A PERFORMANCE COMPARISON OF CLUSTERING ALGORITHMS IN AD HOC NETWORKS

by

CHUN SUM YEUNG
B.S. University of Central Florida, 2005

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2006

ABSTRACT

An ad hoc network is comprised of wireless mobile nodes without the need of wired network infrastructure. Due to the limited transmission range of nodes, the exchange of data between them may not be possible using direct communication. Partitioning the network into clusters and electing a clusterhead for each cluster to assist with the resource allocation and data packet transmissions among its members and neighboring clusterheads is one of the most common ways of providing support for the existing ad hoc routing protocols. This thesis presents the performance comparison of four ad hoc network clustering protocols: Dynamic Mobile Adaptive Clustering (DMAC), Highest-Degree and Lowest-ID algorithms, and Weighted Clustering Algorithm (WCA). Yet Another Extensible Simulation (YAES) was used as the simulator to carry out the simulations.

ACKNOWLEDGMENTS

I would like to give a special thank to my advisor Dr. Turgut for her guidance and patience during this past year. Also, I would like to thank Dr. Bölöni, who is one of the main developers of the YAES simulator, for providing invaluable help with the details of YAES.

I would like to also extend my gratitude to the committee members of the thesis defense, Dr. Bauer and Dr. Sundaram for sparing time from their busy schedules.

Also, I would also like to thank School of Electrical Engineering and Computer Science at University of Central Florida for providing BS+MS fellowships to students such as myself to pursue a graduate degree in the field of Electrical and Computer Engieering.

Finally, I would like to thank my parents for supporting me in all these years. Thanks for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A mobile ad hoc network (MANET) consists of a number of mobile nodes equipped with a transmitter and a receiver. MANET was envisioned to create a network dynamically on-the-fly without relying on any wired infrastructure. That is why, they are also called "infrastructureless networks". Unlike the infrastructure-based networks such as a cellular network, all the components of an ad hoc network is highly mobile and due to this mobility, the topology of the network changes dynamically. The base station in cellular networks is analogous to the clusterhead in ad hoc networks; however, the difference is that base stations are stationary while the clusterhead themselves are also mobile.

Fixed wireless networks usually exist in a form of a master slave relationship. However, MANETs do not share this characteristic. Nodes rely solely on each other to established communication links and act as routers to convey data packets between source and destination pairs. Since the data packet may need to travel from a source to a destination node through a set of intermediate nodes, yet another name for ad hoc networks is "multi-hop networks".

Due to the dynamic nature of mobile ad-hoc networking, restrictions imposed on protocols are challenging. The research groups in academia and in research laboratories are working on design and implementation of protocols with the aim of achieving stability in terms of network formation.

1.1 Motivation of clustering

Clustering is used to partition an ad hoc network into some smaller groups yet all the partitioned clusters function as a whole. Each cluster is comprised of a number of ordinary nodes, gateway nodes, and a clusterhead. If a node wishes to transmit data outside of its own individual partition, it can go through a *clusterhead*, or the clusterhead delegates its authority to a *gateway* of where it belongs to initiate the transmission. Therefore, partitioning is more power efficient. Clustering can also be used for transmission management, backbone formation and routing efficiency. In summary, we can say that cluster-based control structures provides more efficient use of resources for large dynamic networks.

Figure 1.1 - 1.4 shows an example of how a network is partitioned into clusters. In Figure 1.1 all nodes are individuals with no local clusterhead in place. After defining the neighbors of each node in the network, Figure 1.2 shows the minimum coverage needed for all the nodes in the network. In Figure 1.3, the clusterheads are identified. Finally, in Figure 1.4, the network is partitioned.

1.2 Two phases of a clustering algorithm

In [4], Basagni explains that clustering has at least two phases in its existing form: the set up phase and the maintenance phase. Since both are equally important and critical, we provide a brief description of each phase below.

Clustering setup phase

As the name indicates, a setup procedure has to be performed before a network is formed and partitioned as small groups. Each node in the network has to locate its own neighbors.

Generally, there are three types of roles in the clustering algorithm: clusterheads, gateways, and ordinary nodes. Each role is assigned based on the specifications of the clustering algorithm at a specific time frame. Role of each node is dynamically changing corresponding to many criteria such as its current location, weight, unique id, energy level, number of neighbors and so on to calculate its role at a particular instant.

After the setup phase is finished, a set of clusterheads is determined and the set is called a *dominant set*. The dominant set will change dynamically from time to time depending on the formation of the network at any given instant.

Clustering maintenance phase

After the setup phase is completed, the algorithm enters the maintenance phase. Maintaining a network is dynamic and even more challenging since more than ninety percents of a network life time remains in this phase. The maintenance phase makes sure that all the nodes inside the network has at least one connection to its neighbors. If the connection between a node and its neighbor is no longer active due to various reasons, such as topology changes and so on, the node will try to reaffiliate itself to another clusterhead.

When reaffiliation is taking place, all active nodes eliminate any nodes which is no longer active in its cluster from the neighbor list and add new nodes joining to the cluster as neighbors. If a node is not able to reattach to an existing clusterhead in the dominant set, then the clustering algorithm is reinvoked and network is partitioned to new clusters. At this time, the dominant set may change.

1.3 Contributions

This thesis presents a performance evaluation of four clustering algorithms in ad hoc networks. The algorithm studied are Lowest-ID [2, 1], Highest-Degree [12], Dynamic Mobile-Adaptive Clustering (DMAC) [4], and Weighted Clustering Algorithm (WCA) [7]. The simulation study has been performed using YAES [20] simulator. The algorithm have been compared in terms of three simulation metrics: i) average number of clusterheads; ii) reaffiliation counts; and iii) dominant set updates. Each metric has been simulated with varying transmission range and max displacement parameters.

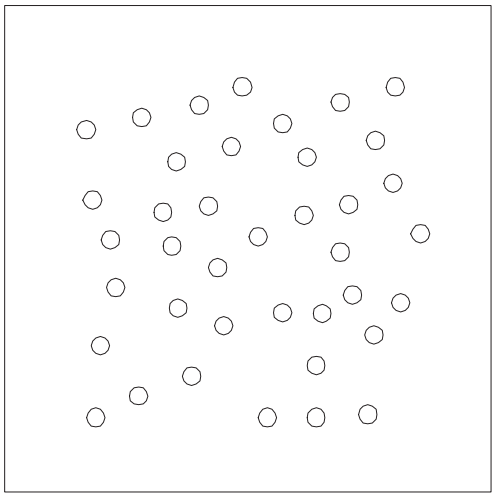

Figure 1.1: Nodes in a network without being clustered.

