

# Self Organizing Sensor Networks Using Intelligent Clustering

Kwangcheol Shin, Ajith Abraham and Sang Yong Han<sup>1</sup>

School of Computer Science and Engineering, Chung-Ang University  
221, Heukseok-dong, Dongjak-gu, Seoul 156-756, Korea  
kcshin@archi.cse.cau.ac.kr, ajith.abraham@ieee.org,  
hansy@cau.ac.kr

**Abstract.** Minimization of the number of cluster heads in a wireless sensor network is a very important problem to reduce channel contention and to improve the efficiency of the algorithm when executed at the level of cluster-heads. This paper proposes a Self Organizing Sensor (SOS) network based on an intelligent clustering algorithm which does not require many user defined parameters and random selection to form clusters like in Algorithm for Cluster Establishment (ACE) [2]. The proposed SOS algorithm is compared with ACE and the empirical results clearly illustrate that the SOS algorithm can reduce the number of cluster heads.

## 1 Introduction and Related Research

Research in wireless sensor networks has been growing rapidly along with the development of low-cost micro devices and wireless communication technologies [1]. Some of the research related to scientific, medical, military and commercial usage has gone to the background [4].

Sensor networks are composed of hundreds to myriads of sensor nodes, which appear to be sprinkled randomly by a car or airplane. Each node has strict limitation in the usage of electric power, computation and memory resources. They typically utilize intermittent wireless communication. Therefore, sensor networks should be well-formed to achieve its purposes. Clustering is a fundamental mechanism to design scalable sensor network protocols. The purpose of clustering is to divide the network by some disjoint clusters. Through clustering, we can reduce routing table sizes, redundancy of exchanged messages, energy consumption and extend a network's lifetime. By introducing the conventional clustering approach to the sensor networks provides a unique challenge due to the fact that cluster-heads, which are communication centers by default, tend to be heavily utilized and thus drained of their battery power rapidly. Algorithm for Cluster Establishment (ACE) [2] clusters the sensor network within a constant number of iterations using the node degree as the main parameter. Some of the weaknesses of ACE are: First, ACE randomly selects candi-

---

<sup>1</sup> Corresponding author

date node in each iteration which creates different results each time on the same sensor network. Second, spawning threshold function is used in ACE to control the formation of new cluster by using two manually adjusted parameters. ACE performance relies on these parameters which are usually manually adjusted according to the size and shape of a sensor network.

In the literature, besides ACE, there are some related works on forming and managing clusters for sensor networks. For example, LEACH [5] rotates the role of a cluster head randomly and periodically over all the nodes to prevent early dying of cluster heads. Guru et al. [6] consider energy minimization of the network as a cost function to form clusters. Mhatre and Rosenberg [7] take into account not only the battery of the nodes but also the manufacturing cost of hardware.

In this paper, we propose a new clustering algorithm that does not require manually adjusted parameters which could also provide identical results in each test on the same sensor network to overcome the weakness of ACE. Rest of the paper is organized as follows. In Section 2, we present the clustering problem followed by Section 3 wherein the new algorithm is illustrated. Experiment results are presented in Section 4 and some conclusions are also provided towards the end

## 2. The Clustering Problem

Clustering problem can be defined as following. Assume that nodes are randomly dispersed in a field. At the end of clustering process, each node belongs to one cluster exactly and be able to communicate with the cluster head directly via a single hop [3]. Each cluster consists of a single cluster head and a bunch of followers as illustrated in Figure 1. The purpose of the clustering algorithm is to form the smallest number of clusters that makes all nodes of network to belong to one cluster. Minimizing the number of cluster heads would not only provide an efficient cover of the whole network but also minimizes the cluster overlaps. This reduces the amount of channel contention between clusters, and also improves the efficiency of algorithms that executes at the level of the cluster-heads.

