing its new location. In the centralized algorithm, the robot updates the manager and the robot's one-hop neighbors of its new location. In the fixed algorithm, the robot updates all the sensors within it subarea of its new location. In the dynamic algorithm, some sensors in the neighbor subareas also need to change their "myrobots" and relay the location update message. In all the three algorithms, the robot updates its location whenever it moves away from the last updated location by a distance threshold. The threshold depends on the sensors' transmission range. In the simulations, we choose this threshold to be $20m$, less than $\frac{1}{3}$ of the sensors' transmission range ($63m$) to ensure that the robots can receive failure messages all the time.

5.3. Simulation results

We run simulations with three different numbers of maintenance robots: 4, 9, and 16. We choose square numbers to make area partition easy. We skip one robot experiments, since there is little difference among the three algorithms. For a simulation with k^2 maintenance robots, the total area size is set to be $200 \times 200 \times k^2 m^2$, and there are a total of $50k^2$ sensors. For the fixed algorithm, we only show the results for the square partition method, because the hexagon partition method shows negligible difference with respect to the overheads.

5.3.1. Motion overhead

Fig. 13 presents the average traveling distance per failure with different numbers of maintenance robots. The dynamic algorithm has similar motion overhead as the centralized algorithm, while the fixed algorithm has higher motion overhead. For example, with 16 maintenance robots, the dynamic algorithm can save 10.8% traveling distance compared with the fixed algorithm. In the fixed algorithm, a sensor's failure is reported to the robot in that subarea. There is no cooperation between robots and a sensor's failure is not reported to the closest robot. In the centralized or the dynamic algorithm, a failure is reported to the closest robot and the robots coordinate with each other indirectly. For this reason, the two algorithms have lower motion overhead than the fixed algorithm.

5.3.2. Messaging overhead

The messaging overhead can be divided into four parts: initialization, failure detection, failure report, and robot location update. Since all three algorithms are similar in

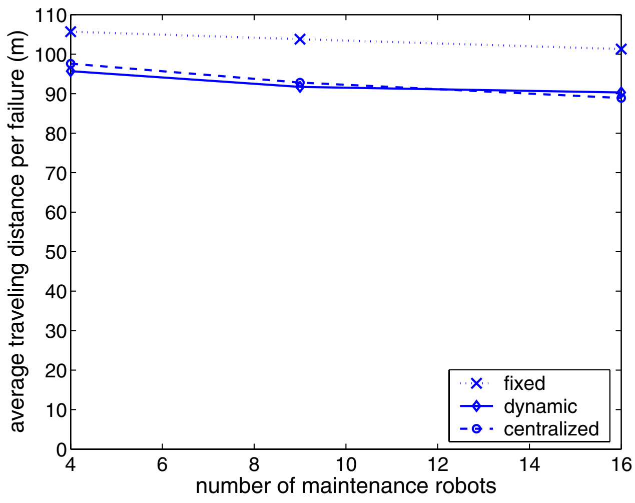

Fig. 13. The average robot traveling distance as a function of the number of robots.

initialization and failure detection, we focus on the overhead from failure report and location update.

Fig. 14 shows the average number of hops for failure reports or replacement requests. Since we use geographic routing, the number of hops is roughly proportional to the distance between the report node (sender) and the manager (receiver). The failure reports and replacement requests are delivered with 100% delivery ratio due to the high density of sensor nodes and low traffic load in the network. In the dynamic or the fixed algorithm, the failure is reported from a sensor node to a robot, and the distance between them is about 100 m on average. This distance has the same average value as the robot traveling distance. Since the sensors' transmission range is 63 m, passing a message cross 100 m requires at least two hops. From the figure, the average number of hops traveled by the failure reports in the dynamic or the fixed algorithm is stable at about 2. This validates the effectiveness of the geographic routing. In the centralized algorithm, we consider both failure report and failure forwarding. Statistically, the distances traveled by failure reports and replacement requests have the same distribution, and they increase as the area size increases because the central manager is always at the area center. However, the average number of hops traveled by failure reports and replacement requests differ. The reason is that sensors and robots have different transmission ranges. Failure reports have larger numbers of hops. The two curves in Fig. 14 for centralized algorithm show the increasing trend and their difference. When the area increases, the number of maintenance robots increases accordingly, and the number of hops in the centralized algorithm also increases. Therefore, compared with the dynamic or the fixed algorithm, the centralized algorithm is less scalable.

