International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:3, No:3, 2009

PoPCoRN: A *Po*wer-Aware *P*eriodic Surveillance Scheme in *Co*nvex *R*egion using Wireless Mobile Sensor *N*etworks

A. K. Prajapati

***Abstract*—**In this paper, the periodic surveillance scheme has been proposed for any convex region using mobile wireless sensor nodes. A sensor network typically consists of fixed number of sensor nodes which report the measurements of sensed data such as temperature, pressure, humidity, etc., of its immediate proximity (the area within its sensing range). For the purpose of sensing an area of interest, there are adequate number of fixed sensor nodes required to cover the entire region of interest. It implies that the number of fixed sensor nodes required to cover a given area will depend on the sensing range of the sensor as well as deployment strategies employed. It is assumed that the sensors to be mobile within the region of surveillance, can be mounted on moving bodies like robots or vehicle. Therefore, in our scheme, the surveillance time period determines the number of sensor nodes required to be deployed in the region of interest. The proposed scheme comprises of three algorithms namely: Hexagonalization, Clustering, and Scheduling, The first algorithm partitions the coverage area into fixed sized hexagons that approximate the sensing range (cell) of individual sensor node. The clustering algorithm groups the cells into clusters, each of which will be covered by a single sensor node. The later determines a schedule for each sensor to serve its respective cluster. Each sensor node traverses all the cells belonging to the cluster assigned to it by oscillating between the first and the last cell for the duration of its life time. Simulation results show that our scheme provides full coverage within a given period of time using few sensors with minimum movement, less power consumption, and relatively less infrastructure cost.

***Keywords*—**Sensor Network, Graph Theory, MSN, Communication.

I. Introduction

THE wireless sensor networks [1, 4] have wide range of application such as environment monitoring (temperature, pressure, humidity, etc.), security and surveillance, fire alarming system, life support system, maritime navigation, and other potential applications [10, 5, 7]. Apart from cost constraint, one fundamental technical limitations of a sensor node is its low energy budget. In order to keep the cost of sensor network down, an application may follow of the following two approaches:

A. K. Prajapati is a PhD student with the Department of Computer Science and Engineering, Oakland University, Rochester, MI, 48309-4401 USA (e-mail: akprajap@oakland.edu or ashokkp@ieee.org; phone: +1- 248-678-2565).

- Use larger quantity of low capacity tiny sensor nodes, or
- Use limited number of sensor nodes with higher capabilities.

Tiny sensor nodes with low battery life will not able to do any complex computation as fast drainage of energy will lead the network to a quick death. Generally, sensor nodes are deployed in inaccessible, non-friendly regions for sensing the data. Therefore, it is not possible to provide service or replace the deployed sensor nodes. Replacing the spent batteries will not only be very costly but at times impossible. Therefore, battery life is crucial to the survival of a sensor network. It leads us to evolve a planning (scheduling both sensing and transmission of data) for operation of sensor nodes with low energy budgets. Therefore, coverage problem is key to any application involving environmental sensing through a wireless sensor network.

Various algorithms exist to handle coverage problems. For example, in *k*-coverage [8], each point in the region is monitored by at least *k* sensor nodes at a time. It implicitly assumes that there are more sensor nodes than the number of cells in the region to be covered. The *k*-set of sensor nodes are activated periodically to sense the data. Therefore, no sensor will remain active for entire duration of sensing. A significant amount of energy can thus be saved while target area is fully covered with active set of sensors [2]. There are many variations of *k*-coverage algorithm, some of which use disjoint sets of sensors and other use non-disjoint sets [3]. When the numbers of available sensors are less than the number of cells in the coverage area then monitoring by static sensors is not possible. The basic problem of coverage by mobile sensor nodes is that it does not allow for continuous monitoring. But if an application is tolerant to discrete periodic monitoring, it is possible to design appropriate schedules for periodic monitoring of the region with only a few mobile sensors. Mobile sensor nodes being more sophisticated can also be designed to support energy-aware operations. For example, with little extra computation they may perform data aggregation as well as compression to avoid unnecessary data transmissions. As a consequence, the

network life also can prolong. The contribution of this work is summarized below.

- It proposes energy efficient area monitoring centralized algorithm i.e., whole sensor network is controlled by a base station controller.
- This scheme is driven by mainly three algorithms, hexagonalization, clustering, and sensor scheduling, the first two algorithms are executed once, at the time of deployment, in the base station, and the scheduling algorithm runs at every sensor node.