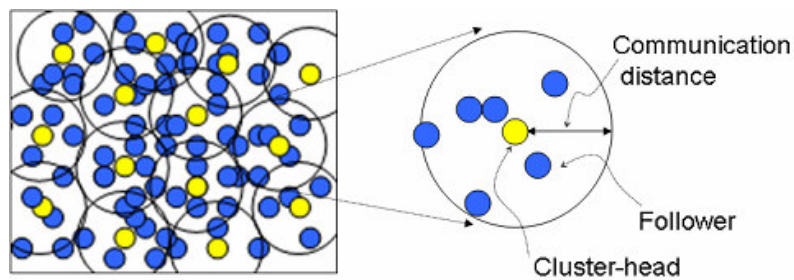


Fig. 1. Clustering in a sensor network

### 3. Self Organizing Sensor (SOS) Networks by Minimization of Cluster Heads Using Intelligent Clustering

#### 3.1 Global level of clustering algorithm

This Section presents the proposed clustering algorithm in a global scale, and the following section describes the algorithm at a node level. The following steps illustrate an overview of the suggested algorithm.

1. Find the node ( $N_o$ ), which has the maximum number of followers, and make a cluster with it.
2. Include clustered nodes into a clustered node set  $G$ .
3. Selects the next head node ( $N_f$ ), which can communicate with a node in  $G$  and has the maximum number of followers, and make a cluster with it.
4. If there exist an unclustered node or nodes then go to step 2
5. Else terminate the algorithm.

At first, it makes a cluster with the center node which has the maximum number of followers. We assume that there is a coordinator which controls globally in the entire network (for easy understanding). So it does not matter to locate the center node during step 1. In step 2, it includes the selected cluster head node and its followers to the clustered node set  $G$ . And in step 3, it selects the node, which can communicate with a node in  $G$  and has the maximum number of followers, and makes a cluster with it as a cluster head and include it and its follower to set  $G$ . Figure 2 illustrates step 3. A node ' $a$ ' is ' $N_o$ ' node and node ' $b$ ' is the node which can communicate with the next head node (that is, node ' $c$ '), which has the maximum number of followers. Then it elects node ' $c$ ' as a next cluster head node and makes a cluster with it. The process is then repeated until all the nodes are clustered.

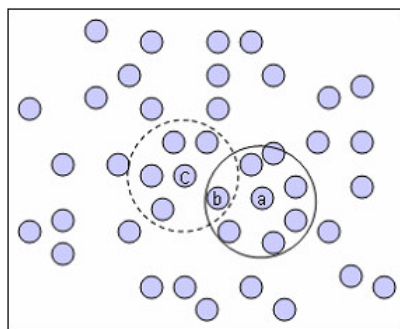


Fig. 2. Clustering example

### 3.2 Node level of Clustering Algorithm

Node level algorithm is mainly divided into two parts, first part for finding out a node which has the most number of followers and makes it as the first cluster head, and the second one for the actual clustering process.

**Table 1. Message and methods**

Message Structure: (command, data, node_id)
Methods description :
broadcast(message) : send a message to everyone which it can communicate with
send(message, destination) : send a message to a destination

To implement the algorithm, we introduce ‘message’ which has three parts, (*command, data, node\_id*) and two methods which are used frequently, broadcast (*message*), which sends a message to everyone, to which it can communicate with and send (*message, destination*) which sends a message to a destination. The concept of *message* and *methods* is illustrated in Table 1.

We also define two concepts:

**Super-node:** The node which is selected as a head of first cluster, to decide which node will be the new cluster head (for example, node ‘a’ in Figure 2).

**Linker node:** The node which communicate between two cluster heads. This node is included in two clusters which it connects (for example, node ‘b’ in Figure 2).

**Table 2.** Algorithm for finding the super-node

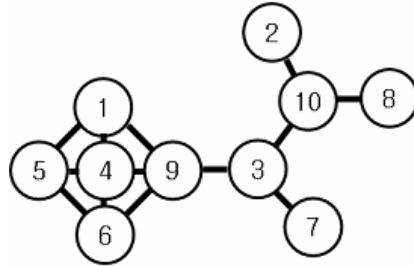
```

myState := Super_Head
n := number of my neighbors
c := myID
while (myState is Super_Head and c is not 0)
    c := c -1
    if (notEmpty(msgQueue)
        message := find_best_one(msgQueue)
        if(message.data >= n)
            myState := Unclustered
            broadcast(message)
    if (myState is Super_Head) broadcast(( ,n, ))
    d := n
    t := sufficient time + myID
    While (t is not 0)
        t := t-1
        message := wait_for_a_message()
        if (message.data > n)
            myState := Unclustered
            if(d < message.data)
                d := message.data
                broadcast(message)
    if (myState is Super_Head) broadcast(("recruit", ,myID))
    Purge(msgQueue)