Fig. 15 shows the messaging overhead of location updates in the three algorithms. We use the number of transmissions as the metric. The dynamic and the fixed algorithms result in much higher average number of transmissions per failure. Also, the dynamic algorithm has a slightly higher number of transmissions than the fixed algorithm, as we discussed in the last paragraph of Section 3.3. This is because many sensors in the fixed or the dynamic algorithm need to relay the robot's location update messages, while there is no sensor relay in the centralized algorithm. The sensor relays greatly increase the probability that each sensor can receive the location updates from its "myrobot". However, requiring all the sensors inside the robot's subarea to rebroadcast the robot's location update may introduce many redundant transmissions. The high messaging overhead in the two distributed algorithms can be reduced by using more efficient broadcast schemes (e.g. [25]) which require only a subset of the sensors in each subarea to relay the location update messages, or rendezvous-based location services such as quorum-based schemes (e.g. [24]), hierarchical hashing based schemes (e.g. [14]), or home-region/geographic hashing based schemes (e.g. [22,7]). This is part of our future work. Combining the two types of messaging overhead, we can see that the dynamic and the fixed algorithms have higher messaging overhead than the centralized algorithm.

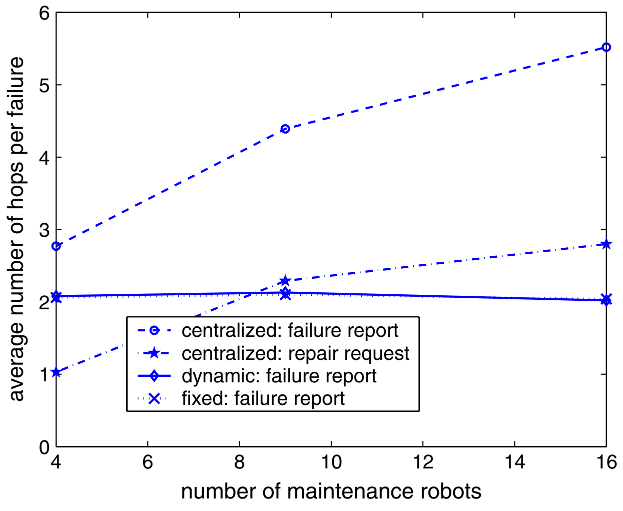

Fig. 14. The average message passing hops per failure.

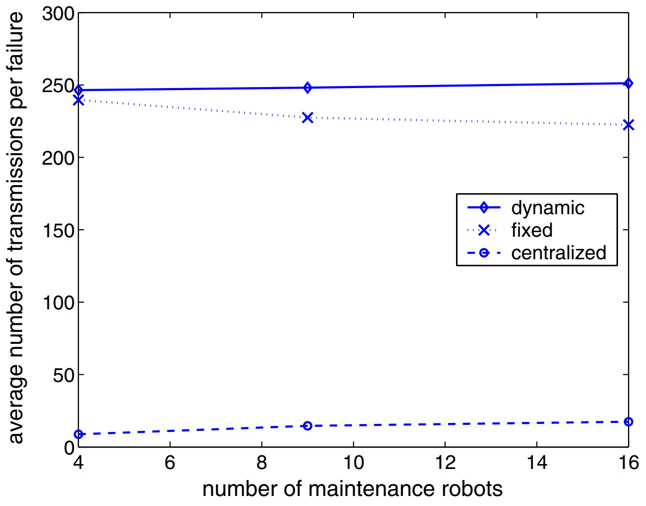

Fig. 15. The average number of transmissions for location update per failure.

