Sensor Relocation with Mobile Sensors: Design, Implementation, and Evaluation

Jie Teng, Tim Bolbrock, Guohong Cao, and Tom La Porta
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
E-mail: {teng,gcao,tlp}@cse.psu.edu timbolbrock@gmail.com

Abstract—**Mobile sensors are useful in many environments because they can move to increase the sensing coverage. In this paper, we present a mobile sensor prototype in which the Mica2 sensor node is used to control the movement of the robot built with commercial off-the-shelf (COTS) components. We use a sensor relocation application to demonstrate the feasibility of our design. In the sensor relocation application, after a sensor node failure creates a coverage hole, a mobile sensor node is relocated to cover the hole in a timely and energy-efficient way. We present a distributed sensor relocation algorithm and provide novel solutions to implement this algorithm in our mobile sensor platform. Experimental results show that our relocation algorithm can reduce the sensor relocation time and balance the energy consumption of the mobile nodes.**

Index Terms: Sensor networks, mobile sensor, mobile robot, mote, sensor relocation.

I. Introduction

Recent advances in hardware design are enabling low-cost sensors that have sophisticated sensing, communication, and computation capabilities. These sensors communicate via radio transmitters/receivers to form a multihop wireless network, i.e., a distributed wireless sensor network. Typical applications of wireless sensor networks are target tracking, environmental monitoring, and surveillance [1].

In order to support application requirements, sensor nodes must be deployed properly to reach an adequate coverage level to sense the phenomena or events of interest [12], [10]. For example, in target tracking [26], there should be enough sensor nodes deployed along the track of the target. In many cases, precise or manual sensor deployment is not possible, especially in some hostile environments such as disaster areas. Further, sensor nodes are subject to power depletion and failures, and may be affected or destroyed by external forces (e.g., wind, fire), creating coverage holes that are not covered by any sensor node [20], [25]. As the network condition or application requirements change, sensor nodes may need to be redeployed to recover failures or respond to occurring events. Since redeploying extra sensors may not be possible in many cases, there is a need to make use of mobile sensors, which can be relocated to achieve the required coverage level or response to new events [22].

Early works on mobile sensor networks [9], [27], [23], [24] focus on designing algorithms to deploy these mobile sensors, and there is not much work on implementation and evaluation due to the difficulty of building mobile sensors. In the literature, some existing mobile sensor prototypes are the Robomote [18] and the mobile robot in [14]. However, in these mobile sensors, the sensor nodes and the robots are integrated, and the sensor nodes are custom designed. As a result, existing popular applications developed for popular sensor platforms such as Mica2/TinyOS [4], [6] cannot be used. To use their mobile sensor, the applications have to be rewritten in their special language.

In this paper, we present our mobile sensor design which is based on commercial off-the-shelf (COTS) components. Our mobile sensor is based on the popular sensor node platform Mica2 [4] and mobile robot is built with COTS. The Mica2 sensor node will run TinyOS, and some instructions are provided to control the mobile robot. As a result, existing sensor network applications can be run on our mobile sensors without any modification. By using the added instructions, the sensor node can control the movement of the robot.

We use a sensor relocation application [22] to demonstrate the feasibility of our design. In the sensor relocation application, after a sensor node failure creates a coverage hole, a mobile sensor node is relocated to cover the hole in a timely and energy-efficient way. The sensor relocation algorithm was proposed in our early work [22]. However, it is still a challenge to design and implement a distributed sensor relocation algorithm in the resource constrained sensor node. In this paper, we will design a distributed sensor relocation algorithm and provide novel solutions to implement this algorithm in our mobile sensor platform. Experimental results show that our relocation algorithm can reduce the sensor relocation time and balance the energy consumption of the mobile nodes.

The rest of the paper is organized as follows. In Section II, we first outlines the requirements for a typical sensor relocation application. Section III discusses some design issues in the relocation algorithm. In Section IV, the hardware and software design, and the implementation of our testbed are described. The results of experimental evaluations are presented in Section V. Section VI discusses related work and Section VII concludes the paper.

II. Application Requirements

Our system design is motivated by the requirements of a typical sensor relocation application. The general objective of such an application is to relocate a mobile sensor to recover a sensor failure or fill a coverage hole as soon as possible, while generating minimum effect on other applications in the network and the sensing topology. Also, the failure recovery should be accomplished within a time constraint to reduce the interruption on the

application. In summary, several application requirements must be satisfied to make this system useful in practice:

- **Timely response:** The relocation latency determines the effectiveness of a sensor relocation scheme. In order to successfully complete a task or minimize the adverse effects, the sensor failure should be recovered within a tolerable time constraint.
- **Energy efficiency:** It is crucial to prolong the lifetime of both the sensor nodes and the network. If the mobile sensor node travels a long distance to replace the failed node, it may run out of power and create a new coverage hole. Therefore, a good scheme should be energy-aware.
- **Dynamic reconfiguration:** The system should be dynamically adjustable to deal with different requirements. Since recollecting the deployed nodes for reprogramming usually takes long time and may not be possible sometimes, it is necessary to enable dynamic reconfiguration in the system. For example, critical tasks usually require a short recovery latency while the latency can be relaxed when the task is not urgent.

