

FEED: Fault Tolerant, Energy Efficient, Distributed Clustering for WSN

Mohammad Mehrani¹, Jamshid Shanbehzadeh², Abdolhossein Sarrafzadeh³, Seyed Javad Mirabedini⁴, Chris Manford⁵

¹Young Researchers Club, Islamic Azad University, Dezful Branch, Iran

²Department of computer Tarbiat Moallem University, Tehran, Iran

³Department of Computing, UNITEC, Auckland, New Zealand

⁴Islamic Azad University, Tehran Markaz Branch, Tehran, Iran

⁵Department of Computing, UNITEC, Auckland, New Zealand

mehrani@iaud.ac.ir, jamshid@tmu.ac.ir, hsarrafzadeh@unitec.ac.nz, jvd2205@yahoo.com, cmanford@unitec.ac.nz

Abstract— Increasing network lifetime, scalability and load balancing are important factors for wireless sensor networks. Clustering is a useful technique through which we can effect these factors. In this paper, we propose a new method of clustering which prolongs network lifetime by using energy, density and centrality factors and also the distances between nodes for making clusters. We assume a supervisor node for every cluster head which is to be its replacement when the cluster head fails. This property causes an increase in network lifetime and also helps the network to be fault tolerant. Simulation results demonstrate that using the proposed method offers significant improvement in clustering especially in network lifetime in comparison with the LEACH and HEED methods.

Keywords— wireless sensor network, cluster head, pivot cluster head, supervisor node, lifetime

I. INTRODUCTION

Wireless Sensor Networks consist of a huge number of sensor nodes, with limitations like their energy resources, dispersed in a region. All network nodes have wireless communications and send information about their region to a base station either directly or via other nodes. Energy efficiency is one of the most important factors in wireless sensor networks. Routing plays a critical role in energy efficiency, so a suitable routing protocol should be selected for this purpose. Clustering is an approach by which, all network nodes are organized into clusters and communicate with a cluster head which forwards received data to base station. Clustering aims at achieving an energy efficient routing protocol. In Clustering every cluster has a cluster head. In every round, some nodes are selected to act as cluster heads and regular nodes try to join to the nearest cluster.

In some clustering algorithms, cluster heads are selected randomly in every round [1]. Some others are sensitive to energy, distance, etc, like the algorithms mentioned in [2] and [3]. Moreover, the algorithm mentioned in [3] has tried to use other techniques including fuzzy logic to form clusters.

Selecting suitable cluster heads can lead to a decrease in overall energy consumption and also an increase in network lifetime. In this article we propose an energy efficient clustering method (FEED) which selects suitable cluster heads by paying attention to energy, density, centrality and the distance between nodes for making clusters. Moreover, in this approach every cluster head has a supervisor node. This property leads to an increase in network life time and also a fault tolerant clustered network.

We compare our approach with two well known approaches LEACH and HEED. Simulation results demonstrate that the proposed algorithm (FEED) performs much better than LEACH and HEED in some clustering issues especially in network life time. In Section 2, we review related works. Then we describe our fault tolerant, energy efficient, distributed clustering algorithm in Section 3. Section 4 reveals experimental results of simulations in comparison with other similar approaches. We conclude in Section 5, with a summary of the contributions of the proposed approach.

II. RELATED WORKS

In this section we briefly review related works.

A. Leach

LEACH (Low Power Adaptive Clustering Hierarchy) is an algorithm used for clustering in WSN. In this algorithm there is a probability formula for every node to be a cluster head in every round. At the beginning of every round every node chooses a random number between 0 and 1. There is a threshold number $T(n)$ which varies in every round. The node can be a cluster head in the current round if the random number chosen by it is less than $T(n)$. If a node decides to be a cluster head in a certain round, it informs other nodes about this fact by broadcasting a message. Then every regular node joins the nearest cluster. The LEACH probability formula is:

$$T(n) = p(n) / (1 - p(n) * (r \bmod (1/p(n)))) \quad \forall n \in G \quad (1)$$

Where n is the number of network nodes, r is the number of the round, G is the set of nodes that haven't been cluster head

in the last $1/p$ rounds and p is the desired percentage of cluster heads which equals to 0.05. This formula lets every node to have the chance of being a cluster head once in every $1/p$ rounds. LEACH enhances network lifetime and energy consumption compared with the Direct algorithm. One shortcoming of LEACH is that it doesn't consider the energy factor in selecting cluster heads. Thus chosen cluster heads aren't always suitable for the network.

