

December 01, 2009

Wireless backbone protocols in homogeneous and heterogeneous ad hoc networks: an ns3 implementation and simulations

Mohamed Ahmed T. Elgalhud

Recommended Citation

Elgalhud, Mohamed Ahmed T., "Wireless backbone protocols in homogeneous and heterogeneous ad hoc networks: an ns3 implementation and simulations" (2009). *Electrical and Computer Engineering Master's Theses*. Paper 71. <http://hdl.handle.net/2047/d20002391>

Wireless Backbone Protocols in Homogeneous and Heterogeneous Ad Hoc Networks: An NS3 Implementation and Simulations

A Thesis Presented by

Mohamed Ahmed T. Elgalhud

To

The Department of Electrical and Computer Engineering

In partial fulfillment of the requirements
For the degree of

Master of Science

in

Electrical and Computer Engineering

In the field of

Wireless Networks

Northeastern University
Boston, Massachusetts

December 2009

Table of Contents

| | |
|--|----|
| Abstract | 4 |
| Introduction..... | 5 |
| Chapter 1 | 7 |
| 1.1 Homogeneous Network..... | 7 |
| 1.2 DCA Algorithm | 7 |
| 1.3 DCA Messages..... | 10 |
| 1.4 Experimental Results | 11 |
| Chapter 2..... | 14 |
| 2.1 Deterministic Connectivity Theory..... | 14 |
| 2.2 3-Hop Coverage Set..... | 14 |
| 2.2.1 Greedy Algorithm | 15 |
| 2.2.2 Individual Path Algorithm | 18 |
| 2.3 Experimental Results | 22 |
| Chapter 3..... | 25 |
| 3.1 2.5-Hop Coverage Set..... | 25 |
| 3.2 Cases of 2.5-Hop Coverage Set | 25 |
| 3.3 Experimental Results | 28 |
| Chapter 4..... | 32 |
| 4.1 2.25-Hop Coverage Set..... | 32 |
| 4.2 3-Hop Clusterheads Cases | 32 |
| 4.3 2-Hop Clusterheads Cases | 34 |
| 4.4 Proof of 2.25-Hop Coverage Set..... | 36 |
| 4.4.1 Proof of the Connectivity of 3-Hop Clusterheads..... | 37 |
| 4.4.2 Proof of the Connectivity of 2-Hop Clusterheads..... | 39 |
| 4.5 Experimental Results | 44 |
| Chapter 5..... | 47 |
| 5.1 3-Hop, 2.5-Hop, and 2.25-Hop Comparison..... | 47 |
| Chapter 6..... | 50 |
| 6.1 Heterogeneous Network..... | 50 |
| 6.2 Problem Description | 50 |
| 6.3 Algorithm..... | 50 |

| | |
|--|----|
| 6.4 Politburo Algorithm Message Types | 52 |
| 6.5 Difficulty of the Algorithm | 54 |
| 6.6 Determining the Number of Red Nodes..... | 54 |
| 6.7 Feasible Network Topology..... | 56 |
| 6.8 Experimental Results | 56 |
| Chapter 7..... | 61 |
| 7.1 Homogeneous and Heterogeneous Networks Comparisons | 61 |
| 7.2 Conclusion | 64 |
| Chapter 8..... | 65 |
| 8.1 NS3 Simulator..... | 65 |
| 8.2 Implementation | 65 |
| 8.3 Collisions | 65 |
| 8.4 Back-off Algorithm..... | 66 |
| 8.5 Broadcasting | 67 |
| 8.5.1 Broadcasting Approach..... | 67 |
| 8.5.2 Unicasting Approach..... | 68 |
| 8.5.3 Customized Broadcast Approach..... | 68 |
| 8.6 Topology Generating | 68 |
| 8.7 Average Route Length Computation | 69 |
| References..... | 71 |

Abstract

This work concern hierarchical organizations of multi-hop wireless communication networks, often called ad hoc networks. These networks are characterized by the lack of a fixed infrastructure, and as such they require algorithms that distributed and localized, and present challenges that are quite different from those of wired networks, or from those of wireless cellular network, where most crucial operations are performed by the fixed, often wired part of the networks. In this thesis we focus on partitioning the nodes of an ad hoc networks into clusters, made up of a clusterhead and its affiliated ordinary nodes. We consider two protocols, one for homogeneous networks (i.e., for networks where all the nodes have the same characteristics and resources) and one for heterogeneous networks (where we have two types of nodes: A few, more expensive and resource rich ones that can act as clusterheads and more simple nodes to be covered). The two algorithms are implemented and simulated by using the network simulator NS3, recently released. More important, for the first algorithm, termed Distributed Clustering Algorithm (DCA), we propose new methods for interconnecting the clusterheads into a connected backbone that reduce the backbone size, a property often sought after by many applications that use clustering. The one we propose here is the first implementation and study of the second algorithm, termed “Politburo” in this work, which determines the number of richer nodes to activate to have the simple nodes covered by multiple rich ones.

Introduction

Ad hoc wireless networks are made up of nodes communicating wirelessly without the aid of a fixed infrastructure. As such they find their primary use in those situations where it would not be viable to install a wired network, or where some infrastructure cannot be found. Typical examples are network deployment for disaster recovery, as well as networks for temporary events.

Research in ad hoc networks has gone in many different directions in the past few years. One that has received particular attention has been the problem of partitioning the network into clusters. As in wired networks, *clustering* has been found useful for optimizing resource allocation as well as for making networks protocols such as broadcast and routing more scalable.

This thesis concerns the software implementation and simulations of two protocols for clustering ad hoc networks with static nodes. Implementation and simulations are performed through the recently released Network Simulator 3, NS3 (<http://www.nsnam.org/>). The two protocols concern two different kinds of ad hoc networks, namely, homogeneous ones, where all nodes are equivalent, and some of them is selected as *clusterhead* (a node in charge of a cluster), and heterogeneous networks, where the nodes are of two kinds, those who can be clusterheads, and those who can only be affiliated to a clusterhead.

In homogeneous networks, there are many algorithms performing clusterhead selection. These algorithms choose the nodes depending on different parameters. Some of them, for instance, promote nodes that have a bigger coverage set or, in other words, promote the nodes that have the biggest number of neighbors (i.e., nodes in a node transmission range). Other algorithms elect the clusterheads depending on a weight metric associated to the node, and the nodes that have the biggest weight are selected to be in charge. Finally, some other algorithms just select the nodes based on the node unique ID (typically nodes with the lowest ID in their neighbors become clusterheads). A survey of these different protocol can be found in [2].

After been selected the clusterheads start choosing *gateway* nodes to act as connectors between clusterheads, so to form a *connected backbone*.

In this thesis clusterhead selection is performed by using the Distributed Clustering Algorithm (DCA), originally presented in [1]. We implement this protocol using NS3 and we perform extensive simulations for assessing its performance. The metrics investigated include the number of clusterheads and the number of *ordinary nodes* (i.e., the nodes that are not clusterheads). The DCA will be discussed in Chapter 1. (The NS3 implementations details of the DCA are also illustrated in the last chapter.)

For interconnecting the clusterheads into a connected backbone with introduce three algorithms, termed 3-hop coverage set, 2.5-hop coverage set and 2.25-hop coverage set. The latter is a contribution of this thesis. The three algorithms will be introduced in Chapter 2, 3 and 4, respectively. In general the difference between these three protocols is the number of neighbor nodes from which each clusterhead has to obtain information.

There are two ways of selecting the gateways. The first one follows a greedy paradigm based on the weights of the nodes, and the second is based on the path between the two clusterhead to be joined. Both were tested on the three coverage sets methods (except for 2.5-hop coverage set scheme, where we implemented only the path based method). More details will be discussed in Chapter 5, where we will also make a comparison between the three types of hop-coverage set. The second choice of gateways has not been implemented before.

Chapter 6 concerns the implementation of clustering for heterogeneous networks, a much less investigated field within ad hoc network research. The focus here is on the reliability and robustness of coverage of non-clusterhead nodes (ordinary, or white nodes in the following) by the nodes that can act as clusterhead (red nodes). Since the algorithm is about selecting a suitable number of red nodes to cover all the white nodes, we call the algorithm the “Politburo protocol.” This is the first time this algorithm has been implemented using a simulator that takes into account different part of a sensor node protocol stack (including physical and MAC layers for interference detection and avoidance).