III. The Sensor Relocation Design

To achieve the design goals illustrated in the last section, a mobile sensor node should be found and relocated to the destination in a timely and energy-efficient way. Using flooding to find the redundant sensors may cause significant message overhead. In [22], we proposed a quorum based approach where the sensor field is divided into cells, and each cell has a cell head. The cells with redundant nodes advertise to other cells in a row. The cells that need redundant sensors send queries to cells in each column. Since there is always an intersection cell between each row and each column, the intersection cell head will be able to serve the query.

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. 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. Next, we first describe the cascaded movement solution proposed in [22], and then describe how to implement this solution in a distributed way.

A. Sensor Relocation based on Cascaded Movement

The idea of *Cascaded Movement* [22] can be explained by Fig. 1. As shown in the figure, sensor S_0 fails and S_3 is the redundant sensor. If S_3 moves directly to the destination, its power may run out due to the long distance movement. If S_3 moves to $S2$, S_2 moves to S_1 and S_1 moves to S_0, the power consumption of these mobile nodes can be balanced. However, the delay is still high. In cascaded movement, messages are first exchanged among S_3, S_2, and S_1. Then three nodes move simultaneously; i.e., S_3 moves to replace S_2, S_2 moves to replace S_1, and S_1 moves to the destination. In this way, delay can be significantly reduced. Next, we present the algorithm which is used to select the cascaded schedule.

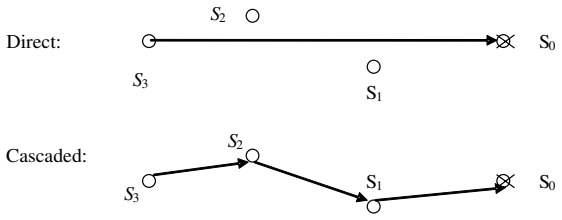


Fig. 1. Cascaded Movement

In the cascaded movement algorithm [22], we assume that the radio links are typically symmetric, and each sensor node knows its location. The location information can be obtained by using GPS [2], triangulation [17] or other means. The sensor network is modeled as a complete weighted graph $G(V, E)$, where a vertex corresponds to a mobile sensor node. The edge weight is the distance between two nodes. The following notations are used for describing the algorithm.

Notations

- S_i: node i's ID. The target position is represented by S_0, and the redundant node is denoted as S_r.
- t_i: the departure time of S_i's movement
- T_i: S_i's recovery delay constraint
- P_i: the remaining energy of S_i before movement
- E_i: the total energy consumption of the cascading schedule S_i calculates
- E_{min}: the minimum of the remaining energy after movement
- d_{ij}: distance between S_i and S_j
- v_i: node S_i's moving speed

Suppose S_j moves to replace S_i. S_j is called S_i's *successor*, and S_i is called S_j's *predecessor*. In order not to interrupt the supported application, each node S_i is associated with a recovery delay constraint T_i, within which S_i's successor must take its place after its movement. T_i is determined by the application based on S_i's sensing task, the size of the coverage hole generated by S_i's movement, and other factors. S_i's departure time t_i is normalized to be the time period after the relocation request is sent and t_0 is set to 0. Due to the recovery delay restriction, an inequality $d_{ji}/v_j - (t_i - t_j) \leq T_i$ must be satisfied if S_j is S_i's successor. t_j is usually set to $T_i + t_i - d_{ji}/v_j$ (the upper limit) such that more nodes can be the candidates to choose for S_i's successor.

A *cascaded movement schedule* is a set of cascading nodes and their departure time in a relocation. For energy-efficiency, the 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. Based on its simulation results, [22] proposes that the best schedule is *the one with the minimum difference between the total energy consumption and the minimum remaining power*.

To find the best schedule, [22] proposes a centralized modified Dijkstra's algorithm to calculate the shortest cascaded movement schedule, which is the schedule with the least total energy consumption. In this algorithm, an edge that does not satisfy the recovery delay constraint is not selected in any cascading path. To find the best schedule, [22] first calculates the shortest cascaded movement schedule and records its total en-

ergy consumption E and its minimum remaining energy E_{min}. Then, all the edges S_iS_j are deleted if $P_i - d_{ij} \leq E_{min}$ and $d_{i0} \geq d_{j0}$. Therefore a new graph is generated. This process continues and a new shortest 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 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.

B. Distributed Cascaded Sensor Relocation Algorithm

The aforementioned modified Dijkstra's algorithm is centralized, and hence has disadvantages such as single point of failure and not scalable. In the following, we present a distributed, dynamic programming-based approach and discuss how to implement it in resource constrained sensor nodes.

- We observe that the shortest cascading schedule should not include a node who is farther away from S_0 than S_r. Based on the direct distance to S_0 (the length of the edge connecting them), nodes that are closer to S_0 than S_r together with S_r are sorted into a sequence denoted as below:

$$N_0, N_1, \cdots, N_n.$$

Here, N_0 refers to S_0, N_n refers to S_r, and the total number of nodes is $n+1$. If two nodes are at the same distance to S_0, the node with a smaller node ID is numbered first.

- Let $E_m(i,l)$ be the minimum total energy cost of a cascading path from N_i to N_0 that has l intermediate nodes from $N_{i-1}, N_{i-2}, \cdots, N_1$ ($l \leq i-1$). The direct distance from N_i to N_j is denoted as $D(i,j)$. Then the shortest cascading path on the current graph, denoted as $E_m(n, n-1)$, can be computed as follows:

$$E_m(i,l) = \begin{cases} D(i,0) & l = 0, \\ min\{\ D(i,j) + E_m(j,l-1)\ \mid \\ D(i,j)/N_i's\ speed \leq T_j + t_j \\ \&\&\ i > j \geq 1\} & otherwise. \end{cases} \quad (1)$$

Here, $1 \leq i \leq n$ and $0 \leq l \leq n-1$.