B. HEED

In HEED (A Hybrid, Energy-Efficient, Distributed Clustering) several iterations are needed to choose a cluster head. The time slice for each iteration should be long enough for a node to receive all sent messages from its neighbor nodes. All nodes assume the initial probability to become a cluster head as follows:

$$CH_{prob} = C_{prob} \times \frac{E_{residual}}{E_{max}} \quad (2)$$

At the beginning of each round all uncovered nodes decide to be cluster heads with probability CH_{prob} . If a node decides to be a cluster head, it broadcasts a message to other nodes. In this message if CH_{prob} is less than 1, the node introduces itself as a tentative cluster head. If CH_{prob} is equal to or greater than 1, the node introduces itself as a final cluster head. At the end of each iteration all nodes double their CH_{prob} . A node assumes itself covered if it is covered by at least one tentative or final cluster head. If at the end of a round, a certain node isn't covered by any tentative or final cluster head, it reveals itself as a final cluster head. Then each node joins a cluster which generates the lowest cost for it.

C. PEGASIS

In the PEGASIS (Power Efficient Gathering in Sensor Information System) algorithm, starting at the farthest node, a chain is formed greedily so that data from each node arrives at the leader node and the leader aggregates the data for transmission to the base station. The leader in PEGASIS is the top confluent node that aggregates multiple chains' data. The leader passes a token to the end node of a branch path to aggregate data along that path. Other confluent nodes exist in different levels of a hierarchy. Only confluent nodes at a level will be active in the next level. We can regard the group of a confluent node along with its sub-chains before it as a special type of cluster in which multi-hop intra-cluster routing is performed and hence we regard confluent nodes as cluster heads such that inter-cluster routing is performed [4].

D. TEEN

TEEN (Threshold sensitive Energy Efficient sensor Network protocol) [5] is a hierarchical clustering protocol which uses three different kinds of nodes: simple nodes, 1st level cluster heads and 2nd level cluster heads. Simple nodes try to send sensed data to the 1st level cluster heads. Then the

1st level cluster heads pass the gathered data to the 2nd level cluster heads and the 2nd level cluster heads send data to base station. This algorithm uses a data-centric mechanism and passes sensed data from different levels to reach the base station.

III. THE PROPOSED ALGORITHM (FEED)

In FEED (Fault tolerant, Energy Efficient, Distributed Clustering) all network nodes are divided into clusters such that at the end of the algorithm there are some cluster heads (CH), some pivot cluster heads (PCH), and some supervisor nodes (SN). A CH node is a regular cluster head which is the head of its cluster. A PCH node is a pivot cluster head with additional capabilities beyond a CH node. All the PCH nodes together cover a big area of the entire network and are also the best nodes for acting as routers. A SN node is a supervisor node for its cluster head (CH or PCH) and will replace its CH or PCH when the CH or PCH fails. So, SN nodes are substitutes for their cluster heads and also try to achieve to a fault tolerant clustered network.

There are five important factors in FEED. We can give weights to them or use fuzzy logic to count their average amount. These factors are:

A. Density

The density factor of every node reveals the number of nodes around that node such that their distances are less than a threshold *Dist1*. It'll be very good if we choose cluster heads from nodes that have the greatest density factors.

B. Centrality

Sometimes a node has a good density factor meaning that there are lots of near nodes around it but they are all on one side of that node. It is desirable to choose cluster heads from those nodes at the center of their neighbors.