5.3.3. Replacement time

Fig. 16 shows the average replacement time defined as the duration between a failure and the time the failed sensor is replaced. The time depends on three steps: detection, report, and handling. On average, the time of failure detection is the same for the three algorithms. The failure report time in the centralized algorithm tends to be longer because the failure report or replacement request travels a larger number of hops. For the third step, the fixed algorithm has longer failure handling time because the robot travels longer to replace a failed node. Adding the three steps together, the dynamic algorithm has shorter average replacement time. This is shown in the Fig. 16. With 16 maintenance robots, the replacement time of the dynamic algorithm is about 10.2% shorter.

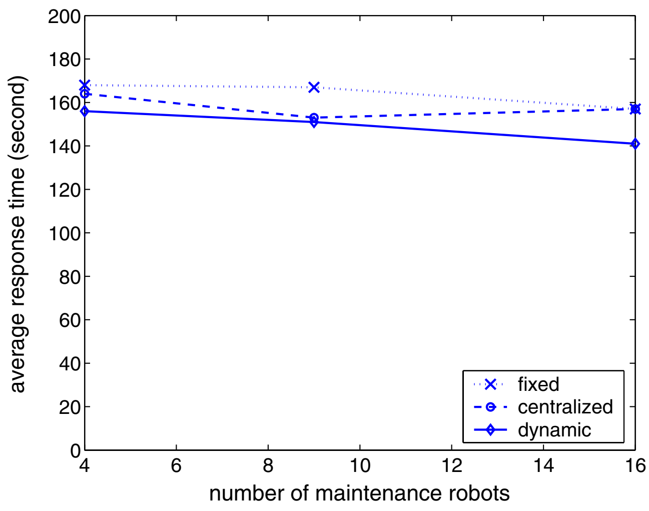

Fig. 16. The average replacement time per failure.

5.3.4. *Unevenly distributed failures*

A basic assumption in this study up to this point is that all the sensors have the same failure probability distribution. However, this may not be true. For example, some nodes may locate in a swamp and some others on a dry land. It is easy to understand that the dynamic algorithm have better adaptivity than the fixed algorithm. Therefore, with unevenly distributed failures, we expect the dynamic algorithm to have better advantages than the fixed algorithm. Fig. 17 shows a scenario where the failures has a 9 times higher probability to be happened in half of the area than in the other half. The failure probability within each half is evenly distributed. The figure presents the average traveling distance of the three different algorithms. Compared with Figs. 13, 17 show larger differences between the fixed algorithm and the dynamic algorithm. With 16 robots, the dynamic algorithm achieves 15.6% lower traveling distance than the fixed algorithm. The dynamic algorithm has better adaptivity to the unevenly distributed failures than the fixed algorithm.

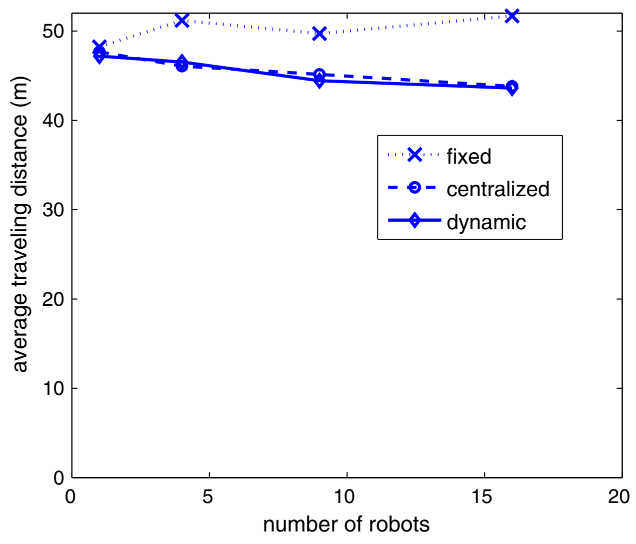

Fig. 17. The average traveling distances with different number of robots. The failures are 9 times more likely happened in one half of the area than in the other half.