Fig. 2 shows an example in which a cascading path consists of S_0, S_1, S_2, S_3 and S_r.

This distributed solution is broadcast-based and needs multiple iterations. In each iteration, S_0 first initiates a schedule computation by broadcasting a request message. A node S_j receiving the request first determines if it can become the successor of the sender S_i based on the following two conditions: (1) it can take S_i's place within T_i; (2) its remaining energy after moving is no larger than the minimum remaining energy in the last schedule. If both conditions are satisfied, it rebroadcasts the request with recalculated current total energy consumption E_j and the minimum remaining energy E_{min}, and remembers its predecessor S_i. If several such messages are received, the one that can minimize the total energy cost is chosen. This iteration terminates when the request arrives at the redundant sensor S_r. The process continues until the best schedule is found.

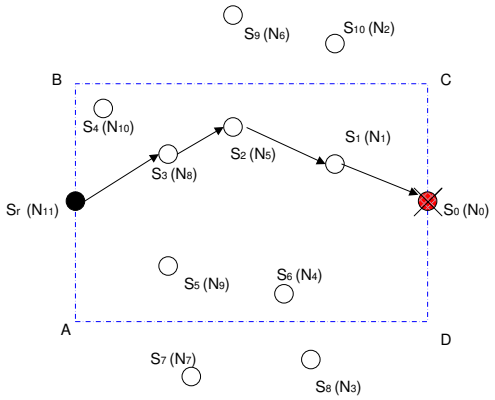


Fig. 2. Primary Search Area

To make the above method work, one issue has to be addressed: how can a node determine that it has received the message that can minimize the total energy consumption before generating and broadcasting its own message. There are two simple solutions. In the first one, if a node finds the newly received message that can reduce E, it immediately broadcasts an updated version. This method may cause a high message overhead since each node may broadcast several times. In the second method, each node waits for a period of time before broadcasting the message. However, it is hard to decide the time threshold: not enough information is received when it is low, while the overall delay may be very long when it is high.

In order to avoid high message overhead and long delay, the geographic information can be utilized to ensure that nodes can make correct decisions before broadcasting with a high probability. The solution needs two data structures: the *primary search area* and the *waiting list*.

B.1 Primary Search Area and Waiting List

The *primary search area* is determined based on the location of the target destination S_0 and the redundant sensor S_r. It should encompass all potential cascading nodes with a high probability, and only nodes in the primary search area are considered as candidates for cascading nodes. 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 S_r than S_i. A node only broadcasts the message after receiving the messages from all nodes in its waiting list.

The primary search area can be in any shape. In our implementation, we use rectangular primary search area. As shown in Fig. 2, the redundant sensor and the event location are the centers of parallel sides AB and CD of the rectangular primary search area. To encompass all the potential cascading nodes, we can simply set AB and CD to the width/length of the whole network. However, this will incur high message overhead because many nodes are involved and the schedule calculation time may be long.

On the other hand, if the area is set too small, many candidate cascading nodes may be excluded and thus we cannot get a good relocation schedule. Therefore, to find an adequate size for the primary search area is very important, which may be a dominating factor affecting the relocation scheduling calculation time.

We propose to choose the size based on the relocation time requirement. The size can be set large if the time requirement is not tight; otherwise we start with a small value. If no qualified schedule is found within this small search area, we can increase the size and initiate another relocation schedule finding process.

B.2 Message Piggybacking and Processing

Due to the limitation of the communication range, the potential successor of a node may not be its communication neighbor. Therefore, a node piggybacks information which may be needed by nodes further away. To do this, each node uses a *waiting list message queue* to cache its received messages from nodes in its wait list. One such message should at least include the following information about the message originator node: S_{org}, T_{org}, t_{org}, E_{org} and E_{min}. That is, each message needs at least 10 bytes if each field is represented by two bytes. However, the memory of a sensor node is limited (e.g., the Mica2 mote only has a 4 KB data memory [4]). Therefore, it is not feasible to cache all the received messages when the number of messages is large. On the other hand, it may not broadcast those messages immediately after receiving them because that will cause high message complexity and then more collisions. Also, the processing cost will be increased since each message may need to be processed individually.

To be more effective and efficient, in our implementation, the received messages are processed in a batch when the queue reaches half full or when messages from all nodes in the waiting list are received. The processed messages are then broadcast in a batch. After all the requested messages are received and processed, the node can update its E, E_{min} and t, and generate its own message to broadcast.

B.3 Message Broadcast

When nodes are close to each other, some broadcasts may not be necessary. For example, in Fig. 2, S_1 is in the waiting lists of S_2 and S_3. S_2 and S_3 both will rebroadcast the message originated from S_1, and therefore S_4 may hear two copies of the same messages. A node can simply drop the redundant copy, however, receiving a message consumes energy. In addition, sending more messages may increase the collision rate.