C. Energy

It is clear that cluster heads should be chosen from those nodes with enough remaining energy.

D. Near to other nodes

As mentioned, all nodes contribute to choosing cluster heads. A node that is going to be a cluster head is called a Volunteer. All nodes vote for volunteers. A regular node tries to vote for the nearest volunteer. Cluster heads should be selected from the nodes that many nodes elect as their nearest one.

E. Not being in border

Cluster heads shouldn't be chosen from nodes at the border of the network because border cluster heads create problems for their members during communication.

How FEED works:

Phase 1:

In this phase all nodes should calculate their densities. Each node is equipped with GPS and will be aware of its coordinates.

It must be noted that if all nodes are aware of their densities this phase can be omitted. After some rounds, some nodes will die and consequently node densities will change. However, Phase 1 executes just once in several iterations of algorithm execution.

For calculating densities, all nodes send their id and geographical coordinates to other nodes around by broadcasting locally. It is clear that local broadcast doesn't need so much energy. Nodes can broadcast their messages according to geographical coordinates in different time slices to avoid overhead problem. Then they compute density factor (*de*). Each node can compute its distance from other nodes by knowing its own coordinates and that of the sender node.

If the computed distance is less than *Dist1* the node increases its density field. At the end of this phase all nodes would know their densities.

Phase 2:

In this phase if the density of node is more than a threshold amount and also if it isn't a border node, it computes its centrality factor (*ce*). On the assumption that *n* is the number of network nodes and by using the following algorithm the centrality factor can be computed.

Proposed algorithm for computing density (centrality algorithm):

Consider Figure 1 in which the red node at the center of the figure is a regular node which has received local broadcast messages from black nodes (other regular nodes). The distance between them is less than *Dist1*. Also there would be a specific angle for each couple of red and black nodes that can be computed easily. Recently, each black node has sent its *id* and coordinates by a local broadcast message. The angle between red and black nodes is computed from:

$$\theta_{NodeID} = \tan^{-1} \left(\frac{(NodeID.Y - me.Y)}{(NodeID.X - me.X)} \right) \quad (3)$$

Thus for each black node, the red node calculates its angle and increments the corresponding element in an array of 24 elements. According to figure 2, element 1 is for angles between 0° to 15°, element 2 is for angles between 15° to 30°, . . . and consequently element *i* is for angles between 15(*i*-1)° to 15(*i*)°.

First all array elements are set to zero. The corresponding element is incremented when the angle is computed. The best case occurs when elements *i* and *i*+12 are the same. It means the best case for centrality field is when the absolute value of difference between elements *i* and *i*+12 equals to zero. We save these absolute differences in array *C* as follows:

$$C(i) = |B(i) - B(i + 12)| \quad (4)$$

Then we sum all elements of array *C* and save it in variable *ce* as the centrality factor. A smaller value of *ce* creates a better situation for the centrality factor. So, the best

case for the centrality factor occurs when it equals zero. In this condition the node is exactly at the center of its neighbors. We give negative sign to *ce* when we want to compute average value of all factors.

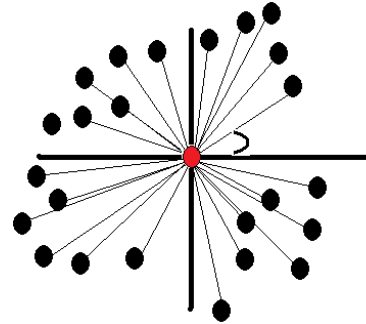


Figure 1. A regular node and its neighbours

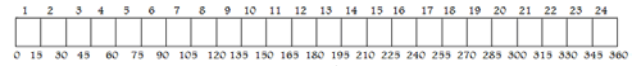


Figure 2. Array B for saving angles

All nodes are aware of their remaining energy (*en*). In continuation of this phase each node gives itself a score according to *ce*, *de* and *en* factors as follows:

$$y = (a * en) + (b * de) - (c * ce) \quad (5)$$

a, *b* and *c* are coefficients for giving weights to energy, density and centrality factors, respectively. If the node is located on the border, it cannot be a volunteer. Taking this and all factors into consideration the node decides whether or not to be a volunteer. Then each volunteer sends a message consisting of *y*, *co* and *id* as follows:

$$y_local_broadcast_msg[myNodeID, co, y] \quad (6)$$

Phase 3:

