



RESEARCH ARTICLE

EQUAL WORKLOAD DISTRIBUTION TREE CONSTRUCTION ALGORITHM FOR  
WIRELESS SENSOR NETWORKS

\*<sup>1</sup>Laxmaiah.Kocharla and <sup>2</sup>Dr. Raghava Rao, K.

<sup>1</sup>Department of Computer Science and Engineering, KL University, Vijayavada, India

<sup>2</sup>Reaserach Group Head Sensor Networks, KL University Gree Fields, Vaddeswaram, Guntur Dt, A.P, India

ARTICLE INFO

Article History:

Received 09<sup>th</sup> December, 2013  
Received in revised form  
15<sup>th</sup> January, 2014  
Accepted 18<sup>th</sup> February, 2014  
Published online 25<sup>th</sup> March, 2014

Key words:

Wireless Sensor Network,  
Energy efficiency,  
Workload balancing,  
Tree construction.

ABSTRACT

Wireless Sensor Networks play a vital role in applications like disaster management and human relief, habitat monitoring, studying the weather and eco systems, etc. Since the location of deployment of these WSNs is usually remote, the source of energy is restricted to battery. A Significant amount of work has been done by researchers in the past to achieve energy efficiency in WSNs. In this paper I propose a scheme to optimize the power utilization in a WSN. Many of the WSN applications form a tree topology for communication. In a WSN, that adopts tree topology, it is observed that the nodes at higher levels of the tree tend to consume more power when compared to those at lower levels. In our proposed workload-balance query/result routing tree construction scheme, I construct the final routing tree by keeping in mind the workload of nodes at various levels of the tree. This way, the tree construction takes place with workload at each level being evenly distributed among the nodes at that level. The proposed approach not only increases the lifetime of the network, but also utilizes the battery power optimally. Simulation results show a considerable increase in the lifetime, and effectiveness of the wireless sensor network, as a result of applying our proposed tree construction technique.

Copyright © 2014 Laxmaiah.Kocharla and Dr. Raghava Rao, K. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

INTRODUCTION

With the ever growing demand for automation systems, the sensor devices are widely used. A sensor is a tiny electronic device with a small microcontroller, energy source (usually battery), and a radio transceiver for wireless communication. Depending upon the application, each sensor device is deployed in the physical environment for monitoring certain conditions such as temperature, motion, sound, vibration, pollutants, etc. A wireless sensor device can communicate with other sensor devices within its transmission range. In real-time, as an application demands, many such wireless sensor devices collaborate to form a Wireless Sensor Network (WSN). Coal mine monitoring (Mo Li and Yunhao Liu 2009) is one of the important applications of Wireless Sensor Networks. Coal mines are usually long narrow tunnels under ground. Human miners go on foot to work in these tunnels. A disaster in a coal mine is fatal. Many of the disasters happen due to the roof and/or wall collapses. Normally it is the ceiling structure of an underground tunnel that gives way due to the pressure from the material that is above it. Wireless sensor nodes are placed on the walls of the tunnel to detect the environment inside the mine. In some underground mines, Wireless Sensor Networks are used as Structural Positioning Systems (Ning Xu *et al.*, 2004; Mo Li and Yunhao Liu 2007). These Structural

Positioning Systems help in predicting/forecasting landslides and wall/roof collapses, and locating the places where such collapses have occurred. If a proper forecast is made, I can avoid loss of property and human life. Further, if the locations where collapses have occurred are identified properly, it facilitates early and effective rescue operations which minimize the damage. The higher the precision of positioning system, greater will be the chances of performing successful rescue operations.

In some other cases, WSNs are deployed in underground mines to sense the physical conditions such as existence of hazardous gases, decreased availability of oxygen, etc. This helps in taking required precautionary measures to avoid accidents or to reduce the effect, and to provide effective rescue. In a WSN, one of the nodes is designated as the *Base-Station* (BS) that connects the WSN with the outside world. Each Sensor node is equipped with computation and communication capabilities. Sensor nodes sense data continuously and store in their built-in memory, and/or transmit to the BS. Hence, all queries from outside, first arrive at the BS, and then routed to the concerned nodes in the network. Similarly, the results of a query will be sent back to the querying entity through the BS. I assume that the BS is abundant in computational, storage and energy resources. The query/result transmission between the BS and the nodes in a WSN can be continuous or periodical. As the nodes are battery operated, and are deployed in hard-to-reach areas, battery replacement is extremely difficult or impossible.

\*Corresponding author: Laxmaiah.Kocharla, Department of Computer Science and Engineering, KL University, Vijayavada, India.