Both algorithms implementation are completely localized and distributed, thus been suitable for implementation and deployment of actual ad hoc networks.

Chapter 1

1.1 Homogeneous Network

In this chapter, one of the algorithms which is dedicated for homogenous network is discussed and tested using NS3 simulator. In the homogeneous network, there is no predetermined clusterheads that control the cluster, so all the nodes are similar and designed in the same way. Any node can be Clusterhead or ordinary node depending on the nodes distribution. This kind of network is good in the application at which determining the clusterheads automatically required. Although this may cost more because each node will have the design of being clusterhead or ordinary, it is good due to its self-independence of nominating the clusterheads.

1.2 DCA Algorithm

The aim of the algorithm Distributed Clustering Algorithm is to divide the network into clusters and selecting the clusterheads for each cluster, and it is intended to get the minimum number of clusterheads as possible. Each node has unique ID, and different weight, and the nodes that has maximum weight in the coverage set is selected to be clusterheads. The advantage of using the weights is that it is general metric which in implantation could be any parameter, e.g. power consumed, position.

Each node decide its role, the nodes that have biggest weight promote themselves as clusterheads and other nodes choose to be ordinary nodes.

There are some rules for the DCA as mentioned in [1]:

- Every node has at least one clusterhead in its neighborhood.
- There are no two clusterheads neighbors to each other.
- Every ordinary node joins the biggest clusterhead among its neighbor.

In the implementation of the Algorithm, I assumed that each node already knows its neighbor and the weight and ID of the neighbor. And actually this is can be done by exchange messages between neighbors till all neighbors are discovered.

There are two main types of exchanged messages between the nodes, the first message is CH(), and the other is Joint() message.

The nodes start compare its weight with the weight of all its neighbor, if it has the maximum weight among these neighbors it send CH() message, if it does not have, it waits for the nodes with bigger weight to decide their role.

Now each node that still its role not determined yet wait for knowing the roles of all neighbor nodes with bigger weight, and depending on these information decide its role. There are different scenarios for these nodes

Scenario 1: if all the neighbor nodes with bigger weight have already joint to different clusterheads, then this node will select itself as clusterhead and send CH() message.

Scenario 2: if all the neighbor nodes with bigger weight have sent CH messages, then the node will decide its role to be ordinary and joint the clusterhead with maximum weight.

Scenario 3: if some of the neighbor nodes with bigger weight have decided their role and other not, then the node checks if one of the decided nodes has announce itself as Clusterhead, if this is the case then, the node checks if this Clusterhead has bigger weight than any other undecided nodes, if yes, then the node will join to it, if not, then the node will wait for the undecided nodes to determine their role.

The following figure shows example of DCA algorithm:

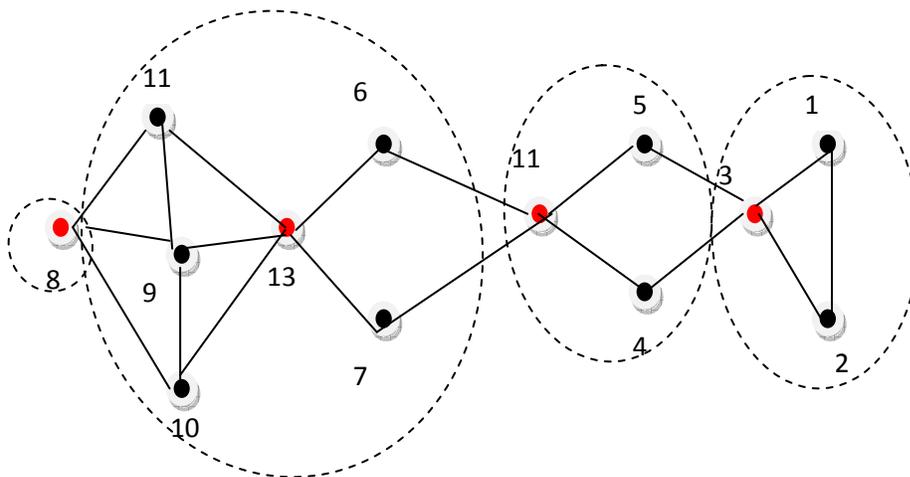


Figure 1.1 shows the clusters and their clusterheads

Figure 1.1 an example of network and how it is clustered, in this figure every two neighbor has an edge, and there is no edge for non-neighbor nodes, the nodes with Red color are nominated to be the clusterhead, the dash circle is the cluster belongs to the red clusterhead. For simplicity each node has the same ID and weight. Node 13 is nominated to be the clusterheads since it is the biggest one, and all its neighbor joined to it, Node 8 is single clusterhead which has no members, although node 8 has weight less than node 9,10 and 11, it is a clusterhead node. And this is because node 11,9 & 10 joined to Node 13, therefore node 8 is now the biggest non-determined node, so it decides its role to be a clusterhead. Node 11 is also a clusterhead , because it is the biggest one in the neighborhood, and node 3 becomes a clusterhead because node 4 and 5 joined to node 11.

Init ()

begin

Checks my weight among my neighbor

if I have maximum wait send CH()

else wait for neighbor

end

Do Action on Received CH(source)

begin

if (Weight[source] is Max[Weights of Neighbors])

send Join message to source

else

begin

if (Neighbors[Weight > Weight[source]] are joined)

send Join message to source

else

Wait for Neighbors[Weight > Weight[source]] to determine their role

end

end

Do Action for Received Join(source, Dest)

begin

```

if(Dest==MyId) Add this node to my cluster members
else
    if (Neighbors[Weight > MyWeight ] are joined)
        send CH()
End

```

Every Clusterhead quits the Algorithm if all its neighbors determine their roles, and Ordinary nodes quit the algorithm if it joins to a clusterhead.

since each node either sends Join or CH messages and since each node need to receive the message from all neighbors to decide whether to be ordinary node in case the node undecided yet or to quite the algorithm in case the node is clusterhead, therefore each message needs to be received by all nodes, to guarantee that Ack message is required after each message.

Each clusterhead needs to know whether its neighbor going to joint its cluster or not, so whether the node joins or not, the clusterheads need to make sure that the CH message has been received by all its neighbors. Instead of sending Ack message to the clusterhead then sending Join message, the ordinary node will send only join if it decided to join any clusterhead and it will send Ack if it still waiting the statue of other nodes.

1.3 DCA Messages

CH message format:

| | | |
|-----------|--------------|--------|
| Source ID | Message Type | Weight |
|-----------|--------------|--------|

Figure 1.1 Shows the CH message format

The first portion of the message is the MAC address of the source. Message Type: refers to which type of the message is, and there are many types could be CH , Join , Ack or any type.

Weight: this portion is used especially when the clusterhead does not receive any response from some nodes; in this case the clusterhead assumes that these nodes did not receive the message, and then send the CH message again with weight number, this weight number is equal to the minimum weight of the nodes that the clusterhead did not receive any Ack from.

Join message format:

| | | |
|-----------|---------------|--------------|
| Source ID | CusterHead ID | Message Type |
|-----------|---------------|--------------|

Figure 1.2 Shows the Join message format

Source ID: it is the MAC Address of the node that sends Join message.

Clusterhead ID: it is the MAC address of the clusterhead which this node joins to.

Message Type: it refers the Message type.

After each nodes finishes DCA algorithm will have table like this:

| Neighbor ID | Weight | Is it Ordinary | Its Clusterhead |
|-------------|--------|----------------|-----------------|
| 1 | 2 | yes | 4 |
| 3 | 3 | yes | 3 |
| 2 | 1 | No | - |

Table 1.1 shows the neighbor information table at the End OF DCA Algorithm

Each node will have some information about its neighbors like whether it in the same cluster or not and each nodes will know all its 2-hop clusterheads and some of the 3-hop clusterheads. (n-hop nodes are the nodes that are n hops far).

1.4 Experimental Results

In this experiment, the nodes are distributed along fixed Area A (850 m²), and they are uniformly distributed and the algorithm was run for number of nodes n=50,100,150,200,250,300. For each number of n, the experiment was repeated more than hundred times, and the average was taken for the all metrics.

