A Node-Centric Load Balancing Algorithm for Wireless Sensor Networks

Hui Dai , Richard Han
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado, 80302
Email: {huid, rhan}@cs.colorado.edu

***Abstract*— By spreading the workload across a sensor network, load balancing reduces hot spots in the sensor network and increases the energy lifetime of the sensor network. In this paper, we design a node-centric algorithm that constructs a load-balanced tree in sensor networks of asymmetric architecture. We utilize a Chebyshev Sum metric to evaluate via simulation the balance of the routing trees produced by our algorithm. We find that our algorithm achieves routing trees that are more effectively balanced than the routing based on breadth-first search(BFS) and shortest-path obtained by Dijkstra's algorithm.**

I. Introduction

Wireless sensor networks(WSNs) have recently emerged as an active research area. Typically, a WSN consists of a large number of nodes that sense the environment and collaboratively work to process and route the sensor data. [4] [6] A large number of application scenarios for such WSNs have emerged, including battlefield monitoring, habitat monitoring, tracking of office equipment, and medical/health deployments in the home [1] [3]. As sensor networks scale up in size, effectively managing the distribution of the networking load will be of great concern. By spreading the workload across the sensor network, load balancing averages the energy consumption. This extends the expected lifespan of the whole sensor network by extending the time until the first node is out of energy. Load balancing is also useful for reducing congestion hot spots, thereby reducing wireless collisions.

We focus on WSNs with an asymmetric architecture, i.e. a powerful base station collects data through a multi-hop routing framework of distributed wireless sensor nodes. This centralized architecture rooted in the base station is common to sensor networks [7] [5] [12] [13]. A base station serves as the data aggregation point or the sink of the data in the network. Typically, the base station has more resources in terms of power, computation, memory, and bandwidth than the individual sensor nodes. The base station has thus been proposed as the resource-rich focal point for tracking failed nodes [7], securing the sensor network against compromised sensor nodes [5], hosting services such as data aggregation [12] [13], or monitoring of WSNs [8] [15]. In this paper, we also assume the common case of static sensor networks in which the position of the sensor nodes are fixed.

Previous work in sensor network routing [10] [2] as well as QoS routing in wireless ad hoc Networks [14] largely ignores load balancing issues. Hsiao et al [9] designs a rebalancing scheme to achieve load balancing for wireless access networks, though not wireless sensor networks. This work assumes the existence of an initial unbalanced tree that is then readjusted or rebalanced using their algorithm, and selects a random neighbor for rebalancing. In contrast, our algorithm presents a complete solution that forms the initial tree, and rebalances this tree using topological knowledge rather than random selection.

The key contributions of this paper are in the following three areas. First, we identify the importance of the node-centric approach. Second, we formulate a node-centric load-balancing problem that helps construct the routing and monitoring structures for an asymmetric sensor network. Third, we present the construction algorithms for load balancing.

II. Load Balancing In Sensor Networks

A node-centric load balancing strategy considers the cumulative load of data traffic from child nodes in a routing tree on their parent nodes. The WSN routing tree is rooted in the base station. The load of child sensor nodes adds to the load of each upstream parent in the tree. Hence, the sensor nodes nearest the base station will be the most heavily loaded. The goal of node-centric load balancing is to evenly distribute packet traffic generated by sensor nodes across the different branches of the routing tree.

A shortest path routing algorithm executed on a sensor grid rooted in a base station doesnt guarantee that the resulting shortest path tree is load balanced. Figure 1(a) illustrates that a shortest path tree that minimizes the number of hops can result in a highly unbalanced tree. This is because selecting the shortest path doesn't account for the effect of load aggregation on upstream links. The base station is the clear node at the root of the tree, and the assumption is of a uniform grid with each node generating the same load, i.e. generating the same amount of periodic sensor data. In comparison, a balanced tree is shown in Figure 1(b), where the load is precisely the same on each of the branches emanating from the root. The tradeoff of achieving a load-balanced tree is that some nodes will have a longer path to the root than the shortest path, e.g. node X in Figure 1.