```

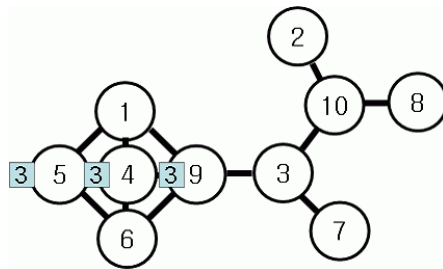
### 3.2.1 Discovery of nodes which has the most followers

To find the node which has the maximum number of followers, we suggest a method as illustrated in Table 2. In the first stage, state of every node is considered as a super-head. Each node counts the number of its neighbors and it sets variable  $c$  as its unique identification number (ID) to execute the algorithm one by one without collisions. This unique ID for the individual sensors is decided when the sensors are spread.



**Fig. 3.** Illustration for finding the super-node

For the sensor network illustrated in Figure 3 (the number in each circle is a unique ID for each sensor), we present how the proposed algorithm could set up node 4 as a super-node. It is important to remember that each node performs its own algorithm operation independently to setup the super-nodes. At first, node 1 sends message to its neighbors and nodes 5, 4 and 9 will receive the message which node 1 sent. Message queue of nodes 5, 4 and 9 are shown in Figure 4 and node 1 will get into the state of waiting for a message. After that, node 2 broadcasts its number of neighbors to its neighbor node 10, and node 3 to nodes 7, 9 and 10. Node 2 and 3 will also get into the state of waiting for a message as shown in Figure 5.



**Fig. 4.** Message queue of nodes 5, 4 and 9 after node 1 broadcasts

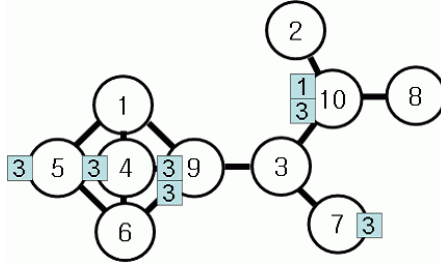


Fig. 5. State after nodes 2 and 3 broadcast a message

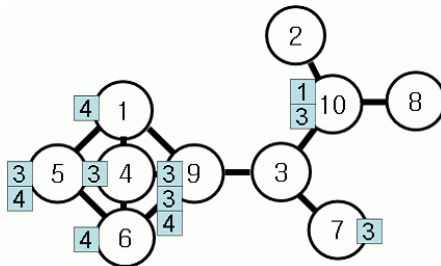


Fig. 6. After node 4 broadcasts a message

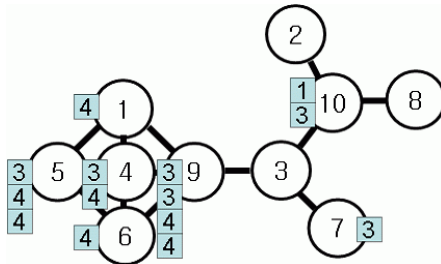


Fig. 7. After node 1 broadcasts a message which it received

By turn, node 4 performs its operation and its message queue is not empty. So, node 4 finds the message, which has the biggest data value, in its message queue and compares it with its number of neighbors. In this case, node 4's number of neighbors is 4 and the biggest one in message queue is 3, so node 4 broadcasts its number of neighbors as shown in Figure 6. Node 1 will now receive the message, which node 4 sent, and it changes its status as unclustered since arrived *'message.data'* is bigger than its number of neighbors and broadcast arrived *'message.data'* again. The procedure is illustrated in Figure 7.

Node 5 executes its algorithm and the number of its neighbors is 3 and the biggest one in message queue is 4, so it changes its status as unclustered and broadcasts *'message.data'*, which is 4. Node 6 executes its algorithm and its number of followers is smaller than the biggest one in queue, and it changes its status as unclustered and broadcasts the biggest *'message.data'*. After doing all of procedures, node 4 will

remain as super-node and all of rest will be unclustered status. And finally, node 4 broadcasts a recruit message to its neighbors to make a cluster with node 4 as cluster head.

### 3.2.2 Self Organizing Sensor (SOS) Clustering Algorithm

Table 3 illustrates the pseudo code of the SOS clustering algorithm and it consists of 5 parts.