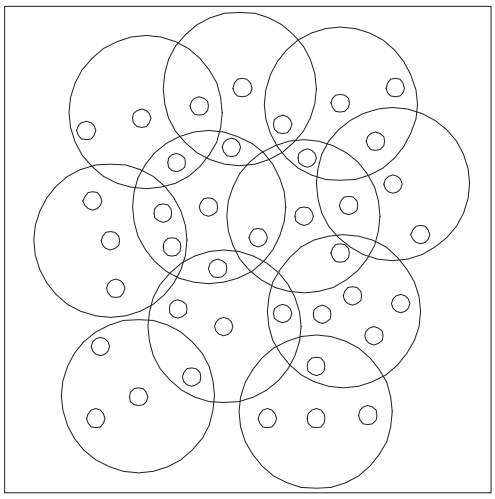

Figure 1.2: Nodes in a network with transimission ranges defined.

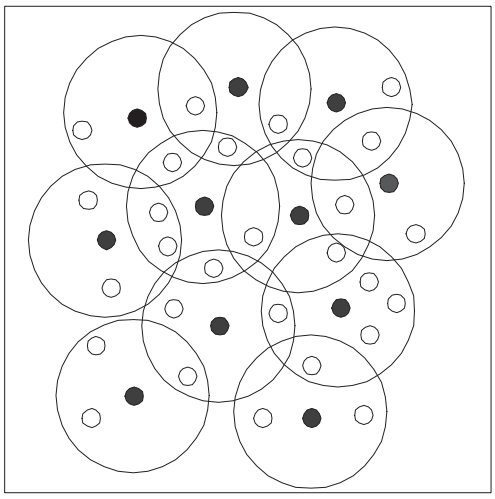

Figure 1.3: Nodes in a network with clusterheads identified.

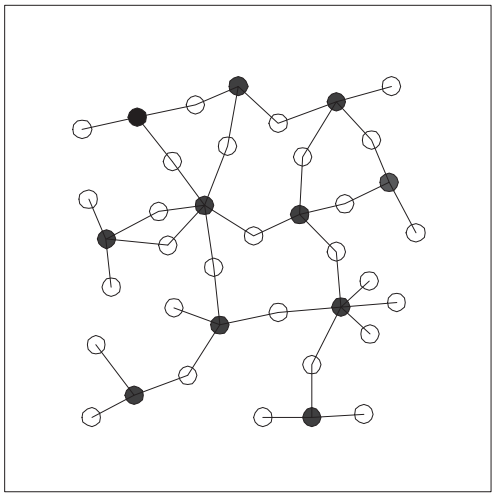

Figure 1.4: Nodes partitioned after clustering algorithm applied.

CHAPTER 2

RELATED WORK

In this chapter, we present the related work of the clustering algorithms in mobile ad hoc networks.

In [8], the authors have proposed a clustering algorithm that aims at maximizing the lifetime of the network by determining optimal cluster size and optimal assignment of nodes to clusterheads. They assume that the number of clusterheads and the location of the clusterheads are known, which is not possible in all scenarios. Moreover, the algorithm requires each node to know the complete topology of the network.

In [21], the authors presented an energy-efficient distributed clustering approach, called *Hybrid Energy-Efficient Distributed clustering* (HEED), for the ad hoc sensor networks. HEED operates in quasi-stationary networks. Their approach randomly selects clusterheads based on their residual energy and to minimize communication cost. There are no assumptions made about the node dispersion. The result of this work shows that HEED prolongs network lifetime and supports data aggregation.

Mobility-based d-hop clustering algorithm (MoDHop) [10] is a distributed algorithm which stresses stability. Similar to other clustering algorithms MoDHop first generates nonoverlapping clusters in the network. After all the nodes are partitioned, gateways are elected and a merging process takes place if the node can hear more than one clusterhead at a time. However, the merging process will start when certain stability of the cluster is met. MobDHop is designed to be used in large scale networks.

In [11], authors have introduced an algorithm for efficient and energy-balanced clustering of mobile ad hoc networks, called Lowest ID with Adaptive ID Reassignment algorithm (LIDAR). The algorithm is a modified version of Lowest-ID algorithm. LIDAR takes both mobility and the energy consumption of nodes into consideration to provide a more stable cluster set. Their algorithm reduces control traffic volume and also increases broadcast frequency for highly mobile network configurations.

In [16], the authors have optimized the weighted clustering algorithm (WCA) [7] with genetic algorithms. Each chromosome contains information about the clusterheads and their members as it is obtained in the original WCA. Then genetic algorithm uses the information collected to obtain the chromosome defined by the fitness function. Each clusterhead is expected to handle the maximum possible number of nodes in its cluster to obtain the optimal operation of the medium access control (MAC) protocol. Therefore, it leads to the minimum number of clusterheads. Simulation results show that improved performance of the optimized WCA is better than the original WCA.

In [17], the authors demonstrate how simulated annealing algorithm can be applied to optimize performance of WCA [7]. The simulated annealing stands to be a powerful stochastic search method. Simulated annealing uses the information collected from the original WCA in [7] to find the best solution defined by computing the objective function and obtaining the best fitness value. Each clusterhead handles the maximum possible number of mobile nodes in its cluster in order to obtain the optimal operation of the MAC protocol. Consequently, it results in the minimum number of clusterheads. Simulation results show that the performance of the optimized WCA is better than that of the original WCA.

In [14], the authors propose a new multicast routing protocol, "Weight-Based Clustering Multicast Protocol" (WCMP) for the mobile ad hoc networks and compared the performance of WCMP with ODMRP protocol. Both protocols degrade with the increasing number of mobile nodes. However, WCMP can keep the stability of the clusterheads while ODMRP is not able to.

In [5], the authors compare protocols such as DCA, DCA-S, WuLi and WAF. These protocols are all distributed, localized and deterministic algorithms for computing a backbone. DCA-S stands for *sparsification* DCA and it has duration similar to original DCA. WuLi [19] is the fastest protocol due to its low complexity; WAF is the slowest protocol in the paper due to the nontrivial complexity of the first phase.

In [13], the authors investigated through simulation on the impact of mobility over both DMAC and GDMAC. GDMAC stands for Generalized DMAC which is the specific version of DMAC. The two protocols have been implemented in the ns-2 simulator [18]. The results show that lifetime for both DMAC and GDMAC decreases with the increasing speed as nodes tend to drift away faster with high velocity. GDMAC tends to be more stable than its specific variation DMAC. In addition, it was observed GDMAC is also effective in reducing the clustering overhead imposed by mobility and its maintenance cost.

