An Efficient Management Algorithm for Clustering in Mobile Ad hoc Network

Zouhair El-Bazzal, Michel Kadoch, Basile L. Agba, François Gagnon and Maria Bennani
École de technologie supérieure
1100 Notre Dame O., Montreal, Qc.
H3C 1K3 Canada
1(514)396-8800 # 7983
zouhair.elbazzal@lagrit.etsmtl.ca, {mkadoch, bagba, fgagnon, mbennani}@ele.etsmtl.ca

ABSTRACT
Clustering of mobile nodes among separate domains has been proposed as an efficient approach to mimic the operation of the fixed infrastructure and manage the resources in multi-hop networks. In this work, we propose a new clustering algorithm, namely Efficient Management Algorithm for Clustering (EMAC) based on weighting parameters. The goals are yielding low number of clusters, maintaining stable clusters, minimizing the number of invocations for the algorithm and maximizing lifetime of mobile nodes in the system. Through simulations we have compared the performance of our algorithm with that of WCA in terms of the number of clusters formed and number of states transitions on each clusterhead. The results demonstrate the superior performance of the proposed algorithm.

Categories and Subject Descriptors
C.2.1 [**Computer-Communications Networks**]: Wireless communication

General Terms
Algorithms, Management, Performance, Design.

Keywords
MANET, Clustering Algorithm, Weight, Election.

1. INTRODUCTION
Recently, most research is focusing on clustering in multi-hop networks in order to build a virtual backbone formed by a set of suitable representative nodes. The main challenge is to elect the clusterheads (CHs) which guarantee the communications across the formed clusters. The clusters are able to store minimum topology information; each CH acts as a temporary base station within its zone or cluster and communicates with other CHs. An example of multi-hop networks is a mobile ad hoc network (MANET) characterized by a collection of wireless nodes that are arbitrary and randomly changing their locations and capabilities without the existence of any centralized entity. Therefore, any clustering scheme should be adaptive to such changes with minimum clustering management overhead incurred by changes in the network topology. As election of optimal clusterheads is an NP-hard problem [1], many heuristic mechanisms have been proposed. Recent works suggest CH election exclusively based on nodes' IDs or location information; however these algorithms suffer from single point (CH) of bottleneck especially in highly mobile environments; hence initially elected CHs have to collect excessive amounts of information and soon reach battery exhaustion. Other works take into account additional metrics (such as energy and mobility) and optimize initial clustering. However, in many situations re-clustering procedure involves frequent broadcasting of control packets even when network topology remains unchanged.

PM²HW²N'06, October 6, 2006, Torremolinos, Malaga, Spain.

In addition, a topology control mechanism is required to mitigate the vulnerability of such clusters due to node joining/leaving and link failures. It aims to reduce interference and energy consumption, to increase the effective network capacity, and to reduce the end to end delay. In order to simplify the maintenance, especially in high mobility scenarios, we investigate an algorithm that generates one-hop clusters. In this way, the goals of this paper are to maintain stable clusters with a lowest number of clusterheads, to minimize the number of invocations for the clustering formation/maintenance and to maximize the lifetime of mobile nodes in the system. To achieve these goals, we propose an Efficient Management Algorithm for Clustering (EMAC) which utilizes factors like the node degree, remaining battery power, transmission power, and node mobility for the clusterheads' election. Our algorithm differs from others in that it is based on the clusters' capacity and it uses the link lifetime instead of the node mobility for the maintenance procedure. We refer this to the fact that the node mobility metric does not affect the election of a CH as much as the link stability metric does. The simulations results show that the proposed algorithm provides better performance in terms of number of formed clusters and average number of transition (state change) on CHs when compared to that of other weight based algorithms such as WCA.

The paper is organized as follows. In Section 2, we review several clustering algorithms proposed previously. Section 3 presents the proposed algorithm for ad hoc networks. Section 4 presents the analyzed performance of the proposed algorithm. Finally, Section 5 concludes this paper.