Unreliable message delivery is another issue we have to deal with. Currently, the default TinyOS [6] implementation uses a unreliable CSMA-like media access control protocol with a random backoff. In our algorithm, a node should receive messages from all nodes in its waiting list. Although we can add an acknowledgment scheme to achieve reliable delivery, the message overhead will be significantly increased if ACK is used for each message.

In our system, one message may be broadcast multiple times. In order to reduce the unnecessary rebroadcasts and the message redundancy, we adopt the distance-based scheme which is used to address the broadcast storm problem [13]. However, there is some difference between our solution and that in [13] since a node drops the messages which contain no information about the nodes in its waiting list in our scheme.

When a node S_i hears a broadcast message from node S_j, it calculates the distance d_{ij}. Since each node may receive a message for multiple times, it records the distance (d_{min}) to the nearest node from which the same message is heard. As in Fig. 2, if S_4 hears two copies of the same message from S_2 and S_3, its recorded d_{min} is $min\{d_{24}, d_{34}\}$. A node compares d_{min} with a predefined distance threshold D to decide whether to rebroadcast the message. It rebroadcasts the message if $d_{min} > D$; otherwise, it will not. To ensure receiving all the requested messages, each node has a message waiting timer. When the timer expires, it sends inquiries to those nodes whose messages it is waiting for.

IV. Prototype and Implementation

Fig. 3 shows the testbed of our mobile sensor network, which consists of 4 mobile sensor nodes and 11 static sensor nodes (Mica2 motes [4]). The static sensor nodes can be used to relay data between mobile nodes when they are far away from each other. Although the wireless radio is sufficient for the motes to communicate with each other in a small indoor field, it is not enough for long-range (> 500 ft) communication in an outdoor environment. A laptop with an attached mote can communicate with the sensor nodes through wireless communication, which mainly aims at debugging/visualizing. In our testbed, the static nodes also intentionally relay the heard messages to the laptop to aid debugging/visualizing. The right side of Fig. 3 shows an abstract view of the mobile sensor network on the laptop.

As shown in Fig. 4, the system is organized into a layered architecture. The mote controls the robot via serial commands through its UART interface. Next, we present the hardware and software design, and the details of this prototype.

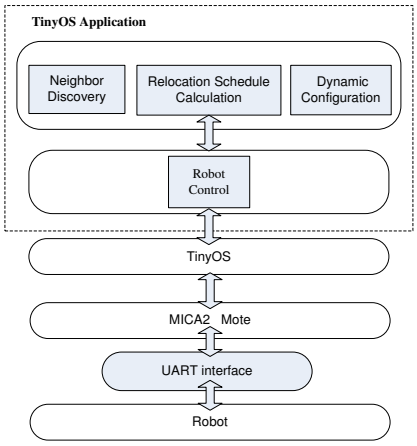


Fig. 4. Overview of mobile sensor relocation system

A. The Hardware

In this part, we briefly describe the Mica2 mote, and the hardware architecture of the mobile node.

Mica2 is the third generation mote built for wireless sensor networks [4]. It is equipped with a 4Mhz Atmel microprocessor with 4 KB of RAM and 128 KB of code space, a 868/916 MHz RFM radio, and 512KB of flash EEPROM. The outdoor radio range is 500 ft. A 51-pin connector accommodates a wide vari-

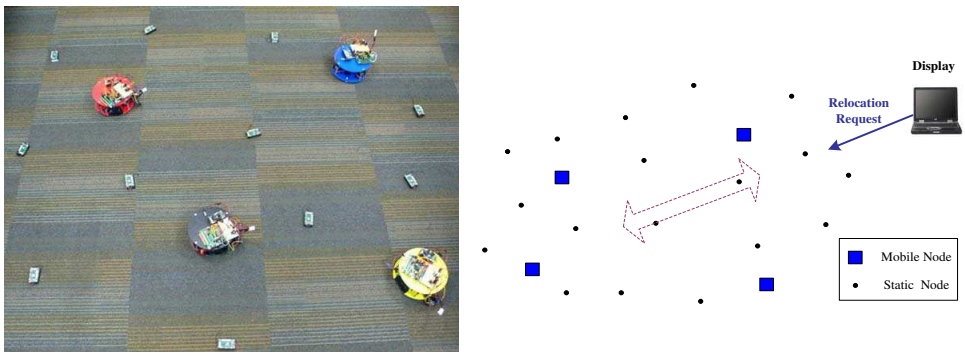


Fig. 3. The prototype which consists of mobile sensors and static sensors

ety of external peripherals by exposing a number of input lines as well as popular serial interface.

Each mobile node comprises a Mica2 mote and a robot platform, as shown in Fig. 5. All computation related to applications is done by the motes. The communication between different nodes is via the RFM radio of motes, which uses a CSMA-like media access control protocol with random backoff [6], [4]. The robot platform has two 6" plastic bases. The lower base consists of the motor, odometry encoders, wheels, line detectors, and batteries. On the upper base, the Mica2 mote is wired to a MC9S12DP256 microcontroller [19], which features a 16-bit HCS12 CPU with a 256 KB Flash EEPROM, 4 KB EEPROM and 12 KB RAM. Via its UART interface, the mote sends serial commands to control the robot. Any feedback from the robot goes through the UART interface to reach the mote. This mote-robot API will be discussed in Section IV-B.