This paper is organized into following five sections: section II presents a quick review of some previous work related to it. Section III presents an analysis of some theoretical background. Subsequently, section IV discusses the details of the proposed coverage scheme for periodic surveillance. Section V describes simulation and results of experiments. The paper ends with concluding remarks in section VI.

II. Related Work

Basically gathering of sensory data from environment is made possible by deploying a set of static sensors [11, 2, 3, 15, 6]. These low cost sensors are, as described earlier; operate with low power and low resources in hazardous environment or those inaccessible by human beings. Since replacement of node batteries in hostile environment is not possible, a full re-deployment has to be carried out every time after the deployed network dies. An exercise that should best be avoided as long as possible. Therefore, most of the work in the sensor network is centered around the survival of the network [1]. As a result many energy efficient area coverage strategies have been developed for sensor networks. Mobility of sensor nodes [12, 9, 14] adds another important dimension to the wireless sensor networks.

The mobile sensor nodes introduce dynamic coverage and measurement of environmental data [12, 9, 14]. In the paper [13], authors have designed an energy efficient mobile wireless sensor network, which is also not cost effective for certain kind of applications. Such networks are able to give full coverage of the sensing region for continuous monitoring and one region may be monitored by different sensor nodes at different times as dictated by front-end applications, if they are algorithmically well-designed. The PoPCoRN has been designed in such a way which includes full coverage along with power-awareness but it is more suitable for the applications which need periodic surveillance as well as relatively high power, high computational ability, slightly more storage, high sensing ability, etc. Because use of such high power sensor nodes would increase overall cost of a sensor network but on the other hand, requirement of the mobile sensor nodes are very less due to the dynamic nature of the nodes which makes PoPCoRN more suitable for such applications and turns out to be efficient in terms of cost, energy, coverage, capacity and network life with respect to tradition sensor networks.

III. Background

This section is concerned with the theoretical background that will help in understanding how mobility of sensor nodes can be program controlled to cover a sensing region by a small number of mobile sensor nodes within a pre-specified interval. It is assumed that all mobile sensor nodes movable with same speed. The interval of sensing that would guarantee coverage of the entire area is dependent on the number of available sensor nodes. Therefore, once the interval of sensing is specified, number of sensor nodes can easily be estimated to be deployed for gathering sensory data from the entire coverage area within the specified interval as per the application's requirements.

The area within the sensing range of a sensor node is considered as a unit sensing area. Each unit sensing area is approximated to a hexagon referred to as a *cell*, here-in-after. Each sensor has to cover a single cluster or a set of connected cells within the interval as per an application's requirement.

A Partitioning of Coverage Area

But, before the theory behind the process of gathering sensory data by a sensor from its assigned cluster of cells, is discussed, the coverage area is to be partitioned into hexagonal cells which represent a close packing of the area as in Fig. 2. This partitioning process is named here as hexagonalization.

For hexagonalizing any convex region, first it is needed to find the bounding box of the region. The bounding box is a rectangular region defined by coordinates of its left-bottom corner (0*;* 0), and right-top corner (*xmax; ymax*). Let sensor range be *R*. Then a regular hexagon with circum-radius *R* approximates the cell area. The in-radius *r* of such a hexagon with circum-radius *R* can be found with the help of the following simple geometrical formula.

$$r = R \sin 30^0$$

$$r = R \frac{\sqrt{3}}{2}$$

The rectangular region representing the bounding box is divided into an $m \times n$ grid of rectangular cells. Since the sides of bounding box may not be exact multiples of *R*, The approximation errors due to division is distributed equally among all the cells of the grid. So each cell is of size $(R + e) \times (R + e`)$, where $e = (xmax \% R) = m$ and $e` = (ymax \% R) = n$. In order to find the closed hexagon packing of the bounding box using the above grid, sides of hexagon can be categorized into three ways,

(i) Type 1 line, parallel to *x*-axis (horizontal lines),

(ii) Type 2 lines, intersecting x-axis at an angle of 30^0
(iii) Type 3 lines, intersecting x-axis at an angle of 120^0.