Hence, at each node, the battery power available is limited. Many applications form tree topology for communication in WSN, and as a result, each communication from the base-station to a leaf node and vice-versa will follow a specified path. For the routing tree of a WSN, the base-station is always considered as the root node.

In Figure 1, a sample communication tree structure in a WSN is shown. The intermediate nodes are always taxed with extra burden of query dissemination (Samuel Madden *et al.*, 2003) and results forwarding as well. The extra burden taken by an intermediate node is directly proportional to the size of its subtree. These nodes tend to expend more energy and die early. This may become even worse when the workload distribution among the nodes at same level is not uniform. In this paper, I present a scheme to form a routing tree in which the workload distribution among the nodes of a given level is uniform. The rest of the paper is organized as follows. I discuss the related work in Section 2, and Section 3 describes the proposed scheme with algorithms. Section 4 shows the simulation setup and experimental results. Finally, I conclude the paper in Section 5.

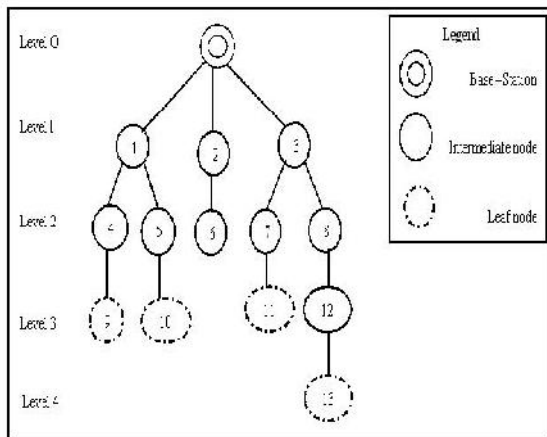


Figure 1. Tree topology for WSN

2. RELATED WORK

The work described in (M'ario Macedo 2009) shows that the average number of children of a given node can be expressed as a function of the tree level. Several functions are derived in 2D and 3D deployment environments. One important parameter it takes into account is the distance between any node and the BS in terms of number of hops (hop-count). The work in (M'ario Macedo 2009) derives approximate estimations for the average number of children a node can have in the data aggregate tree of WSN. The Energy-driven Tree Construction (ETC) algorithm (Andreou *et al.*, 2009) uses First-Heard-From (FHF) approach in its first phase, to construct a query/result routing tree for a WSN, where each node after hearing from other nodes within its transmission range, selects one among them as its parent. Each of the remaining nodes will become an Alternate Parent (AP) for that node. The list of such alternate parents for a given node is maintained as the Alternate Parent List (APL). Each node stores its APL and Child Node Lists (CNL) locally. In ETC

algorithm, the maximum number of children for a node is indicated by *branching factor* ( $\beta$ ), which is considered to be the threshold value to indicate the maximum number of children a node can have. The branching factor ( $\beta$ ) is calculated at the base-station using the formula  $d \cdot n$  where  $d$  is depth of the tree and  $n$  is number of nodes in the tree. Later, in the second phase, this  $\beta$  value is disseminated to all the nodes in the tree for load balancing. The candidate nodes for balancing are those which exceed the threshold

( $\beta$ ). The excessive workload of the candidate nodes is distributed amongst the other nodes of the tree. The candidate node instructs some of its children to look for a new parent. Then such child nodes will select the first node of their respective APLs as their new parent. This process of balancing the workload continues for all nodes of the tree.

2.1. Drawbacks in existing ETC approach

The ETC algorithm, computes the *branching factor* ( $\beta$ ), only once during the initial tree construction. During the workload balancing, each node is asked to maintain the size of its subtree as recommended by branching factor. After this balancing, some of the children get accommodated with new parents. Sometimes a child node looking for an alternate parent can choose some AP, which is at a higher depth than the self, which results in increase in the number of levels (height) of the tree. This increases the hop-count from the node (which has chosen a new parent) to the BS. Thus, change in depth causes branching factor to change dynamically in the balancing phase which is not taken care of by the ETC algorithm. Another drawback in ETC approach is that, while reorganizing the tree, if a node's branching factor exceeds the *threshold* ( $\beta$ ), some of the children will be attached to a new parent (picked from the APL) whose branching factor is less than the *threshold*. Unfortunately, this algorithm will not choose the parent with minimal branching factor; instead it just selects the first possible node that satisfies the required conditions in terms of branching factor, from the APL as its parent. As a result, in the reorganized tree, some nodes are overburdened while some are under loaded. This situation leads to early termination of some nodes due to overload, resulting in reduction of the effectiveness of WSN in terms of its lifetime and coverage. To address the above issues, I propose a scheme, in which the initial routing tree is constructed using the FHF approach (Andreou *et al.*, 2009), and then the reorganization of the tree takes place to evenly distribute the workload at a given level, among the nodes of that level. This is done by computing the average workload for each level. In this process, I detach some children from parent nodes which are overburdened, and attach the same to some other nodes, with lesser workload. This eventually results in a tree with reduced number of collisions and reduced communication path lengths.