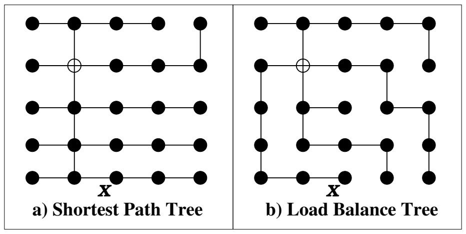

Fig. 1. Unbalanced shortest path tree vs. Top-level balanced tree

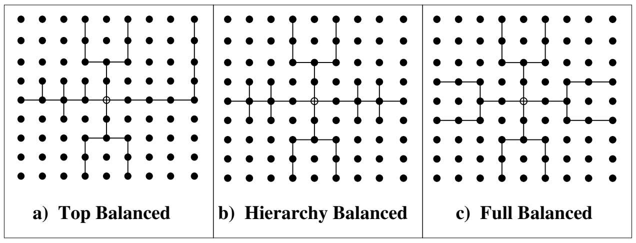

Fig. 2. Load Balanced Trees on the Grid Topology

All nodes in the sensor network periodically broadcast their existence, e.g. an IMA message in the WSNDiag protocol [11], and neighboring information. After collecting this information, the base station constructs the graph $G(V, E)$ (where V is the vertex set while E is the set of all edges). An algorithm is executed on G to construct a load-balanced tree. The "load" associated with a given sensor node represents the amount of data periodically generated by that sensor node.

Load balanced trees can be classified into different categories. We define the "level" to be the distance from a node to the base station. A load-balanced tree could be fully balanced, top-level balanced or hierarchy-balanced. A fully balanced tree is a tree in which the branches on the same level carry the same total amount of load. A top-level balanced tree is a tree such that each branch at the top level closest to the base station carries the same amount of load [9]. Both fully balanced trees and top-balanced trees are extreme cases of hierarchy-balanced trees, i.e. a tree in which the branches in certain levels carry the same total amount of load. Figure 2 provides examples that illustrate these three versions of balanced trees in a sensor grid. In this paper, our node-centric load balancing technique focuses on constructing a top-balanced tree over the sensor network. Figure 3 shows a top-balanced tree constructed using our load balancing algorithm on a 10x10 grid sensor network.

III. Load Balancing Algorithm

This section discusses the construction and adjustment of the top-level balanced tree for a WSN. A grid topology is assumed for simplicity, though the algorithm is not limited to a sensor grid. In the grid, one of the nodes is randomly selected and assigned to be the base station.

To measure how well the load is balanced across different branches of a routing tree, the Chebyshev Sum Inequality is selected as the load balancing metric. The definition of the Chebyshev Sum Inequality is as follows: for all $a \subseteq \mathbf{C}^N$ and

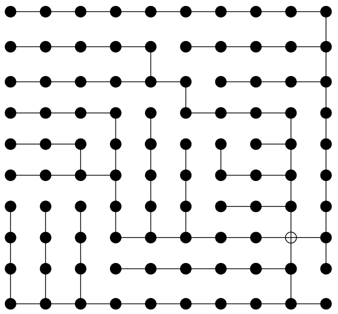

Fig. 3. A top-level load balanced tree on a 10x10 grid network

$b \subseteq \mathbf{C}^N$,

$$a = \{a_1, a_2, a_3, \ldots, a_n\}$$

$$b = \{b_1, b_2, b_3, \ldots, b_n\}$$

and

$$a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_n$$

$$b_1 \geq b_2 \geq b_3 \geq \ldots \geq b_n$$

Consequently:

$$n \sum_{k=1}^{n} a_k b_k \geq (\sum_{k=1}^{n} a_k)(\sum_{k=1}^{n} b_k)$$

Let W_{bi} be the weight (cumulative load) on the i_{th} branch of the routing tree. Form a vector of the weights $w = \{W_{b1}, W_{b2}, W_{b3}, \ldots, W_{bn}\}$. For example, in a square grid, the base station in the center of the grid will have four neighbors, so there will be four weights.