2. OVERVIEW OF THE EXISTING ALGORITHMS

A large number of approaches have been proposed for the election of clusterheads in mobile ad hoc networks. The Highest-Degree [2] uses the degree of a node as a metric for the selection of clusterheads. The degree of a node is the number of neighbors each node has. The node with maximum degree is chosen as a clusterhead; since the degree of a node changes very frequently, the CHs are not likely to play their role as clusterheads for very long. In addition, as the number of ordinary nodes in a cluster is increased, the throughput drops and system performance degrades. The Lowest-Identifier (LID) [3, 4, 5] chooses the node with the lowest ID as a clusterhead, the system performance is better than Highest-Degree in terms of throughput. However, those CHs with smaller IDs suffer from the battery drainage, resulting short lifetime of the system.

The Distributed Clustering Algorithm (DCA) [6] and Distributed Mobility Adaptive clustering algorithm (DMAC) [7] are enhanced versions of LID; each node has a unique weight instead of just the node's ID, these weights are used for the selection of CHs. A node is chosen to be a clusterhead if its weight is higher than any of its neighbor's weight; otherwise, it joins a neighboring clusterhead. The DCA makes an assumption that the network topology does not change during the execution of the algorithm. Thus, it is proven to be useful for static networks when the nodes either do not move or move very slowly. The DMAC algorithm, on the other hand, adapts itself to the network topology changes and therefore can be used for any mobile networks. However, the assignment of weights has not been discussed in the both algorithms and there are no optimizations on the system parameters such as throughput and power control.

Instead of static weights, MOBIC [8] uses a new mobility metric; Aggregate Local Mobility (ALM) to elect CH. ALM is computed as the ratio of received power levels of successive transmissions (periodic Hello messages) between a pair of nodes, which means the relative mobility between neighboring nodes. Least Clusterhead Change Algorithm (LCC) [9] allows minimizing clusterhead changes that occur when two CHs come into direct contact. In such a case, one of them will give up its role and some of the nodes in one cluster may not be members of the other CH's cluster. Therefore, some nodes must become CH while causing a lot of re-elections because of the propagation of such changes across the entire network. Maximum Connectivity Clustering (MCC) [10] is based on the degree of connectivity. A node is elected as CH if it is the highest connected node. This is not suitable in dynamic network topologies where the degree of connectivity changes rapidly.

The Weighted Clustering Algorithm (WCA) [11] is based on the use of a combined weight metric that takes into account several parameters like the node-degree, distances with all its neighbors, node speed and the time spent as a clusterhead. Although WCA has proved better performance than all the previous algorithms, it lacks a drawback in knowing the weights of all the nodes before starting the clustering process and in draining the CHs rapidly. As a result, the overhead induced by WCA is very high.

3. THE PROPOSED MODEL

The goal is to build an architecture based on clusters. Every cluster has a limited number of nodes which defines its size. The nodes collaborate to select the best CH. A CH must be able to manage its members, to accept or to refuse the adhesion of new arrivals based on its capacity without perturbing the functionality of the other members. We note that this architecture may generate some isolated nodes that fail to reach any CH (even if they are in the transmission range of other member nodes) and some single node clusters because we privilege the inter-clusters' communication in order to facilitate the building of an entire backbone in future works.

3.1 Setup procedure

First, we allocate IDs for the nodes and the clusterheads. We use the MAC Address as the node ID in order to avoid the conflicts between IDs in the zone. Hence, the node ID (My_MAC) is unique within a cluster; the CH ID is the node ID of the CH (CH_MAC) in the cluster. The CH ID appended with the node ID forms a unique identifier for every node in the ad hoc network. Every node in the cluster will have information about its CH so that it can communicate across the cluster. We also use a 'Counter' parameter maintained by each node to guarantee that the cluster does not exceed the cluster size limit in terms of number of nodes 'N' inside a cluster.

