BIOINFO Sensor Networks
ISSN: 2249-944X & E-ISSN: 2249-9458 Volume 2, Issue 1, 2012, pp.-10-15.
Available online at http://www.bioinfo.in/contents.php?id=301

BUILDING A SENSOR NETWORK OF MOBILE PHONES AND SENSOR RELOCATION

DESHMUKH P.R.[1] AND WASANKAR S.W.[2]

[1]Department of Computer Science and Engineering, Sipna Area, Amravti, MS, India.
[2]College of Engineering and Technology, Sipna Area, Amravati, MS, India.
*Corresponding Author: Email- pr_deshmukh@yahoo.com[1], snehal.2104@gmail.com[2]

Received: February 21, 2012; Accepted: March 15, 2012

Abstract- All Mobile phones have a microphone, and most have a camera. Other sensors can be connected to a phone using Bluetooth. Mobile phones are connected to a network. They have a power supply, based on human user initiated charging.Some Technique providing efficient methods for the sensor nodes to make their data available to the network, allowing the sensor network applications to access the data from potentially disconnected and highly mobile devices. We demonstrate an initial system prototype that addresses some of these concerns. Recently there has been a great deal of research on using mobility in sensor networks to assist in the initial deployment of nodes. Mobile sensors are useful in this environment because they can move to locations that meet sensing coverage requirements. This paper explores the motion capability to relocate sensors to deal with sensor failure or respond to new events. We define the problem of sensor relocation and propose a two-phase sensor relocation solution redundant sensors are first identified and then relocated to the target location. We propose a Grid-Quorum solution to quickly locate the closest redundant sensor with low message complexity, and propose to use cascaded movement to relocate the redundant sensor in a timely, efficient and balanced way.

Citation: Deshmukh P.R. and Wasankar S.W. (2012) Building A Sensor Network of Mobile Phones and Sensor Relocation. BIOINFO Sensor Networks, ISSN: 2249-944X & E-ISSN: 2249-9458 Volume 2, Issue 1, pp.-10-15.

Introduction

Such mobile devices as sensors has a significant advantage over unattended wireless sensor networks in that deploying the sensing hardware and providing it with network and power is already taken care of. Secondly, mobile phones can provide coverage where static sensors are hard to deploy and maintain. No single entity may have the access rights to place sensors across the complete coverage domain required by an application, such as a combination of subway stations, public parks, and shopping malls. Thirdly, each mobile device is associated with a human user, whose assistance can sometimes be used to enhance application functionality. For instance, a human user may help by pointing the camera appropriately at the target object to be sensed. Our goal is to enhance the utility of the existing swarm of mobile devices by presenting it as a physical sensing substrate that sensor networking applications may program for their sensing objectives. Several applications can be enabled using sensor networks of mobile phones.

System Description

We build a sensor network of mobile phones that is used as a shared system, as opposed to a system where a single application owns and uses a dedicated set of mobile device carried by users or vehicles directly involved with that application. In the shared the phones are carried and used by their respective owners as they need. The sensor networking applications use the mobile devices when available. The key advantages of a shared system come from the vast coverage expanse achieved that a

single dedicated system may never be able to match, and the re-use of physical resources by multiple applications. The related trade-offs are difficulty in providing performance guarantees and limitations in control of the sensor nodes.

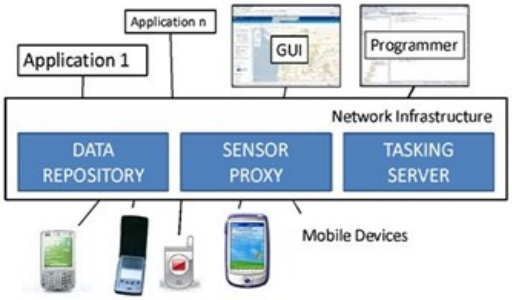

Fig. 1- A shared sensor network of mobile phones

The system (Figure 1) consists of the following key entities:

1. Sensors: mobile devices that sense the physical world
2. Network Infrastructure: In our system, this includes: a data repository that stores all the sensor data provided by the mobile devices, a tasking server that enables applications to program the sensor network as per their requirements, and a sensor proxy that allows disconnected operation for sensors while making their data available to applications.
3. Users: the applications that access our shared sensor network, or human users who access the sensor data through a graphical user interface We demonstrate the following aspects of this system.