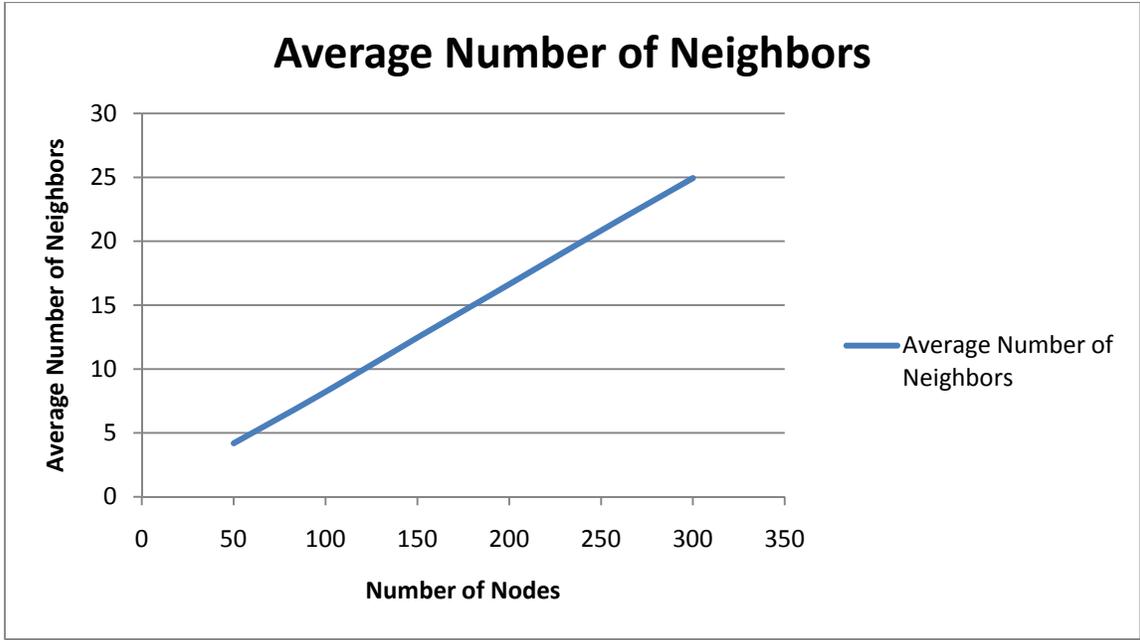


Figure 1.3 Shows the Average number of Neighbors

Figure 1.3 shows the network density per each node, the network density is the number of neighbor nodes per each node.

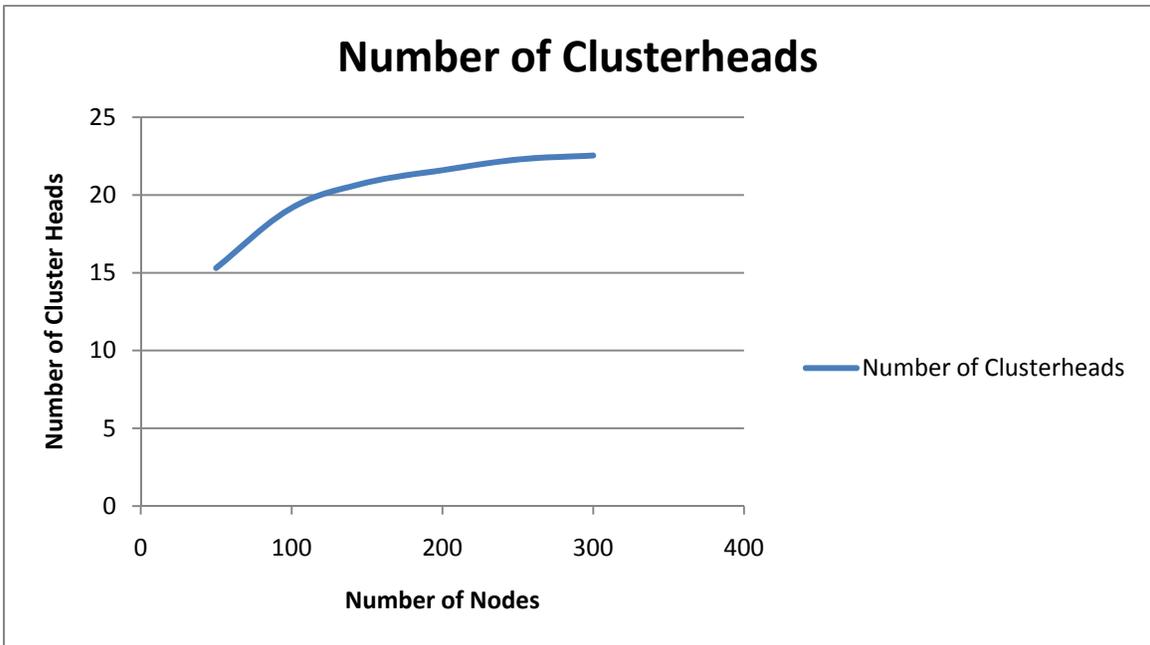


Figure 1.4 Shows the Average number of Clusterheads

Figure 1.4 shows the average number of clusterheads versus the number of nodes, the number of clusterheads increases as the number of nodes increases till reach certain number, and then the number of cluster clusterheads stay constant.

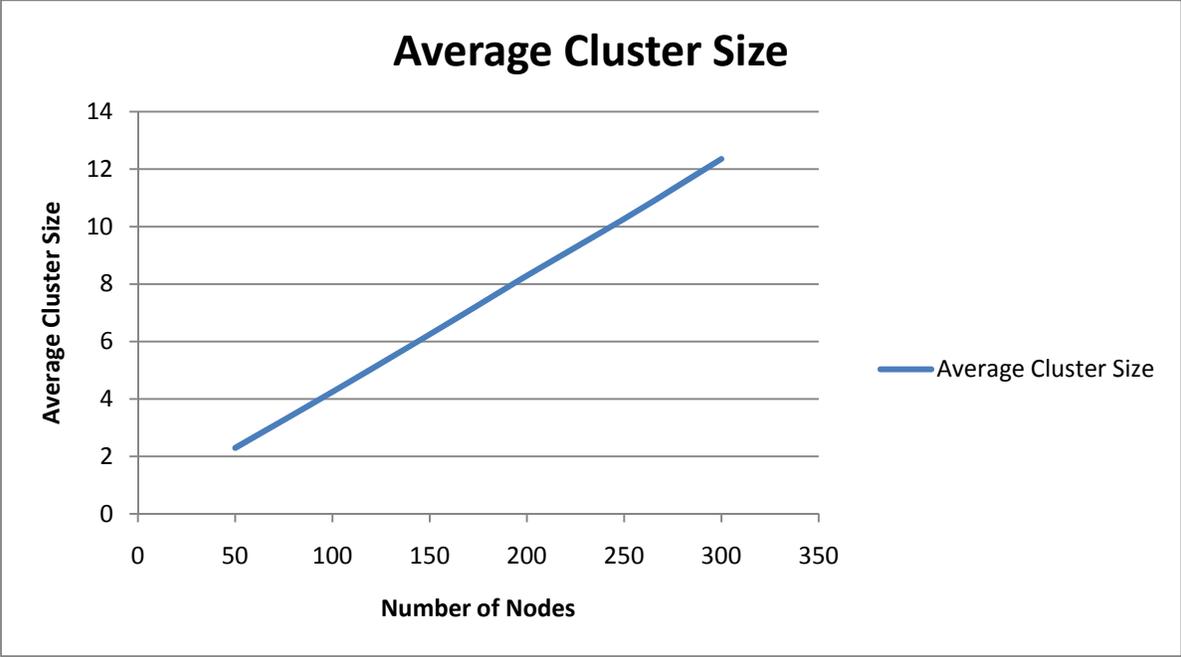


Figure 1.5 shows the average cluster size per clusterhead

Figure 1.5 shows the average cluster size per clusterhead, the cluster size is the number of nodes that joins to the same clusterhead. The figure shows the linear increase in Average cluster size and this is due to the linear increase in the Average Number of Neighbors

Chapter 2

In this chapter, the second step to build the backbone will be discussed. After finishing DCA algorithm which divides the network into clusters and for each cluster there is a clusterhead, and each clusterhead has some members or no members, in this case the clusterhead called single-clusterhead. Now to build the backbone, it is needed to connect these clusterheads.