In the proposed model, each node N_i (member or CH) is identified by a state such as: N_i (id_{node}, id_{CH}, Weight, Counter, N), it also has to maintain a 'node_table' wherein the information of the local members is stored. However, the CHs maintain another clusterhead information table 'CH_table' wherein the information about the other CHs is stored. The format of these tables is defined as: node_table (id_{node}, Weight) and CH_table (id_{CH}, Weight).

In complex networks, the nodes must coordinate between each other to update their tables. The Hello messages are used to complete this role. A Hello contains the state of the node; it is periodically exchanged either between CHs or between each CH and its members in order to update the 'CH_tables' and the 'node_tables' respectively. The weight parameter is periodically calculated by each node as shown in figure 1 in order to indicate the suitability of a node for playing clusterhead's role.

- Find the degree d_i of the node i by counting its neighbours;
- Compute the degree difference $\Delta_i = |d_i - N|$ for the node i, where N is a threshold for the cluster's size in terms of number of nodes;
- Compute the remaining battery power E_i for the node i;
- Compute the actual transmission power P_i of the node i;
- Compute the average speed S_i of the node i until the current time T;

$$S_i = \frac{1}{T}\sum_{t=1}^{T}\sqrt{(X_t - X_{t-1})^2 + (Y_t - Y_{t-1})^2}$$

  where (X_t, Y_t) is the coordinate of node i at time t;
- Calculate the combined weight W_i of the node i :
  $W_i = a*\Delta_i + b*E_i + c*P_i + d*S_i$
  where a, b, c et d are the weighting factors and a+b+c+d=1;

Figure 1. Procedure for calculating the weight of node i

These parameters are inspired from those used in WCA [11], except the actual transmission power 'P_i' and the remaining battery power 'E_i'. We focus on P_i instead of the sum of distance used in WCA in order to elect the node which can cover the largest range, thus, minimizing the number of clusters generated. In addition, the E_i factor is a better measure than the cumulative time during which the node acts as a CH that is used in WCA, because it allows to extend the lifetime of nodes by relinquish the role as a CH in case of insufficient battery power. In this manner, we are able to combine each of the above system parameters with certain weighing factors depending on the system needs. The flexibility of changing the weighting factors helps us apply our algorithm to various networks. For example, in low mobility environment, we can privilege the remaining battery of a node, thus the factor ‘b’ can be made smaller.

On the other hand, we believe that the relative mobility M_i is a better factor than the average speed S_i. In the first stage, we still use S_i instead of M_i because it is impossible to estimate M_i when the node is alone in the zone without any other reference point (neighbors). In the second stage, the re-election is more sophisticated. Therefore, we use the link lifetime metric which seems more realistic than the relative mobility metric to see whether it is worth to re-elect or to continue with the old CH. Initially, each node broadcasts its state (Join with id_{node} = My_MAC) to notify its presence to the neighbors. Each node builds its neighbors’ list based on the received states. After that, the election procedure is executed once the topology is stabled, and the node having the lowest weight is chosen as CH.

3.2 New Arrival Nodes Mechanism

Once a wireless node is activated, its id_{CH} field is equal to NULL since it does not belong to any cluster. The node continuously monitors the channel until it figures out that there is some activity in its neighborhood. This is due to the ability to receive the signals from other present nodes in the network. The node still has no stable state, thus its state is not full identified. In this case, it broadcasts a Join_Request in order to join the most powerful clusterhead. Thus, it waits either for a welcome_ACK or for a welcome_NACK.

When the entry node does receive neither welcome_ACK nor welcome_NACK, it may increase its transmission power in order to broadcast another Join_Request that may reach the farthest clusterheads. If this persists for certain number of attempts, the node declares itself as an isolated node, readjusts its transmission power and restarts by broadcasting a new Join_Request after a period of time. We note that just the CHs may response by a welcome_ACK or welcome_NACK; the ordinary members have to ignore any Join_Request received even if they are in the transmission range of the new entry node. This allows simplifying the management of the clusters.

In the case where the node receives a response (welcome_ACK or welcome_NACK), it does not take immediately any decision, this allows the node to be certain that it has received all the responses from all the neighboring CHs. The welcome_ACK and welcome_NACK messages do not indicate that the CH has added the node to its table; they just signify that the CH is waiting for a Join_Accept in order to add the node to its table.