CHAPTER 3

AD HOC NETWORK CLUSTERING ALGORITHMS STUDIES

In this chapter, we present the descriptions including the pseudocodes of all four clustering algorithms in detail. For consistency, all pseudocode presented in the section will use v as the node which is currently executing the procedure, node$u$ is the neighbor of v which is being applied the procedure on and z is the neighbors of node v respectively.

3.1 Lowest-ID algorithm

The *Lowest-ID algorithm* [1, 2] , also known as *identifier-based clustering*, provides three different roles for the nodes: *original, gateway* and *clusterhead* nodes. The algorithm of electing a clusterhead is as follows:

Step 1. Each node is assigned an unique id in the setup process.

Step 2. The node assigned with lowest id in its group will be the clusterhead of that group. Therefore, the ids in the group will be higher than that of the clusterhead.

Step 3. If a node lies in two different clusters, it will be set as a gateway. The functionality of a gateway is to act as a bridge or a connection between two or more clusters in the network to ease the workload of a clusterhead.

Step 4. No clusterhead can be neighbors of each other. In order to have an inter-cluster connection, messages have to go through a gateway node.

The pseudocode of the algorithm is given below. To identify a gateway node candidate, the node v checks if any of its neighbors z is a clusterhead. If there is more than one neighbor which is a clusterhead, meaning that v lies between different clusters, v will be set as a gateway and the boolean variables isClusterHead(-) and isGateWay(-) will be set to false and true accordingly, and v will be withdrawn from the election process. The neighboring clusterheads of the node v is denoted as neighboringClusterHeads(v).

```
definingGateways( )
    start
        for  v : neighbors(z)
            if  isClusterHead(u) == true
                    neighboringClusterHeads(v).add(u)
            end if
        end for
        if  neighboringClusterHeads(v) > = 2
            isClusterHead(v) == false
            isGateWay(v) == true
        end if
    end
```

In order to look for a possible clusterhead, the *creatingClusters*( ) is run until the entire network is partitioned. The node v broadcasts its id to the network to see if there are any neighbors surrounding it. If there are not any, v becomes a clusterhead itself. Once a node decided its own role, it will quit the algorithm and the boolean variable isMarked(-) will be set to true.

```
creatingClusters( )
    start
        while condition == true
            for v : activeNodes
                if isMarked(v) == false
                        possibleClusterHeads.add(v)
                end if
            end for
            if possibleClusterHeads == 0
                condition == false
            end if
            for u : possibleClusterHeads(x)
                if nodeID(u) < nodeID(possibleClusterHeads(x))
                    x = u
                end if
            end for
            v = x
            isMarked(v) = true
            findNeighbors(v)
            for v : neighobrs(z)
                if isMarked(u) == false
                    clusterContent(v).add(u)
                    isMarked(u) == true
                end if
            end for
        end while
    end
```

findNeighbors( ) is used to find all the neighbors a node v has at that instant. v will broadcast its node id to the neighbors z which lie in the transmission range. if this is the case the neighbor will be added to v's neighborList.

```
findNeighbors( )
    start
        for  v : activeNodes
            if  distance(v, x) <= tx_range(v)
                neighborList(v).add(x)
            end if
        end for
    end
```

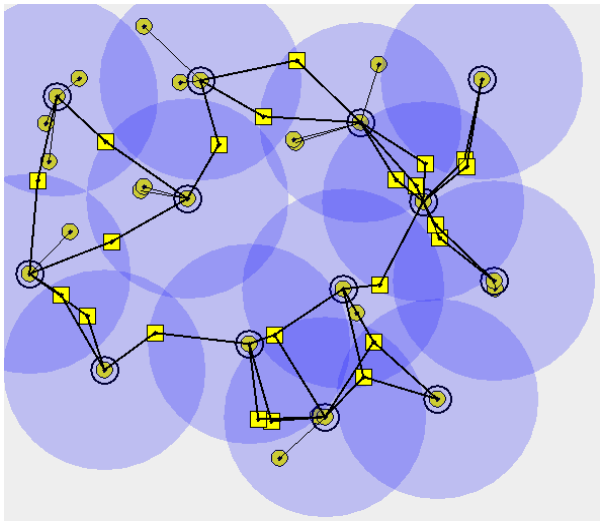

Figure 3.1: Lowest-ID algorithm sample cluster formation.

3.2 Highest-Degree algorithm

Highest-Degree algorithm [12], as known as *connectivity-based clustering*, was one of the first developed clustering algorithms used in ad hoc networks. Similar to Lowest-ID algorithm, a network consists of two major components, clusterhead and ordinary node. Functionality of a clusterhead node is to control the local traffic of the nodes in the cluster. The algorithm of electing a clusterhead is as follows:

Step 1. Each node is assigned a unique id in the network.

Step 2. After an id is obtained, the node broadcasts its id to other nodes which are inside of its transmission range.

Step 3. Any node receiving the signal is included as a part of the neighbors list of that node.

Step 4. The most number of neighbors a node has determines the node being elected as the clusterhead of the group.

Step 5. If there is a tie, which means there are multiple nodes with the some number of maximum neighbor nodes in that group, the node with lower id is elected as the clusterhead of that group.

Step 6. Similar to the Lowest-ID algorithm, two or more clusterheads cannot be neighbors to each other simultaneously.

Step 7. Repeat steps 2 through 6 until the remaining nodes in the network become a clusterhead or join the clusterhead.

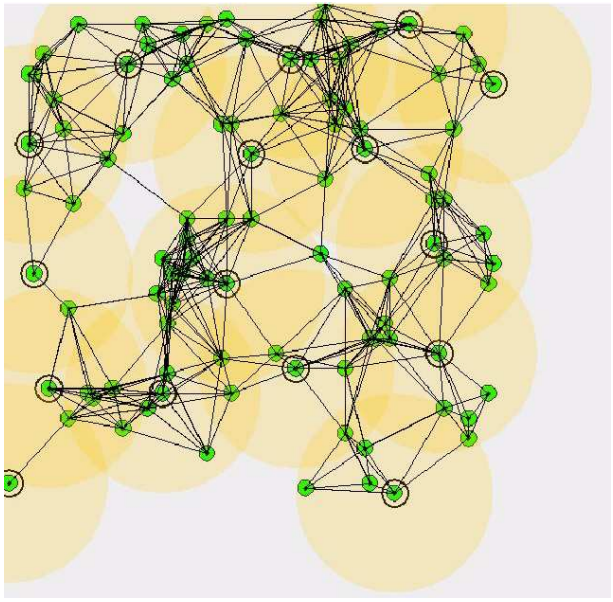

Figure 3.2: Highest-Degree algorithm sample cluster formation.