All the nodes are aware about the messages of their neighbors mentioned by "(6)". The received messages consist of *y*, *co* and *id* of volunteers. Each node enters the distance factor to other factors for each volunteer and computes a second score for that volunteer. The second score is:

$$z = (d * y) - (e * dis) \quad (7)$$

d and *e* are coefficients for giving weights to *y* and distance, respectively. Each node gives a negative score to the distance factor to limit far volunteers of being the cluster head for that node. For the rest of this phase, each node selects the neighbor volunteer with the best second score (*z*) as its deputy volunteer and reveals it to the other nodes around. The number of voters is very important for computing the final scores for volunteers. So, if a volunteer is selected as a deputy by several nodes, it

will have a higher chance of being selected as a CH, PCH or SN. The way final scores are calculated is shown in Table 1.

Phase 4:

Now, each volunteer knows its final score and final scores of its neighbors as well. So, it can decide to be a CH, PCH, SN or even a regular node.

For making such decision there are some rules for location of CH, PCH and SN as shown in Figure 3. In Figure 3, we have: $Dist1 \leq Dist2 \leq 1.5 * Dist1$

The most important rules in Figure 3 are as follows.

In every local area if the node has the highest final score it reveals itself as a PCH, else if the node doesn't have the highest final score and also if its distance from nearest PCH is less than $Dist1$, it'll check to know whether there isn't any SN with better final score for that PCH and by paying attention to this condition it introduces itself as a SN for that PCH.

Table 1. Giving Final Scores to Volunteers

Volunteers' id and co	Given Scores by Other Nodes (nodeid : given score)	Num of Selectors (P)	Sum of Received Scores(S)	Final Score (F=P*S)
1(x1,y1)	⋅(8,90)⋅(2,120) (22,100)	3	310	930
3(x3,y3)	(17,98)⋅(4,100)	2	198	396
7(x7,y7)	(43,143)⋅(33,112) (44,123)⋅	3	378	756
9(x9,y9)	⋅(34,93) ⋅(6,109)⋅(45,111) (61,233)	4	546	2184
5(x5,y5)	(84,153)⋅(21,103)	2	256	512
4(x4,y4)	(12,196)	1	196	196

But, if its distance from nearest PCH is between $Dist1$ and $Dist2$ and if there isn't any CH with better final score at that area, it will try to exhibit itself as a CH. If there is any CH in this area but there isn't any SN with better final score for the nearest CH, the node introduces itself as a SN for that CH.

Another situation for the nodes is that its distance from nearest PCH is greater than $Dist2$, so if its distance from nearest CH is also greater than $Dist1$, it'll do an attempt for exhibiting itself as a PCH, but if its distance from nearest CH is less than $Dist1$ and in the situation that there isn't any SN with better final score for that CH, the node introduces itself as a SN for that CH.

In parallel with the above operations, regular nodes just listen to received messages from volunteers and distinguish nearest ones from others and finally join the nearest cluster heads.

Note that the lowest distance between every couple of PCH shouldn't be less than $Dist2$, the lowest distance between every couple of CH shouldn't be less than $Dist1$ and the lowest distance between any PCH and CH shouldn't be less than $Dist1$.

