# Backbone Tree based Data Gathering protocol for Wireless Sensor Networks

[1] Yu Niu, [2] Jin Wang, [*3] Sungyoung Lee, [4] Young-Koo Lee
[1]*Student, niuyu@oslab.khu.ac.kr*
[2]*Post Doctor, wangjin@oslab.khu.ac.kr*
[*3 Corresponding Author] *Professor, sylee@oslab.khu.ac.kr*
[4]*Professor, yklee@oslab.khu.ac.kr*
*Ubiquitous Computing Laboratory, Dept. of Computer Engineering,*
*Kyung Hee University, Seocheon-dong, Giheung-Gu, Yongin-Si, 449-701, Korea,*

## *Abstract*

*In this paper, we proposed a novel Backbone tree topology for the data gathering in Wireless Sensor Networks. By integrating the ideas of both linear chain and tree topology, the new Backbone tree topology obtains quicker network responding speed and longer lifetime results.*

**Keywords***: Backbone tree, data gathering, wireless sensor networks*

## 1. Introduction

As one of the generic type of applications for sensor networks, the data gathering [1] periodically collects data from the sensing field and reports to the sink node ( or Base Station) which is not energy or computing power limited. At the sink the collected information can be further processed for end-user queries. The key challenge in such data gathering is conserving the sensor energies, so as to maximize their lifetime. There are three classes of methods to collect data. One is direct transmission; the second method is clustering [2-6]. The third one is multi-hop routing [7]; all nodes transmit their data locally along the given routing plan. The routing method has good scalability and load balancing and it also can be used in the clustered network situation [8]. The algorithm proposed in this paper belongs to the routing method.

Based on the periodical and low bit rate characteristics of data gathering application in wireless sensor networks, this paper proposes a novel Backbone tree algorithm. The algorithm constructs a chain-like backbone in the spanning tree. With the backbone, the head duty of long distance transmission can be rotated in each round without reconstructing the whole topology. Meanwhile, the higher connection degree of bone node makes the data can be parallel transmitted, which greatly decreases the network procession latency.

## 2. The Backbone Tree

### 2.1. Motivation and Tree Description

The spanning tree for constructing the whole network routing plan is a proper choice, because it can not only connect all member nodes but self grow along the physical shape of the field. However, the root node in a tree is prone to resource overburden because of the long distance transmission to the base station (Figure 1(a)). Once the root node fails, the whole topology will collapse. To avoid the root-node-die-early, the system has to change the root node and reconstruct the tree frequently. This will increase the constructing cost. Based on this, Lindesey and Raghavendra [9] proposed the "Power-Efficient Gathering in Sensor Information Systems" (PEGASIS), in which all nodes are constructed into a linear chain. Data flows along the chain to arrive at the head node. The head node is changed in each turn (Figure 1(c)). The system does not need to reconstruct the whole topology to rotate the head duty, thus reducing the amount of energy spent per round. However, the data collecting latency of it is significant. In the chain topology, all nodes can only transmit their data serially. When the node

number is not small, the time for waiting the data being transmitted from the ends of chain to the head may be very long.

To solve the problems of the above two methods, a Backbone Tree for the data gathering application in sensor networks is proposed in this paper. The Backbone tree is a special kind of spanning tree. There are two kinds of nodes in the tree: bone nodes which connect to each other to form a chain-like backbone, and child nodes which connect themselves to some bone node (Figure 1(b)). The bone nodes are in charge of collecting the raw data from all their children and transporting the packed message along the backbone chain to the head which is one of the bone nodes. The head duty shifts on the backbone by turns.

The advantage of constructing a backbone in a tree is that the network can rotate the head duty without reconstructing the whole tree every round. On the other hand, the backbone tree shorten the length of 'chain' by allowing bone nodes to recruit children, on which the nodes can parallel transmit data so as ease the long latency problem.