**Table 3.** SOS clustering algorithm

```

myHead := NONE // my cluster head
nextHead := NONE // for linker node, which has two head

// for unclustered node
while (myState is Unclustered)
    message := wait_for_a_message()
    if (message.command is "survey")
        uf := calculate_number_of_followers(myID)
        send("report",uf,myID),message.node_id)
    if (message.command is "recruit")
        myHead := message.node_id
        myState := Clustered
    if (message.command is "notify" and message.node_id is myID)
        myState := Cluster_Head
        broadcast("recruit", ,myID))

// for clustered node
while (myState is Clustered)
    message := wait_for_a_message()
    followers := NONE // array for follower nodes
    if (message.command is "survey")
        followers := update_my_followers(myID)
        if(followers is not NONE)
            send("survey", ,myID),followers)
            msgQueue := wait_for_followers_reports()
            nodeBest := find_best_node(msgQueue)
            message := (message.command,message.data,myID)
            send(nodeBest,myHeader)
            purge(msgQueue)
        else
            send("report",NONE,NONE),myHead)
            terminate()
    if (message.command is "notify" and message.node_id is myID)
        myState := Linker
        nextHead := nodeBest.node_id
        send(message,nodeBest.node_id)

// for super-head
while (myState is Super_Head)

```

```

        broadcast(("survey", , ))
        msgQueue := wait_for_followers_reports()
        networkBest := fine_best_node(msgQueue)
        if(networkBest.node_id is NONE) terminate()
        else broadcast_to_followers(("notify",,networkBest.node_id))
        purge(msg_queue)

// for cluster head
while (myState is a Cluster_Head)
    message := wait_for_a_message()
    if (message.command is "survey")
        broadcast_to_followers("survey", ,myID)
        msgQueue := wait_for_followers_reports()
        clusterBest := find_best_node(msgQueue)
        send(clusterBest,message.node_id)
        if (clusterBest.node_id is NONE) terminate()
        purge(msgQueue)
    if(message.command is "notify" and message.node_id is clusterBest.node_id)
        broadcast_to_followers(message)

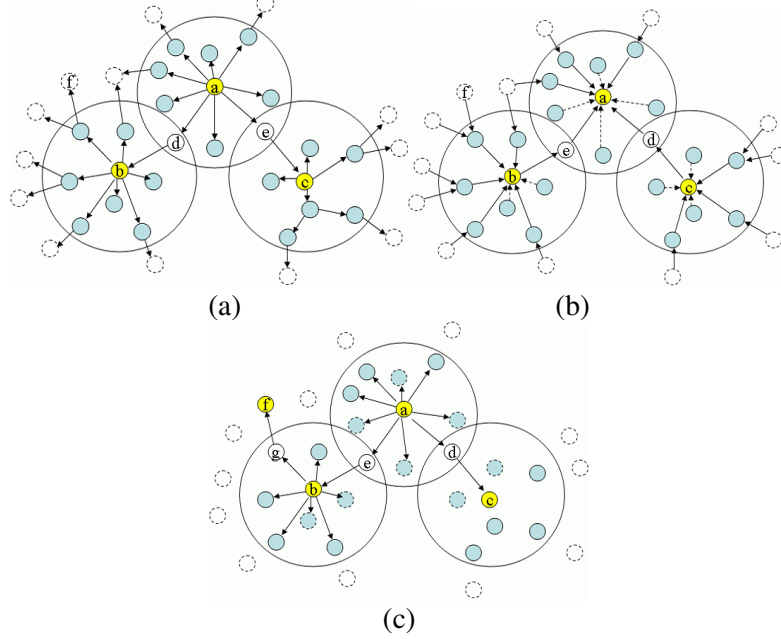
// for linker
while (myState is a Linker)
    message := wait_for_a_message()
    if (message.command is "survey")
        message.node_id = myID
        send(message, nextHead)
    if (message.command is "notify") send(message, nextHead)
    if (message.command is "report")
        send(message, myHead)
        if(message.node_id is NONE) terminate()