$3V$ working voltage for the mote is supplied via a free-hanging $9V$ battery pack which also provides power for the robot. The effective lifetime of a sensor node is determined by the power supply. That is, the power consumption of each node tends to be dominated by the cost of transmitting and receiving messages when it is still, and by the cost of moving when it is moving.

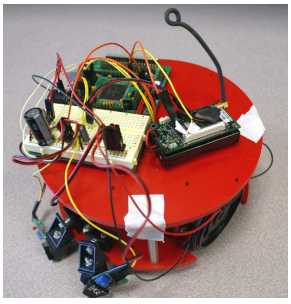


Fig. 5. Mobile Node

B. The Software

The software architecture has four major components: 1) the low-level program in the microcontroller on the robot; 2) the robot control program on the mote; 3) the relocation application program on the mote; and 4) the program running on the base station (laptop) for debugging/visualization. In the following, we will introduce them in detail.

B.1 The robot program

The robot program consists of a main loop which deals with sending and receiving messages, handling the navigation queue, and calling for odometry updates. Hardware-specific interface is handled through memory-mapped I/O and interrupts. Current code size is nearly 18 KB, which is too large for the 12 KB RAM or 4 KB EEPROM and can only be programmed into the flash memory.

B.2 The robot control program

The robot control program is an added TinyOS component, written in nesC [5], which sends and receives messages to/from the robot via the UART interface. TinyOS is an event-driven operating system specifically designed for the mote platform. TinyOS provides a set of essential components such as scheduler and communication protocols, which provide low-level support for application modules. Packets in the current generation of TinyOS are in a fixed size, by default, 36 bytes with a 29-byte payload.

nesC [5], [21] is a C-like language that enables users to define components and the relations among them. A nesC application consists of one or more components linked together to form an executable image [21]. A component provides and uses *interfaces*, which are the only point of access to the component and are bi-directional. There are two types of components in nesC: modules and configurations. Modules provide application code, implementing one or more interfaces. Configurations are used to assemble other components together, connecting interfaces used by components to interfaces provided by others [21].

The relocation application uses this robot control component to make the robot move in a controlled manner. Currently 21 serial commands are available to be sent from the mote to the microcontroller on the robot such as "turn a angle" and "go to a point". There are 5 feedback instructions that the robot can provide to the mote. With these serial commends and feedback instructions, the mote can control the motion of the robot, check its status and retrieve parametric values. Upon receiving a message, an user-defined event will be signaled and proper processing will then be performed on the mote.

B.3 The sensor relocation algorithm

The primary goal of our work is to relocate a mobile sensor node to a target destination in a timely and energy-efficient way. In the last section, we presented the relocation algorithm. To implement this algorithm, we need two other functions: neighbor discovery, network reconfiguration. Next, we present the details of these three modules.

Neighbor Discovery: After deployment, the nodes discover and notify its neighbors periodically by locally broadcasting beacon messages. In the beacon message, a sender sends its ID, its location and its neighbor list. This local information helps each node to build a neighborhood table and acquire certain knowledge of the network topology, which helps select the waiting list.

Network Reconfiguration: In this phase, we can configure the network by dynamic reconfiguration, which is conducted by adjusting the values of some control parameters. This capability facilitates re-tasking for application requirement change, and it simplifies system tuning and debugging. For example, it usually takes at least 30 seconds to recollect and reprogram one mote manually. With dynamic reconfiguration, the time can be reduced to several seconds.

Dynamic reconfiguration is supported with the help of beacon messages, which piggyback the new parametric values. Upon receiving the messages, the nodes adopt the new values. Such a piggybacking strategy obviates the need of another specialized type of messages to re-parameterize the nodes and thus saves energy. However, during the relocation schedule calculating process, the reconfiguration is forbidden. An example of the reconfigurable parameters is the recovery delay constraint. Changing a node' recovery delay constraint will impose a direct impact on the relocation schedule calculation for a node failure, because it determines whether a path can be chosen according to the algorithm stated in Section III-B.

Sensor relocation: The sensor relocation application architecture described in Fig. 4 is implemented on top of TinyOS. Fig. 6 shows the component architecture of the relocation application in nesC. The whole program occupies 21450-byte code space and 2104-byte data memory. RobotCommM is responsible for robot motion control. MoveM is the main module, which uses the RobotCommM and some TinyOS-provided components to perform Neighbor Discovery, Dynamic Reconfiguration and Relocation Schedule Calculation.

There may be race conditions between different software modules if they share some resources or try to transmit packets simultaneously. Although application-level methods such as synchronization or packet scheduling can be used to avoid this, in our implementation, we use *tasks* and *atomic statements* [21] provided by TinyOS to do concurrency control. *Tasks* are used to perform general-purpose background processing, which are put into a task queue to execute one by one. The *atomic statements* that are braced by the keyword *atomic* will be executed without preemption. We believe that our solution is more convenient and efficient than the application-level methods. The implementation of our sensor relocation application on the Mica2 motes has the following advantages:

- **Energy Efficiency**: By balancing the total energy cost and the energy cost of individual node, we avoid depleting a single node in a short time and prolong the network lifetime.
- **Simplicity**: Because of the hardware limitation (4K data memory and 128K code memory), the application must be small, simple and effective. In our implementation, the code size is only 21450 bytes and the data memory is only 2104 bytes.
- **Flexibility**: Dynamic reconfiguration is achieved by adjusting values of some parameters (e.g., recovery time constraint), which provides fast performance tuning and debugging.
- **Contention Reduction**: A distance-based method is used to reduce the unnecessary rebroadcasts, and hence reduces the contention for media access between different nodes. At the node level, we use concurrency control methods provided by TinyOS to avoid the possible race conditions.