2.1 Deterministic Connectivity Theory

“If each clusterhead connects itself to all other clusterheads that are at most 3 hops away, a connected backbone is always guaranteed. Moreover, we prove that this is the minimum distance needed to *always* guarantee connectivity. In other words, if only a single clusterhead is left out from all clusterheads that are in the 3-hop neighborhood, then connectivity can no longer be guaranteed in the worst case.” [3].

As result of this theory, an algorithm called 3-hop coverage set is developed, in this chapter this algorithm is implemented into two ways, Greedy Algorithm [4] and Individual-path algorithm [2].

2.2 3-Hop Coverage Set

In 3-hop coverage set each clusterhead has to be connected with all 2-hop clusterheads and 3-hop clusterheads according to the theory.

To make the connection with these clusterheads, each clusterhead needs to nominate gateways that connect it with other clusterheads. So each clusterhead need to know the information about all 1-hop and 2-hop neighbor nodes. The exchange information is done in two rounds.

The first round is the process that every node sends all its 1-hop clusterheads to all its neighbors, and the second round of exchange information is forwarding the all information obtained by first round and send them to all neighbors.

After getting the information of one-hop and 2-hop neighbors, the process of nominating the gateways starts. There are two ways of selecting the gateways, Greedy Algorithm and Individual path Algorithm.

2.2.1 Greedy Algorithm

The way of Greedy algorithm work is done by choosing the 1-hop nodes that have bigger number of clusterheads connected to it, then choosing the 2-hop gateways that are connected to the selected 1-hop nodes, 2-hop gateways is selected if they have biggest number of clusterheads connected to it. this algorithm is briefly mention without details in [4].

| Gateway 1 | Gateway 2 | Head |
|-----------|-----------|------|
| 1 | 1 | 6 |
| | | 11 |
| | 2 | 6 |
| | 3 | 7 |
| | | 8 |
| | | 11 |
| | | 12 |
| | 4 | 9 |
| 10 | 3 | 11 |
| | | 12 |

Table 2.1 shows example of the information collected at each clusterhead

Table 2.1 shows example of the information that each clusterhead may have after getting the information from neighbors, Gateway1 is the neighbor node to this clusterhead, and Gateway2 is the node that connects the clusterhead to Gateway1, and clusterhead columns indicated the clusterheads that resides in 3-hop or 2-hop away from this clusterhead. when Gateway1 is equal to Gateway2, this means that the clusterhead is 2-hop away. The greedy algorithm works as follows:

1. Selecting all Gateways1 that are equal to Gateway2, on other word selecting the clusterhead through 2-hop path has more priority than through 3-hop path. Then delete these clusterheads from the table.
2. Selecting the Gateway1 that have bigger number of clusterheads, and then selecting Gateway2 that are more frequent in the table. After that deleting the all clusterheads from the table.
3. Once the all clusterheads are done, there is a path between this clusterhead and all other clusterheads, then this clusterhead start to inform all gateway1 that are selected.
4. When the gateways1 are selected, they will get information about the clusterheads that will connect to and gateways2 that will communicate through to theses clusterheads.
5. Gateway1 will send to all gateways2 to inform them to be intermediate nodes to specific clusterheads.

Each two clusterheads in the backbone will be connected to each other, they might be connected through different path and that is more likely.

Referring to Table 2.1, the greedy algorithm will choose first Gateway1, then to select gateways 2, the algorithm has 2 choices for clusterhead 6, so Gateway2 either be node 2 or node 1, so the algorithm will choose node 1, as it is more frequent in the table than node 2. So the clusterhead 6 and 11 will be deleted, in this case gateway2 with node 2 will be dropped from the table because its clusterhead has been deleted. Gateways2 with node 3 and 4 will be chosen in the same way. Node 10 as gateway1 will not be selected because its clusterheads are covered through node1.

Now the routing table will be like this:

| Gateway 1 | Gateway 2 | Clusterhead |
|-----------|-----------|-------------|
| 1 | 1 | 6 |
| | | 11 |
| | 3 | 7 |
| | | 8 |
| | | 11 |
| | | 12 |
| | 4 | 9 |

Table 2.2 shows the Routing table for the Greedy Algorithm

Message Types Used in this Algorithm:

There are four type of messages used in this algorithm:

CH1Hop(): This message is sent by all nodes to their neighbors, this message has all 1-hop clusterheads. This message will have only clusterheads.

CH2Hop(): This message is sent after getting CH1Hop from all neighbors, then collecting the information in one message, the collection is done by collecting all clusterheads then selecting the gateways that have more clusterheads. So the message will have pairs of clusterheads and gateways.

AnnounceGateway1: This message is sent by the clusterheads after getting CH2Hop from all neighbors, this message will have gateways1 and beside each gateway1 there are pairs of gateways2 and clusterheads. each node receives this message will look at the Gateways1, if it is equal to its ID, then take the pairs of clusterheads and Gateways2 that corresponds to it, and send AnnounceGateway2.

AnnounceGateway2: This message is sent by each Gateway1 that has been selected to all Gateways2 that have been selected. This message will have clusterheads that belong to each gateway2.

2.2.2 Individual Path Algorithm

This algorithm is different from the previous one, instead of selecting the nodes that have more clusterheads connected to it, the algorithm works by looking at every two 3-hop or 2-hop clusterheads, and choosing the intermediate gateways in such way that each clusterhead will choose the same nodes as gateways.

After the two round of exchange information, each two 2-hop or 3-hop clusterheads see each other. the way that each clusterhead choose the gateways as follows:

- If the two clusterheads has different routes between each other, and some of them are two hops and other three hops, then the routes with two hops will be chosen.
- If there are more than one route with two hops , then the route that has node with the biggest weight will be chosen.
- if the two clusterheads have only routes with three hops (or two intermediate nodes), then each node will take the maximum summation of the weights of the two intermediate nodes among all the intermediate nodes. if there are two routes has the maximum weight, then the route that has smallest ID will be chosen.

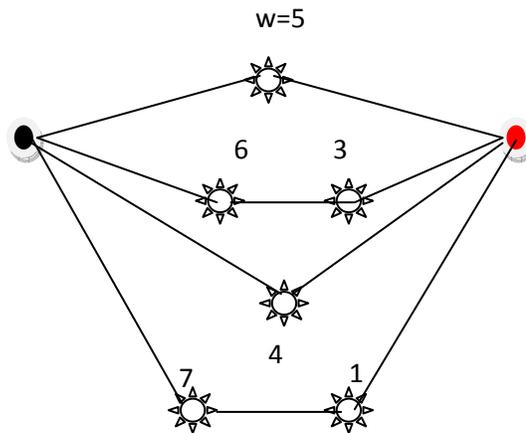


Figure 2.1 shows an example of two clusterhead nodes has routes with two and three hops

Figure 2.1 shows two clusterheads one is red and the other is black, and there are four routes between these two clusterheads, two of them are two hops routes and the other two are three hops route, the red and black clusterheads will choose the path that has the node with weight 5, because the routes with less number of intermediate nodes have more priority. therefore will one

only two routes, the first that has the node with weight 5 and the second with weight 4, the route with node of weight of 5 will be chosen, because it is the route that has the biggest weight.

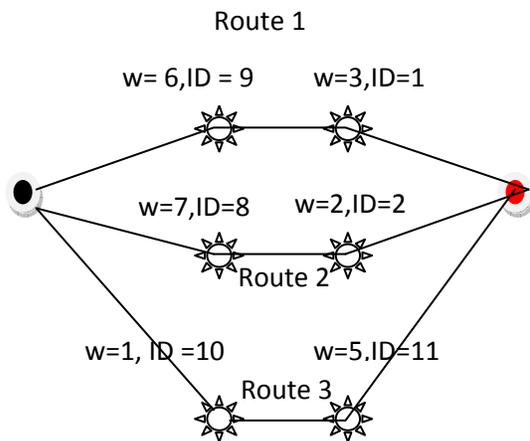


Figure 2.2 shows an example of two clusterheads with routes of 3 hops only.

In figure 2.2, there are three routes between the red and black clusterheads. each node will choose the route with maximum summation of weights, Sum of Route1 is 9, and Sum of Route2 is 9, and Sum of Route 3 is 6, so Route 3 will be dropped, now each node has two routes, Route1 and Route 2. Since these two routes have the same sum of weights, therefore the route with smallest ID will be chosen. Therefore Route1 will be nominated by these two nodes, because it has the node with ID=1 which is smaller than any ID in Route 2.