Figure 3.2 shows how clusters are formed after the Highest-Degree algorithm is applied. Nodes having most neighbors within range will be elected as clusterheads. In the figure, the nodes with a shell are clusterheads and the other are ordinary nodes. In this algorithm, no gateway is presented and clusterheads can not be direct neighbors.

creatingClusters( ) is used to identify clusterheads in the network. The algorithm searches for all the active nodes in the network and see if there exist any possible clusterheads in the network. To be considered a clusterhead candidate, the node cannot be marked as part of a cluster, or it cannot be in a transmission range of a clusrerhead. The pseudocode of the algorithm given below shows that the algorithm keeps looking for all possible clusterheads until all nodes are marked and the whole network is partitioned.

```
creatingClusters( )
    start
        while condition == true
            for v : activeNodes
                if isMarked(v) == false
                        possibleClusterHeads.add(v)
                end if
            end for
            if possibleClusterHeads == 0
                condition == false
            end if
            for v : possibleClusterHeads
            findNeighbors(v)
            end for
            for u : possibleClusterHeads(x)
                if neighborList(u).size() == neighoborList(possibleClusterHeads(x)).size()
                    if nodeID(u) > nodeID(possibleClusterHeads(x))
                        x = u
                    end if
                end if
                if neighborList(u).size() > neighoborList(possibleClusterHeads(x)).size()
                    x = u
                end if
            end for
            v = x
            isMarked(v) = true
            findNeighbors(v)
            for v : neighobrs(z)
```

```
                    if isMarked(u) == false
                        clusterContent(v).add(u)
                        isMarked(u) == true
                    end if
                end for
            end while
        end
```

findNeighbors( ) is used to find all the neighbors a node v has at that instant. v will broadcast its node id to the neighbors z which lie in the transmission range. if this is the case the neighbor will be added to v's neighborList.

```
findNeighbors( )
    start
        for v : activeNodes
            if distance(v, x) <= tx_range(v)
                neighborList(v).add(x)
            end if
        end for
    end
```

This is a relatively straight forward approach to determine a clusterhead. Optimization is not taken place in the election process. This method has a disadvantage that production rate becomes inefficient when the number of nodes in a cluster increases.

3.3 Node-weight algorithm

Distributed Mobility-Adaptive clustering (DMAC) [3] is the enhanced version of *Distributed Clustering Algorithm (DCA)* [4]. DCA works the best in *quasi-static network* meaning that the nodes move with a low speed. However, for a dynamically changing environment, DMAC works better than DCA since the nodes are aware of the new neighbors joined and neighbors which are no longer in the neighbors list.

There are at least three conditions to be met in order to implement DMAC and DCA algorithms.

1. Each partition has to have a clusterhead for its own.
2. Each ordinary node will affiliate with the neighboring clusterhead, and the weight of the clusterhead is bigger than that of it.
3. Clusterheads cannot be direct neighbors.

The algorithm of DCA is given as follows:

Step 1. Every node runs the algorithm simultaneously. There are two types of messages, CH and JOIN messages. The node with highest weight among its neighbors sends a CH message.

Step 2. At least one node in the cluster sends out the CH message.

Step 3. Other nodes in the cluster waits to receive a message. The message may not necessarily be CH message.

Step 4. If a node receives the CH message, which means that its id is not the highest among the neighbors, it checks if it belongs to a cluster yet. If not, it sends a JOIN message and joins the cluster.

Step 5. Repeat steps 2 through 4 until the remaining network is partitioned.

The algorithm of DMAC is similar to that of DCA, yet DCA only takes care of the setup phase of the clustering process and DMAC handles both the setup and maintenance phases. The algorithm of DMAC is as follows:

Step 1. Every node runs the algorithm simultaneously. For simplicity, there are only two types of messages: CH and JOIN messages.

Step 2. A random weight is assigned to each node. The node whose weight is the highest among its neighbors sends a CH message to the neighbors.

Step 3. At least one node in the cluster sends out the CH message to its neighbors.

Step 4. The nodes with lighter weights waits for a CH message to decide which clusterhead to join.

Step 5. By the time a node receives a CH message from a neighbor, it will join the cluster if it has not yet joined a cluster.

Step 6. Upon receiving a CH message, the node checks if it has already heard from all of its neighbors. If this is the case, a JOIN message will be sent to the node which sent the CH message and joins the cluster.

Step 7. Repeat steps 2 through 6 until the remaining network is partitioned.

Due to the dynamic nature of the network, the algorithm has an ability to check if nodes are detached or attached to a cluster at an instant. *failureLink*( ) and *newLink*( ) procedures will be run accordingly depending on the situation.

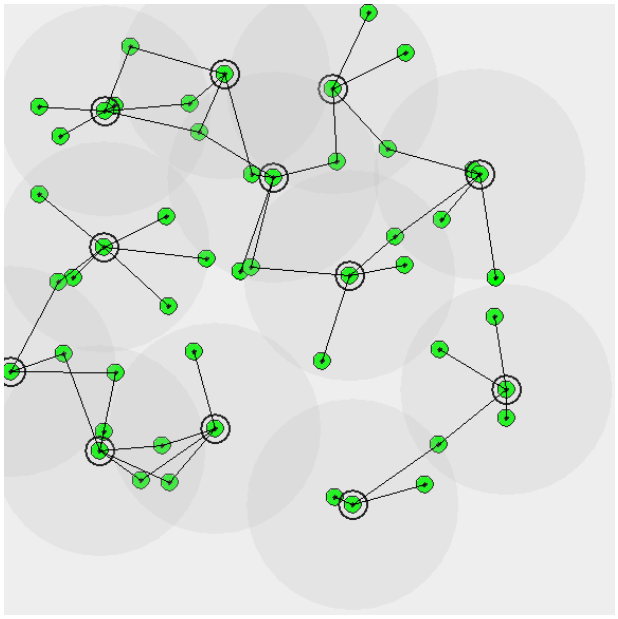

Figure 3.3: DMAC algorithm sample cluster formation.

Both DCA and DMAC are passive algorithms, which means that after the active initialization process, they become message driven. Thus, the nodes in the network act according to the messages received from the neighboring nodes and remain in sleep mode when no events occur. Passive algorithms have an advantage since only minimal energy will be consumed when a node is in either an idle or a sleep mode. This greatly helps prolonging the life span of the network.

Figure 3.3 shows the cluster formation after DMAC is applied. Nodes with a shell are clusterheads and the rest is ordinary nodes. Note that there are no gateway nodes in the network, and being a clusterhead does not necessarily have most neighbors. Moreover, the clusterheads cannot be direct neighbors.