Because the bone nodes take more responsibility than the child nodes, there are some energy threshold requirements for the bone node candidate during the backbone construction. During the backbone growing, the bone nodes take the greedy rule to recruit as many as possible children in its searching area. When the backbone stops growing and if there are still some nodes that are not connected to the tree, these nodes try to connect themselves to the nearest node which is already connected to the backbone. In the tree, each bone node is only allowed to have at most two layers of child nodes, which is to guarantee the dominating role of backbone and shorten the network latency. In each turn, after the head collects all information and sends them out, the head duty will shift to the next node on the backbone chain.

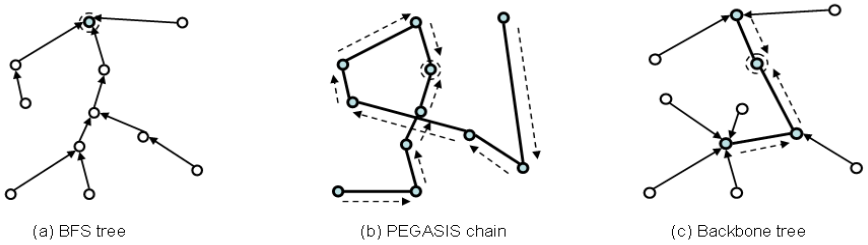


**Figure 1.** Demonstration of three topologies. (Dashed circled node means the current head duty location).

## 2.2. Construction of Backbone Tree

Some preconditions are stated here:

1) Each node in the network has a unique identification number ID.

2) Each node can adjust its transmission range power level.

3) Each node keeps a neighbor table $T_{nbr}$ to store the IDs of all its neighboring nodes located within the transmission range $r$. The content of this table is obtained from the BS message or constructed according to the nodes mutually informing messages. Once the $T_{nbr}$ of each node is constructed, unless some nodes die, the content will not change.

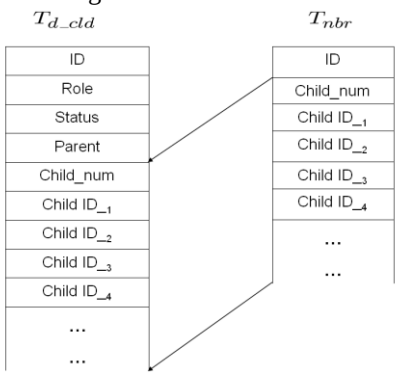


**Figure 2.** Structure of neighbor table $T_{nbr}$ and dynamic child table $T_{d\_cld}$.

4) Each node also keeps a dynamic child table $T_{d\_cld}$. Each time, before constructing the new Backbone tree, the node loads the content of neighbor table $T_{nbr}$ into the child table $T_{d\_cld}$. During the tree constructing procedure, the content of table $T_{d\_cld}$ is updated from time to time, until the node is connected to the tree. Figure 2 shows the structures of two tables and their loading relationship.
5) Each node records the energy threshold of being a bone node, which is refreshed termly by the BS broadcasting.

The following are the construction steps of the Backbone tree:

**Initialization:** All node load new child table $T_{d\_cld}$ from their neighboring table $T_{nbr}$, and reset their 'Status' field as *Unconnected*, 'Role' as 1 which means normal child node.

**Step 1** Randomly select one energy qualified and unconnected node as the initial root and change its 'Role' field as 1 which means the root node. The root also works as the first bone node ($B_0$) of the backbone.

**Step 2** For each new determined bone node, check if there are entries in its child table $T_{d\_cld}$. If it has, then go to Step 3; if hasn't, wait until the time out then go to Step 4.

**Step 3** For each bone node $B_i$ that has child, determine the next bone node $B_{i+1}$. If current bone node is root node, then determine two bone nodes from $B_0$. Return to Step 2.

**Step 4** For all unconnected nodes, broadcast connecting request to all nodes in its neighboring table $T_{nbr}$. If the first acknowledgement message comes from node $j$, change its 'Status' field as *Connected*, 'Parent' filed as node $j$'s ID and 'Role' field as 2 which means it is the second level child node. For any node who receives this connecting request, if its residual energy higher than the current threshold value, send acknowledgement message to the request node.