3. PROPOSED TREE-CONSTRUCTION SCHEME

In our proposed approach, tree construction consists of following steps.

- (i) Construction of the initial tree as per the FHF approach.
- (ii) Computing the workload at each node.

- (iii) Computing the average workload at each level of the tree.
- (iv) Workload balancing (tree reorganization).

Now, I give a detailed account of our proposed tree construction and load balancing schemes.

### 3.1 Initial tree construction

The initial tree is constructed by following the first-heard-from approach as described in ETC algorithm (Andreou *et al.*, 2009) with a small amount of customization to capture the required data for load balancing. Initially, the base-station transmits a 'hello' message to the nodes within its transmission range. Each of the receiving nodes acknowledges the receipt of 'hello' message from the BS, with an 'ack' message and becomes a child node of BS. I assume that the base-station is at level 0, and these set of child nodes, identified as above, will be at level 1. Now, each node at level 1, sends 'hello' message to the neighbouring nodes. In this process, a node (which has not found its parent) may receive hello messages from more than one node. Now each node identifies the node from where it has first heard the 'hello' message and confirms it to be its parent. All other nodes from where it has received 'hello' messages will become Alternate Parents (APs). Thus, the level 2, is established for the tree. This process is repeated to establish all the levels till the complete tree is formed as shown in Figure 1. In this process, every time a child parent relationship is established between two nodes, the same is informed to the BS where the global information about the tree is maintained. I also assume that the APL and Child Node List (CNL) of each node are also available at the BS. For the sake of brevity, in this paper, I do not present the initial tree construction algorithm.

### 3.2 Computing the workload for each node

The workload of a node is the sum of its self-load and the workload due to its children (sub-tree). The base-station computes the workload of each node in the network and stores in the data structure that holds the other information about the nodes of the tree. One important assumption made is that the workload in the network due to external queries is uniformly distributed among the nodes. Further, I also assume that the self-load of each node is 1 unit of work. The workload of a node is computed as given in Procedure 3.1.

#### Procedure: 3.1: Workload (node)

This procedure computes the workload of a node and returns the same. Here, *getChildNodeList(node)* is to extract the list of child nodes of a given node.

1. Set  $WL := 1$
2. Set  $CL := getChildNodeList(node)$
3. Set  $size := |CL| // \text{size of the child list}$
4. If ( $size \neq 0$ )
5.    $size$
6.    $WL = i=1 \text{ Workload}(CL(i-1))+1 // \text{recursive call}$
5. End if
6. Return  $WL$

### 3.3 Computing the Average workload for each level

Next, the base-station computes the average workload (AWL) for each level, as given in Algorithm 3.1, and stores in the data structure (table of data) that holds the complete information about the routing tree, which is available at the BS. The table contains the information about parent, APL, CNL, workload for each node, average workload for each level, etc.

#### Algorithm: 3.1: AverageWorkload(level)

This algorithm computes the average workload of a given level. I use NODES as array of nodes at that level. The method *getNodeList(level)* gives the list of nodes at that level. AWL is the average workload, and *getWorkload(node)* returns the computed and stored value of workload of a given node.

### 3.4 Workload balancing for each level

The workload balancing for each level is performed at the BS, as per the Algorithm 3.2. All the data that is necessary to perform this load balancing is available at the BS. In our scheme, I define the *threshold* value ( ) for the maximum amount of workload for any node at a given level, as 1.5 times the average workload of that level. This *threshold* ( ) is different from the *threshold*

( $\beta$ ) used in the ETC algorithm. If the workload of a node is greater than the *threshold* value given for that level, then it is considered as the *candidate node* for load balancing. The candidate node will then refer to its CNL and select one child node with highest workload for relocation. Now this selected node need to be connected to a new parent. In order to select a new parent for this child node (identified for relocation), its APL needs to be examined. One of the nodes in the APL, will be selected to become its new parent, if it satisfies the below mentioned criteria.

- The node should have least workload among all APs in the APL.
- The sum of the existing workload of the new parent and the workload of the relocated child should be less than the *threshold* ( ) of the level to which the new parent belongs.

#### Algorithm 3.2: WorkloadBalancing(node)

This algorithm balances the workload of the *node*.