Now, consider a cell $c[i; j]$ of the grid, define $modX = i\%4$ and $modY = j\%4$. The line type is determined by use of following rules:

Rule 1. if ($modX\%2 = 0$ ^ $modY\%2 = 0$) then line type=1.
Rule 2. if ($modX = modY$ ^ $modX\%2 \neq 0$) then line type=2,
Rule 3. if ($modX \neq modY$ ^ $modX\%2 \neq 0$) then line type=3

The mid point of a line segment which defines a side of a hexagon is the point where in-circle of that hexagon will touch the corresponding side. Hence, from the knowledge of geometry, the coordinates ($px; py$) of the mid point can be calculated as

$$px = 3i \times cX; py = j \times cY;$$

where $cX = \frac{R}{4}$ and $cY = \frac{r}{2}$. Again from simple geometrical properties, it is easy to compute the end points of a line when mid point of the line ($px; py$), and parameters cX and cY *are* provided.

Consider the line labeled as (2) (i.e., line type 2) in Fig. 1. The end points of this line be denoted by $C(x_1; y_1)$ and $A(x_2; y_2)$. The mid point has coordinates ($px; py$). Therefore, it is obvious from Fig. 1 that the coordinates of end points are ($x_1 = px + cX$, $y_1 = py + cY$), and ($x_2 = px - cX, y_2 = py - cY$). We can actually prove that the line CA intersects x-axis at an angle $30^{\pm}$ as follows. Let the angle be denoted by θ, then

$$\theta = tan^{-1}\frac{CB}{BA} = tan^{-1}\frac{2cY}{2cX} = tan^{-1}\frac{R}{r/2} = tan^{-1}\frac{2(\sqrt{3}R)/2}{R} = 30^0$$

It can also be proved that the end points of:

- The line type 1 are (px-2$cX; py$), and (px+2$cX; py$), and that of
- The line type 3 are (px- $cX; py + cY$), and ($px + cX; py - cY$).

Furthermore, line types 1 and 3 lines will intersect x-axis at angles 0^0 and 120^0 respectively. It can, therefore, be concluded as stated by following theorem.
Theorem III.1: Hexagonalization process described above produces a close packing of a given finite convex region into regular hexagons each with a circum-radius of R.

Proof. It is already known that any finite rectangular region can be closely packed by a set of identical regular hexagons with circum-radii R each. For finding a close packing of any finite convex region, we proceed as follows:

1. Determine its bounding box. This gives a finite rectangular region.
2. Find a close packing of the bounding box.
3. Delete all hexagons which fall outside the boundary of the convex region.

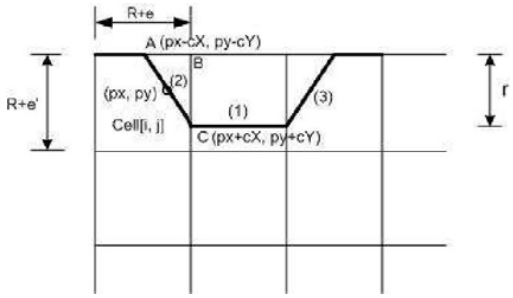

Fig. 1 Hexagon formation

B. Sensing Coverage using Concurrent Graph Search

Once cell partitioning of the coverage area is available from hexagonalization process, the abstract underlying structure of this area can be viewed in the form of a graph as follows:

- Each cell is mapped to a vertex of the graph,
- A pair of vertices in the graph is connected by an edge if and only if the cells represented by these two vertices are adjacent, i.e., they have a common border.

For example, the underlying graph for the coverage area depicted in Fig. 2 will be as shown in Fig. 3.

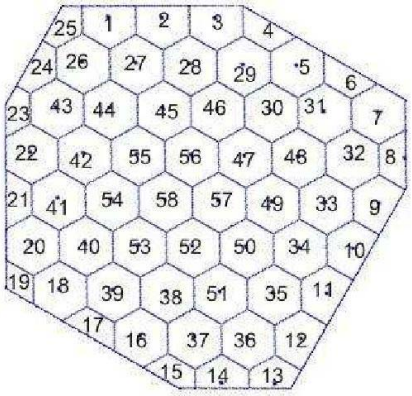

Fig. 2 Hexagonalized region

Using the graph abstraction the coverage problem simply can be viewed as a concurrent graph search problem. The search is initiated by placing the mobile sensor at different nodes and letting each sensor visit a cluster of connected nodes. The search is designed in way that all sensors can complete sensing within required time interval. Assuming that the sensors have identical capabilities, it simply means that the

clusters to be covered by the sensors are approximate of same size. Therefore, the clustering or grouping of connected cells of the region of coverage is performed in such a way that it only produces clusters of approximate same size. However, it is easy to see that the requirement regarding connectedness of cluster nodes is expected to interfere in the process of desired clustering. Fortunately, the underlying graph structure of a coverage area as indicated earlier is quite regular. Taking advantage of this regularity in graph structure, a technique can be devised that will ensure that this clustering process produces clusters of only two sizes which differs by 1 only.