When the node receives multiple welcome_ACKs, it selects the one which has the lowest weight. After that, it sends a Join_Accept to the chosen clusterhead and waits for CH_ACK from this CH. The CH_ACK has to contain a confirmation that the id_{node} has been added to the CH_table. Thus the node can fully-define its state. The reason that we use four ways to confirm the joining procedure is to prevent other CHs that they can serve the entry node to add this node to their tables and cause conflicts.

In the case where the node was just receiving welcome_NACKs, it considers these responses as rejection messages from the CHs. This may occur when the CHs are saturated and decide to reject the adhesion of new nodes. Hence, the node may increase its transmission power and count the number of attempts it tries to reach any CH. When the number of attempts reaches a certain value, the node prefers not to stay isolated, thus it declares itself as CH; in this case, the node must search for the welcome_NACK which has the lowest weight, thus it communicates via a CH_Request message with the CH which is the source of the chosen welcome_NACK and waits for a CH_Response. Table 1 summarizes messages used in the proposed algorithm.

Table 1. Messages used in the algorithm

Message	Description
Hello(id_{node}, id_{CH}, Weight, Counter, N)	To update the tables of the nodes
Join_Request(id_{node}, id_{CH})	To affiliate a cluster
welcome_ACK(id_{node}, id_{CH}, Weight)	The CH accepts a Join_Request
welcome_NACK(id_{node}, id_{CH}, Weight)	The CH rejects a Join_Request
CH_Request(id_{node})	The node declares itself as CH
CH_Response(id_{node})	The CH accepts a CH_Request
Join_Accept(id_{node}, id_{CH}, Weight, Counter, N)	The node accepts the welcome_ACK
CH_ACK(id_{node} , id_{CH}, Weight, Counter, N)	The CH adds the node as a member
Database_info(id_{node} , id_{CH}, Weight, Counter, N)	The current CH sends the database to a new elected CH
Database_ACK(id_{node} , id_{CH}, Weight, Counter, N)	The new elected CH accepts the received database
CH_change(id_{CH})	The CH notifies a CH change
CH_info(id_{node}, id_{CH}, Weight, Counter, N)	The CH accepts the presence of a new CH in the network
Leave_Request(id_{node} , id_{CH}, Weight, Counter, N)	The node leaves the cluster

3.3 Clusterhead Nodes Mechanism

A CH has an id_{node} field is equal to id_{CH} field. As a CH, the node calculates periodically its weight, thus it sends periodically Hello messages to its members and to the neighboring CHs in order to update the node_tables and CH_tables respectively. The CH must monitor the channel for Leave, Hello and Join_Request messages.

The re-election does not necessarily mean that a new CH must be elected even if there is a member node having a lowest weight, we will explain in details this procedure in figure 4. When the CH receives a Join_Request (id_{CH} =NULL) from a new arrival node or a Join_Request (full state) from a node which belongs to another cluster, the CH must invoke the merging procedure explained in figure 3 in order to accept or to reject the request basing on its capacities, the link lifetime and the available resources. This procedure gives more flexibility to the members by allowing them to leave a weak CH and join another one which seems stronger than the current CH. It may not be possible for all the clusters to reach the cluster size N. We have tried to reduce the clusters formed by merging the clusters that have not attained their cluster size limit. However, in order not to rapidly drain the clusterhead's power by accepting a lot of new nodes, we define thresholds which allow the clusterhead to control the number of nodes inside its cluster.