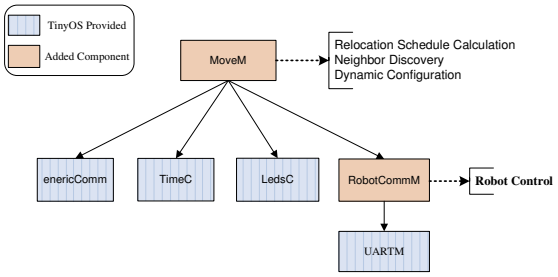


Fig. 6. The structure of the sensor relocation application in nesC

B.4 Programs on the base station

In our testbed (as shown in Fig. 3), we use a laptop connected with a mote as the base station for debugging/visualizing. The mote attached to the laptop passively listens to messages being transmitted between mobile nodes or relayed from the static nodes, then delivers them to the laptop; the laptop program processes these messages and displays the results to the user. Since no node in our testbed is equipped with any localization devices, this program is also responsible for assigning the initial positions to the mobile nodes. During movement, the mobile nodes can reposition themselves autonomously by the odometry update function of the robot program. Additionally, the base station mimics the "failed" node which initiates a relocation schedule calculation process by injecting a request to the network.

V. Experimental Evaluations

We setup a testbed to evaluate the performance of our distributed sensor relocation scheme. In the testbed, 4 mobile sensor nodes are randomly deployed in a $50ft \times 35ft$ flat floor in a research laboratory. A node is randomly chosen as the redundant node. A laptop connected with a mote (the base station) initiates a node relocation process by injecting a request to the network. All four mobile nodes are in the primary search area, i.e., they are involved in the optimal cascaded schedule which may need several rounds. Each round returns a schedule, based on which new requests are generated to search for better schedules until the best one is found.

We run multiple experiments to measure the performance of

the algorithm in terms of the *moving distance*, *energy consumption*, *recovery time* and *message complexity*. For each experiment, a random topology is generated, and the mobile nodes are positioned based on this topology. We compare two schemes: *Cascaded movement* and *Direct movement*. For better observation of the effectiveness of cascaded movement, we relaxed the recovery delay constraint.

Fig. 7 compares the moving distance of the cascaded movement approach and the direct movement approach. Besides the total moving distance, we also use the average moving distance, the minimum moving distance, and the maximum moving distance among the nodes participating in cascaded movement to measure the performance of the cascaded movement approach. As shown in the figure, although the total moving distance of the cascaded movement approach is slightly longer than the direct movement approach, it has a much smaller average moving distance, and hence its mobile nodes can balance their power consumption. Also, the maximum moving distance in cascaded movement is much lower than the moving distance of direct movement, and hence cascaded movement can have a much shorter relocation time.

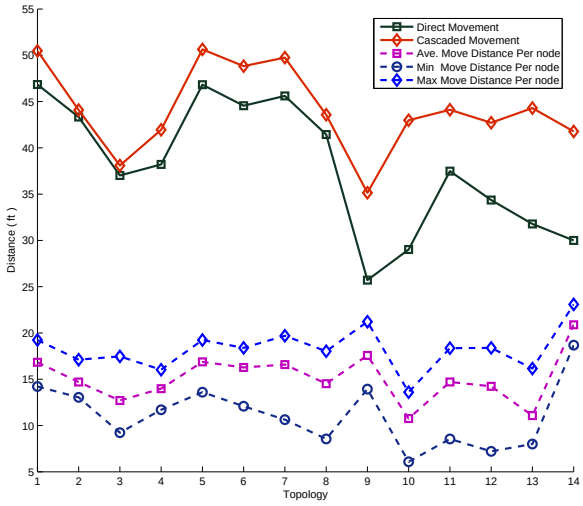


Fig. 7. Comparison of the moving distance in Cascaded Movement and Direct Movement

Fig. 8 compares the energy cost between cascaded movement and direct movement. Here, energy is represented by the distance that the sensor can move with this energy, i.e. one energy unit means that the node can travel one ft with this energy. For easy comparison, the initial energy of each node is set to 100 units in each experiment. As shown in the figure, the total energy cost of cascaded movement is slightly higher than the direct movement. However, the minimum remaining energy in cascaded movement is much higher than that of direct movement, and hence, the cascaded movement approach can balance the energy cost and increase the network lifetime.

For cascaded movement, the total time for relocation includes the relocation schedule calculation time and the physical moving time which is equal to the maximum moving time of the