Sensor Data Sharing

We have developed a data sharing framework that makes the sensor data from mobile devices available to network applications. The mobile devices may be only sporadically connected, or stay behind firewalls. The first is a *publishing client* on the mobile phone that collects samples from the sensors on the phone, applies predefined filters to the data, buffers the data locally, and uploads the data as per network availability using a web service interface. This client runs in parallel with other applications on the phone and can be activated or deactivated as necessary. The client allows the user to configure whether data may be automatically collected or only when the user initiates a sample capture. The client also includes functionality that allows remote applications to program the phones, as described later.

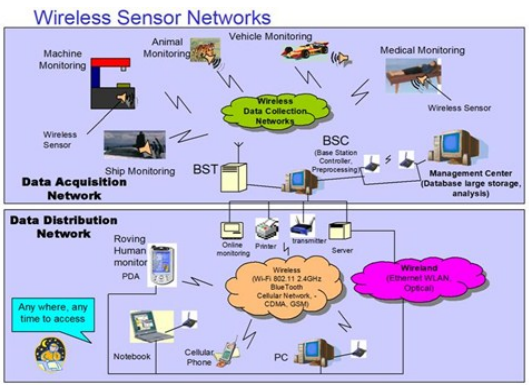

Fig. 2- Wireless Network

We have developed the client software GUI to use only the phone keypad and the small phone screen. The second component is the *data repository* which hosts the web service to accept data samples from the phones and uses a database to index the uploaded data. The third component is a *sensor proxy* that acts as an online representative for the mobile sensors. This proxy provides a fixed sensor URL that can be used by applications to access a specific sensor device directly.

Location Based Indexing

A salient feature of the mobile phone sensor network is that each sensor node is mobile and the motion pattern cannot be easily controlled. Most applications are unlikely to be interested in the data collected by a specific device but rather need data from a specific region and time window. The device identities of the devices that contributed that data may only be of secondary importance. Thus, it is very important to index the uploaded data by location and time. This also helps applications avoid any overheads in discovering specific mobile phones, tracking their motion, or waiting for the device to be connected to the network.

In our prototype we obtain location information using:

1. Cell-tower triangulation: The cell-phone network typically knows the location of a phone using signal strength measurements from one or more cellular base stations.
2. Phone GPS: Many recently released and forthcoming mobile phones have built in GPS receivers. The GPS based location information is accurate to several meters when the phone has good GPS satellite visibility, such as under open sky. We use GPS location also in our prototype.

Programing

Our system enables sensor network application developers to deploy their applications without ever deploying a physical sensor. Instead, the developers may program our shared mobile phone sensor network to carry out required sensing tasks. This is enabled using the *tasking server* that is part of the network infrastructure. We allow mobile phone users to share their sensing resources to varying degrees based on their resources and involvement in the sensor network application. We also allow the applications to program the network without knowing the identities of any specific devices in their region of interest or the device capabilities. The tasking server presents the sensor network as a single object to the applications. Applications can call this object's methods to carry our their required sensing tasks, specifying their spatiotemporal region of interest, measurement tolerances, and delay constraints.

Sensor Location

In addition, once deployed, sensor nodes may fail, requiring nodes to be moved to overcome the coverage hole created by the failed sensor. In these scenarios, it is necessary to make use of mobile sensors which can move to provide the required coverage. One example of a mobile sensor is the Robomot. These sensors are smaller than 0.000047$m3$ and cost less than 150 dollars. In this paper we address the problem of *sensor relocation*.

Related Work

There have been several research efforts on deploying mobile

sensors. For example, the work in assumes that a powerful cluster head is available to collect information and determine the target location of the mobile sensors. Sensor deployment has also been addressed in the field of robotics where sensors are deployed one by one, utilizing the Recently, we proposed three mobility-assisted sensor deployment protocols where mobile sensors move from densely deployed areas to sparse areas to increase the coverage. The protocols run iteratively In each round, sensors first detect coverage holes around them by utilizing the Voronoi diagram If coverage holes exist, sensors decide where to move to heal or reduce the holes by three different distributed algorithms called VOR, VEC and Minimax.