6. Related work

Meguerdichian et al. [15] first address the coverage problem and use Voronoi diagram to establish an optimal polynomial-time algorithm for coverage calculation. For static sensor network, Wang et al. [30] present the Coverage Configuration Protocol (CCP) that determines which sensors should be turned on to achieve the desired coverage and the rest sensors can be turned off to save energy. Zhang et al. [32] propose the Optimal Geographical Density Control (OGDC) protocol to minimize the overlap of sensing areas of all sensors. Tian et al. [26] propose turning off a node if its sensing area can be completely covered by its neighbors. To avoid the possibility of multiple neighbors turning off and creating a coverage hole, the nodes use a random back off algorithm before going to sleep. Several studies have considered using robots for initial sensor deployment, for example, [3,6].

The above techniques are insufficient to handle the case where a coverage hole exists even all sensors around the hole use their maximum sensing ranges. This problem can be solved by relocating some sensors from the densely deployed subarea to the holes to amend the coverage. Wang et al. [27] propose relocating redundant mobile sensors to fill the coverage holes and using a cascading movement method to balance the energy cost and the replacement time of sensor relocation. Ganeriwal et al. [9] propose to make low energy nodes, on predicting death, broadcast a Panic-Request message. Nodes with high energy levels respond with the Panic-Reply message if they can move without losing existing coverage. Howard et al. [11] propose a potential fields based approach for self-deployment of mobile sensor networks. In such potential fields, nodes are treated as virtual particles and the virtual force between two nodes is larger if the two nodes are closer. As the virtual forces repel the nodes from each other, the sensors tend to form a uniform distribution in the sensor area without coverage holes. These methods require nodes equipped with motors, steering devices, and/or GPS. These components are very expensive while a large-scale sensor network usually assumes that the sensors are small and cheap. Meanwhile, higher-capacity batteries may be required for mobile sensors because mobility is also an energy-consuming feature. Recently, some studies have been focusing on hybrid sensor networks with both static and mobile sensors [20,28]. Hybrid sensor network can compensate the disadvantages of pure static sensor networks without enabling all sensors mobility. Ingelrest et al. [12] describe several broadcasting strategies for hybrid sensor networks in a more general context. However, their strategies primarily optimize broadcasting for fixed infrastructure points. Our solutions also require multi-hop broadcasting but are targeted at mobile robot nodes.

Our previous work [17] presents three coordination algorithms for using mobile robots to replace failed sensor nodes. Sensor networks and robots together form an autonomous system with self maintenance capability. Compared with the above solutions which use mobile sensors, this approach has more flexibility in designing sensor networks and are more cost effective. This paper is an extension over the previous paper. We use random variables to analyze the main properties of the algorithms. We add more figures and pseudocodes to explain the three algorithms. We adopt a new metric, replacement time, to evaluate the performance of the algorithms. Finally, we conduct simulations to show adaptivity of the algorithms to uneven distributed failures.

7. Conclusion

In this paper, we have proposed using mobile robots for sensor replacement. Sensors detect failures of their neighbors and report to robots. Robots move and replace failed nodes with functional ones. We have presented three different algorithms and analyzed their performance by both random variables and simulations. Simulation results confirm performance analysis using random variables. It has been shown that the centralized algorithm is not scalable as the message passing distance increases with the sensor network area. The dynamic and the centralized algorithms have lower motion overhead. The fixed and the dynamic algorithms have higher messaging overhead. The dynamic algorithm can achieve 10% lower replacement time than the other two algorithms. The centralized and the dynamic algorithms can adapt to unevenly distributed failures better. In general, the three algorithms have different properties, and the optimal choice of the algorithm depends on specific scenarios and objectives being optimized. In our future work, we will study more efficient location update mechanisms to reduce the messaging overhead in the dynamic and the fixed algorithms.