Here I use *getChildNodeList()* to extract the list of child nodes of a given node, *getAverageWorkload(level)* to extract the average workload at a given level, *getWorkload(node)* to extract the stored value of workload of a given node, *getAlternateParentList(node)* extracts the alternate parents of the given node, *getNode(workload)* extracts the node id of the node with the given workload, *addChild(m,n)* adds a node *n* to the Child Node List of node *m*, *getLevel(node)* extracts the level of a given node.

1. If ( $getWorkload(node) > 1.5 * \text{getAverageWorkload}(getLevel(node))$ )  $CNL := getChildNodeList(node)$ ;
2. For each  $j := 1$  to  $|CNL|$
3.  $WL(j) := getWorkload(CNL(j))$
4. End for

5.  $bestchild := getNode(getLargest(WL()))$
6.  $APL := getAlternateParentList(bestchild)$
7. For each  $k := 1$  to  $|APL|$
8.  $W(k) = getWorkload(APL(k))$
9. End for
10.  $bestnewparent := getNode(getLeast(W()))$
11. If  $(getWorkload(bestchild) + getWorkload(bestparent)) <$
12.  $1.5 * getAverageWorkload(getLevel(bestnewparent))$
13. Add Child ( $bestnewparent, bestchild$ )
14. End if
15. End if
16. End Procedure

Once the new parent is found, the newly established parent child relationship is captured into the table at BS by appropriate updates to the data structure. In case, if a node doesn't find a new parent satisfying both the above criteria, it may be due to selection of dependent node (of candidate node) with highest workload. As an alternative, the dependent node with second highest workload will become a candidate for relocation, and looks for the new parent as per the above mentioned criteria. This process of identifying a dependent of the candidate node, for relocation will repeat until the workload of the candidate node goes below threshold. In some rare case, I may not find a suitable new parent. In such cases the tree remains unchanged. This way I balance the tree by distributing the workload uniformly amongst the nodes at same level. Thus, our algorithm ensures uniformity in workload distribution in the network, thereby increasing efficiency and lifetime. The complete algorithm is presented in Algorithm 3.2. The base-station will apply this algorithm to all the nodes of the tree starting from the top.

**3.5 Illustration for the proposed Workload balancing technique**

Here, using an example, I illustrate the working of our proposed approach. I consider a set of 11 nodes deployed as shown in Figure 2a. First, I explain how an initial tree is constructed by FHF approach as given in ETC (Andreou *et al.*, 2009), then describe separately, how load balancing is applied to the initial tree, using ETC algorithm and our approach.

**3.5.1 Initial tree construction**

The base-station (node 0), sends a 'hello' message to all its neighbours (nodes within transmission range) i.e., nodes 1, 2, 3. Each of these child nodes reply with 'ack' message back to the parent node (node 0). This is to confirm the parent-child relationship. Further, nodes 1, 2 and 3 send 'hello' message to other nodes within their transmission range (except to their parents). This process continues till it reaches the leaf nodes of the network. The initial tree structure formed is shown in Figure 2b. Figure 2a. Example node deployment When an external entity wants to query the WSN, submits its query to the base-station. The base-station then transmits the received query to the concerned nodes. Each node processes the query on its local data and sends the results back to the base-station. The network normally works as long as every intermediate node is alive. Hence, the network lifetime is highly dependent on the battery life of its nodes. If I analyse the energy

consumption rate of every node in the network, I find that the intermediate nodes are being taxed more when compared to leaf nodes of the tree. In our example, I may note that the intermediate nodes of the tree at level 1 (node 1) and at level 2 (nodes 4, 5) not only transmit self-data, but also take the responsibility of forwarding the query results produced by their descendants. The parent-child relationship among the nodes is also depicted in Figure 2b. Hence, any intermediate node falls on the routing path between the base-station and its descendants. initial tree is as per the FHF approach (Andreou *et al.*, 2009).

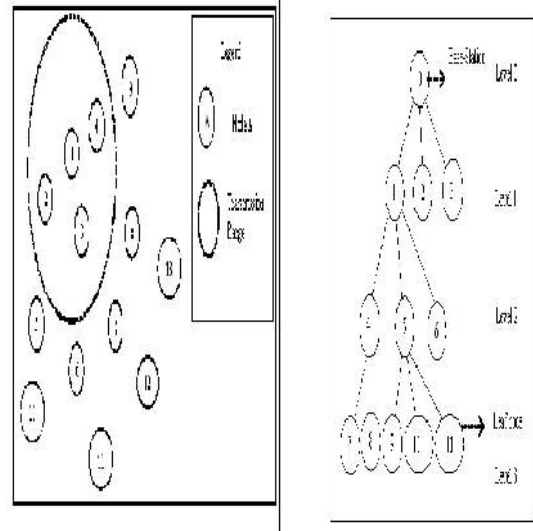


Figure 2b. Example tree structure of WSN

**3.5.2 ETC approach**

I apply ETC algorithm (Andreou *et al.*, 2009) on the initial tree of the WSN shown in Figure 2b and the resulting tree is shown in Figure 3. The Figure 4 is the redrawn tree structure for the tree shown in Figure 3.