To assess the degree of balance among the different branch weights of w, let $a = b = w$. In this case, the inequalities will become

$$n \sum_{k=1}^{n} W_{bk}^2 \geq (\sum_{k=1}^{n} W_{bk})^2$$

or,

$$1 \geq \frac{(\sum_{k=1}^{n} W_{bk})^2}{n \sum_{k=1}^{n} W_{bk}^2}$$

with equality if and only if $W_{b1} = W_{b2} = W_{b3} = \ldots = W_{bn}$ for all $W_{bk}, k \in [1, n]$. The balance factor θ used in the algorithm is defined as

$$\theta = \frac{(\sum_{k=1}^{n} W_{bk})^2}{n \sum_{k=1}^{n} W_{bk}^2}$$

As the weights in each branch converge to the same value, i.e. the load across the different branches of the routing tree becomes more balanced, the balance factor monotonically increases towards 1. When the weights of all the branches are equal, the result of the inequality will be 1, i.e. the maximum value.

TABLE I

BASIC LOAD BALANCING ALGORITHM

```
$M \Leftarrow Allnodes$;
while(M is not empty) do
    step 0: Select the lightest most restricted branch
    B = B[0]
    for each B[i] do
        if (Weight(B) $\neq$ Weight(B[i]))
            /* select lightest branch */
            $B \leftarrow$ lighter $(B[i], B)$;
        else
            /* if same load, select most restricted branch */
            $B \leftarrow$ minFreedom $(B[i], B)$;

    step 1: Select the heaviest border node with most growth space
    $n' = n_0 \subseteq N$, N is Bś border node list
    for each $n_i \subseteq N$
        if Weight(ń) $\neq$ Weight($n_i$)
            /* Select heaviest border node */
            $n' \leftarrow$ heavier($n', n_i$);
        else
            /* Select border node with max growth space */
            $n' \leftarrow$maxFreedom($n', n_i$);

    step 2: graft node and update metrics
    T = T + $\{n'\}$
    $N = N - \{n'\}$
    $M = M - \{n'\}$;
    for each unmarked border node $i$ of $n'$
        $N = N + \{i\}$;
done
```

A. Basic Algorithm

The node-centric algorithm iteratively grows a load-balanced tree outwards from the base station root. At each step, the algorithm first selects the branch with the lightest load, and then grafts onto this branch the unassigned/unmarked border node generating the heaviest load. Intuitively, the algorithm absorbs the nodes generating the greatest load to the lightest branches to achieve balance. In addition, a crucial observation is that absorbing the heaviest nodes at the earliest possible step maintains the greatest flexibility for future balance. In comparison, absorbing the heaviest nodes at the end of the algorithm could lead to highly unbalanced trees.

If there are multiple "heaviest" border nodes, as in a uniform sensor network, then select the unmarked border node with the greatest "growth space". Intuitively, the algorithm expands or grows the routing tree into the most open areas in the sensor grid before filling in the crowded areas. The motivation is to maximize the flexibility in terms of routing options at each step. This approach reduces the chance that a given branch will become hemmed in, unable to grow, which would create an unbalanced tree with an abridged branch carrying an unusually light load.

The pseudo code for the algorithm is shown in Table I. The base station collects the initial topology and load information and computes the backbone tree from graph G. We define the following variables: the current tree T; the array of branches $B[i]$; the selected branch B; lists of the border nodes for each branch $N[]$, and the set of unmarked nodes M.

A key concept in this algorithm is the "growth space" of a node, i.e. a measure of the freedom to grow the tree towards this node. The greater the growth space, the more open area there is to expand the load-balanced routing tree through this node. Each sensor node has a number of unmarked neighbors. The growth space of a node is the sum of the number of unmarked neighbors of all the node's unmarked neighbors minus common links. For example, Figure 4 shows a load balanced tree in the process of construction, with four branches emanating from the root base station. In Figure 4(a), node Z has 2 unmarked neighbors to its east and south, while each of these unmarked neighbors has 3 unmarked neighbors themselves. Therefore, the growth space of Z equals 3 + 3 - (2 common links) = 4, as shown in Figure 4(b).

The growth space is used in the algorithm of Table I in two places. First, it is used in step 0 to break ties between equally lightly loaded branches. The growth space of a *branch* is defined as the sum of the growth spaces of all nodes within the branch. This is a measure of how much freedom the branch has to expand. If there is a tie between multiple lightly loaded branches, then the branch with the most restrictions, i.e. smallest growth space, is selected. This enables restricted branches the most opportunity to grow, and avoid being hemmed in, which would lead to unbalanced trees. Second, the growth space is used in step 1 to break ties between equally heavily loaded border nodes, for the reasons outlined earlier.