Acknowledgments

This work is supported in part by the National Science Foundation IIS-0329061. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–114.
[2] S. Basagni, I. Chlamtac, V.R. Syrotiuk, B.A. Woodward, A distance routing effect algorithm for mobility (DREAM), in: International Conference on Mobile Computing and Networking, pp. 76–84, 1998.
[3] M.A. Batalin, G.S. Sukhtame, Coverage, exploration and deployment by a mobile robot and communication network, Telecommunication Systems Journal, Special Issue on Wireless Sensor Networks 26 (2) (2004) 181–196.
[4] P. Bose, P. Morin, I. Stojmenovic, J. Urrutia, Routing with guaranteed delivery in ad hoc wireless networks, in: International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, pp. 48–55, 1999.
[5] B. Carbunar, A. Grama, J. Vitek, Distributed and dynamic voronoi overlay maintenance for coverage detection and distributed hash tables in ad hoc networks, in: IEEE International Conference on Parallel and Distributed Systems, pp. 549–556, 2004.
[6] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, G. Sukhatme, Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle, in: IEEE 2004 International Conference on Robotics and Automation, pp. 3602–3608, May 2004.
[7] S.M. Das, H. Pucha, Y.C. Hu, Performance comparison of scalable location services for geographic ad hoc routing, in: IEEE INFOCOM, pp. 1228–1239, March 2005.
[8] S.S. Dhilon, K. Chakrabarty, S.S. Iyengar, Sensor placement for grid coverage under imprecise detections, in: International Conference on Information Fusion, pp. 1581–5187, 2002.
[9] S. Ganeriwal, A. Kansal, M.B. Srivastava, Self aware actuation for fault repair in sensor networks, in: IEEE International Conference on Robotics and Automation, pp. 5244–5249, May 2004.
[10] H. Gnzalez-Banos, J.-C. Latombe, A randomized art-gallery algorithm for sensor placement, in: Annual Symposium on Computational Geometry, pp. 232–240, 2001.
[11] A. Howard, M.J. Mataric, G.S. Sukhatme, Mobile sensor network deployment using potential fields: a distributed, scalable solution to the area coverage problem, in: International Symposium on Distributed Autonomous Robotics Systems, pp. 299–308, June 2002.
[12] F. Ingelrest, D. Simplot-Ryl, I. Stojmenovic, Routing and broadcasting in hybrid ad hoc and sensor networks, in: Jie Wu (Ed.), Handbook on Theoretical and Algorithmic Aspects of Sensor Ad Hoc Wireless and Peer-to-Peer Networks, Auerbach Publications, Boca Raton, Florida, USA, 2006.
[13] B. Karp, H.T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, in: International Conference on Mobile Computing and Networking, pp. 243–254, 2000.
[14] J. Li, J. Jannotti, D.S.J.D. Couto, D.R. Karger, R. Morris, A scalable location service for geographic ad hoc routing, in: Proceeding of ACM Mobicom, pp. 120–130, 2000.
[15] S. Meguerdichian, F. Koushanfar, M. Potkonjak, M. Srivastava, Coverage problems in wireless ad-hoc sensor networks, in: INFOCOM, pp. 1380–1387, April 2001.
[16] Y. Mei, Y.-H. Lu, Y.C. Hu, C.G. Lee, A case study of mobile robot's energy consumption and conservation techniques, in: International Conference on Advanced Robotics, pp. 492–497, 2005.
[17] Y. Mei, C. Xian, S.M. Das, Y.C. Hu, Y.-H. Lu, Replacing failed sensor nodes by mobile robots, in: Workshop on Wireless Ad Hoc and Sensor Networks, 2006.
[18] S. Poduri, G.S. Sukhatme, Constrained coverage for mobile sensor networks, in: IEEE International Conference on Robotics and Automation, pp. 165–171, 2004.
[19] G.J. Pottie, W.J. Kaiser, Wireless integrated network sensors, Communications of the ACM 43 (5) (2000) 51–58.
[20] X. Shan, J. Tan, Mobile sensor deployment for a dynamic cluster-based target tracking sensor network, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 741–746, 2005.
[21] K. Sohrabi, J. Gao, V. Ailawadhi, G.J. Pottie, Protocols for self-organization of a wireless sensor network, IEEE Personal Communication 7 (5) (2000) 16–27.
[22] I. Stojmenovic, Home region based location updates and destination search schemes in ad hoc wireless networks, Technical Report TR-99-10, Computer Science, SITE, University of Ottawa, September 1999.
[23] I. Stojmenovic, X. Lin, Power-aware localized routing in wireless networks, IEEE Transactions on Parallel and Distributed Systems 12 (11) (2001) 1122–1133.
[24] I. Stojmenovic, B. Vukojevic, A routing strategy and quorum based location update scheme for ad hoc wireless networks, Technical

Report TR-99-09, Computer Science, SITE, University of Ottawa, September 1999.