Problem Statement

In theory, the two protocols we previously proposed can be used for sensor relocation. For example, after a sensor failure, the sensors neighboring the failed node can execute the algorithms. After several rounds, the neighbor sensors will move to cover the area initially covered by the failed sensor. However, moving neighbor sensors may create new holes in that area. To heal these new holes, more sensors must move. In addition, since many sensors are involved, it may take a long time for the algorithm to terminate. Based on this observation, we propose to first find the locations of the redundant sensors, and then design an efficient route for them to move to the destination. To determine which sensor(s) is redundant is a challenging problem. It is hard for a single sensor to independently decide whether its movement will generate a coverage hole. problems: finding the redundant

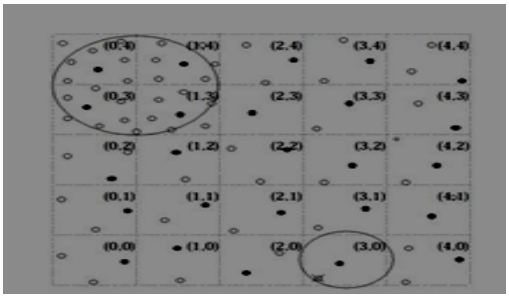

Fig. 3- System model (manually)

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

Finding The Redundant Sensor

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

Background and Motivation

The problem of finding redundant sensors has some similarity to the publish/subscribe problem where the publisher advertises some information and the subscriber requests the information. Mapping the terminology to our problem, the grids that need more sensors are the subscribers, and the grids that have redundant sensors are the publishers. In the publish/subscribe system, the matching of a request to an advertisement is called matchmaking. Generally, there are three types of solutions for matchmaking.

(1) Matchmaking occurs at the subscriber, which is referred as “broadcast advertisement” . In our problem, this is similar to letting the grids having redundant sensors flood this information. Later, when some grid needs redundant sensors,

it can get the information quickly.

(2) Matchmaking occurs

at the publisher, which is referred as “broadcast request”

In our problem, this is similar to letting the grids that need sensors flood the request. The grid that has redundant sensors replies after receiving the request. (3) Matchmaking happens in the middle of the network In our problem, this is similar to letting the supply grid advertise the information to some intermediate grids from which the demand grid obtains the information. Different from the traditional publish/subscribe problem, the information in our system is not reusable. The information about the redundant sensor can only be used once, since it may be changed after the redundant sensor moves to the requesting place. Due to this special property, the message complexity will be very high if we use the broadcast advertisement approach, which requires two network-wide broadcasts for each redundant sensor: one for advertisement and the other for data update after the redundant sensor moves column. For example, suppose grid (0,3) has redundant sensors, it only sends the advertisement to grids in a row ((0,3), (1,3), (2,3), (3,3), (4,3)) and a column ((0,4), (0,3), (0,2), (0,1), (0,0)).When grid (3,0) is looking for redundant sensors, it only needs to send a request to grids in a row ((0,0), (1,0), (2,0),(3,0), (4,0)) and a column ((3,4), (3,3), (3,2), (3,1), (3,0)). The intersection node (0,0) will be able to match the request to the advertisement. Suppose N is the number of grids in the network. By using this quorum based system, the message overhead can be reduced from $O(N)$ to $O(pN)$. Although the message overhead is very low compared to flooding, we can further reduce the message overhead by observing the specialty of our problem.

The Grid-Quorum Solution

In our Grid-Quorum system, we do not require the intersection of any two quorums. Instead, we deploy two coterie, called *supply coterie* and *demand coterie* separately, and only require that the quorum belong to the supply coterie intersects with all quorums in the demand coterie, and vice versa.To construct a Grid-Quorum, the grid heads belong to the grids in one row are organized into one quorum, called supply quorum and the grid heads belong to the grids in a column are organized into one quorum, called demand quorum. We can see that using the geographic information reduces the cost of building Grid-Quorum to almost zero. Still Grids (0,4), (1,4), (0,3) and (1,3) have redundant sensors, while grid (3,0) needs more sensors. The grid head of (1,3) propagates its redundant sensor information through its supply quorum ((1,4), (1,3), (1,2),(1,1), (1,0)). The grid head in grid (3,0) searches its

demand quorum ((0,0), (1,0), (2,0), (3,0), (4,0)). Grid (1,0) can reply the information about redundant sensors.