The pseudocode of the algorithm is presented below for the convenience [3, 4]. Before *init*( ) is run, all nodes, which denotes as v, in the network have already been assigned with a random weight and a node id. Each of them also knows their neighbors, denoted as z. All nodes with its variables masterClusterHead(-), clusterContent(-), and isCHMessageSent(-) are initialized as null, zero and false respectively. When a node has not decided its role, it is treated as an ordinary node.

When a network is in the setup phase, or there is a new node joining to the network, the algorithm runs the *init*( ) procedure to cluster the network properly. The node v checks if there are any neighbors whose weights are greater than of node v's. If this is the case, node v joins that node and quits the algorithm. If there is no neighbor whose weight is greater than that of node v's, v becomes a clusterhead itself.

```
init( )
    Start
        if ! isCHMessageSent(z == 0)
            possibleClusterHead = w_max(z){neighborList(v)}
            if nodeWeight(v) > nodeWeight(possibleClusterHead)
                possibleClusterHead.receivedJoin(v)
                masterClusterHead(v) = possibleClusterHead
            else
                for v : neighbors(z)
                    neighbor(v).receivedCHMessage(v)
                end for
                isCHMessageSent(v) = true
            end if
        else
            for v : neighbors(z)
                neighbor(v).receivedCHMessage(v)
            end for
            isCHMessageSent(v) = true
        end if
    end
```

When a node v receives a JOIN message from the node u, v checks if it contains u. If this is true, v will remove u from its cluster. If u was not in the cluster, the node v will add u to the cluster.

```
receivedJoin(u)
    start
        if isCHMessageSent(v) == true
            if clusterContent(v).contains(u)
                clusterContent(v).remove(u)
            else
                clusterContent(v).add(u)
            end if
        else if isCHMessageSent(v) == (true)
            init( )
        end if
    end
```

If v receives the CH message from a node u, it checks to see if the weight of u is greater than that of its clusterhead. If this is the case, v tells its neighbors z that it is leaving the current cluster and joining node u's cluster.

receivedCHMessage(u)

start

if nodeWeight(u) > nodeWeight(v)

for v : neighbors(z)

neighbor(v).*receivedJoin*(v)

end for

masterClusterHead(v) = u

end if

if isCHMessageSent(v) == **true**

isCHMessageSent(v) == **false**

end if

end

If there is a link failure, the node v checks if it has sent a CH message and the failure link is within its cluster. If this is the case, the node v removes it from the cluster. If not, the algorithm checks if the failure link is its clusterhead. If and only if this is the case the node tries to implement the *init*( ) again to affiliate with the neighbor which has the biggest weight.

failureLink(u)

start

if isCHMessageSent(v) == **true and** clusterContent(v).contains(u)

clusterContent(v).remove(u)

else if masterClusterHead(v) == u

init( )

end if

end

If there is a new node u appears, the node v checks if the node u is sent a CH Message. If this is the case, the node v only join node u if and only if the weight of node u is bigger than the weight of node v's clusterhead. Node v will not perform any action if both v and u are ordinary nodes.

newLink(u)( )

start

if weight(u) > weight(v)

v.receivedCHMessage(u)

masterClusterHead(v) == u

if isCHMessageSent(v) == **true**

isCHMessageSent(v) = **false**

end if

end if

end

3.4 Weighted clustering algorithm

Weighted Clustering Algorithm (WCA) [7] elects clusterheads based on four specific parameters, namely, degree-difference, sum of the distances, mobility, and battery power.

According to [7], there are eight steps in the clusterhead election process. The steps are described as follows:

Step 1. Find neighbors v of each node in its transmission range which is defined as

$$d_v = |N(v)| = \sum_{v' \in V,\ v' \neq v} \{dist(v, v') < tx_{range}\} \tag{3.1}$$

Step 2. Find the degree-difference, Δ_v, for every node v as

$$\Delta_v = |d_v - \delta| \tag{3.2}$$

where δ is the ideal degree representing the idea number of neighbors a node can have.

Step 3. Find the sum of the distances, D_v, for every node and its neighbors.

$$D_v = \sum_{v' \in\ N(v)} \{dist(v, v') \tag{3.3}$$

Step 4. Find the running average of the speed, M_v, for every node until the current time T to

calculate mobility, M_v, as

$$M_v = \frac{1}{T} \sum_{t=1}^{T} \sqrt{(X_t - X_{t-1})^2 + (Y_t - Y_{t-1})^2} \tag{3.4}$$

where as (X_t, Y_t) is the coordinate of the node v at time t and (X_{t-1}, Y_{t-1}) is the coordinate at time $(t-1)$. Component with less mobility is a better choice for being a clusterhead.

Step 5. For which a node v acts as a clusterhead, find the cumulative time, P_v, which represents the amount of battery power consumed. (assumed it is that a clusterhead drain more power than an ordinary node).

Step 6. Find the combined weight, W_v, for each node v as

$$W_v = w_1\Delta_v + w_2 D_v + w_3 M_v + w_4 P_v \tag{3.5}$$

where the w_1, w_2, w_3, and w_4 are the weighing factors for the corresponding system parameters.

Step 7. Choose the smallest w_v as a clusterhead. This means that the neighboring nodes of the selected clusterhead

are no longer allowed to participate in the election procedure.

Step 8. Repeat Steps 2 through Step 3 for the remaining nodes which have not been selected neither as a clusterhead nor assigned to a cluster.

The weighting factors are defined as

$$\sum_{i=1}^{4} w_i = 1 \tag{3.6}$$

where each weighting factor in is determined depending upon the application and implementation.

Figure 3.4 is an example of how a network is partitioned according to WCA. The thicker lines connect the clusterheads, or dominant set, together. Note that the clusterheads can be neighbors of each other. The thinner line shows the connections of the clusterhead to the ordinary nodes connecting to it.

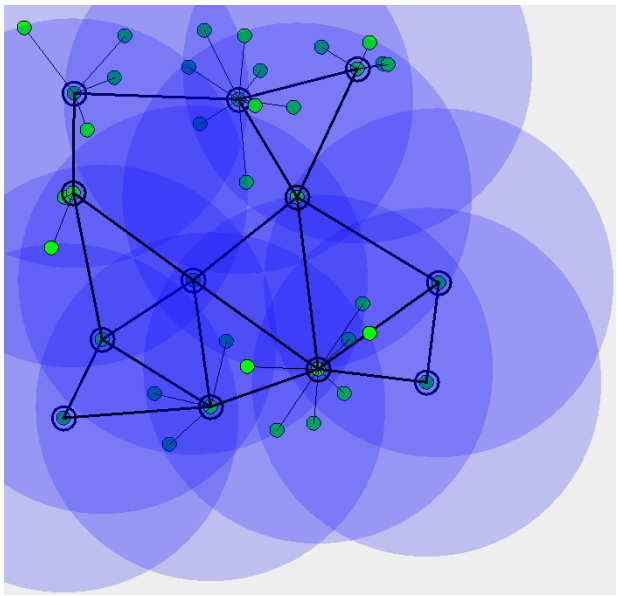

Figure 3.4: WCA sample cluster formation.