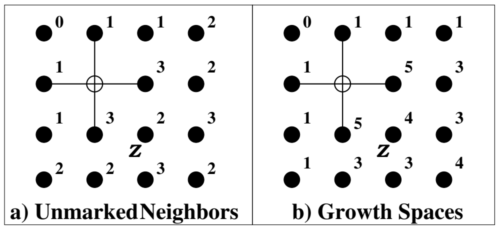

Fig. 4. (a) Number of unmarked neighbors of each node (b) growth space of each node

The time complexity of the algorithm is comparable to the algorithm constructing the shortest path tree, because at the end of each iteration, information related to neighbor nodes and branches needs to be updated. The modifications are similar to those on the path length after each iteration in the Dijkstra algorithm. With appropriate data structures supported, the time complexity could be $O(nlogn)$.

B. Adjustment

While the basic algorithm produces a roughly load balanced tree at the top level, an additional adjustment algorithm is needed to achieve further load balancing. There are several adjustment techniques available. The random adjustment method [9] has been previously used to re-adjust a roughly balanced tree. However, these algorithms are blind to the topology information. Here we propose a spiral adjustment algorithm that uses the topology information obtained in the first phase.

TABLE II
ADJUSTMENT ALGORITHM

```
$Avr \longleftarrow$ the average number of the nodes on a brunch
$B \longleftarrow$ Heaviest Brunch that has maximum neighbors
While (Not meet the stop criteria) do
    if Weight($B$) is bigger than average
        $\delta = |B| - Avr$;
        if there is node $m$ that has load close to $\delta$
            Push $m$ to $B's$ unmarked neighbor
        else
            connect all leaf nodes to neighboring branches
            that can improve the balance factor
    if Weight($B$) is smaller than average
        Pull the leave nodes from the neighbor
    $B =$ the next connected unmarked neighbor
```

After the first phase, i.e. after the basic algorithm has been completed, the adjustment algorithm iteratively rebalances the tree by moving nodes from the heaviest loaded branches to more lightly loaded neighboring branches. First, the most heavily loaded branch is found, and the deviation from the optimally balanced load is calculated by subtracting the branch's load from the average load of an optimally balanced tree. Second, a node within this heaviest branch is selected to be moved to a neighboring branch. The algorithm first attempts to move a node whose load most closely matches the deviation. If such a node cannot be moved, e.g. it is an interior node, then the algorithm searches among all of its border nodes that are also leaves. The algorithm continues to migrate leaf nodes to neighboring branches as long as the balance factor is improved. The algorithm finishes after a stop criterion is reached, e.g. stop after 10 iterations.

The algorithm spirals in the sense that it rotates through each of the tree's top-level branches, either pushing neighbors from heavily loaded branches to lighter ones, or by pulling neighbors to lightly loaded branches from heavier ones. Table II provides the pseudo-code for the adjustment algorithm.

IV. Simulation And Performance Evaluation

We have built a Java simulator to evaluate the load balancing performance of our spiral node-centric strategy, and have compared the balance factor resulting from our basic algorithm with the routing trees produced by breadth-first-search (BFS) and shortest path routing in sensor networks. Square sensor grids with both uniform and non-uniform load distributions were studied. The convergence speed of our spiral adjustment algorithm was also compared to a random adjustment algorithm.

First we evaluate the load balancing performance of our basic algorithm compared with shortest path tree (SPT) and the tree created by BFS. The shortest path tree was generated using Dijkstra's algorithm using a link cost of 1 for each link. Since there will be many nodes at each iteration that are an equal distance from the SPT, i.e. all border nodes are exactly distance 1 from the SPT, then a node is randomly selected. In practice, heuristics such as lowest address ID are used to break such ties. The BFS algorithm grows the tree from the root in a rotational fashion, rotating between branches. All nodes a distance N hops from the root are appended to the branches in rotational fashion, before proceeding to all nodes $N + 1$ hops from the root.