Figure 2 explains the procedure to execute by each CH in the network. We restrict this procedure to the CHs in order to simplify the maintenance and the complexity of the cluster management.

```
Calculate periodically its weights;
Send periodically Hello messages to its members and its neighboring CHs;
if (no hellos from a specific member during a time interval)
   The member is down;
   Update node_table and CH_table;
else //hello is received from a member
   Update node_table;
   Perform re-election procedure;
end
if (no hellos from a specific neighboring CH during a time interval)
   The neighboring CH is down;
   Update CH_table;
else // hello is received from neighoring CH
   Update CH_table;
end
if (Join_Request is received from a member)
   Perform merging procedure;
else
   if (Leave_Request is received from a member)
      Counter = Counter - 1;
      Update node_table and CH_table;
   end
   if (CH_Request is received from a node which desires to be CH)
      Accept the request;
      Add a new entry for this CH in the CH_table;
      Send CH_Response to this CH;
      Update the CH_table;
   end
end
```

Figure 2. Clusterhead node algorithm

When the CH receives a Join_Request, it verifies the threshold N which defines the size of the cluster, then it verifies the ratio of power levels of the successive Join_Request messages received from the requester member, which allows getting good knowledge about the link lifetime metric between the CH and the requester node. Hence, the clusterhead does not definitively accept the merging until it is certain that the power level of the last received messages from the member is greater than the power level of the first received messages. In this way, the CH is sure that the member is moving closer to it; otherwise, the CH realizes that the link is going to break and it is no need to add this node in the node_table because it is going to leave soon. Figure 3 shows the merging procedure used to join or to merge multiples nodes within a cluster.

```
Define the threshold of the link durability while storing a historic about the last
Join_Request messages received;
A = The historic indicates that the power signal of the Join_Request decrements rapidely;
B = Verify the threshold on N: the total number of nodes in the cluster;
if (A or B is false)
   Send welcome_NACK to the Join request's source;
else
   if (Join_Request is received from a new arrival node) // the state is not identified
      Send welcome_ACK to the source of Join_Request;
      Wait during a time interval for Join_Accept from the new arrival node;
      if (no Join accept)
         Ignore the request;
      end
   end
   //Join_Request is received from an identified member node or a new arrival node
   Counter = Counter + 1;
   Add the node to the node_table;
   Send a CH_ACK to the new added member;
   Update node_table and CH_table;
   Broadcast hello message to the members and the neighboring CHs;
   Perform re-election procedure;
end
```

Figure 3. Merging procedure

Figure 4 shows the re-election procedure used in order to decide whether to elect or not a CH.

```
Define the threshold of the link durability while storing a historic about the last
Join_Request messages received;
A = The historic indicates that the power signal of the Join_Request decrements rapidely;
B = Verify the threshold on N: the total number of nodes in the cluster;
Verify periodically the hellos received weights from neighbors;
if (there is a new Weight < Weight (CH))
   if (A or B is false)
      Don't perform the re-election and the CH continues its role;
   else // Prepare to the re-election;
      The old CH stays CH until the reception of a confirmation;
      Send database_info(old CH) to the new elected node;
      Wait during a time interval for a database_ACK() from the new elected CH;
      if (no database_ACK)
         Don't perform the re-election and the CH continues playing its role;
      else
         Store in the id CH field the MAC Address of the new elected node;
         Update node_table and CH_table;
         Send CH_info() to the elected node;
         Broadcast CH_change() to the members and the neighboring CHs;
      end
   end
else
   Don't perform the re-election and the CH continues playing its role;
end
```

Figure 4. Re-election procedure

It is favorable when the CH stays in the cluster for a longer time, as time need not be spent in re-election of a CH frequently. The re-election is not periodically invoked; it is performed just in case of a lowest received weight, it allows minimizing the generated overhead encountered in previous works. As we explained above, the re-election may not result a new CH, it depends on the stability of the new node for playing the CH's role. In the case where a new CH must be elected, the procedure should be soft and flexible in order not to perturb the clusters while to copying the databases from the old CH to the new CH.

We guarantee that the re-election is done in a transparent manner vis-à-vis the members and the neighboring clusters, we limit the execution of the algorithm where there is a CH or a network change in order not to impact the whole ad hoc topology. Thus the furthest nodes are not affected by any problem which occurs in other clusters; therefore they are up to date about the size's changes of any cluster in the network. This helps us in our future works to accomplish the inter-clusters communications in the entire network.