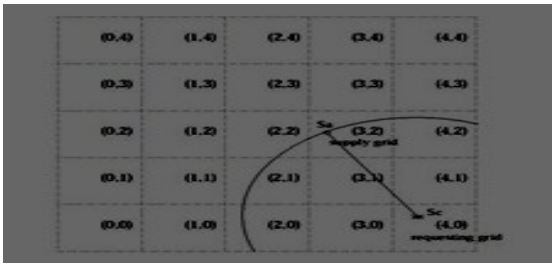

Fig 4- Stopping criteria

and radius of the distance between *sa* and *sc* must be farther to *sc* than *sa*. Cluster head of grid (3,0) attaches the location of *sa* in the requesting message before forwarding it. When the grid head in grid (2,0) receives the request message, it will not forward it further since no closer redundant sensor can be found.. compares the performance between the Grid-Quorum solution and the "Broadcast Request" method in a 10_10 grid. The results were obtained from a simple ns2 simulation; more details on the simulation environment are given later. Cluster heads of neighbor grids communicate through a gateway sensor. Suppose there is no other traffic in the network. From the figures, we can see that the Grid-Quorum solution can significantly reduce the message complexity compared to the "Broadcast request" approach.

Sensor Relocation

General Idea: Cascaded Movement

Having obtained the location of the redundant sensor, we need to determine how to move the sensor to the target location (destination). Moving it directly to the destination is a possible solution. If the redundant sensor is 100 meters away and it takes at least one minute for the sensor to reach its destination, the application requirement cannot be met. Moreover, moving a sensor for a long distance consumes too much energy. If the sensor dies shortly after it reaches the destination, this movement is wasted and another sensor has to be found and relocated. We propose to use a *cascaded movement* to address the problem. The idea is to find some cascading (intermediate) nodes, and use them for relocation to reduce the delay and balance the power. redundant sensor *s*3 move directly to the destination, *s*1 and *s*2 are chosen as cascading nodes. As a result, *s*3 moves to replace *s*2, *s*2 moves to replace *s*1, and *s*1 moves to the destination. Since the sensors can first exchange communication messages (i.e., logically move), and ask all relevant sensors

a. Message complexity
b. Response time

(physically) move at the same time, the relocation time is much shorter. A node *si* which moves to replace another node *sj*, is referred to as *sj* 's *successor*, and *sj* is referred to as *si*'s *predecessor*. In Fig. *s*3 is *s*2's successor and *s*2 is *s*3's predecessor. We also introduce a virtual node *s*0, which is used to represent the target location. It may represent the failed sensor or the location where an extra sensor is needed to increase the sensing accuracy. In Fig. we say *s*0 is *s*1's predecessor and *s*1 is *s*0's successor. Selecting cascading nodes is not easy since the sensor nodes may be used by some application and their movement may affect the sensing or communication tasks they are performing. To ensure that this effect is within application's requirement, each sensor *si* is associated with a *recovery delay Ti*. After *si*'s movement, its successor must take its place within *Ti*. *Ti* is determined by the application based on the critical level of *si*'s sensing task, the size of the coverage hole generated by *si*'s movement, and other application factors. We use *T*0 to represent the recovery delay of the relocation event. It can bethe maximum recovery delay of the failed sensor or the time limit for an additional sensor being placed at *s*0. The *T* value imposes restrictions on the spatial relationship and departure time of the cascading nodes. We use *ti* to denotethe departure time of *si* and *dji* to denote the distance between *si* and *sj* . The following Inequality must be satisfied if *sj* is *si*'s successor. dji/speed − (ti − tj) _ Ti (1) For simplicity, *ti* is normalized to be the time period after the relocation request is sent and *t*0 (for *s*0) is set to be 0. Based on Inequality (1), whether *sj* can be the the successor of *si* is not determined solely by its distance to *si*, but also *si*'s departure time. If *si* moves at *t*0 (0), *sj* must be within *speed* _ *Ti* from *si*; if *si* moves after another *t* minutes, *sj* can be farther away from *si* as long as *dji* _ *speed* _ (*Ti*+*t*). Whether *si* can stay at its place for this *t* minutes or must move immediately is determined by its own predecessor. For example, if *si* is the successor of *s*0, and *di*0 is shorter than *speed*_*T*0, *si* can flexibly move between (0*, T*0−*di*0*/speed*). In this case, we normally