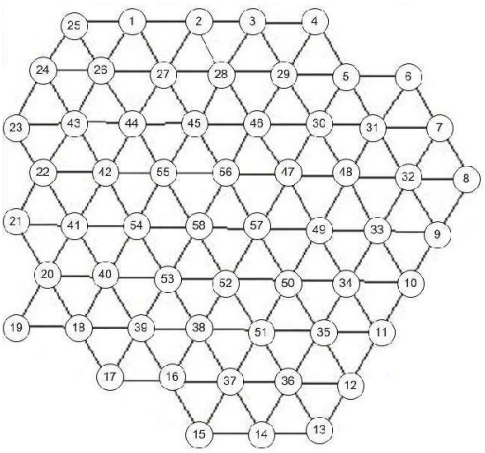

Fig. 3 Underlying graph of hexagonalized region

To provide a clear abstraction of the problem of clustering and scheduling sensor movements, certain theoretical results are discussed here in the form of three theorems which appear below. The results of these theorems form the basis of our approach in designing sensing coverage of a given region by mobile sensors.

Theorem III.2: Suppose k mobile sensors are available for gathering sensory data from n cells of a convex region. Then a sensor will cover a cluster of size either $\lfloor \frac{n}{k} \rfloor$ or $\lceil \frac{n}{k} \rceil$.

Proof: Let $n = mk + r$, where $r < k$.

(i) If n is a multiple of k then $r = 0$, so each cluster is of size n/k.

(ii) If n is not divisible by k, then $r < k$, and we can rewrite the expression for n as

$$n = m(k - r) + r(m + 1)$$
$$n = (k - r) \lfloor \frac{n}{k} \rfloor + r (\lfloor \frac{n}{k} \rfloor + 1)$$
$$n = (k - r) \lfloor \frac{n}{k} \rfloor + r \lceil \frac{n}{k} \rceil$$

Since $\lceil x \rceil = \lfloor x \rfloor + 1$,

Hence, at most two clusters sizes $\lfloor \frac{n}{k} \rfloor$ (i.e. m) and $\lceil \frac{n}{k} \rceil$ (i.e. $(m + 1)$) exist.

Theorem III.3: Consider the underlying graph G for a given convex region C. Let s be a boundary vertex of G where the search commences. Suppose, the search proceeds by visiting next unvisited vertex with least number of unexplored edges, then search will always proceed from boundary G to its inside.

Proof: Let all the vertices of the underlying graph G be stored in a list. The information stored in each node of the list is:

1. Degree of the vertex (number of unexplored edges incident at the node),
2. Pointers to its neighbors

Start searching G by picking a random vertex s from its boundary. Since each of the boundary vertexes has a degree strictly lower than six, the next vertex of least degree which is connected to the chosen start vertex s should belong to boundary of G. Only when no boundary vertex is left the search enters an interior vertex (a vertex with degree six). But the chosen interior vertex, being connected to an already visited boundary vertex, will have a degree strictly less than six; a fact, obvious from Fig. 4. Once a vertex is visited, it is marked "visited" as shown in the Fig. 4 by dotted lines. A visited vertex will not be considered for search again. So, the remaining set of unvisited vertices together can be viewed as a reduced graph $G`$ obtained by deleting the visited vertex along with edges incident on it (see Fig. 5). The deletion of edges reduces degree of the neighbors of visited vertex by one as shown in Fig. 5. Therefore, every time a boundary vertex $s$ of a graph $G$ is visited, each time different vertex, say $s`$ of the reduced graph $G` = G - \{s\}$ is picked as in Fig. 5. The process is repeatedly executed until unvisited vertex is left. Therefore, the search will always progress from boundary of G to its inside as stated.

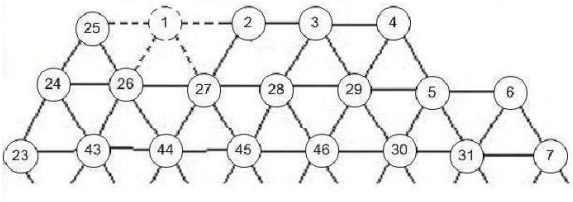

Fig. 4 Start vertex visited

The execution of graph search as explained in theorem III.3, produces a search tree in form of a simple open path is obtained as shown in the Fig. 6 by the sequence of nodes connected by thick lines. This simple path is partitioned into k simple paths each representing a cluster of desired size. The clusters and the vertices in each cluster are connected (indicated by separate dotted contours) as obvious from the Fig. 6.