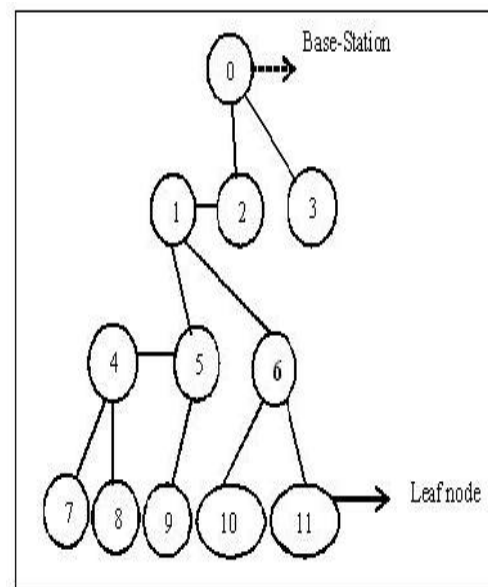


Figure 3. The balanced tree for the initial tree in Figure 2b. (Result of ETC algorithm)

In the ETC process, it is evident that the reorganization of the tree is done only to distribute the child nodes of an intermediate node whose branching factor exceeds the *threshold* ( $\beta$ ), among the other suitable nodes of the tree irrespective of the level. It is observed that the above explained reorganization doesn't consider the workload. Due to this, it may lead to a situation where some intermediate nodes are overburdened and tend to exhaust their energy faster. I consider this as one of the major drawbacks of the ETC algorithm.

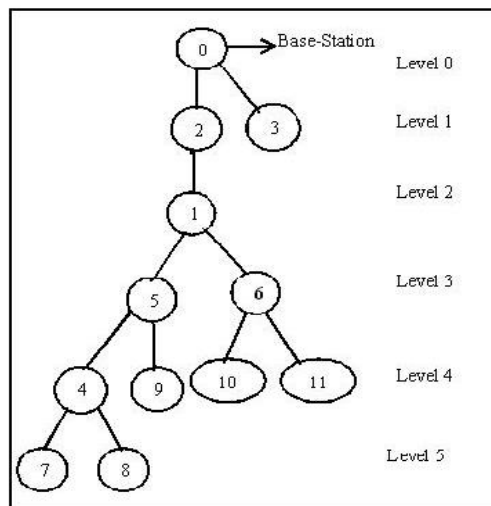


Figure 4. The redrawn tree from Figure 3

Table 1. Details of the tree structure for the network given in Figure 2b.

Node	Level No.	No. of Child	Child Node List (CNL)	Workload	AlternateParent List (APL)	Average Workload (AWL)
0	0	3	1,2,3	12	1,2,3	-
1	1	3	4,5,6	9	0,2,3,5,6	1.6
2	1	0	None	1	0,1,3,5,6	1.6
3	1	0	None	1	0,1,2,6	1.6
4	2	2	7,8	3	5,6,7,8,11	2.6
5	2	3	9,10,11	4	2,4,6,7,8,9,10,11	2.6
6	2	0	None	1	1,2,3,5,6,7,8,9,10,11	2.6
7	3	0	None	1	4,5,6,8,9,10,11	1
8	3	0	None	1	4,5,6,7,9,10,11	1
9	3	0	None	1	5,6,7,8,10,11	1
10	3	0	None	1	5,6,7,8,9	1
11	3	0	None	1	4,5,6,7,8,9	1

### 3.5.3 Our proposed workload balancing technique

Our proposed approach is designed to overcome the limitations of the ETC algorithm explained in the previous discussion. After constructing the initial tree as discussed in Section 3.5.1, now I am apply our technique to construct the final tree. I know that every node has a parent node and a list of alternate parents. All nodes maintain their CNL as well as APL. As mentioned earlier, all this information about each node of the tree is available at the base-station. For instance, at node 4, the child nodes are 7 and 8. Therefore, the workload associated with node 4 is the sum of the workloads of node 7 and 8, and the workload of node 4, itself. This is computed using the method given in Algorithm 3.1. The computed workloads and other relevant details for all the nodes of the tree in Figure 2b, are presented in Table 1. Next, the Average Workload (AWL) of each level of the tree is computed from top to bottom by