**Step 5** If the unconnected node in Step 4 didn't receive any acknowledgement message, then enlarge its transmission radius meanwhile decrease the threshold requirement and rebroadcast the request message. This time, only the node whose 'Role' field value is less than 2 reacts to this request. The receiving node who meets the both new threshold and 'Role' requirement sends back the acknowledgement message.

Repeat Step 5, until all nodes' status become *Connected*. Then terminate the algorithm.

To show how the current bone node $B_i$ decides the next bone node $B_{i+1}$ in Step 3. Here lists the detail actions of the bone nodes and the related normal nodes respectively.

For the current bone node $B_i$

1) $B_i$ receives the appointment message and changes its 'Role' field as 0 which means normal bone node, 'Parent' field as the sending node ID;

2) Assuming $B_i$ has $m_i$ entries in its current $T_{d\_cld}$. $B_i$ sends the grandchildren number inquiry request to all these $m_i$ children.

3) $B_i$ waits for all the $m_i$ children's reply messages within the valid waiting time (VWT) and picks up the one who has the most grandchildren number.

4) $B_i$ sends the appointment message to the picked up child and affirms it as the next bone node $B_{i+1}$.

For each node in the bone node $B_i$'s $T_{d\_cld}$ table:

1) After receiving the grandchildren number inquiry from $B_i$, mark its 'Parent' field as $B_i$.

2) Broadcast the updating message to all nodes in its $T_{d\_cld}$ table, which claims it has been recruited and asks all the receivers to delete its entry in their $T_{d\_cld}$ tables.

3) Meanwhile, wait for the updating messages from other child nodes and delete the corresponding entries in its $T_{d\_cld}$ until the valid waiting time (VWT) ends.

4) After the waiting time ends, clean up its $T_{d\_cld}$ and count the new child number. If the node's residual energy is higher than the energy threshold value, then send the reply message to its parent node.

5) Change the 'Status' field as *Connected*. Unless receiving the bone node appointment message or latter connecting request in Step 4 and 5, the node no longer receives any updating or inquiry requests.

## 2.3. Time and Message exchange Complexity

In above algorithm, the Step 4 and 5 mean to connect the residual nodes to the tree. To connect one isolated node, it may request all the other nodes in network, which costs $O(n)$ time. However, the unconnected node number is constant, so the total processing time to connect all the residual nodes is at worst $O(n)$. In the algorithm, Step 3 is the most time consuming part, in which each node will be

inquired at most once. To reply the inquiry, it has to communicate with all its neighbors. So the processing time for arbitrating the bone nodes is at most $O(n)$ for each node and $O(n^2)$ for the whole network.

Next, the energy cost and message numbers used in Step 3 are analyzed. For each child of $B_i$, to make it connected, there cost one receiving from its parent node, one reply to its parent, one broadcast to all nodes in its $T_{d\_cld}$ table, and at most mi times receiving from the updating messages. Among them, for the chosen new bone node, besides the above mentioned cost, it also needs one receiving of the appointment message; one new appointment message sending; one inquiry request broadcast to all nodes in $T_{d\_cld}$; and $m_i$ times reply message receiving from each child. For the nodes that are not the children of $B_i$ but receive the updating messages from the children of $B_i$, they just delete the corresponding entries in their $T_{d\_cld}$ tables, but do not reply anything.

During this procedure, the system uses four types of control messages: (1) Bone node appointment message, (2) Grandchildren number inquiry message, (3) Table updating message, and (4) Child number reply message. Among these, (1) and (4) are uni-cast that have specific destination ID; (2) and (3) can be whole transmission area broadcasts or multi-casts with multiple destination IDs. The following describes a general network message format (Figure 3).

| Type | Subtype | Source _ID | Dest. _ID | TTL | Type related data | Data_ length | ... ... |
|---|---|---|---|---|---|---|---|

**Figure 3.** The format of system general control messages.

The 'Type' designates the property of message which is either control message or data message. Only if it is control message, the following 'Subtype' and 'Type related data' fields have meanings. The 'Subtype' specifies the kind of control message which could be one of the above four or the connecting request or acknowledgement message used in Step 4 and 5. And the 'Type related data' field content is related to the 'Subtype' field value. Whatever the control message type is, the packet length is short and small amount of energy is needed. If it is data message with long data length, more energy is necessary and the data information is attached in the latter part of message structure.