The Metrics to Choose Cascading Nodes

The cascading schedule should minimize the total energy consumption and maximize the minimum remaining energy so that no individual sensor is penalized. However, in most cases, these two goals cannot be satisfied at the same time.

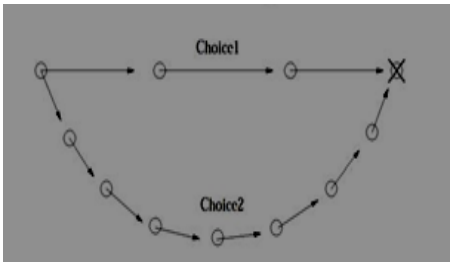

Fig. 5-

As shown in Fig.5 suppose all sensors have the same amount of power. Choice1 consumes less energy, but the involved sensors will have lower remaining energy. Sensors in Choice2 have higher remaining energy, but the total energy consumption of Choice 2 is higher than that in choice1. There is a tradeoff between minimizing the total energy consumption and maximizing the minimum remaining energy, and we want to find a balance between them. Before presenting our solution, we first show some observations.Based on the sensor deployment result generated by running VOR, we randomly choose some sensor and deplete its energy. Then, all cascading schedules to recoverthe failed sensor are enumerated and compared in terms of the total energy consumption and the minimum remaining energy. Here, the recovery delay (*Ti, i 6*= 0) is relaxed for better observation, but the relocation time (*T*0) is calculated for reference. The cascading schedules which are worse than some other schedule in both metrics

(total energy consumption and minimum remaining energy) will be ignored; that is, only keep the cascading schedules which perform better than others at least in terms of one metric. Fig. shows the total energy consumption and the minimum remaining energy of these schedules in an increasing order. As shown in the figure, the total energy consumption is almost flat at the beginning and then significantly increased, whereas the minimum remaining energy has a steep increase at the beginning and then becomes flat. This new metric can be explained in a mathematical way. Suppose there are two cascading schedules with *E*1 and *E*2 as their total energy consumption, and Emin1 and Emin2 as their minimum remaining energy. Schedule 1 is chosen since E1 −Emin1 _ E2 −Emin2. This inequality can also be expressed as E1−E2 _ Emin1−Emin2; i.e., the cascading schedule with more advantage and less disadvantage should be chosen.

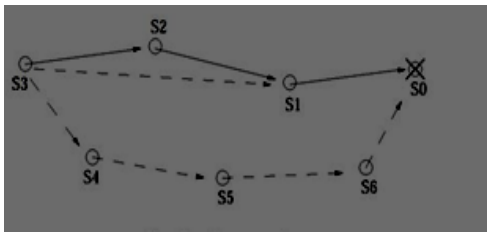

Fig. 6-

In Fig 6. moving *s*3 directly to the target location is the most energy efficient solution. However, in this way, *s*3 will be penalized, and its minimum remaining energy will be significantly reduced. If *s*1 is added as a cascading node, the load of *s*3 can be shared and the minimum remaining energy can be improved. Since the total length of the zigzag line *s*3*s*1*s*0 is only a little bit longer than the length of *s*3*s*0.