considering the workload of all the nodes at respective levels. The average workload is computed as- the sum of workloads of all the nodes divided by the no. of nodes at that level. The procedure for computing the average workload is given in algorithm 3.2. Now, for illustration, I describe the computations involved in deriving the average workload, and the *threshold* ( $\beta$ ) for each level. For level 1, the no. of nodes is 3 (node 2, 3 and 4), and the sum of the workloads of all its dependents is 5 (1+1+3). Now, the average workload for level 1 is 1.6. Similarly, for level 2 it is 2.6. Since the level 3 contains only leaf nodes I need not compute the average workload. For a given level, I define the *threshold* ( $\beta$ ) as 1.5 times the average workload of that level (which is the result of observations made during the simulations). Nodes 1, 2 and 3 at level 1, need to maintain 2.4 as the maximum workload. Similarly, nodes at level 2 are to maintain 3.9 as their maximum workload. If the workload of a particular node is more than *threshold* ( $\beta$ ), then, I am identify such node as the candidate node for applying load balancing. Now, some dependent node of the candidate node needs to be attached to a new parent. The candidate node selects the dependent with the highest workload to be the suitable child node for reorganization. This child node is to be detached from the candidate node. To reduce the workload of the candidate node, I attach this child to a new parent. In our example, I am observe that the workload of node 1 at level 1 exceeds the threshold; hence, node 1 becomes the candidate node for applying load balancing. For node 1, I find that the node 5 is the suitable

node for relocation. This node 5 has an APL with nodes-{2, 4, 6, 7, 8, 9, 10 and 11}. Now, the node 5 looks for a node from the APL, which satisfies the criteria mentioned earlier in Section 3.4. Among the nodes in APL, the node with least workload becomes the suitable node to become the new parent of node 5. Further, this addition of node 5 should not increase the workload of its new parent beyond its *threshold*. In case if the workload of the new parent exceeds the *threshold*, then select the next suitable node from the APL, and check the above conditions for this new choice as well. Repeat this until the parent is found. In our example, for node 5 it is evident that node 2 becomes the new parent. I am also observe that the new workload of node 2 is less than the *threshold*. The above technique is applied repeatedly to all the levels. The Figure 5 gives us the complete details about the reorganized tree. According to this, the intermediate nodes {1, 4 and 5} will now have uniform workload. The nodes {2, 3 and 6} which were

under-loaded before reorganization are now given some workload to maintain uniformity in workload allocation.

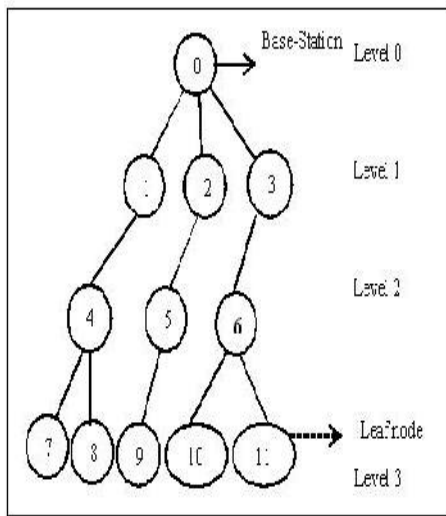


Figure 5. Final Workload balance tree

#### 4. PERFORMANCE EVALUATION

In this section, i present a brief report on the series of experiments i have conducted on our custom built simulator to assess the effectiveness of our technique and compare the same against the performance of the ETC algorithm.

##### 4.1 Simulator

i have developed a custom-based simulator, which is implemented using Java Technology. Our simulator is meant for Windows platform and is console-based. This simulator allows us to define the geographical range of the network along with the number of nodes. i can also define the transmission range of each node. Since i plan to conduct experiments to assess the power consumption of our technique, i have also designed our simulator wherein the power allotted to each node can also be defined. The deployment of nodes using our simulator can be either random or predefined. Our simulator also facilitates the generation and assignment of workload (number of queries) to the network, which also can be either random or predefined. For the simulation purpose i assume the connectivity to be constant.

##### 4.2 Experimental set-up

The simulator was run for networks of sizes 20, 50 and 70 nodes. The input to the simulator is the location of nodes specified by their x & y co-ordinates. The radio communication range of each node is set as 3m. Each sensor node is initialized with 1J of energy. The network is given some query workload so that the energy of the nodes gets depleted. The queries are randomly generated at the base-station. Each query is targeted to randomly selected nodes of the tree. Here, the term *querying* means sending a dummy packet to a specified node. Further, a node responds to its queries by sending another dummy packet back to the base-station. Thus, i would like to make it clear that this experimentation is not intended to query for the real data.