CHAPTER 4

SIMULATION STUDY

4.1 Simulation environment and metrics

We have run simulations to achieve performance evaluation among four clustering algorithms: Lowest-ID, Highest-Degree, DMAC and WCA algorithms with a simulator called, Yet Another Extensible Simulation (YAES) [20]. YAES was developed by Networking and Mobile Computing Research Laboratory (NetMoC) [15] at the school of Electrical Engineering and Computer Science at University of Central Florida. YAES is Java-based and it was specifically developed to simulate environments related to mobile computing and networking applications. It supports real-time experimentation and refactoring [6]. Moreover, other than the sample simulations, YAES provides a wide variety of abstract classes, interfaces, and a comprehensive set of libraries to expand its functionalities depending upon the user's needs.

YAES also can generate MATLAB files used to produce simulation graphs. Simulation graphs, including the sample formations of each algorithm were generated by YAES. YAES is intentionally run with Eclipse [9], which is a software development kit (SDK) to run Java applets and applications. Eclipse has a user-friendly interface which helps to develop applets easier.

To test the algorithm more thoroughly, different scenarios will be used. The simulation parameters, their default values and ranges are given in Table 4.1. Simulations take place in a 100 x 100 square area. All nodes are placed in random positions at the beginning and they move randomly within predefined max displacement, mobility, and transmission range.

Each node has an energy level of 100 unit to start with. Energy consumption will be calculated differently depending on the role of a node. Clusterhead nodes consume more energy in a time step than gateway nodes, and the gateway nodes consume more energy than ordinary nodes in a time step. Nodes are assumed to have bi-directional communications.

Transmission range, *tx_range*, is varied from 0 to 100 with an increment of 5 units while keeping the maximum displacement constant. The max displacement, *max_displacement*, is varied from 1 to 10 with an increment of 1 unit while keeping the transmission range constant. The random waypoint mobility model is used with the network size of N = 30. Table 4.1 shows the metrics used in the simulations.

Table 4.1: Simulation Metrics

Simulation Parameters	**Value**
simulation area	$100 \times 100(m^2)$
number of nodes	30
transmission range	0 - 100
maximum displacement	1 - 10(m/s)
simulation time	1000(s)
WCA parameters	$w_1 = 0.7$, $w_2 = 0.2$
	$w_3 = 0.05$, $w_4 = 0.05$

4.2 Simulation results

This section covers the simulation results generated from the algorithms. Figure 4.1 shows the relative performance of the algorithms in terms of the average number of clusters. The transmission range has been varied, but the number of nodes and the max displacement

parameters are kept as constants with the values of 30 and 5 respectively. As we can see, the average number of clusterheads is inversely proportional to the transmission range of each node increases. This is due to the fact that the bigger transmission range a node can cover, the less numbers of clusterheads are required. However, the cluster size will increase correspondingly as the average number of clusterheads decreases. Bigger cluster size implies more neighbors a clusterhead has to handle, which means that the traffic will increase, and there may be excessive drainage of battery of a clusterhead.

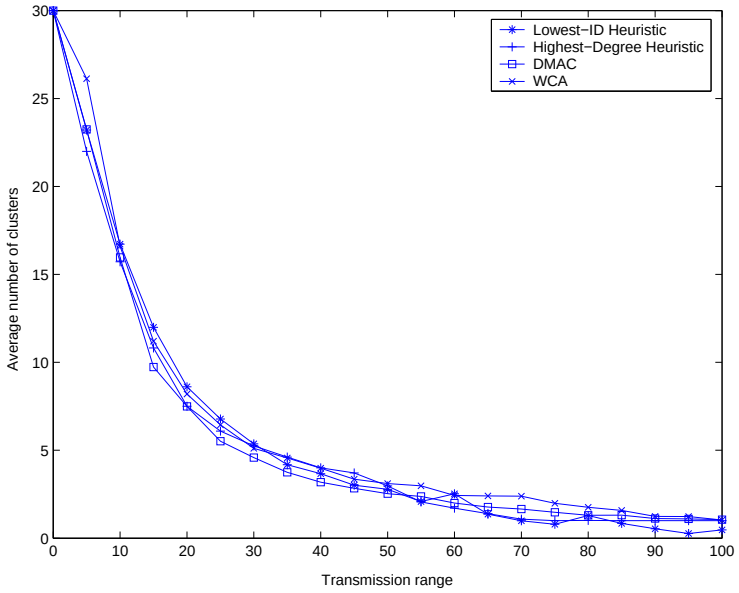

Figure 4.1: Average number of clusters vs. transmission range.

Figure 4.2 shows the relative performance of the algorithms in terms of the reaffiliation counts per time step. The transmission range has been varied, but the number of nodes and the max displacement parameters are kept as constants with the values of 30 and 5 respectively. As we can see, the reaffiliation counts appear to be less when the transmission range increases. This is due to the fact that a node is not likely to detach from its clusterhead as the transmission range is relatively large. The chances for a node moving from one cluster to the other cluster is relatively small with the increased transmission range.

Highest-Degree algorithm shows the worst performance among all the algorithms.

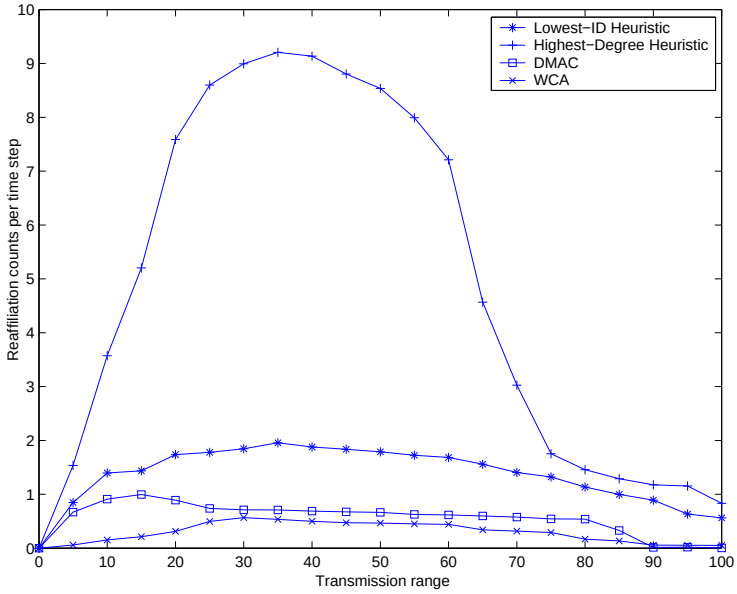

Figure 4.2: Reaffiliation vs. transmission range.