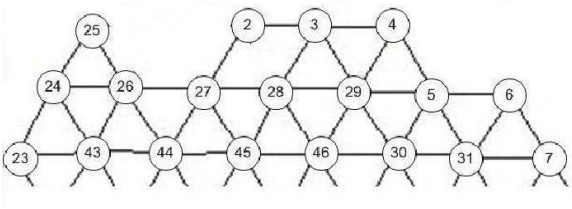

Fig. 5 Reduced graph after visiting start vertex

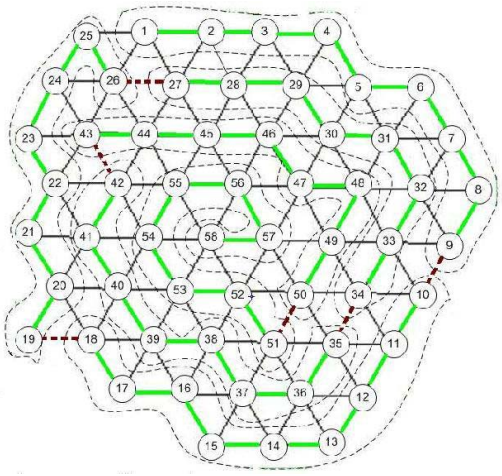

Fig. 6 Path produced by graph search and the resulting clusters

IV. Coverage Scheme

For simplicity, the coverage process is broken down into three distinct tasks, namely, hexagonalization, clustering, and *sensor scheduling*. Here each algorithm has been discussed separately.

A Hexagonalization Algorithm

This algorithm finds a close packing of the coverage region into minimum number of hexagons with circum-radius of each being equal to sensing range of the sensor type. It executes once at the central controller before deployment of the sensor nodes. Effectively, it finds the minimum number of static sensor nodes that will be required to cover a given region. On the basis of underlying geometrical properties for hexagonalization explained in section III, we divide the process of hexagonalization into several parts:

- One part determines the line type of a hexagonal cell,
- The other part determines the end points of the respective lines.

Algorithm: Hexagonalization

```
hexgonalization() {

        // This algorithm is executed only once
        // at base station before deployment of
        // sensor nodes. It divides the area into
        // hexagons. A convex polygonal area P is
        // provided as input to this algorithm.
        // bbox: bounding box.
        // xmax: maximum x-coordinate of bbox.
        // ymax: maximum y-coordinate of bbox.
        // R: sensing range of each sensor node.
        // rin: in-radius of hexagon approximating
        //      unit sensing area.

        p[2]; // Array of type point for storing end
              //points of the line segments.

        rin = p3(R+e`)/2;
        row_bbox = xmax/(R+e);
        col_bbox = ymax/(R+e`);

        for (i=1; i<=row_bbox; i++) {
                for (j=1; j<=col_bbox; j++) {
                        type = setType(i%4, j%4);
                        p = findPoint(type, i, j);
                        draw line segment with end
                        points
                        p[0],and p[1];
                }
        }
        for (each hexgon h) if (h ∉ P) exclude h;
}
```

The procedure setType is invoked to determine the type of the line to drawn while the procedure findPoint is required to determine the end points of the lines to be drawn.

Procedure: setType.

```
linetype setType(x,y) {

        // This procedure is called by
        // Hexagonalization for calculating
        // type of line(see algo. A).
        // type: line type.

        switch(x) {
                case 0: if (y==2) type=1;
                case 1: if (y==1) type=2;
                        if (y==3) type=3;
                case 2: if (y==0) type=1;
                case 3: if(y==3)type=2;
                        if (y==1) type=3;
        }
        return type;
}
```

Procedure: findPoint

```
point findPoint(type, i, j) {

        // This procedure is called by
        // Hexagonalization for calculating
        // end points of given line type.
        // (px, py): mid point of the line
```

```
        // whose ends, points are to be computed.

        p[2]; // Array of point type for storing
              // end points of the line segments
        cX = (R+e)/4;
        cY = rin/2;
        px = 3i*cX;
        py = j*cY;

        switch (type) {
                case 1:  p[0].x = px-2*cX;
                         p[0].y = py;
                         p[1].x = px+2*cX;
                         p[1].y = py;

                case 2:  p[0].x = px+cX;
                         p[0].y = py-cY;
                         p[1].x = px-cX;i
                         p[1].y = py+cY;

                case 3:  p[0].x = px-cX;
                         p[0].y = py-cY;
                         p[1].x = px+cX;
                         p[1].y = py+cY;
        }
        return p;
}
```