Pseudo code:

Assumptions:

All the nodes know their coordinates. (They have GPS)

n: the number of nodes

en: remained energy of the node.

ce: centrality of the node.

co: coordinates of the node.

de: density of the node.

volunteer.F: the final score of the volunteer

F subset: the set of all volunteers' final scores

node.deputy: the deputy volunteer of the node

node.volunteer: is set to 1 if node is a volunteer

volunteer.PCH: is set to 1 if volunteer is a PCH

volunteer.CH: is set to 1 if volunteer is a CH

volunteer.SN: is set to 1 if volunteer is a SN

p: the number of nodes selected a volunteer

s: sum of received scores for a volunteer

a, b, c, d, e: coefficients to give weights

// Phase 1: Computing density:

1. each node:

2. sending a local broadcast packet, receiving packets from others and density calculating.

// Phase 2: Volunteers send advertisement messages locally:

1. if the density of the node (ce) is greater than n/10 and node isn't a border node:

2. node.volunteer = true

3. computing centrality by using centrality algorithm

4. $y = (a * en) + (b * de) - (c * ce)$

5. local broadcasting y as the first score

// Phase 3: all nodes give scores to volunteers:

1. each node for each volunteer:

2. $z = (d * y) - (e * dis)$

3. node.deputy = volunteer with maximum z

4. local broadcasting node.deputy message

5. listen to received node.deputy messages

// Phase 4: Selecting PCH, CH and also SN:

1. if (node.volunteer = true)

2. listen to received messages

3. computing p and s

4. $volunteer.F = \sum p * s$

5. if volunteer.F is the highest in F subset

6. volunteer.PCH=1

7. else

8. if volunteer distance from nearest PCH is less than $Dist1$ and if there is no SN for that PCH:

9. volunteer.SN=1

10. else

11. if volunteer distance from nearest PCH is between Dist1 and Dist2 and if there is no CH in this area:
12. volunteer.CH=1
13. else
14. if volunteer distance from nearest PCH is between Dist1 and Dist2 but there is any CH in this area:
15. if volunteer distance from nearest CH is less than Dist2 and if there is no SN for that CH:
16. volunteer.SN=1
17. else
18. if volunteer distance from nearest PCH is more than Dist2 and also volunteer distance from nearest CH is more than Dist1:
19. volunteer.PCH=1
20. if (node.volunteer == false) it joins to nearest CH or PCH

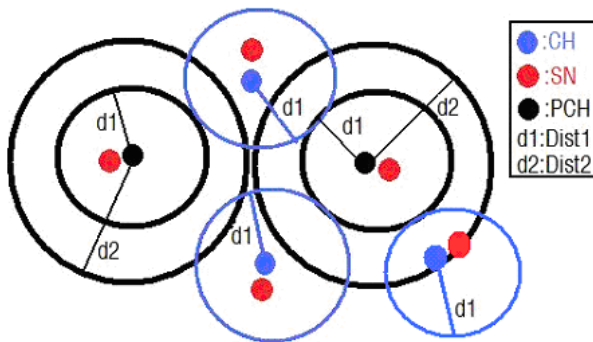


Figure 3. Position of CH, PCH and SN

IV. SIMULATION

To compare LEACH, HEED and FEED algorithms, we simulated and executed them in the Matlab environment. We used three different network models: in the first model we dispersed 100 nodes randomly in a 100*100 region; in the second model we dispersed 200 nodes randomly in a 200*200 region; and in the third model we dispersed 300 nodes randomly in a 300*300 region.

All the initial conditions for the three algorithms mentioned were exactly the same. The radio model is shown in Table 2. In simulation, we considered not only the *Setup Phase* but also the *Steady State Phase* was considered in almost real conditions. In Steady State Phase regular nodes receive (sense) data from the region and send them to their cluster heads. Each cluster head gathers and aggregates the received data and sends it to the Base Station. The Steady State Phase continues until the remaining energy of at least a certain percentage of cluster heads fall less than the threshold.

When the remaining energy of a cluster head falls less than the threshold amount, its SN would be replaced and that cluster can continue its operation until the end of that round. This feature of FEED algorithm leads the region to be completely covered to the end of a certain round. Moreover, this feature leads to an increase in network lifetime and also leads to fault tolerance in clusters.