Now after selecting the route, the nodes or gateways on this route will be informed, each clusterhead will inform only its gateway neighbor.

Message Types Used in this Algorithm

The type of messages used:

CH1Hop: the same as in greedy Algorithm.

CH2Hop: The same as in Greedy Algorithm.

AnnounceGateway: This message is sent by the clusterhead to its neighbor nodes to inform it to be as a gateway, and also the clusterhead that connected to.

The difference in this algorithm and the greedy one is that the greedy one has two types of AnnounceGateway messages, because in the Greedy algorithm, there might two routes between two clusterheads, while in this algorithm, there is only one route between and two clusterheads.

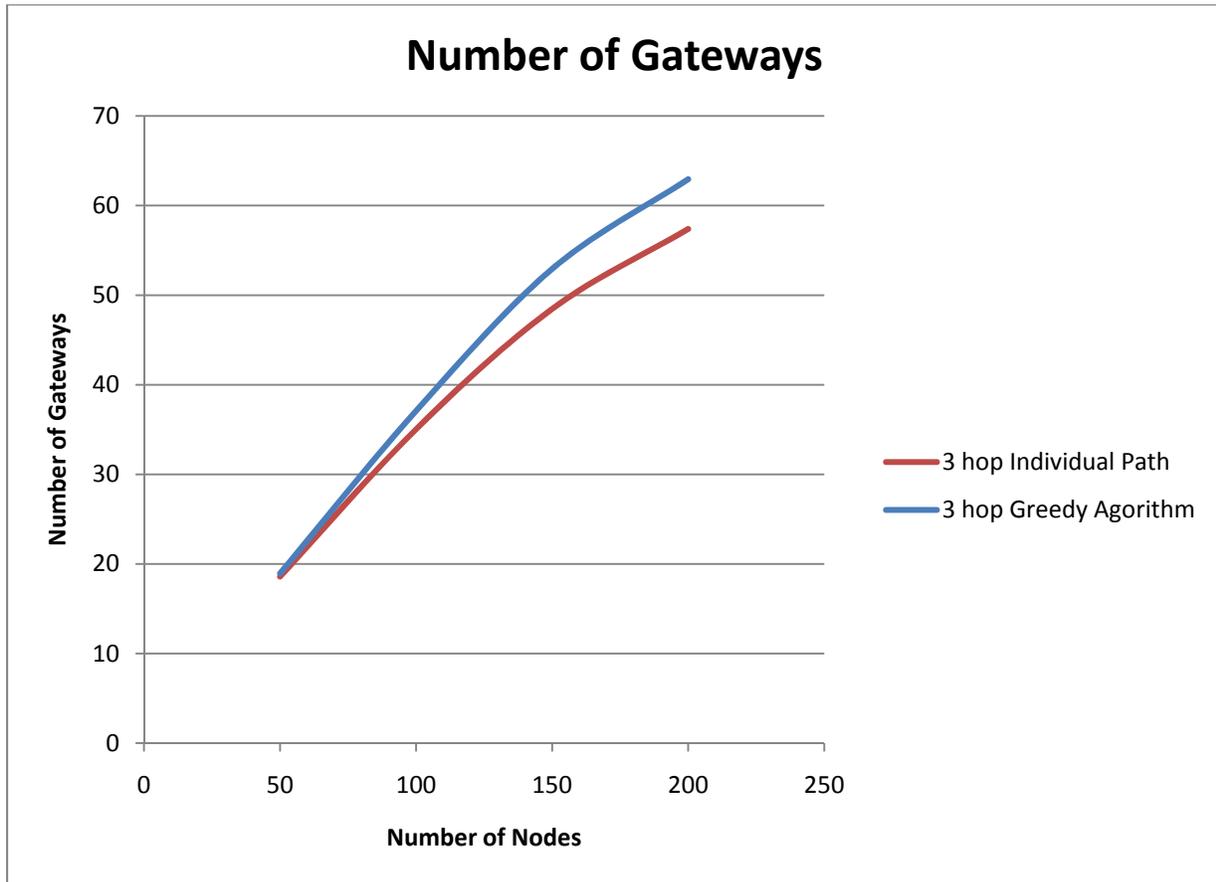


Figure 2.3 shows the number of gateways for the two Algorithms

In figure 2.3, it is noticeable that the Individual-path algorithm outperforms the greedy algorithm, and this is because in the greedy algorithm, the clusterheads different routes, as in figure 2.4 shows.

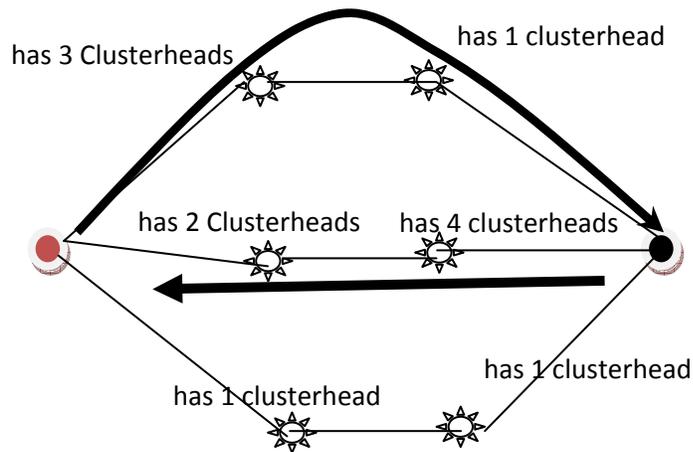


Figure 2.4 shows an example of how two clusterheads in greedy algorithm choose different routes

Figure 2.4 shows the example of how two clusterheads choose different routes in greedy algorithm, in this figure there are three routes, the upper route, the middle route, and the lower route. Since in the greedy algorithm, the clusterheads choose the nodes with higher number of clusterheads connected to it. The black node will chose the middle route, because the middle route has a node with four clusterheads connected to the black clusterhead, while the red node will choose the upper route. In this case will be four gateways chosen to make the connection between the red and black clusterhead. And this is not the case in the Individual path.