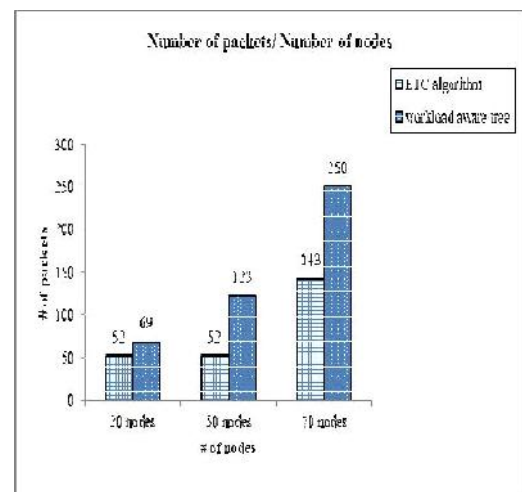
Rather, i just focus on simulating the communication load due to transmission of query/result

#### 4.3 Performance Analysis

A set of experiments were conducted to compare the performance of our workload-balance tree construction scheme against that of the technique described in ETC (Andreou *et al.*, 2009). The comparison is made based on the following metrics.

1. The Number of packets transmitted during the lifetime of the network.
2. The lifetime of the network.
3. The residual power available in the network after the network comes to halt.

i have done our experimentation with trees of 20, 50 and 70 nodes. In each case, i simulated five different topologies. The presented reading for each performance metric is the average of the five simulation runs conducted on above trees. i have computed: (i) number of packets delivered, (ii) network lifetime, and (iii) residual power in the network for each simulation. The average values of our results for each experiment are depicted in the graphs presented in this section. According to Figure 5, i observe that our workload-balance tree construction algorithm shows a good increase in the number of packets transmitted, when compared to the ETC algorithm. From the results it is evident that as the number of nodes in a network increases, the number of packets delivered in the network also increases. Figure 5. Graph depicting the total number of packets transferred till the network dies for a network of 20, 50 and 70 nodes.



The graphs in Figure 6, show the network lifetime which i consider as a significant parameter to quantify the effectiveness of the network. The network lifetime is the time elapsed between starting of the network and the moment it halts. A network is considered to have reached a halt state when not even a single node is connected to the BS. Otherwise, i say that the network is alive if at least one node in the network is active and connected to the base-station. The graph shows excellent results associated with our approach when applied on ntwork with 20, 50 and 70 nodes. A significant increase in the network



lifetime can be seen in network with 70 nodes. This shows that the workload-balance algorithm increases the longevity of WSN as the network size increases. The longer the lifetime, higher is the energy efficiency of the network. As the network spends the maximum amount of energy for packet transmission, battery power is reduced at each of these nodes after each transmission. Optimizing the battery power utilization is one of the major issues in WSN. I measure the power utilization of WSN by considering the residual power (power left) of each node after the network dies. The following graph as shown in Figure 7 shows the total residual power in the network. In the case of ETC algorithm, when the intermediate nodes die quickly, the network will come to a halt early. It is found that a huge amount of battery power of nodes in WSN is left unutilized. Our algorithm has less residual power when compared to the ETC algorithm. More residual power in case of ETC is due to early termination of intermediate nodes. Next, the Figure 8 shows the number of nodes alive in the network after executing 40 queries in three networks with 20, 50 and 70 nodes. The graph clearly shows the difference between ETC and our algorithm. The number of nodes alive after executing 40

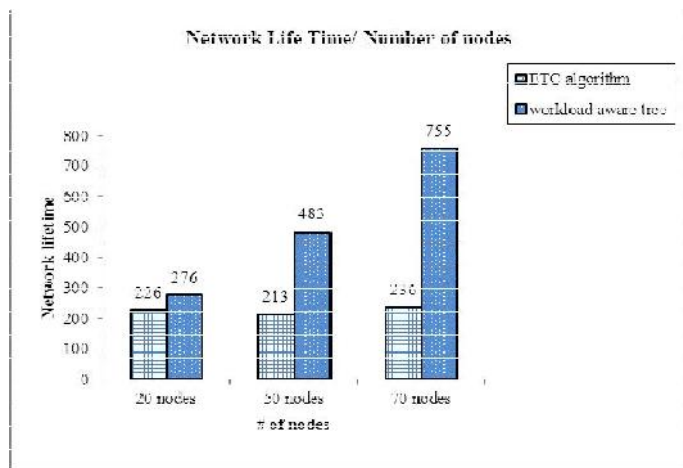


Figure 6. Graph depicting the network lifetime till the network dies for a network of 20, 50 and 70 nodes