```

The clustering process is illustrated in Figure 8. Every node, whose status is unclustered, waits for a message. The super-node (node 'a' in Figure 8) broadcasts 'survey' message to its followers. Every node, which receives 'survey' message from its cluster head (include super-node), investigates that how many unclustered nodes exist within the area of its communication range. If there are no existing nodes that can communicate with, then it reports it to their head and terminates its algorithm. If some nodes exist, it send 'survey' message to every follower and waits for its 'report' messages as shown in Figure 8. When every follower reports about it, the node selects follower's ID, which has the biggest number of neighbors, and save that follower's ID and sends a 'report' back to its head recursively. This works in a recursive way and every 'report' message arrives in super-node (Figure 8-b). If super-node get all report from every follower, then it selects a message contains the follower's id, which has the biggest number of neighbors, and broadcasts 'notify' message with that follower's ID to its followers. Every clustered nodes, which receive 'notify' message, compares 'notify.node\_id' with saved id and if it is same, then it changes its status as 'linker' and set its next-head as saved node id, and sends a 'notify' message to its next-head. If cluster-head received a 'notify' message, then it compares 'notify.node\_id' with stored ID and if it is same then it broadcasts otherwise just drop it. If unclustered node received



'notify' message then it changes its status as cluster-head and broadcasts a 'recruit' message to its followers to make a cluster with it. If super-head get every 'report' message with 'none' then it terminates its algorithm (Figure 8-c).



**Fig. 8.** Illustration of (a) 'survey' process (b) 'report' process and (c) 'notify' process

**Table 4.** Test results for 2500 nodes

Case of 2500 nodes (500*500 rectangle space)			
Communication distance of a node	Number of generated clusters		Improvement ((ACE-SOS)/ACE)
	ACE ( $k_1=2.3, k_2=0.1$ )	SOS	
30	308	255	17.21%
50	126	114	9.52%
70	68	59	13.24%
100	38	35	7.89%
Average			11.97%

## 4. Experiment Results

The proposed SOS algorithm was implemented and compared with the ACE algorithm. We randomly scattered 2500 nodes in a 500\*500rectangle space. Table 4 illustrates the performance results for 2,500 nodes. For comparison purposes, we set the communication range of each node as 30, 50, 70 and 100. In case of ACE, we manu-

ally adjusted  $k_1$  and  $k_2$  to achieve the best results. As shown in Table 4, the number of cluster heads could be reduced by about 11.97% (average) for 2,500 nodes when compared to the ACE approach. By using the SOS approach, we can efficiently reduce the routing table sizes, redundancy of exchanged messages, energy consumption and extends the network's lifetime.

## 5. Conclusions

In this paper, we presented a new clustering algorithm for minimizing the number of cluster heads. The proposed algorithm produces identical results every time for same network without using any network dependent parameters. Empirical results clearly show that the SOS algorithm could reduce the number of cluster heads by about 11.97% for 2,500 nodes when compared to the ACE approach.

Although our algorithm efficiently formulated the required clusters, there are several things to consider such as problems related to fast dying cluster heads and so on. We are also planning to incorporate more heuristic techniques to make the clustering process more efficient.

## Acknowledgements

Work supported by the MIC (Ministry of Information and Communication), Korea, under the Chung-Ang University HNRC-ITRC (Home Network Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

## References

1. J.M. Kahn, R.H. Katz and K.S. Pister, Next Century Challenges : Mobile Networking for "Smart Dust", Proceedings of Mobicom, August 1999.
2. H. Chan, A. Perrig, ACE: An Emergent Algorithm for Highly Uniform Cluster Formation. In 2004 European Workshop on Sensor Networks. pp. 154-171.
3. Younis and S. Fahmy. Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach. In Proceedings of IEEE INFOCOM, March 2004.
4. I.F. Akyildiz, W. Su, Y. Sankarsubramaniam and E. Cayirci, "Wireless Sensor Networks : a survey", Computer Networks, Vol. 38, pp. 393-422, March 2002.
5. W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "An Application Specific Protocol Architecture for Wireless Microsensor Networks," IEEE Transactions on Wireless Communications, Vol. 1, No. 4, October 2002.
6. S. M. Guru, A. Hsu, S. Halgamuge, S. Fernando, "An Extended Growing Self-Organizing Map for Selection of Clusters in Sensor Networks", International Journal of Distributed Sensor Networks, Volume 1, Number 2 / April-June 2005
7. V. Mhatre and C. Rosenberg, "Homogeneous vs. Heterogeneous Clustered Sensor Networks: A Comparative Study", 2004 IEEE International Conference on Communications (ICC 2004), Paris France, June 2004.