### 2.4. Reconstruction and Threshold Refresh

As the data collecting topology of the network, the energy consumption on each Backbone tree node is not equilibrious. To balance the network energy consumption and extend the lifetime, the tree needs to be reconstructed in some frequency. In addition, before each reconstruction, the energy threshold value '*ths*' for selecting the new generation of bone node should also be dynamically updated to suit the current network situation.

In each round of data gathering, bone nodes piggyback their residual energy information in the data packets and transport them to the head. As a part of the final head packet, the energy information of all bone nodes is also reported to the base station. Since the bone nodes usually consume more energy than the normal ones, based on that base station could estimate the possible lowest current energy level in the network. By setting the new energy threshold value higher than that in the next construction, the system could avoid always choosing the same group of nodes as the bone nodes. This method effectively balances the network node energy consumption, which is also proved in our later simulation results.

Therefore, before each reconstruction, base station calculates the new bone node energy threshold value '*ths*' and broadcasts it to the whole network. Once nodes receive the new threshold value '*ths*', they enter the new constructing stage. The reconstruction could be launched after all the current bone nodes having taken the head duty once. In this case the reconstruction period is dependent on the chain length of the tree. The reconstruction also could be activated by the Base Station, and then the reconstruction period is uncertain.

## 3. Implementation

This part gives out the implementation pseudo-codes of Backbone tree on individual node, which demonstrate the different operations and message transmission on either root node (Algorithm 1) or normal node (Algorithm 2).

**Algorithm 1 Root ()**

```
1.  if (Receive Threshold_refresh message) then
2.      Root_Competition ()
3.      if (root) then
4.          S.root Ã 1
5.          Broadcast Grandchild_number_inquiry (RootID)
6.          repeat
7.              Receive Grandchil_report () message
8.          until Time out
9.          Choose two most popular child nodes a and b
10.         Bonenode_assignment (RootID, a)
11.         Bonenode_assignment (RootID, b)
12.     else
13.         Node ()
14. end if
```

**Algorithm 2 Node ()**

```
1.  if (Receive Bonenode_assignment (BnodeID, NodeID)) then
2.      S.parent Ã BnodeID
3.      S.bone Ã 1
4.      Broadcast Grandchild_number_inquiry (NodeID)
5.      repeat
6.          Receive Grandchild_report (m_i, NodeID)
7.      until Time out
8.      Choose the most popular child nodes a
9.      Bonenode_assignment (NodeID, a)
10. end if
11. if (Receive Grandchild_number_inquiry (BnodeID)) then
12.     S.parent Ã BnodeID
13.     S.connect Ã 1
14.     Collect child node number m
15.     Send back Grandchild_report (m, BnodeID)
16. end if
17. while (Time Out) AND (S.connect == 0) do
18.     Broadcast Connecting_request (NodeID, ths, r)
19.     if (Receive Ack (ID, NodeID)) then
20.         S.parent Ã ID
```

```
21.          S.connect à 2
22.      end if
23.    ths à ths-△ths
24.    r  à r+△r
25. end while
26. if (Receive Connecting_request (ID, ths, r )) then
27.      if (S.erg > ths) AND (S.connect < 2) then
28.          Send back Ack (NodeID, ID)
29.      end if
30. end if
```

## 4. Simulation Results

In this section, we check the performance of the Backbone tree from the transmission latency (system reacting speed) and the network lifetime aspects, respectively. First, an example is described that consists of a Backbone tree topology in a general sensor network in which 100 nodes are randomly deployed in a $50 \times 50\ m^2$ sized field, and the initial transmission radius uses $r=16$ (Figure 4).