2.3 Experimental Results

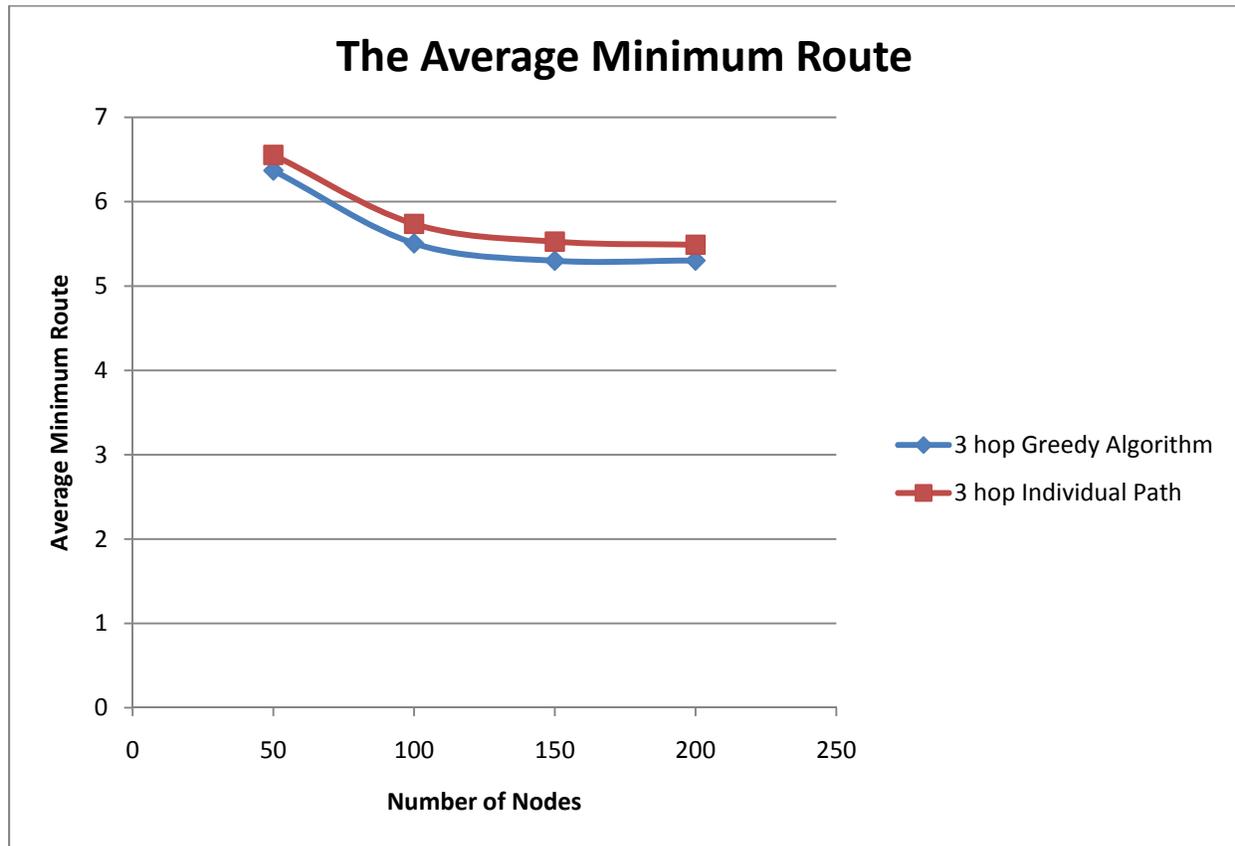


Figure 2.5 shows the average minimum route for the two algorithms.

Figure 2.5 shows that the average minimum route for the two algorithms, the figure shows that the average minimum route for the greedy algorithm is lower than the average for the individual path algorithm. The average minimum route is the average of the number of the links between all the nodes in the built back bone, and this number depends on the selected gateways and how many they are. Since the greedy algorithm has more number of gateways as Figure 2.3 shows, therefore the number of links between any two nodes will be less.

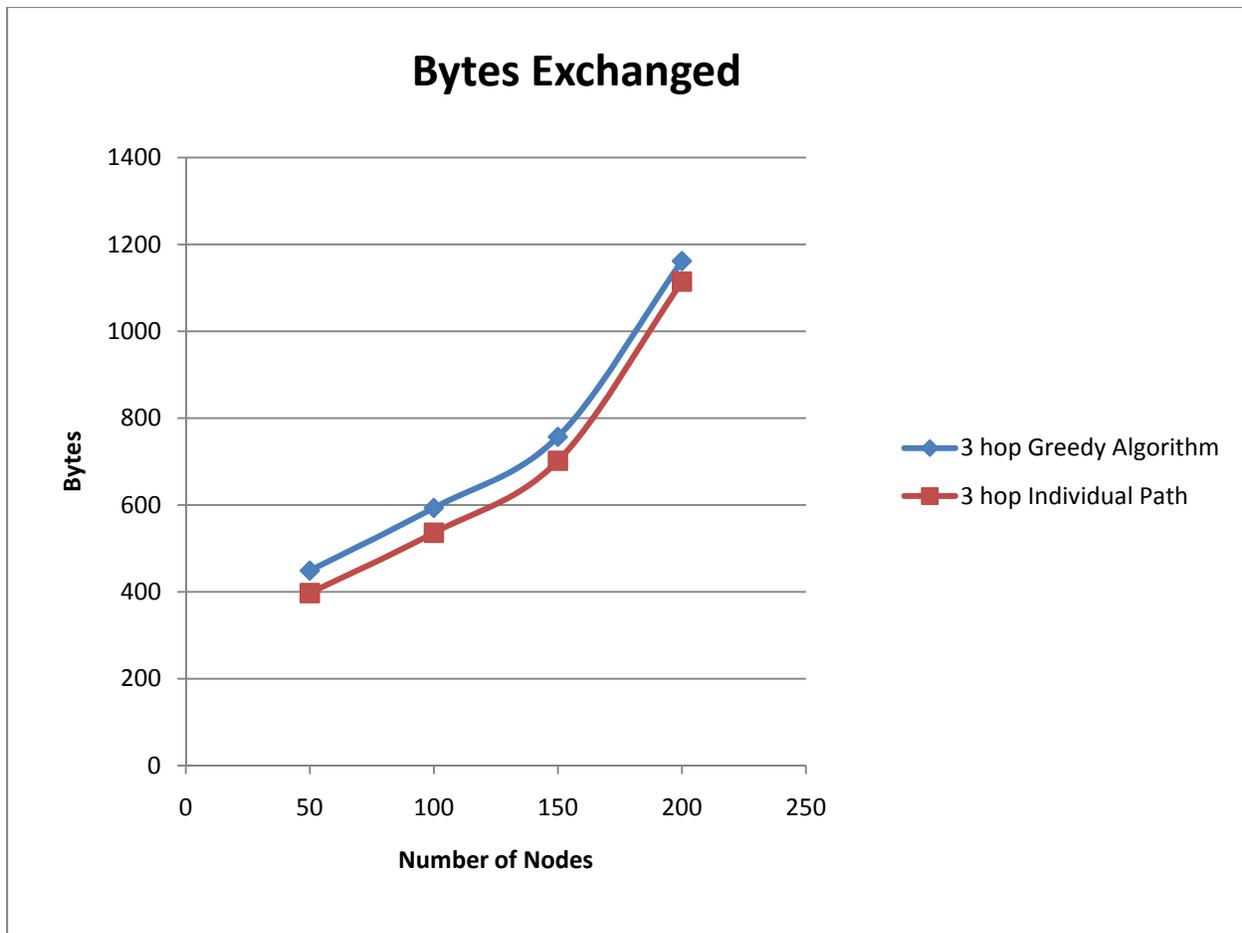


Figure 2.6 shows the number of bytes for the greedy and individual path algorithms

Figure 2.6 shows the number of bytes exchanged for the two algorithms, the greedy algorithm has more number of bytes exchanged, the difference not that much more. And this difference is due to the more number of gateways selected also the selection process is done using two types of message in the greedy while one message in the individual path.

in the greedy Algorithm, the gateway 1 and gateway 2 has to be informed by each clusterhead, while in the individual path algorithm, gateway 1 is only informed, because the gateway2 will be informed using the other clusterhead.