3.4 Member Nodes Mechanism

Note that after joining a cluster, the node declares itself as a member of this cluster. Hence, it calculates periodically its weight and sends periodically Hello messages to its CH. As a member, this node should just handle the Hello, the welcome_NACK, the CH_info and the database_info messages received from the clusterhead nodes (see table 1). This allows optimizing the resources (bandwidth, battery, etc) and minimizing the job of the nodes.

When the node receives a Hello from its CH, the node has to update its node_table. When the node receives Hellos from the neighboring CHs, the node has the possibility to migrate to another CH if there is a Hello which has a smaller weight than the current CH's weight, it sends a Join_Request to the CH which is Hello's source and continues as a member of the current CH until the reception of CH_ACK. In this case, the node can send a Leave_Request to the last CH. This method allows us to minimize the number of the formed clusters in the network.

When the node member receives a CH_info message as a result of the re-election procedure, thus it realizes that it is going to become the new CH in the cluster. When a node member does not receive any message from its CH, it considers that the CH has gone brusquely down; in this case, the nodes have no choice and must restart the clustering setup procedure.

4. PERFORMANCE ANALYSIS

We are interested in studying the performance of the following:

- Average number of clusters: taken as stand for the quality of the cluster maintenance algorithm.
- Average transition number on each CH: defines the number of times an elected CH changes its state from CH to a node member.

4.1 Simulation environment and parameters

The simulations scenarios were randomly generated using our scenarios generator [12] which allows inputting parameters such as min and max speed, pause times, area, Hello interval, number of nodes, terrain configuration and the mobility model. At the physical layer, the generator uses a radio model that takes into account the path-loss, the used terrain which is a 3D environment, the frequency and the transmission range defined for the scenario.

We note that the maximum radius of a mobile radio when operating at full transmission power and having an effective communication range is 300 meters which is a design parameter of some IEEE 802.11 products. The simulation parameters have been listed in table 2. In the following simulation, all the nodes follow the Random Walk Mobility Model used in the scenario generator with speed ranging from 3 Km/h to 10 Km/h. In order to study the effect of the network density on the resulting topologies and to evaluate the cluster maintenance algorithm, we varied the number of the nodes inside the terrain and the power transmission range parameter. We study the stability of the ad hoc network in terms of number of formed clusters and number of transition on each CH for different transmission ranges and network densities.

Table 2. Simulation Parameters

Parameter	Meaning	Value
N	Number of nodes	20 - 100
X x Y	Size of the network	500m x 500m
Speed	Speed of the nodes	3 – 10 Km/h
R	Transmission range	30 – 300 m
PT	Pause Time	0 sec
HI	Hello Interval	5 sec
Frequency	Frequency band	5.4 GHZ
Cluster size	Number of members	15
Duration	Time of simulation	300 sec

4.2 Simulation results and discussion

The number of nodes used in the simulation results varies between 20 and 100. The simulations were run for 300 seconds. The cluster size was fixed at 15. We depict some statistics on the formed clusters for different transmission ranges. In the first set of simulations, the scalability of the algorithm is measured in terms of nodes density and transmission range. Figure 5 and 6 show the performance of EMAC for networks which are different in number of nodes and transmission ranges while varying the nodes speed between 3 and 10 Km/h throughout the entire simulation period.

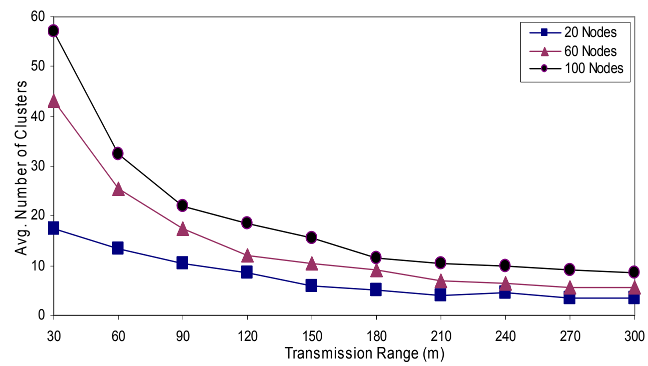

Figure 5. Transmission range vs. Avg. Number of Clusters

Figure 5 shows that for small ranges, most of nodes remain out of each other's transmission range, thus the number of clusters is relatively high and the network may become disconnected because there are no other choices. When transmission range increases, more nodes can hear each other. Hence, the average number of clusters formed decreases and the clusters become larger in size.