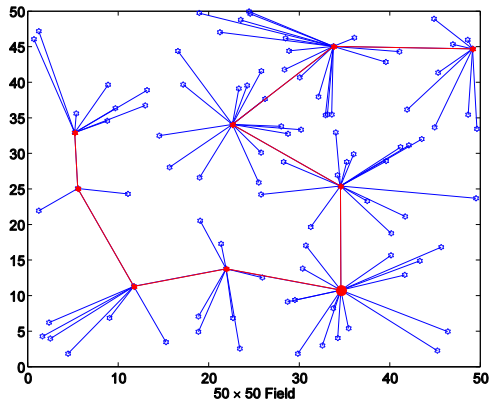


**Figure 4.** Backbone tree in a $50 \times 50\ m^2$ network field *(n=100, r=16)*.

### 4.1. Transmission Latency

During the data gathering, data information is transmitted in a multi-hop way. Except for the propagation delays in air, each relay node also need time to receive, repack and transmit the data package. All these cause the latency in the whole gathering procedure. Therefore, network hops are used to measure the transmission latency.

The transmission latency should be measured by the worst case that could happen during the data gathering. In PEGASIS, all member nodes are constructed into one chain. For chain-liked topology, all the receiving and transmitting is serial on the chain. The best case is when the head duty flows to the midpoint of chain that the latency is around the half length of chain; the worst case is when the head duty flows to one end of the chain that the latency would be the whole length of the chain. Also, the latency on a linear chain is not related with the transmission radius. So the transmission latency of PEGASIS is the number of node in the network and will not change until some node dies early. In the Backbone tree, the bone nodes at most have two layers of child nodes, so the longest latency of the Backbone tree topology is the length of backbone chain plus one. For Breadth First Spanning (BFS) tree, the latency is the layer number of the deepest leaf nodes in the tree. For both Backbone and BFS tree, the final

tree topology is influenced by the transmission radius.

Figure 5 shows the transmission latency comparison of three algorithms under different transmission radii. All the data in this figure are averaged from 100 times network distribution. Compare to the other two, the latency value of PEGASIS is very high, which always equals the node number. BFS tree has the shortest latency time because of its breadth first constructing rule. The Backbone tree's latency is a little longer than BFS tree but still much shorter than PEGASIS.

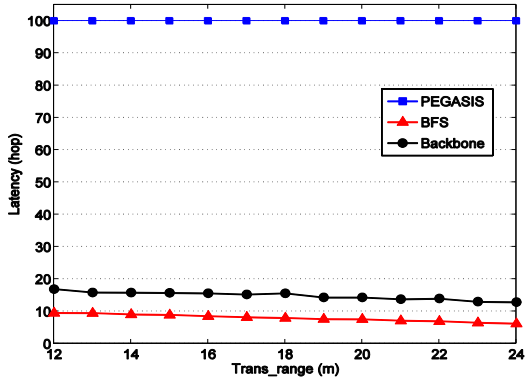


**Figure 5.** Latency comparison of three algorithms. ($n=100$ in $100 \times 100$ $m^2$ field).

## 4.2. Network Lifetime

In this part, we compare the lifetime of the three algorithms' topology. The working round number is used to measure the network lifetime. Energy consumption of sensor nodes is calculated using the common first order radio model [10, 11] in wireless sensor network. The parameters and network settings are listed in Table 1.

**Table 1.** Simulation environment

| Type | Parameter | Value |
|---|---|---|
| Network | Network grid area [x,y] | From (0,0) to (100,100) $m^2$ |
| | Base Station | (200,200) |
| | Initial energy | 1 *J*/battery |
| | Head file length | 25 *byte* |
| | Message length | 2000 *bits* |