B. Clustering Algorithm

This algorithm takes vertices of the planer graph (i.e. centers of hexagons) as an input and groups the nodes into as many clusters as the number of available mobile sensor nodes. Each cluster has disjoint set of nodes (see theorem III.3) and nodes in the each cluster are connected. It finds k disjoint, homogeneous or heterogeneous size clusters depending on whether k is even or odd respectively. When k is odd, (k-r) clusters of size m and r clusters of size (m + 1) are produced, where $r = n\%k$ (see theorem III.2). For convenience in description, a top level algorithm only describes the search process as explained in section III. The data structures which the top level algorithm requires as input are constructed by the algorithm *buildClusterLists* with the input from hexagonalization process described in the previous section.

Algorithm**:** Clustering.

```
buildClusters() {

        // This algorithm is executed only
        // once at base station before
        // deployment of sensor nodes.
        // It partitions the underlying graph
        // of hexagonalized area into k
        // simple open paths or clusters
        // where k is the number of available
        // sensors. A cluster i is provided by
        // list[i], its sequence of vertices.
        // m: cluster size, i.e., number of
        //      vertices in a cluster.
        // k1: number of clusters of size
        //      m, k = k1  when r = n%k=0.
        // k2: number of clusters of size m + 1.

        if (r ≠0) {
                k1 = k-r;
                k2 = r;
        }
        if (r == 0)

                buildClusterLists(m,k, list[1..k]);
        else {
                buildClusterLists(m,k1, list[1..k1]);
                buildClusterLists(m,k2, list[1..k2]);
        }
        for (i=1; i<=k; i++){
                send list[i] to sensor i;
   }
}
```

Procedure: buildClusterLists.

```
        // This procedure is called by
        // clustering for computing clusters.
        // It uses a standard method to create
        // a doubly linked list which is referred
        // to as doublyLinkedList(info,prev,next)
        // n: number of clusters to be computed.
        // m: cluster size.

        buildClusterLists(m, n) {
           if (n≠ 0) {
              pick a vertex v from boundary;
                 while(n > 0) {
                    mark v;
                        nbr ← reference to neighbor
                        of v with least
                        unvisited degree;
                        list[n]=new doublyLinkedList
                        (v, null, nbr);
                        p ← list[n].next;
                        for (j=m ¡ 1;j>1;j-) {
                            mark p.vertex;
                            nbr← reference to
                            neighbor of p.vertex
                            with least unvisited
                            degree;
                            p=new doublyLinkedList
                            (p.vertex, p.prev, nbr);
                             p ← p.next;
                        }
                        if (j==1){
                            p=new
                            doublyLinkedList(p.vertex,
                            p.prev, null);
                      }
                        n = n-1;
```

```
                    pick next vertex with least
                    unvisited degree ;
}
                }
        }
}
```

C. Node Scheduling Algorithm

The cluster lists obtained from the previous algorithm are used as an input to this algorithm. Note that each node of a list corresponds to a node of the cluster represented by the list. One list is allocated to each sensor node. The scheduler uses this list for traversing the cells (vertices) corresponding cluster. Each list is organized in the form of a doubly linked list with following information at each node of linked list:

- Address of cell corresponding to the node,
- Left and right pointers.

Each sensor checks the current node of the list assigned to it. The sensor picks the address of cell in the current node of the linked list, senses data from that cell. The sensed data is then transmitted to base station. After that the sensor picks up the address of the cell corresponding to the list node pointed to by its right pointer and continues both sensing and transmitting next cell's data. When the right pointer of the current node points to null, the left pointer is picked up and the process continues using successive left pointers till left pointer of the current node points to null. After that the right pointer is picked up again. Essentially, each sensor node oscillates between two end vertices of the cluster assigned to it. The details of the traversal scheme of a single sensor as outlined above are provided by the scheduler algorithm below.