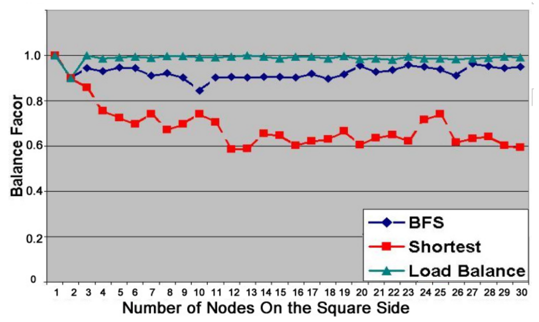

Fig. 5. Average Performance Comparison

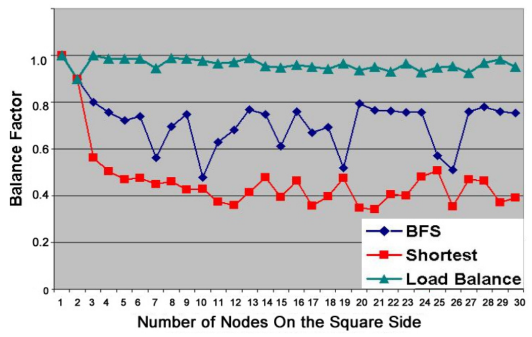

Fig. 6. Worst Performance Comparison

Figure 5 and Figure 6 assess the balance factor of the routing trees produced by the three algorithms as a function of the square grid's length on a side for a uniform load. For each of the square lengths, the experiment is executed 20 times. The root is assigned randomly in each set of the experiments. From Figure 5, the shortest path algorithm produces the most unbalanced trees, while our basic algorithm is slightly better balanced on average than BFS. In the worst, our basic algorithm considerably outperforms both SPT and BFS. Worst cases occur when the root base station is located near the edge or corner of the square grid, so that both BFS and SPT produce highly unbalanced trees, i.e. some branches are hemmed in and therefore especially short. In contrast, our basic algorithm attempts to expand the lightest branches into open space, to avoid confining the growth of branches.

In Figure 7, we randomly assign different load-generating weights to several nodes. Each of the selected nodes is assigned weight 20, while all the other nodes have weight 1. The number of "heavy nodes" is increased from 1 to 10. In each case, we repeat the experiment for 20 times. The square grid is 20x20. Figure 8 shows that our basic algorithm achieves much better load balancing than BFS and SPT for uneven sensor networks, because BFS and SPT are not accounting for load aggregation. In comparison to uniform sensor networks,

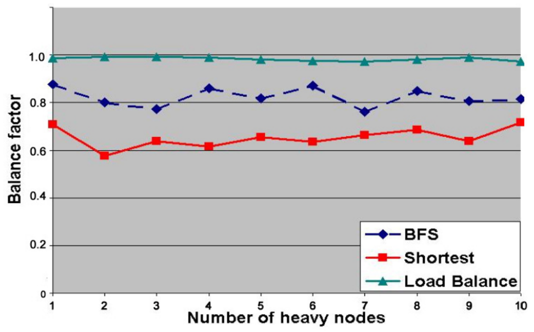

Fig. 7. Average Performance Comparison in an uneven sensor network

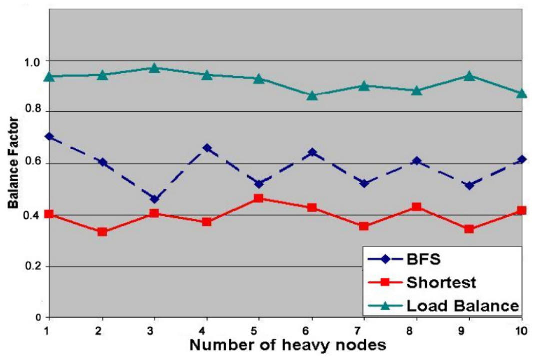

Fig. 8. Worst Performance Comparison in an uneven sensor network

our basic algorithm has more trouble balancing uneven load generators yet is still relatively successful.

Figure 9 shows that the convergence speed of our spiral adjustment algorithm is at least twice as fast as a random adjustment algorithm, because our node-centric approach uses topology information like branch load factors.