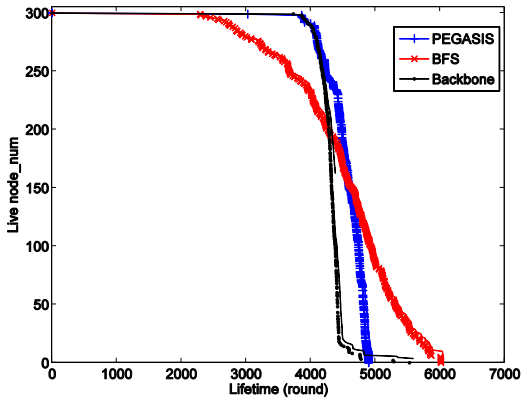

**Figure 6.** Lifetime of all nodes in three algorithms. (*n=300, r=16*).

Figure 6 shows the working round numbers of a network lifetime performance and the energy consuming trend of the three algorithms. In the figure, the later the first node dies, the better load balancing the algorithm has. And, the later the last node dies, the better 'survival ability' the algorithm has. Backbone tree shows good load and energy consuming balancing. From the first to the early 30% of died node, their lifetimes in Backbone tree are much longer than in BFS tree. Also the early 16.7% of died nodes' lifetimes in Backbone tree are longer or similar to PEGASIS. In Backbone tree, after the first node dies, the whole network also dies rather quickly. This property is preferred by some applications which require the whole network work cooperatively. On the other hand, although the first node dies early, BFS tree shows longer whole network lifetime than the other two. After a portion of nodes die, the remainder nodes can continue work for a rather long time with relatively high energy levels. Choosing what kind of property depends on the application requirement. From the above two criteria, PEGASIS achieve a relative best performance among the three. It has later died first node than BFS tree and longer whole network lifetime than Backbone tree. However, its long latency is fatal and restricts its applicability.

After knowing the energy depletion trends of the three algorithms, we discuss their lifetime performance under different node densities (Figure 7) and transmission radii (Figure 8). Here we mainly focus on the situation before the first node dies. All the data in these two figures are averaged from 100 times network distribution.

In Figure 7, PEGASIS and Backbone tree's lifetime are shorter in sparse network. This is because a sparse network usually requires longer transmission distance between two nodes on the chain. The node density has not much influence on BFS tree. In Figure 8, for Backbone tree, when the transmission radius $r \leqslant 24$, the lifetime keeps almost similar level. But after $r>24$, the lifetime decrease with the transmission radius's increase. This is because in backbone tree, with the increase of transmission radius, the neighboring nodes also increased. Increased neighboring nodes in a sparse network (like in Figure 7) helps to rotate and distribute the bone node duty to more nodes so as to prolong the whole network lifetime. However, continue adding neighboring nodes (like enlarging transmission radius in Figure 8) will cause the bone nodes to become overburdened in one reconstruction period. Because the backbone tree is not reconstructed every round like BFS tree, the bone node may die before the next reconstruction happened. So the large transmission radius does not influence BFS tree as much as Backbone tree. Combine the above discussion and the results in Figure 7 and Figure 8, we conjecture there exists a local extreme for Backbone tree's first node die lifetime under some certain neighboring number nodes (in Figure 7 and 8's network setting, we calculate the maximum neighboring node is around 50). And this extreme gives a clue to the network's node saturation for obtaining the Backbone tree's best performance.