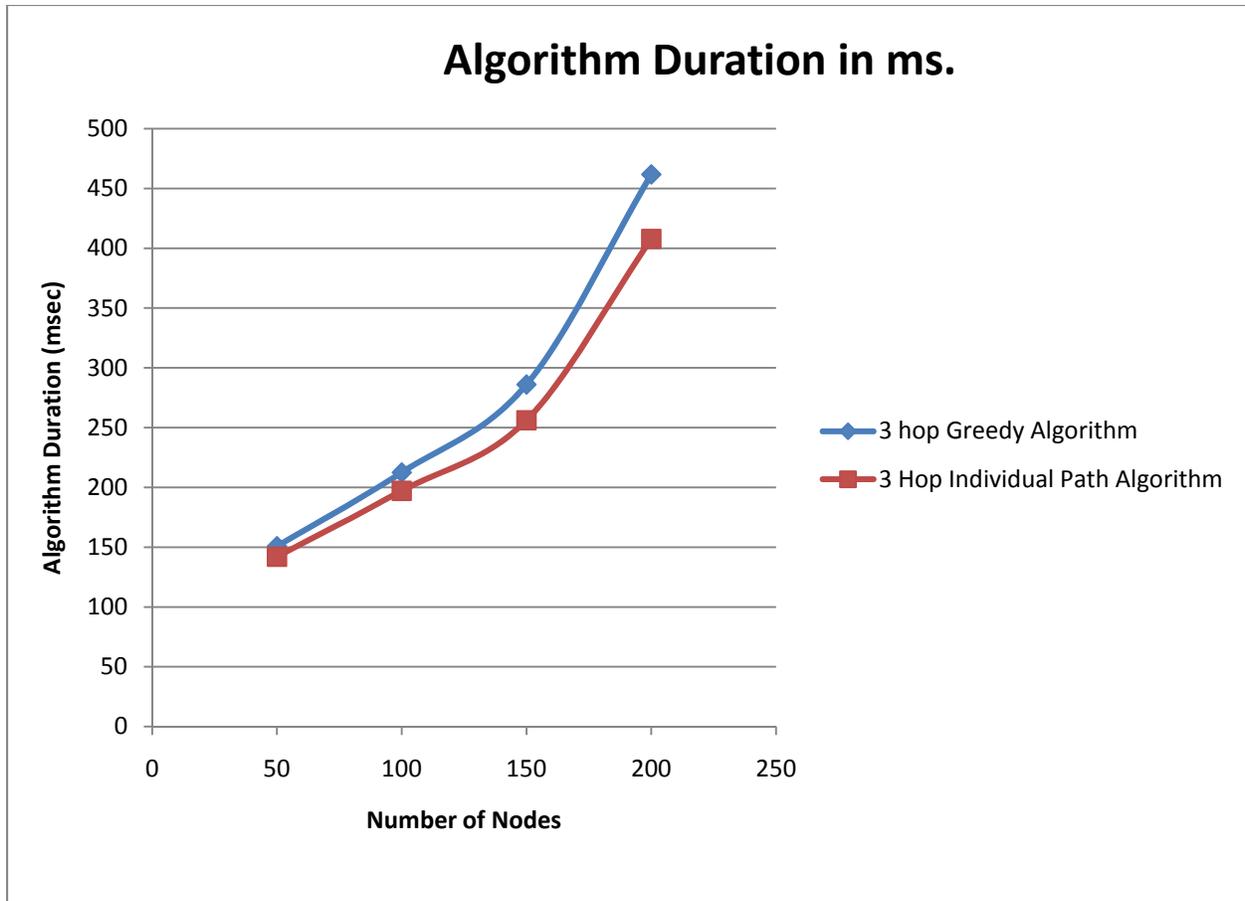


Figure 2.7 shows the Algorithm duration for both algorithms.

Figure 2.7 shows that the algorithm duration for the Individual path algorithm is less than the greedy algorithm, and this is because the greedy algorithm has to inform the gateway1 and gateway2 and this takes two rounds of exchange messages, while the individual path only informs the gateway1 which takes only one round of exchange information.

at the end of this chapter, the gateways of the 3-hop cover set can be determined using two algorithms ,the greedy and Individual-path algorithm. In general the individual path algorithm outperforms the greedy in terms of Number of gates, message exchanged, and Algorithm duration. While the greedy algorithm has less average minimum route, and this is due to more number of gateways.

Chapter 3

In this chapter, another approach for determining the coverage set for the clusterheads will be addressed. In Chapter 3, the 3-hop coverage set is addressed, and this chapter, another approach which is 2.5 hop coverage set will be presented.

3.1 2.5-Hop Coverage Set

In this coverage set, all two-hop clusterheads are connected and also the clusterhead will be connected only to the 3-hop clusterheads that have members neighbor to the clusterhead's neighbor. In other word, a clusterhead will be connected to another 3-hop clusterhead if it has members resides in 2-hop neighbor of the other clusterhead [4]. The proof exists in [5]. And according to this definition, there are three cases

3.2 Cases of 2.5-Hop Coverage Set

Case 1:

This case happens when the two nodes between the two clusterheads are members to these clusterheads.

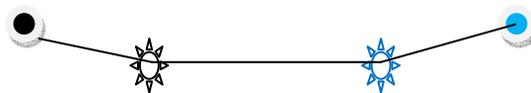


Figure 3.1 shows the Case 1

Figure 3.1 shows the case 1, the Circle presents the clusterhead and the sun presents the ordinary nodes, when the sun and circle nodes have the same color, it means the node belong to this clusterhead.

In this case the two ordinary nodes belong to two clusterheads, each one belong to different clusterheads, and according to the definition of 2.5 hop coverage set, both of them will be connected.

Case 2:

This case occurs when the one of the two ordinary nodes between the two clusterheads belong to the third clusterhead.

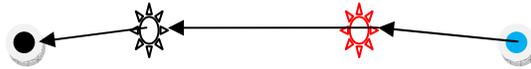


Figure 3.2 shows the Case 2

The figure 3.2 shows the connection between the blue clusterhead and the black clusterhead, it is noticeable that the connection is directed, and this is according to the definition of the two-hop nodes, where the blue clusterhead does not have a member in 2-hop neighbor of the black clusterhead, while the black clusterhead has a member which is 2 hops away from the blue clusterhead. and this shows why the 2.5 coverage set is directed graph. Here the black clusterhead can't see the blue while the blue can see the black clusterhead.

Case 3:

This case happens when between any two clusterheads there are no any nodes belong to the one of the clusterheads.



Figure 3.3 shows the Case 3 of 2.5 Hop coverage set

Figure 3.3 shows two clusterheads black and blue has no members which is 2 hops away from the other clusterhead. And according to the definition of 2.5 hop coverage set, these two clusterheads don't need to be connected, and this will drop the number of the gateways. And this is the only thing differs from the 3-hop coverage set.

The way of exchange information is the same as in 3-hop coverage set, it takes two rounds of exchange information. The only difference is the number of clusterheads sent in the first round, in the 3-hop coverage set all clusterheads that are 1-hop neighbor will be forwarded to the all nodes, while in 2.5 hop not all the clusterheads will be forwarded in the first round, only one clusterhead will be forwarded.

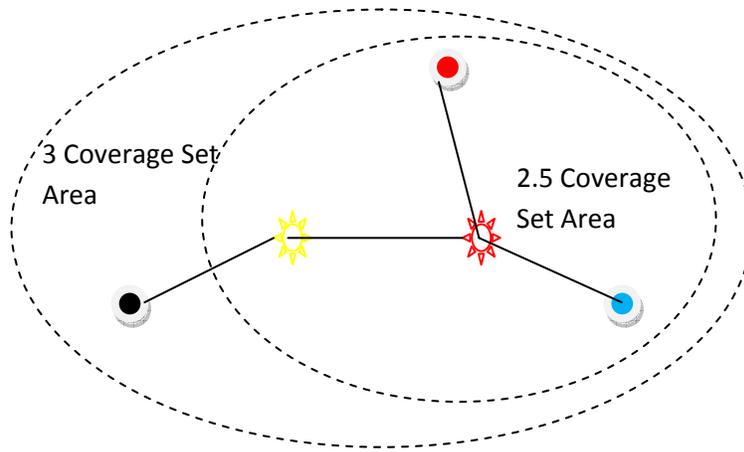


Figure 3.4 shows the coverage set Area for 3-hop and 2.5 hop for the blue clusterhead

In figure 3.4, the coverage set area of the blue clusterhead differs in 3-hop and 2.5 hop, the figure shows the area of the 2.5 hop coverage set is less than 3-hop coverage set, while in 3-hop coverage set, the blue clusterhead is connected to both the red and black clusterheads in 3-hop while in 2.5 the blue clusterhead is only connected to the red clusterhead.

Since the 2.5 hop coverage set is directed graph, therefore the individual path algorithm will not work here, because the individual path algorithm assumes that the two clusterheads see each other. And only the Greedy algorithm will be presented here.

3.3 Experimental Results

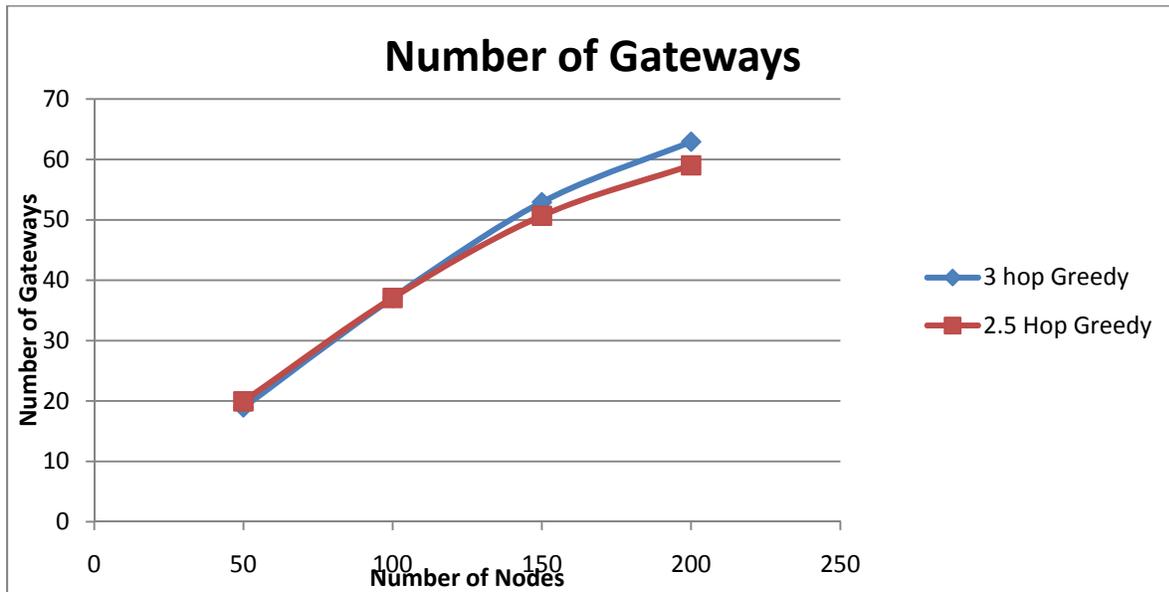


Figure 3.5 shows the number of gateways for 3 hop and 2.5 hop greedy Algorithm

In Figure 3.5, the number of gateways for 2.5 hop is less than the number of gateways in 3 hop, but as obvious from the figure the number of gateways is close in both cases, at 200 nodes the difference is only one node, and this shows the occurrence of the ‘case 3’ which defined earlier is rarely happens, anyway it helps in the bytes exchanged and algorithm duration.