On the other hand, when the transmission range begins to be larger, mobile nodes tend to remain in the range of their neighbors and the number of transitions decreases. In figure 6, when the transmission range is very small, most of nodes form one node cluster which only consists of itself. Due to our algorithm design, which requires one-node clusters to attempt to merge with neighboring clusters with less number of nodes whenever possible, clusterhead will switch their status to non-clustered state in order to merge with their neighbors (if any). This causes the high rate of transitions in disconnected networks. However, we argue that this will not affect network performance as this will only occur when the network is disconnected (A disconnected network is unable to function too).

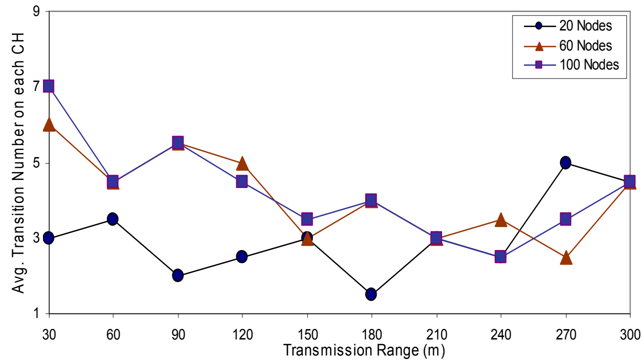

Figure 6. Transmission range vs. Avg. Transition Number on each CH

We also compare the performance of our approach with the corresponding performance of the WCA algorithm while the nodes are moving under the same conditions. In figure 7, we note that the performance difference is small between WCA and EMAC with respect to the average number of clusters. This is because both algorithms are variations of a local weight based clustering technique that forms one-hop clusters.

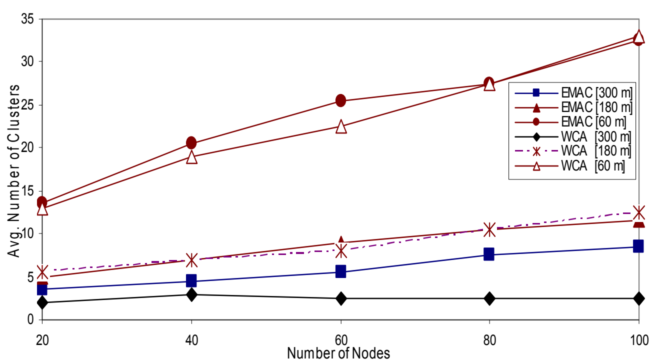

Figure 7. Number of Nodes vs. Avg. Number of Clusters

For high transmission range (more than 250 m), WCA generates less CH than EMAC but to the detriment of a large number of transition on each CH, where the stability is one of the important criteria in clustering because the frequent changes of CH adversely affect the performance of the clustering algorithm.

As shown in figure 8, with 100 nodes in the ad hoc network and for a transmission range equal to 180m, the proposed algorithm produced about 50.0% to 83.3% less transitions than WCA. As a result, our algorithm gives better performance in terms of stability when the node density in the network is high.