Figures 4, 5, and 6 show the simulation results for the three models mentioned above in completely same conditions for LEACH, HEED and FEED algorithms. As shown, FEED algorithm increased network lifetime about six times (five hundred percent) more than the LEACH algorithm and about two times (one hundred percent) more than the HEED algorithm. Supervisor node replacement can be a reason for this enhancement.

Clustering algorithms should make clusters such that the number of single clusters (a cluster with only one member) is minimized. By applying the FEED algorithm we almost never see single clusters, so cluster distribution of this algorithm is suitable.

Figure 7 shows the percentage of total remaining energy of the network nodes after 1, 20 and 50 rounds. After one round, the HEED algorithm has the best outcome, but in later rounds the FEED algorithm has the best outcome. Note that after round 300 only FEED algorithm is still executing whereas the LEACH and HEED algorithms have terminated.

As a shortcoming of FEED, we can point to exchanging necessary messages sent by nodes creating local overhead for the network. But fortunately this overhead is local and doesn't cause huge problems.

Table 2. Radio Model Used in Simulation

Operation	Energy Dissipated
Transmitter Electronics	Eelec=50nJ/bit
Receiver Electronics	Eelec=50nJ/bit
Transmit Amplifier	100 pJ/bit/m ²

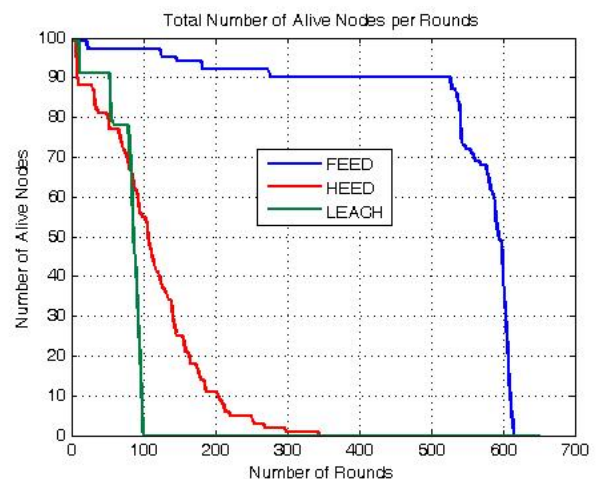


Figure 4. Total number of nodes per rounds in first model

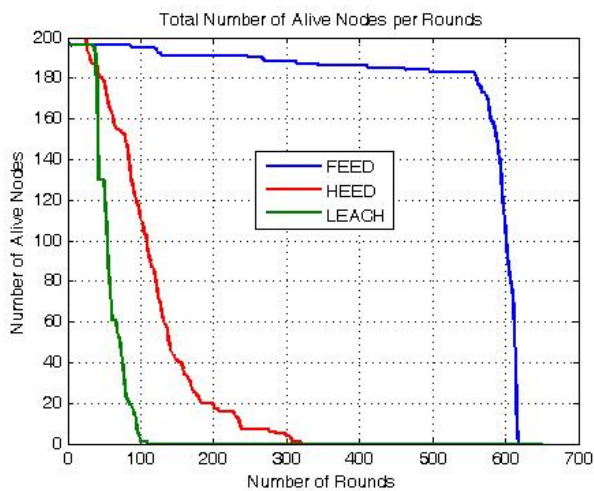


Figure 5. Total number of nodes per rounds in second model

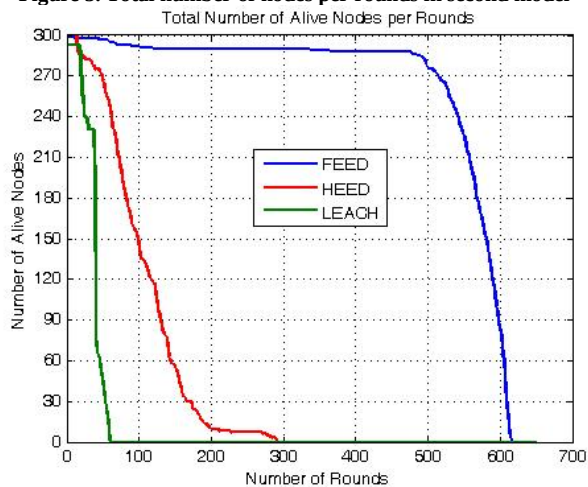


Figure 6. Total number of nodes per rounds in third model

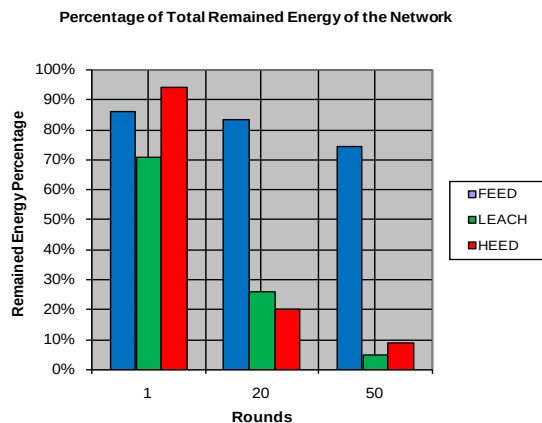


Figure 7. Percentage of total remained energy of the network nodes

V. CONCLUSION

In this paper, we described FEED, a fault tolerant, energy efficient, distributed clustering algorithm. After simulation and comparing this algorithm with two well known algorithms, we found that FEED prolongs network lifetime. FEED pays attention to energy, density and centrality factors and also distance between nodes for making clusters. Furthermore, FEED selects a supervisor node for each cluster head. This selection causes the entire network to be completely covered until the end of a round, and the network will be fault tolerant. By using FEED, the number of single member clusters is near zero, thus FEED provides well distributed clusters.