Algorithm: Sensor Scheduler.

```
sensorScheduler() {

        // This algorithm is to be executed
        // by a sensor node. The node traverses
        // the cluster or the path using the
        // list provided by the base station.

        if(list==null) error;
        while (true) {
                while (list.next ≠null) {
                        store the data;
                        list ← list.next;
                }
                Transmit data to the base station;
                while(list.prev≠null) {
                        store the data;
                        list ← list.prev;
                }
        Transmit data to the base station;
}
```

V. Experiments

The simulation has been done using Java 1.5 on a P4 Machine. The coverage area considered is 72,683 (pix) units. With a homogeneous circular cell of size approximately 1810 units, i.e., with a sensor range of 24 units the coverage region will consists of approximately 41 cells. But since each cell area was approximated by its inscribed hexagon (to cover the whole sensing area without holes) exactly 58 hexagonal cells are required. So the underlying graph representing the coverage area consists of 58 vertices. As Fig. 7 depict, indeed the result of hexagonalization process (see section A) on the coverage area produced a packing of the area by exactly 58 hexagons.

In order to sense environmental data from the cells we experimented by varying mobility of sensors. It has been considered that the factors like speed of a mobile sensor, and the distance traveled by it for moving from one cell to an adjacent cell. These parameters are assumed to be unity. Although we have carried out experiments with actual distances and different speeds of mobile sensor, the behavior of the plot remained unchanged, only traversal time varied with distance and speed. The results has been taken by increasing the number of sensor nodes and calculated time taken in traversal of whole region at every period. In Fig. 8, in-circled hexagons show the one end node of every cluster and number inside circle shows the cluster number. As in our simulation, there are total 7 clusters and first two clusters have 9 nodes and last five clusters have 8 nodes.

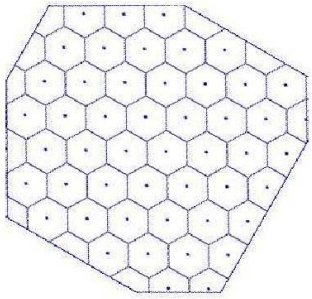

Fig. 7 Hexagonalized area

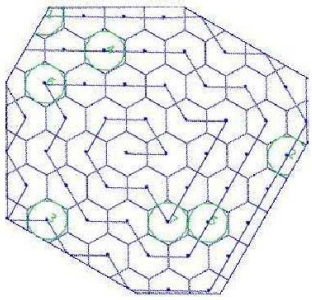

Fig. 8 Cluster Formation

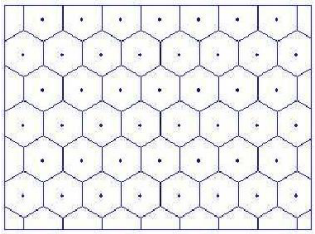

Fig. 9 Rectangular Region of Same Area as Convex Region in Fig. 7 with relative error 0.004%

The plot shown in the Fig. 10 is for the number of sensor nodes versus the traversal time for any convex region (including rectangular regions). As the number of sensor nodes increases, traversal time decreases, i.e., traversal time is inversely proportional to the number of sensor nodes. As it is clear from the graph, when number of sensor nodes are 58 (for convex region), traversal time is zero because each sensor node is responsible for sensing data from just one cell assigned to it. Whereas if only single mobile sensor is assigned to sense data from all the 58 cells, the traversal time is maximum, i.e., 57 units. The rectangular region of approximate same area (72,680) as shown in Fig. 7, hexagonalized rectangular region (produced from the simulation) looked like the one in Fig. 9. The number of sensors required to cover entire region statically is 60. A single sensor takes 59 units of time in one cycle for traversing the entire region.

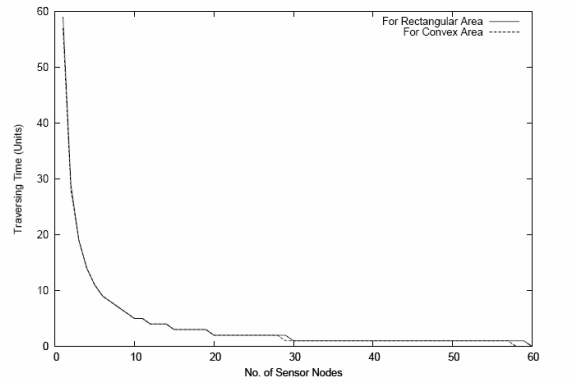

Fig. 10 This graph shows the time taken in traversing the given area with the increase of sensor nodes using Speed=1 unit, and Separation between Two Adjacent Nodes=1 unit for *Rectangular Region* and *Convex Region* of same areas with the relative error of 0.004%

The plot in Fig. 11 shows the performance of mobile sensor network in terms of the network cost. As the size of region increases, cost of network increases. The cost has been computed by assuming that every sensor costs one unit, and the maximum allowed interval for surveillance monitoring of the region is fixed at 5 units.