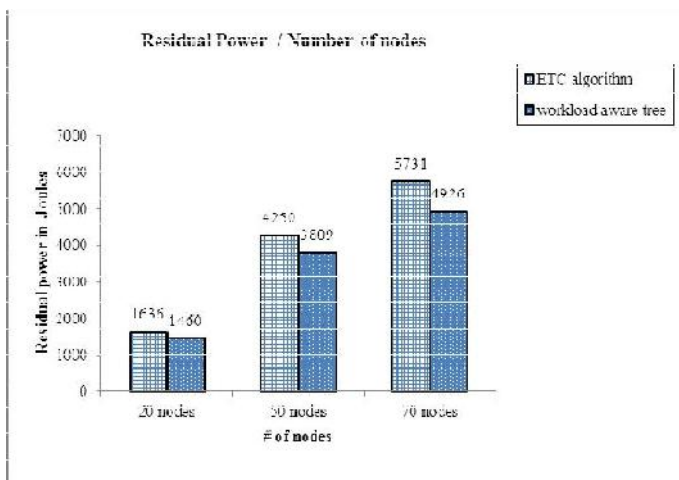


Figure 7. Graph depicting the total residual power of all nodes in the network for a fixed 40 packets delivered in the network of 20, 50 and 70 nodes

Queries is less in case of ETC because of the quick termination

of intermediate nodes and its children. I see that more number of nodes are alive in case of our workload-balance algorithm, which implies increase in lifetime of the WSN, proving that our algorithm is more effective than the ETC algorithm. From the above experimental results, it is evident that I have successfully achieved the goal of optimal power utilization at intermediate nodes, using our workload-balance algorithm. I have showed that how intermediate nodes can live for a longer time when reorganization of the initial tree is done by considering the workload at different levels. Hence, I conclude that workload-balance algorithm is successful in optimizing the energy consumption, and increasing effectiveness of the network through its load balancing strategy. Various comparisons have been provided to show how our scheme works better than ETC algorithm in terms of lifetime, throughput, residual power and power utilization.

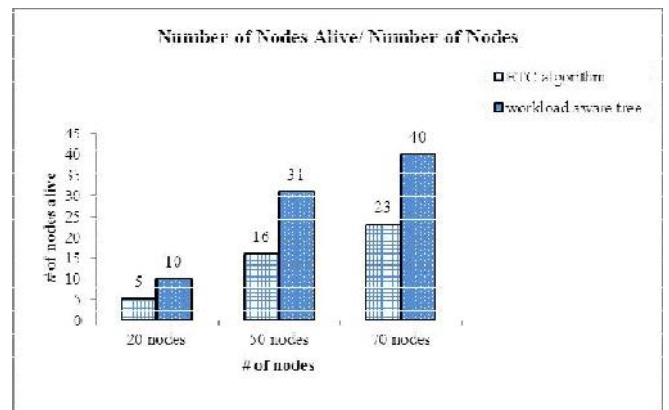


Figure 8. Graph depicting the total number of nodes alive for 40 packets delivered in the network of 20, 50 and 70 nodes

5. Conclusions

In this paper, I have proposed a workload-balance tree construction approach in wireless sensor networks, which uses an efficient strategy to construct a query routing tree. In our algorithm, I compute workload of every node and reorganize the tree by considering the average workload of a level and distributing the workload uniformly among intermediate nodes at same level. I compared the effectiveness of our scheme against the ETC approach. It is found that our technique works better than ETC approach in terms of parameters like throughput, lifetime and residual power. A network with higher number of nodes shows a better performance. I further plan to conduct more intensive experiments to assess the effectiveness of our approach, when real query/result content is considered for transmission in a WSN.

REFERENCES

Andreou P., A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras, 2009. ETC: Energy-driven Tree Construction in Wireless Sensor Networks, Tenth International Conference on Mobile Data Management: Systems, Services and Middleware.  
 International Journal on applications of graph theory in wireless ad hoc networks and sensor networks (GRAPH-HOC) Vol.4, No.1, March 2012.

- Mário Macedo, 2009. Are There So Many Sons per Node in a Wireless Sensor Network Data Aggregation Tree?, IEEE Communications Letters, Vol. 13, No. 4.
- Mo Li, Yunhao Liu, 2007. Underground structure monitoring with wireless sensor networks, Proceedings of the 6th international conference on Information processing in sensor networks, Cambridge, Massachusetts, USA
- Mo Li, Yunhao Liu, 2009. Underground coal mine monitoring with wireless sensor networks, ACM Transactions on Sensor Networks (TOSN), v.5 n.2, p.1-29.
- Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, Deborah Estrin, 2004. A wireless sensor network For structural monitoring, Proceedings of the 2nd international conference on Embedded networked sensor systems, Baltimore, MD, USA.
- Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, 2003. The design of an acquisitional query processor for sensor networks, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California.

\*\*\*\*\*