REFERENCES

- [1] W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proc. of the 33rd Annual Hawaii International Conference on System Sciences (HICSS)*, Maui, HI, Jan. 2000, pp. 3005 – 3014.
- [2] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad-hoc Sensor Networks", sponsored in part by NSF grant ANI-0238294 (CAREER) and the Schlumberger Foundation
- [3] I. Gupta, D. Riordan and S. Sampalli, "Cluster-head Election using Fuzzy Logic for Wireless Sensor Networks", *Proceedings of the 3rd Annual Communication Networks and Services Research Conference (CNSR'05)*, IEEE 2005
- [4] C. Chen, K.R. Lee, J.H. Wang and Y.H. Kuo, "Energy-proportional routing for lifetime extension of clusteringbased wireless sensor networks", *International Journal of Pervasive Computing and Communications*, Vol. 3, pp. 304-305, No. 3, 2007.
- [5] A. Manjeshwar and D.P. Agrawal, "TEEN: a protocol for enhanced efficiency in wireless sensor networks", *Proceedings of the 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, San Francisco, CA, April 2001.
- [6] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks", *Ad Hoc Networks* 3, pp. 325–349, 2005.
- [7] K. Matrouk and B. Landfeldt, "RETT-gen: A globally efficient routing protocol for wireless sensor networks by equalising sensor energy and avoiding energy holes", *Ad Hoc Networks* 7, pp. 514–536, 2009.
- [8] M.Q. Rieck and S. Dhar, "Constructing special k-dominating sets using variations on the greedy algorithm", *Pervasive and Mobile Computing* 5, pp. 64-76, 2009.
- [9] R. Langar, N. Bouabdallah and R. Boutaba, "Mobility-aware clustering algorithms with interference constraints in wireless mesh networks", *Computer Networks* 53, pp. 25 – 44, 2009.
- [10] R. Yu, X. Wang and S. K. Das, "EEDTC: Energy-efficient dominating tree construction in multi-hop wireless networks", *Pervasive and Mobile Computing* 7, 2008