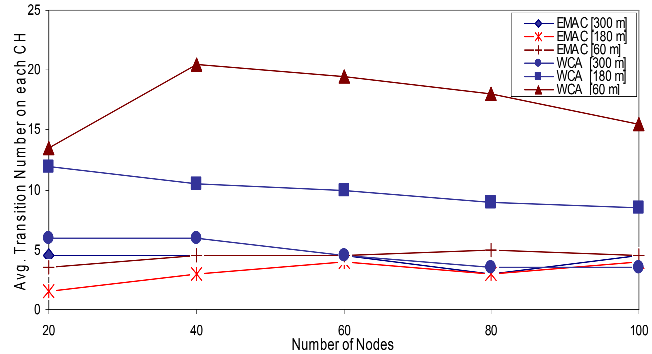

Figure 8. Number of Nodes vs. Avg. Transition Number on each CH

This also resulted in more lifetimes of the CH. In the WCA algorithm, WCA will keep changing with changes in topology. The CH of WCA algorithm relinquishes its position when another node having lower weight joins the cluster. In our algorithm, the CH has to verify the suitability of a new election even if a new node having lower weight has joined the cluster.

5. CONCLUSION AND FUTURE WORKS

This paper has presented an Efficient Management Algorithm for Clustering in ad hoc networks. It has the flexibility of assigning different weights and takes into account a combined metrics to form clusters automatically. Limiting the number of nodes inside a cluster allows restricting the number of nodes catered by a clusterhead so that it does not degrade the MAC functioning. For a fixed clusterhead election scheme, a clusterhead with constrained energy may drain its battery quickly due to heavy utilization. In order to spread the energy usage over the network and achieve a better load balancing among clusterheads, re-election of the clusterheads may be a useful strategy; the algorithm is executed only when there is a demand. Also, if a node is moving away from the clusterhead, then the algorithm is flexible and cheap enough to be applied iteratively as the network configuration changes. Therefore, such approach provides a reliable method of cluster organization for wireless ad hoc networks. Simulation results indicated that the model agrees well with the behavior of the algorithm. Eventually, the route between two nodes changes constantly as the clusterhead set changes. In future works, we will study the impact of the overhead generated by the routing and allow a load balancing between the clusters to complete the inter-clusters communications.

6. REFERENCES

[1] Das B., Bharghavan V., "Routing in ad-hoc networks using minimum connected dominating sets," IEEE International Conference on Communications (ICC Montreal 97), vol. 1, Jun. 1997, pp. 376-380.

[2] Gerla M., Tsai J. T. C., "Multicluster, Mobile, Multimedia Radio Network," ACM/Baltzer Wireless Networks Journal 95, vol. 1, Oct. 1995, pp. 255-265.

[3] Baker D.J., Ephremides A., "A distributed algorithm for organizing mobile radio telecommunication networks," Proceedings of the 2nd International Conference on Distributed Computer Systems, Apr. 1981, pp. 476-483.

[4] Baker D.J., Ephremides A., "The architectural organization of a mobile radio network via a distributed algorithm," IEEE Transactions on Communications, Nov. 1981, pp. 1694-1701.

[5] Ephremides A., Wieselthier J. E., Baker D. J., "A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling," Proceedings of the IEEE, vol. 75, no. 1, Jan. 1987, pp. 56-73.

[6] Basagni S., "Distributed clustering for ad hoc networks," Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks, Jun. 1999, pp. 310-315.

[7] Basagni S., "Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks," Proceedings of Vehicular Technology Conference, VTC, vol. 2, fall 1999, pp. 889-893.

[8] Basu P., Kham N., Little T. D. C., "A mobility based metric for clustering in mobile ad hoc networks," International Conference on Distributed Computing Systems Workshop, Apr. 2001.

[9] Chiang C. C., Wu H. K., Liu W., Gerla M., "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel," IEEE SICON, Singapore, Apr. 1997.

[10] Parekh A.K., "Selecting routers in ad-hoc wireless networks," Proceedings of the SBT/IEEE International Telecommunications Symposium, Aug. 1994.

[11] Chatterjee M., Das S.K., Turgut D., "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks," Cluster Computing Journal, vol. 5, no. 2, Apr. 2002, pp. 193-204.

[12] Agba L., Gagnon F., Kouki A., "Scenarios generator for ad hoc networks," International Symposium on Industrial Electronics, Montreal, Canada, Jul. 2006