Algorithm

Before presenting the algorithm for calculating the best cascading schedule, we first introduce some notations, and describe how to modify Dijkstra's algorithm to calculate the shortest cascading schedule. The sensor network can be modeled as a complete weighted graph *G*(*V,E*), where vertices correspond to the sensor nodes. There are edges between any pair of nodes, and the weight of edge *sisj* is the distance between *si* and *sj* . The remaining power of *si* before relocation and after relocation is denoted by *Pi* and *P0 i* separately. We represent the energy as the distance that the sensor can move with this energy. In this way, if *si* moves to *sj* in the relocation, *P0 i* = *Pi* − *dij* . To calculate the shortest cascading schedule, we cannot simply apply Dijkstra's algorithm due to the constraint of the

Modified Dijkstra's

To calculate the best cascading schedule, we first calculate the shortest cascading schedule and record its total energy consumption E and its minimum remaining energy Emin. Then, we delete all the edges sisj if Pi − dij _ Emin and a new graph is generated. This process continues and a new shortest cascading schedule is calculated as long as the difference between the total energy consumption and the minimum remaining energy is increased compared to the previously calculated cascading schedule. When the process terminates, the schedule calculated before the last schedule is

Modified Dijkstra's Algorithm

Initialization: E = 0, Emin = −2, E0 = 0, Emin0 = −1
while (1)
1. find the shortest cascading schedule using
the Modified Dijkstra's algorithm
2. record the minimum remaining power as Emin0
3. delete all edges sisj if Pi − dij _ Emin0
4 if E0 − Emin0 < E − Emin then
E = E0,Emin = Emin0
else
return the previously calculated schedule the best schedule, i.e., the schedule with the smallest difference between the last two schedules As shown

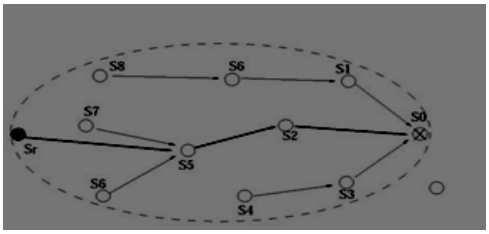

Fig. 7-

Distributed Protocol

In this section, we describe how to implement the algorithm presented above in a distributed way. We first present a distributed protocol to calculate the shortest cascading schedule and then describe how to use it to get the best cascading schedule. To calculate the shortest cascading schedule, the grid head of s0 initiates a dynamic programming computation by broadcasting a request message, which includes T0, t0, the redundant sensor sr, E0, and Emin0, where E0 and t0 are set to 0, and Emin0 is set to infinity. A node si receiving the request first determines if it can be the successor of the sender sj .If the answer is yes, it sets Ei = dij + Ej , Emini = min(Pi − dij,Eminj), and ti = Tj + tj − dij/speed. Then, it rebroadcasts Ti, ti, sr, Ei and Emini, and remembers its predecessor sj . If a node sj receives several such messages, it will choose the one from sk which can minimize Ej, whichis the energy consumption of the shortest cascading schedule from sj to s0. Then, sj calculates the other fields of the message, broadcasts the message and sets its predecessor to be sk. is shown in Fig. . To calculate the best cascading schedule, we only need to execute the distributed calculation of the shortest schedule iteratively similar to the algorithm shown after the following modifications: The minimum remaining energy calculated in the previous iteration, Emin0 is attached. When *si* receives message from *sj* , it will check whether Pi − dij _ Emin0**.** If yes, *si* and *sj* cannot be the successor and predecessor of each other. In addition to the current predecessor, each node also needs to record the predecessor in the previous iteration.

Acknowledgement

We would like to thank our Inspiration our principle Dr. Sidharth ladhake sir, our project Guide Dr. P.R. Deshmukh Sir, and all teacher staffs and friends who supported us for the completion of project.

References

[1] Jeff Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy and Srivastava M.B. (2006) *ACM Sensys World Sensor Web Workshop*, Boulder, Colorado, USA.
[2] Sensorplanet. *http://www.sensorplanet.org/*.
[3] Carlo Ratti, Andres Sevtsuk, Sonya Huang, and Rudolf Pailer. *http://senseable.mit.edu/graz/*.
[4] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Miu A.K., Eugene Shih, Hari Balakrishnan, and Samuel Madden (2006) *4th ACM SenSys*, Boulder, CO.