Figure 4.3 shows the relative performance of the algorithms in terms of the dominant set update. The transmission range has been varied, but the number of nodes and the max displacement parameters are kept as constants with the values of 30 and 5 respectively. As we can see from the graph, the smaller the transmission range, the higher the dominant set update is. This is due to the fact that small transmission range makes a node detach from its clusterhead easily. On the other hand, when the transmission range increases, the dominant set update drops significantly since nodes do not easily move out of their clusterheads' transmission ranges.

Figure 4.4 shows the relative performance of the algorithms in terms of the average number of clusters. The maximum displacement has been varied, but the number of nodes and the transmission range parameters are kept as constants with the values of 30 and 15

respectively. As we can see, the average cluster size slightly increases with respect to the increased maximum displacement. The farther away a node can move, the more likely it detaches from its cluster and joins to another cluster.

Figure 4.5 shows the relative performance of the algorithms in terms of the reaffiliation counts per time step. The maximum displacement has been varied, but the number of nodes and the transmission range parameters are kept as constants with the values of 30 and 15 respectively. As we can see, the reaffiliation counts per time step increase significantly with the increased of maximum displacement.

Figure 4.6 shows the relative performance of the algorithms in terms of their the dominant set update. The maximum displacement has been varied, but the number of nodes and the transmission range are kept as constants with values of 30 and 15 respectively. As we can see that as the maximum displacement becomes larger, the nodes move farther away from their clusterhead. Thus, we can draw a conclusion that once the maximum displacement becomes larger, the nodes tend to detach from their clusterheads and attach to other clusterheads.

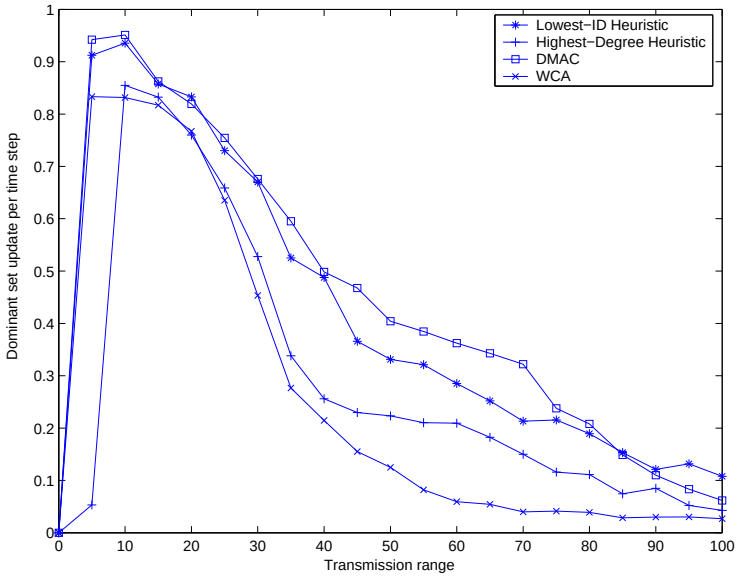

Figure 4.3: Dominant set vs. transmission range.

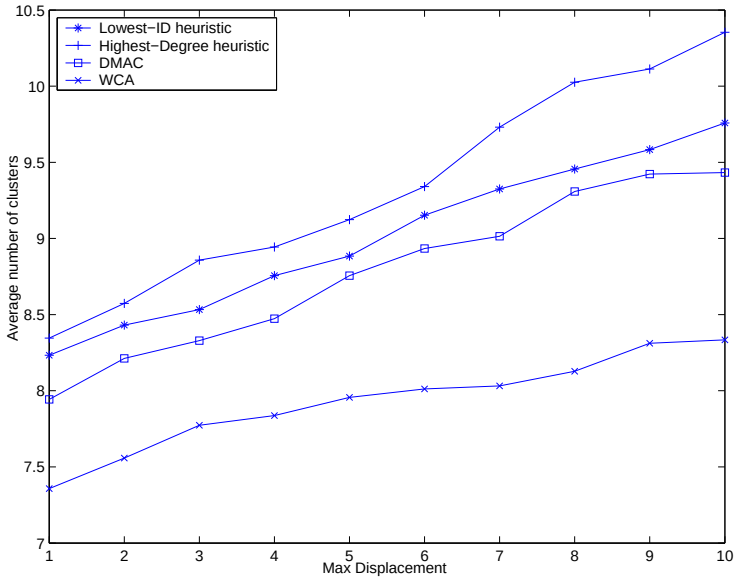

Figure 4.4: Average number of clusters vs. max displacement.

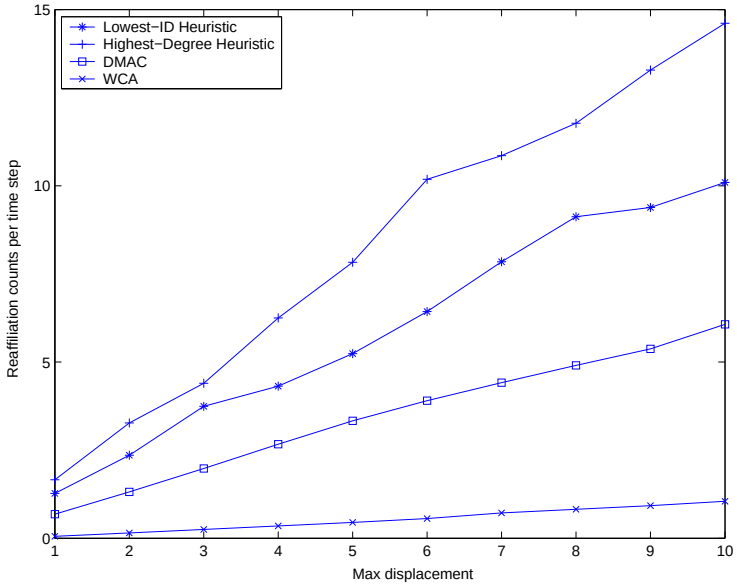

Figure 4.5: Reaffiliation vs. max displacement.