[25] I. Stojmenovic, J. Wu, Broadcasting and activity-scheduling in ad hoc networks, in: S. Basagni, M. Conti, S. Giordano, I. Stojmenovic (Eds.), Mobile Ad Hoc Networking, IEEE/Wiley, Hoboken, New Jersey, USA, 2004.

[26] D. Tian, N.D. Georganas, A coverage-preserving node scheduling scheme for large wireless sensor networks, in: The 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp. 32–41, September 2002.

[27] G. Wang, G. Cao, T.L. Porta, Movement-assisted sensor deployment, in: INFOCOM, vol. 4, pp. 2469–2479, 2004.

[28] G. Wang, G. Cao, T.L. Porta, A bidding protocol for deploying mobile sensors, in: The 11th IEEE International Conference on Network Protocol, pp. 315–324, November 2003.

[29] G. Wang, G. Cao, T.L. Porta, W. Zhang, Sensor relation in mobile sensor networks, in: INFOCOM, vol. 4, pp. 2302–2312, 2005.

[30] X. Wang, G. Xing, Y. Zhang, C. Lu, C. Pless, C. Gill, Integrated coverage and connectivity configuration in wireless sensor networks, in: ACM SenSys, pp. 28–39, November 2003.

[31] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks, in: Workshop on Parallel and Distributed Simulation, pp. 154–161, 1998.

[32] H. Zhang, J.C. Hou, Maintaining sensing coverage and connectivity in large sensor networks, International Journal of Wireless Ad Hoc and Sensor Networks 1 (1–2) (2005) 89–124.

[33] Y. Zou, K. Chakrabarty, Sensor deployment and target localization based on virtual forces, in: INFOCOM, vol. 2, pp. 1293–1303, 2003.

Yongguo Mei received the M.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2001, and the Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, USA in 2007. Currently he is with the Windows fundamentals client performance group at Microsoft. His research interests include sensor networks, mobile robotics, energy-efficient compter systems, and performance optimization for operating systems. Dr. Mei is a member of IEEE.

Changjiu Xian is a Ph.D. candidate in the Department of Computer Science at Purdue University, West Lafayette, IN, USA. He received his BS degree from Beijing University in 1998 and MS degree from Purdue University in 2001. His research interests include low power design for handhelds, mobile robots, and sensor networks.

Saumitra Das is currently a Ph.D. candidate in the School of Electrical and Computer Engineering at Purdue University, USA. Previously, he received a MS degree from Carnegie Mellon University, USA and a B.Engg. degree from the University of Bombay, India. His research interests include cross-layer system design for multi-hop wireless networks, scalable routing strategies in wireless ad hoc networks, and mobile robotics.

Y. Charlie Hu is an Associate Professor of Electrical and Computer Engineering and Computer Science at Purdue University. He received his M.S. and M.Phil. degrees from Yale University in 1992 and his Ph.D. degree in Computer Science from Harvard University in 1997. From 1997 to 2001, he was a research scientist at Rice University. Dr. Hu's research interests include operating systems, distributed systems, Internet routing and measurement, and wireless networking. He has published over 100 papers in these areas. Dr. Hu received the NSF CAREER Award in 2003. He served as a TPC Vice Chair for ICDCS 2007 and the 2004 International Conference on Parallel Processing and a co-founder and TPC co-chair for the International Workshop on Mobile Peer-to-Peer Computing. Dr. Hu is a member of USENIX and ACM and a senior member of IEEE.

Yung-Hsiang Lu is an Assistant Professor in the School of Electrical and Computer Engineering, and by courtesy, the Department of Computer Science at Purdue University, West Lafayette, IN, USA. In 2004, he obtained NSF CAREER Award for studying energy management by operating systems. His research focuses on energy management for sensor networks, mobile robots, computer systems, and image processing. He received Ph.D. in Electrical Engineering from Stanford University, California, USA in 2002.