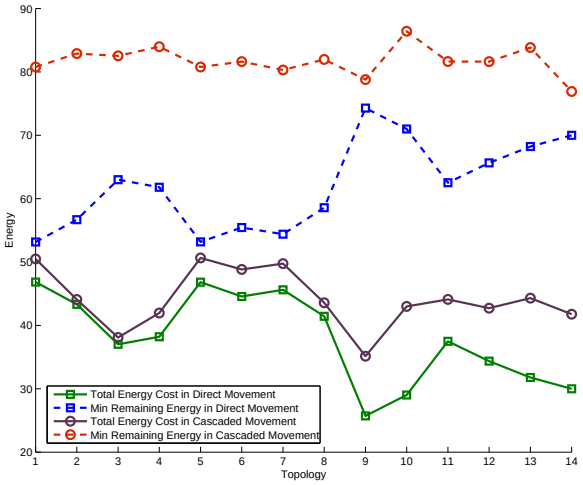


Fig. 8. Comparison of energy cost in Cascaded Movement and Direct Movement

cascading nodes if they start moving at the same time. For direct movement, the total time is the physical moving time since the message routing time is pretty short. As shown in Fig. 9, the cascaded movement approach takes much shorter time to do the relocation compared to the direct movement approach. Fig. 10 (a) shows the schedule calculation time and the average time needed for one round in the relocation schedule calculation. Fig. 10 (b) shows the percentage of schedule calculation time and the moving time in cascaded movement. As can be seen, the calculation time is pretty short compared to the physical moving time.

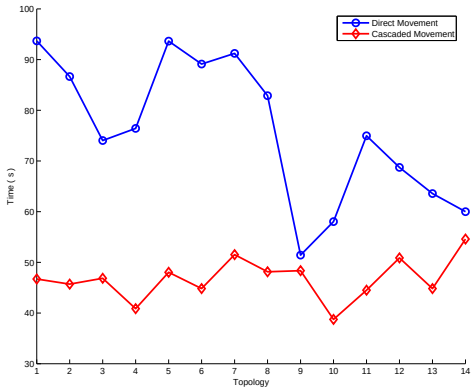
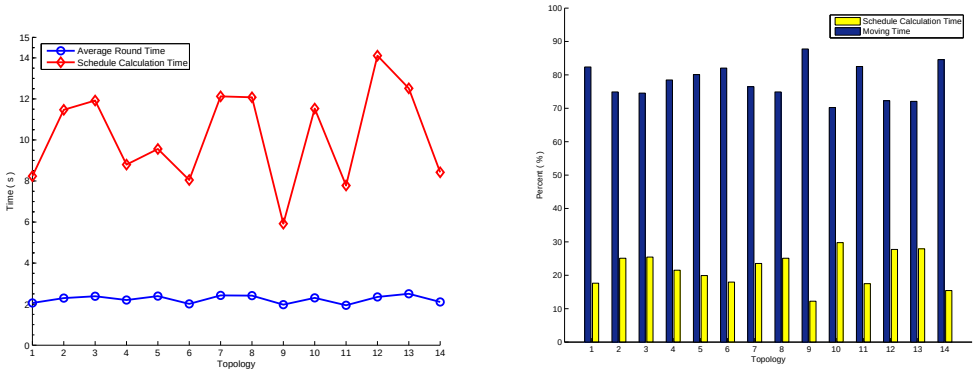


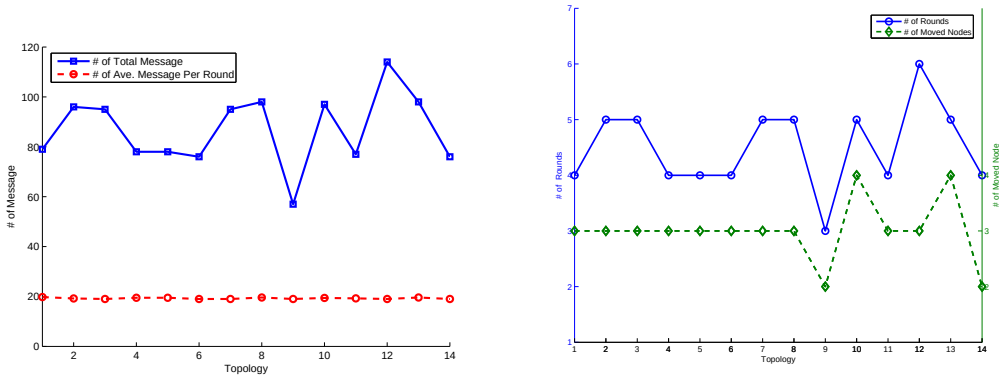
Fig. 9. Comparison of total time cost in cascaded movement and direct movement

Fig. 11 shows the message complexity of the cascaded movement approach. As shown in Fig. 11(a), different topologies may require 3 to 6 rounds to find the best relocation schedule, and each round needs approximately 20 messages for all topologies. From Fig. 11 (b), we can see it generally needs more rounds to calculate the relocation schedule if more nodes are involved in the cascading path.



(a) (b)

Fig. 10. Relocation time in cascaded movement



(a) (b)

Fig. 11. The message complexity in cascaded movement

VI. Related Work

Many research efforts have been made on mobile sensor network. Howard *et al.* [9] proposed algorithms to incrementally deploy sensor nodes one-at-a-time into an unknown environment using information gathered by previously deployed nodes. Based on the notion of potential fields, deployment strategies for mobile sensors are presented in [7], [16], [15] to maximize the coverage with certain constraints. Zou and Chakrabarty [27] utilized the virtual force to enhance the coverage with a given number of sensor nodes after an initial random placement. Three mobility-assisted sensor deployment protocols [24], called VOR, VEC and Minimax, were proposed to increase the coverage by making mobile sensor nodes move from densely deployed areas to sparse areas. To balance sensor cost and sensor coverage, a bidding protocol was presented in [23] for mobile sensor deployment in sensor networks consisting of mobile and static sensor nodes. However, all these approaches did not consider the response time requirements in the sensor relocation problem. Mesh-based relocation algorithms in [11] can guarantee the latency, but some nodes may be penalized when moving too often. In [22], Wang *et al.* presented a sensor relocation algorithm based on cascaded movement, however, many real implementation issues are not addressed.