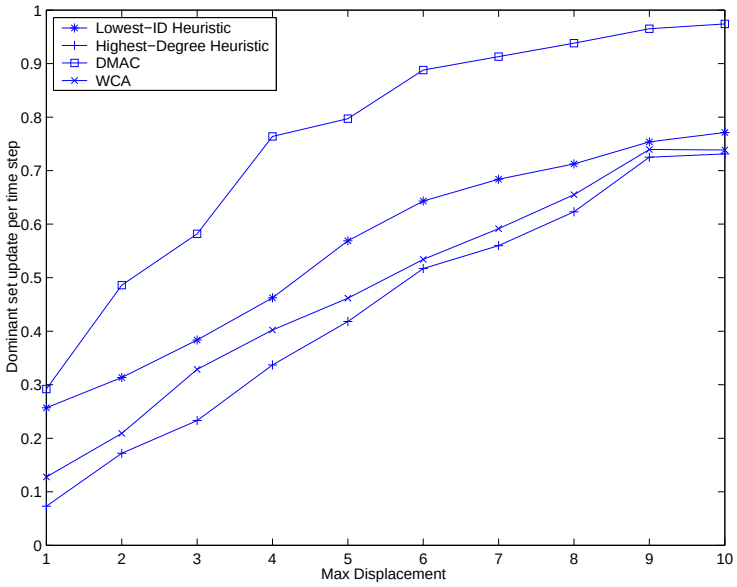

Figure 4.6: Dominant set vs. max displacement.

CHAPTER 5

CONCLUSION

Different algorithms have their own advantages and disadvantages. The Lowest-ID algorithm has its tendency of excessive battery drainage of the nodes with lower node ids due to the clusterhead election criteria. Moreover, it does not balance the load among clusterheads. In Highest-Degree algorithm, a clusterhead may not handle a large number of nodes due to resource limitations since the load handling capacity of the clusterhead puts an upper bound on the node-degree. Therefore, the throughput of the system drops as the number of nodes in cluster increases. DMAC algorithm on the other hand, does not provide a concrete criteria for assigning the node-weights and it works specifically for "quasi-static" networks where the nodes do not move much or move very slowly. The computation overhead of WCA may be high due to the invocation of the election algorithm criteria. More localized solutions may reduce the computation needed to maintain the algorithm.

This thesis presented a performance comparison of the ad hoc clustering algorithms discussed above. These algorithms were compared in terms of average numbe of clusters, reaffiliation counts, and dominant set update for the varying transmission range and the maximum displacement. YAES simulation was used to carry out the experiments and the results demonstrate that WCA performs the best among these algorithms. This is due to the fact that WCA takes into consideration the node degree, sum of the distances, mobility, and power consumption for the clusterhead selection.

REFERENCES

[1] D.J. Baker and A. Ephremides. A Distributed Algorithm for Oraganizing Mobile Radio Telecommunication Networks. In *Proceedings of the 2nd International Conference on Distributed Computer Systems*, pages 476–483, April 1981.

[2] D.J. Baker and A. Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, COM-29 11:1694–1701, November 1981.

[3] S. Basagni. Distributed and Mobility-Adaptive Clustering for Multimedia Support in Multi-Hop Wireless Networks. In *Proceedings of Vehicular Technology Conference, VTC*, volume 2, pages 889–893, Fall 1999.

[4] S. Basagni. Distributed Clustering for Ad Hoc Networks. In *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 310–315, June 1999.

[5] S. Basagni, M. Mastrogiovanni, and C. Petrioli. A Performance Comparison of Protocols for Clustering and Backbone Formation in Large Scale Ad Hoc Networks. In *Proceedings of IEEE International Conference on Mobile Ad hoc and Sensor Systems*, pages 70–79, October 2004.

[6] L. Bölöni and D. Turgut. YAES - A Modular Simulator for Mobile Networks. In *Proceedings of the 8th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, pages 169–173, October 2005.

[7] M. Chatterjee, S.K. Das, and D. Turgut. WCA: A Weighted Clustering Algorithm for Mobile Ad-hoc Networks. *Journal of Cluster Computing (Special Issue on Mobile Ad Hoc Networks)*, 5(2):193–204, April 2002.

[8] C.F. Chiasserini, I. Chlamtac, P. Monti, and A. Nucci. Energy Efficient Design of Wireless Ad Hoc Networks. In *European Wireless*, pages 376–386, February 2002.

[9] Eclipse. URL http://www.eclipse.org.

[10] I.I. Er and W.K.G. Seah. Mobility-Based D-hop Clustering Algorithm for Mobile Ad-hoc Networks. In *Proceedings of IEEE Wireless Communications and Networking Conference, (WCNC)*, volume 4, pages 2359–2364, March 2004.

[11] D. Gavalas G. Pantziou, C. Konstantopoulos and B. Mamalis. Lowest-ID with Adaptive ID Reassignment: A Novel Mobile Ad-hoc Networks Clustering Algorithm. *1st International Symposium on Wireless Pervasive Computing*, pages 1–5, January 2006.

[12] M. Gerla and J. Tsai. Multicluster, Mobile, Multimedia Radio Network. *Journal of Wireless Networks*, 1(3):255–265, 1995.

[13] R. Ghosh and S. Basagni. Limiting the Impact of Mobility on Ad Hoc Clustering. In *Proceedings of The 2nd ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, pages 197–204, October 2005.

[14] C.C. Huang, R.S. Chang, and M.H. Guo. Weight-Based Clustering Multicast Routing Protocol for Mobile Ad Hoc Networks. In *Proceedings of IEEE Wreless Communications and Networking (WCNC)*, volume 2, pages 1112–1117. IEEE, March 2003.

[15] Networking and Mobile Computing (NetMoC)laboratory. URL http://netmoc.cpe.ucf.edu.

[16] D. Turgut, S.K. Das, R. Elmasri, and B. Turgut. Optimizing Clustering Algorithm in Mobile Ad hoc Networks Using Genetic Algorithmic Approach. In *Proceedings of Global Communication Conference*, volume 1, pages 62–66, November 2002.

[17] D. Turgut, B. Turgut, R. Elmasri, and Than V. Le. Optimizing Clustering Algorithm in Mobile Ad hoc Networks Using Simulated Annealing. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1492–1497, March 2003.

[18] VINT project. the ucb/lbnl/vint network simulator-ns (version 2),. URL http://www.isi.edu/nsnam/ns.

[19] J. Wu and H. Li. On Calculating Connected Dominant Set for Efficient Routing in Ad Hoc Wireless Networks. In *Telecommunication Sysrems, Special Issue on Mobile Computing and Wireless Networks*, volume 18, pages 13–36, September 2001.

[20] Yet Another Extensible Simulation Framework (YAES). URL http://netmoc.cpe.ucf.edu/Yaes/index.html.

[21] O. Younis and S. Fahmy. HEED: A Hybrid, Energy - Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks. In *Proceedings of IEEE Transactions on Mobile Computing*, volume 3, pages 366–379, October - December 2004.