Figure 3 illustrates the tree structure after applying our entire algorithm with adjustment to a 10x10 grid. At the end of the basic construction algorithm, the branch numbers are 27, 26, 21, and 25. The balance factor is 0.996. After adjustment, the number of nodes on each branch is 25, 25, 25, 24, and the balance factor is increased to 0.9997. The final tree achieves maximum balance as shown in Figure 3.

V. Conclusion

In this paper, we provide a node-centric model for load balancing of wireless sensor networks. Our load balancing mechanism grows trees iteratively from the base station. First, the most lightly loaded and most confined branches are selected for growth. Second, the heaviest nodes with the maximum growth space are selected. After establishing a loosely balanced tree, we run adjustments that migrate subtrees to neighboring trees, and thereby achieve greater balance. Our algorithm achieves considerably better balanced trees than bread-first-search and shortest-path routing, both for uniform and unevenly weighted nodes in a square topology. We introduce a load balancing metric based on the Chebyshev Sum inequality.

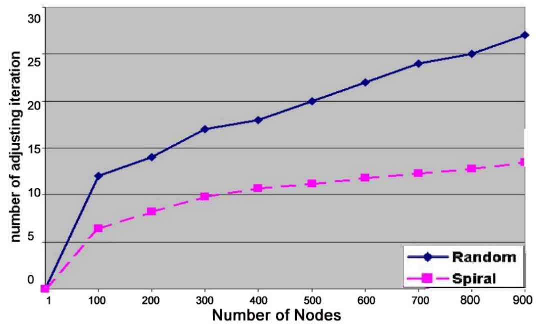

Fig. 9. Performance Comparison between the Random adjustment and the spiral adjustment

References

[1] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat Monitoring: Application Driver for Wireless Communications Technology", Proceedings of the First ACM SIGCOMM Workshop on Data Communications in Latin America, 2001.

[2] A. Manjeshwar and D. P. Agrawal. "Teen: A routing protocol for enhanced efficiency in wireless sensor networks". In 1st International Workshop on Parallel and Distributed Computing Issues in Wireless, 2001.

[3] D. Estrin, L. Girod, G. Pottie, M. Srivastava , "Instrumenting the world with wireless sensor networks" In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001), Salt Lake City, Utah, May 2001.

[4] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci. "Wireless Sensor Networks: A Survey", Computer Networks, 38(4): 393-422, March 2002.

[5] J. Deng, R. Han, S. Mishra, "A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks", IEEE 2nd International Workshop on Information Processing in Sensor Networks (IPSN '03), 2003, Palo Alto, California.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, "System Architecture Directions for Networked Sensors". Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, November 2000.

[7] J. Staddon, D. Balfanz and G.Durfee "Efficient tracing of failed nodes in sensor networks" In WSNA '02.

[8] J. Zhao, R. Govindan and D. Estrin, "Residual Energy Scans for Monitoring Wireless Sensor Networks", IEEE Wireless Communications and Networking Conference (WCNC'02), Orange County Convention Center, Orlando, FL, USA, 17-21 March, 2002.

[9] P. H. Hsiao, A. Hwang, H. T. Kung, and D. Vlah. "Load-Balancing Routing for Wireless Access Networks". IEEE Infocom, April 2001.

[10] Q. Li, J. Aslam, and D. Rus. "Hierarchical power-aware routing in sensor networks". In Proceedings of the DIMACS Workshop on Pervasive Networking, May 2001.

[11] S. Chessa, P.Santi, "Crash Faults Identification in Wireless Sensor Networks", Computer Communications, Vol. 25, No. 14, pp. 1273-1282, Sept. 2002.

[12] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad hoc sensor networks", in USENIX Symposium on Operating Systems Design and Implementation, 2002.

[13] S. R. Madden, R. Szewczyk, M. J. Franklin, and D. Culler, "Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks", in Workshop on Mobile Computing Systems and Applications, 2002.

[14] S. S. Lee, M. Gerla "Fault Tolerance and Load Balancing in QoS Provisioning with Multiple MPLS Paths" IWQoS 2001: 155-169.

[15] Y. J. Zhao, R. Govindan and D. Estrin, "Sensor Network Tomography: Monitoring Wireless Sensor Networks" Student Research Poster, ACM SIGCOMM 2001, San Diego, CA, USA. August 2001