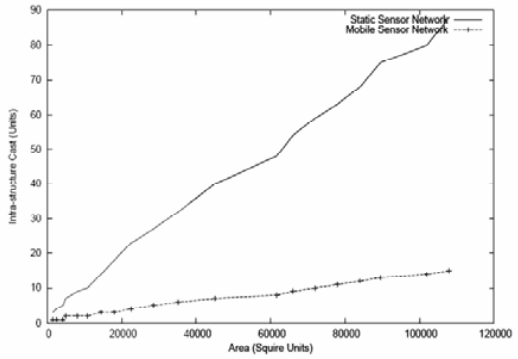

Fig. 11 This graph is drawn for the cost of the sensor network with the increase in the Area of the region, Maximum Traversal Time=5 units, and the cost per sensor node=1 unit

VI. Conclusion

The periodic surveillance by wireless mobile sensor nodes represents a special class of sensor network applications. It has been shown that for periodic monitoring, very low cost mobile sensor networks can be designed that are able to fulfill the desired task and even with the same cost as tradition sensor networks, high power mobile sensor net- works can be designed. PoPCoRN is better trade off between cast and period (between two consecutive surveillances). Main drawback with this schemes are, number of sensor nodes totally depend on the period of surveillance and sensor nodes require some more memory to keep the cluster information and some small processing unit for local processing because processing is always cheaper than communication. If every time node accesses information from any central controller, it would increase the communication cast as well.

Acknowledgment

The author thanks to Dr. G.Sanyal, Dr. R.K. Ghosh, Dr. F. Mili, and the anonymous reviewers for their continued and valuable suggestions in improving the work throughout.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks:a survey. Computer Networks, 38:393– 422, 2002.
[2] M. Cardei, M. T. Thai, Y. Li, and W. Wu. Energy-efficient target coverage in wireless sensor networks. 24th IEEE Conference on Computer Communications, 3:1976 – 1984, March 2005.
[3] J. Carle and D. Symplot-Ryl. Energy-efficient area mon- itoring for sensor networks. Ad-hoc networking by IEEE Computer Society, 37(2):40 – 46, February 2004.
[4] C.-Y. Chong and K. S.P. Sensor networks: Evolution, opportunities, and challenges. Proc. of IEEE, 91(8):1247–1256, August 2003.
[5] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In Proceedings of ACM International Conference on Mobile Computing and Networking, pages 263–270, August 1999.
[6] T. Huang and Y. Tseng. Coverage problems in wireless sensor networks. ACM Mobile Networks and Applications (MONET), special issue on Wireless Sensor Networks, 10(4):519– 528, Aug 2003.
[7] J. M. Kahna, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for 'smart dust'. In 23rd IEEE Conference on Computer Communications, pages 271 – 278, March 2004.

[8] S. Kumar, T. H. Lai, and J. Balogh. On k-coverage in a mostly sleeping sensor network. In Proceedings of the 10th annual international conference on Mobile computing and networking, pages 144 – 158, Sep 2004.
[9] B. Liu, P. Brass, and O. Dousse. Mobility improves coverage of sensor networks. In Proceedings of ACM Mobihoc'05, pages 300 – 308, March 2005.
[10] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson. Wireless sensor networks for habitat monitoring. In Proceedings of ACM International Conference on Mobile Computing and Networking, pages 88 – 97, September 2002.
[11] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'01), 3:1380–1387, April 2001.
[12] T. F. L. Porta, G. Cao, and G. Wang. Movement-assisted sensor deployment. IEEE Transactions on Mobile Computing, 5(6):640 – 652, June 2006.
[13] P. S. and L. F.L. Energy efficient mobile wireless sensor networks. In ASME International Mechanical Engineering Congress and Exposition, 2006.
[14] C. Schindelhauer. Mobility in wireless networks. In 32nd International Conference on Current Trends in Theory and Practice of Computer Science, pages 100–116, Sep 2006.
[15] L. Xiang-Yang, W. Peng-Jun, and O. Frieder. Coverage in wireless adhoc sensor networks. IEEE Transactions on Computers, 52:753– 763, June 2003.

A. K. Prajapati is a PhD student in the Oakland University Michigan, and received the B. Tech.(CSE) from KNIT Sultanpur India and M.Tech.(CSE) from NIT Durgapur India. He has been with UP Technical University, Wipro Technologies, and Hughes before joining the PhD programme. His research interests include Wireless Networks, WiMAX, Optimizations, etc. He is a member of the IEEE.