In order to demonstrate mobile sensors really work, some researchers started to develop prototypes. For example, Robomote [18] can provide a sequence of fundamental functionality such as sensing and communication. Howard et al. [8] used a team of mobile robots coupled with acoustic sensors and 802.11b WiFi to form a sensor network in a spacial indoor environment. Hence, a heterogeneous large-scale sensor network, which consists of cheap unattended ground sensors and relatively expensive mobile sensor nodes, can be envisioned to solve complex tasks [3].

VII. Conclusions

In this paper, we presented our mobile sensor design where the mobile sensor node is based on the popular sensor node platform Mica2 mote [4] and mobile robots are built with commercial off-the-shelf components. We used a sensor relocation application to demonstrate the feasibility of the design. Although the sensor relocation algorithm is based on our early work, we addressed many issues when implementing this algorithm distributedly in the resource constrained sensor motes. Experimental results show that our relocation algorithm can reduce the sen-

sor relocation time. Although the total moving distance may increase a little bit, each mobile node moves much less to balance the energy consumption and hence increase the network lifetime.

ACKNOWLEDGMENT

This work was supported in part by Army Research Office (W911NF-05-1-0270) and the National Science Foundation (CNS-0092770, CNS-0519460, CNS-0721479).

REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, March 2002.

[2] Nirupama Bulusu, John Heidemann, and Deborah Estrin, "GPS-less Low Cost Outdoor Localization for Very Small Devices," in *IEEE Personal Communications Magazine*, Oct 2000, vol. 7(5), pp. 28–34.

[3] T. Christensen, M. Noergaard, C. Madsen, and A. Hoover, "Sensor Networked Mobile Robotics," in *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, June 2000, pp. 82–783.

[4] MICA2 Datasheet, "http://www.xbow.com/products/productdetails .aspx?sid=174," 2006.

[5] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in *Proceedings of Programming Language Design and Implementation(PLDI)*, Feb 2003.

[6] J. Hilland, R. Szewczykand, A. Wooand, S. Hollarand, D.E. Cullerand, and K.S.J. Pister, "System Architecture Directions for Networked Sensors," in *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000, pp. 93–104.

[7] 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," in *the 6th International Symposium on Distributed Autonomous Robotics Systems*, June 2002.

[8] A. Howard, L. E. Parker, and G. Sukhatme, "Experiments with a large Heterogeneous Mobile Robot Team: Exploration, Mapping, Deployment, and Detection," *International Journal of Robotics Research*, 2006.

[9] 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.

[10] Santosh Kumar, Ten H. Lai, and Anish Arora, "Barrier Coverage with Wireless Sensors," in *Mobicom*, 2005.

[11] X. Li, N. Santoro, and I. Stojmenovic, "Mesh-based Sensor Relocation for Coverage Maintenance in Mobile Sensor Networks," in *Proc. of the 4th International Conference on Ubiquitous Intelligence and Computing (UIC)*, 2007.

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

[13] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," in *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom)*, 1999, pp. 152–162.

[14] L. E. Parker, B. Kannan, X. Fu, and Y. Tang, "Heterogeneous Mobile Sensor Net Deployment Using Robot Herding and Line-of-Sight Formations," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2003, pp. 152–162.

[15] Sameera Poduri and Gaurav S. Sukhatme, "Constrained Coverage for Mobile Sensor Networks," in *IEEE International Conference on Robotics and Automation*, Apr 2004.

[16] D. Popa, C. Helm, H. Stephanou, and A. Sanderson, "Robotic Deployment of Sensor Networks Using Potential Fields," in *Proceedings of the International Conference on Robotics and Automation*, Apr 2004.

[17] Andreas Savvides, Chih-Chieh Han, and Mani B.Strivastava, "Dynamic Fine-Grained Localization in Adhoc Networks of Sensors," in *Mobicom*, 2001.

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

[19] MC9S12DP256 Microcontroller Produce Summary, "http://www.freescale.com/webapp/sps/site/prod_summary.jsp ?code=mc9s12dp256&nodeid=01624686363176," 2005.

[20] D. Tian and N. Georganas, "A Coverage-preserving Node Scheduling Scheme for Large Wireless Sensor Networks," in *ACM international workshop on Wireless sensor networks and applications*, 2002.

[21] TinyOS Tutorial, "http://www.tinyos.net/tinyos-1.x/doc/tutorial," 2003.

[22] G. Wang, G. Cao, T. La Porta, and W. Zhang, "Sensor Relocation in Mobile Sensor Networks," in *IEEE INFOCOM*, March 2005.

[23] G. Wang, G. Cao, and T. La Porta, "A Bidding Protocol for Deploying Mobile Sensors," in *ICNP*, Nov 2003.

[24] G. Wang, G. Cao, and T. La Porta, "Movement-Assisted Sensor Deployment," in *IEEE Transactions on Mobile Computing*, Vol. 5, No. 6, pp. 640 - 652, June 2006.

[25] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks," in *ACM SenSys*, 2003.

[26] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communication*, vol. 3, no. 5, pp. 1689–1701, September 2004.

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