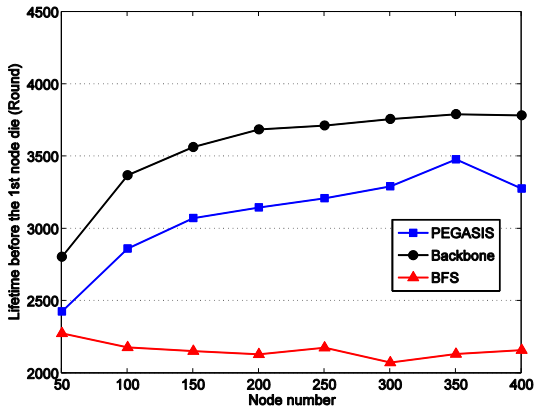


**Figure 7.** Lifetime comparison of three algorithms under different node density before the first node die (*r=17*).

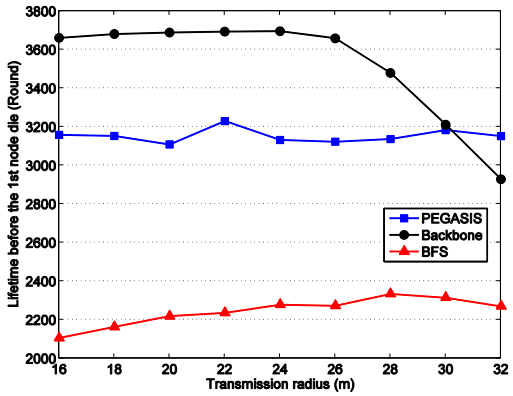


**Figure 8.** Lifetime comparison of three algorithms under different transmission radius before the first node die (*n=200*).

## 6. Conclusion

In this paper, a novel Backbone tree idea and its constructing algorithm was proposed. The Backbone tree topology shows good performance in data gathering application. It shortened the chain length by allowing the chain node to have child nodes, which greatly decreases the processing latency and helps to increase the network responding speed. Meanwhile the head duty rotating on the chain-like Backbone decreases the reconstruction frequency, so as to decrease the node energy consumption per round. Integrating with the bone node energy threshold control scheme, Backbone tree achieved good load balancing property.

## 7. Acknowledgment

This research was supported by Basic Science Research Program Through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0016042).

## 8. References

[1] K.Ramanan, E.Baburaj "Data Gathering Algorithms for Wireless Sensor Networks: A Survey", International Journal of Ad hoc, Sensor & Ubiquitous Computing, IJASUC, vol.1, no.4, pp. 102-114, 2010

[2] Ossama Younis, Sonia Fahmy, "HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," IEEE Transactions on Mobile Computing, IEEE, vol. 3, no. 4, pp. 366–379, Oct 2004.

[3] Fabian Kuhn, Thomas Moscibroda, Roger Wattenhofer, "Initializing newly deployed ad hoc and sensor networks," In Proceedings of the 10th annual international conference on Mobile computing and networking (MOBICOM), pp. 260–274, 2004.

[4] Alan D. Amis, Ravi Prakash, Thai.H.P. Vuong, Dung T. Huynh, "Max-Min D-cluster formation in wireless ad hoc networks," In Proceedings IEEE Conference on Computer Communications INFOCOM 2000, vol. 1, pp. 32-41, 2000.

[5] HaoWen Chan, Adrian Perrig, "ACE: An emergent algorithm for highly uniform cluster formation," In Proceedings First European Workshop Sensor Networks (EWSN), pp. 154–171, 2004.

[6] Mainak Chatterjee, Sajal K. Das, Damla Turgut, "WCA: A weighted clustering algorithm for mobile ad hoc networks," Cluster Computing. Springer, vol. 5, no. 2, pp. 193–204, 2002.

[7] Ming Zhang, "An Novel Energy Balanced Dynamic Routing Protocol Based on Probability in Wireless Sensor Networks", Journal of Convergence Information Technology, AICIT, vol. 6, no. 3. pp.10-17, 2011
[8] Qi Yang, Yuxiang Zhuang, Hui Li, "An Multi-hop Cluster Based Routing Protocol for Wireless Sensor Networks" Journal of Convergence Information Technology, AICIT, vol. 6, no. 3, pp. 318-325, 2011.
[9] Stephanie Lindsey, Cauligi S. Raghavendra, "PEGASIS: Power-efficient gathering in sensor information systems," In Proceedings of Aerospace Conference, IEEE, pp. 3–1125–3–1130, 2002.
[10] Wendi Heinzelman, Anantha Chandrakasan, Hari Balakrishnan, "Energy-efficient communication protocol for wireless sensor networks" In Proceedings of the 33th Hawaii International Conference on System Sciences, IEEE, pp. 10, 2000.
[11] Wendi Heinzelman, Anantha Chandrakasan, Hari Balakrishnan, "An application-specific protocol architecture for wireless micro-sensor networks," IEEE Transactions on wireless communications, IEEE, vol. 1, no. 4, pp. 660–670, 2002.