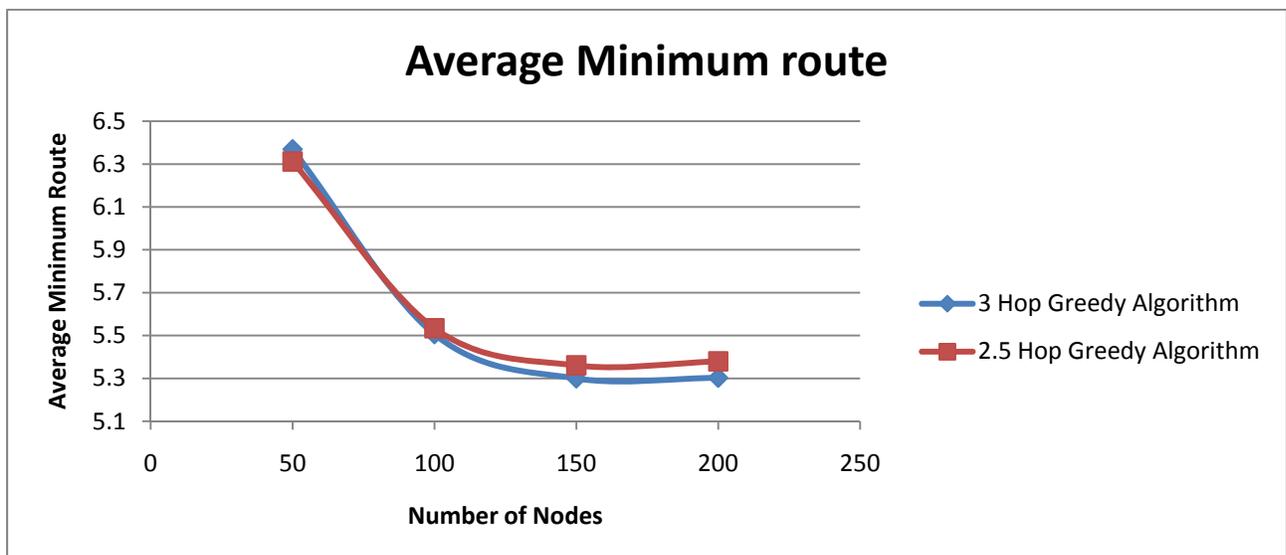


Figure 3.6 shows the Average minimum route for both Algorithms

As figure 3.6 indicates, the average minimum route for 2.5 is higher than the average minimum router for the 3-hop, and this is due to the more number of gateways for the 3 hop which give for each node more choice to go other nodes and this may lead to less number of links between any two nodes. However the difference between 3-hop and 2.5-hop is not that much, since the difference in the number of gateways is not big.

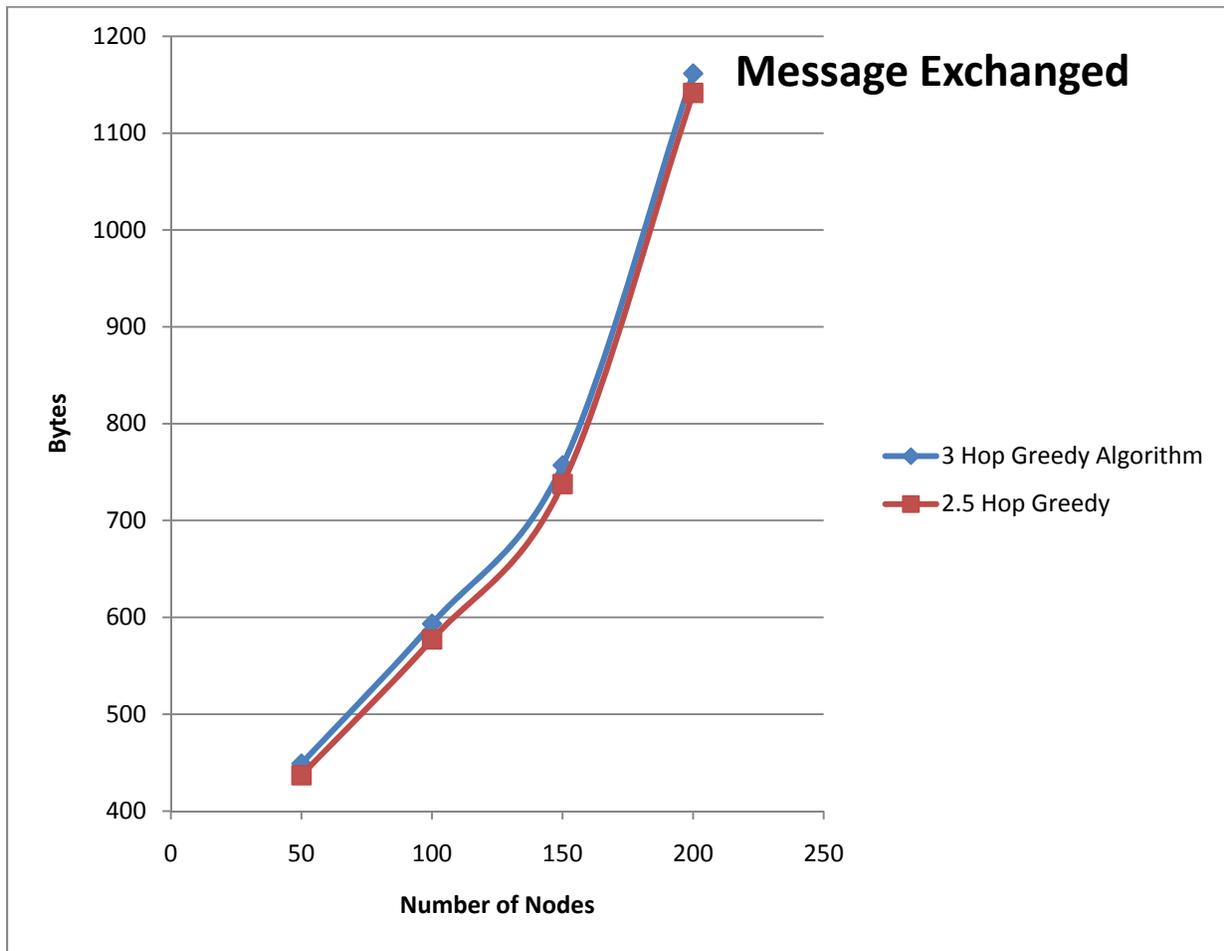


Figure 3.7 shows the Bytes exchanged for the 3 hop Greedy and 2.5 Greedy.

Figure 3.7 shows that bytes exchanged for 2.5 hop is less than the bytes exchanged in the 3 hop, and this exchange is very big, because the only difference between the 2.5 hop and 3 hop is at CH1Hop message, in the 3-hop , this message includes all 2-hop clusterheads, while in 2.5 it includes only one clusterhead, and on average the number of clusterheads in CH1hop is between 2 and 3, so the difference is not very big, especially the bytes represented in the graph represents all the bytes sent by all types of algorithm from the DCA stage and Gateway nomination.

And It is noticeable in the figure, when number of nodes is 200 , the number of bytes jumps more highly than the smaller number of nodes for both algorithm, and this is due to density of the network increases, and due to this increase the collision will increase more and more and this leads to more resent messages.

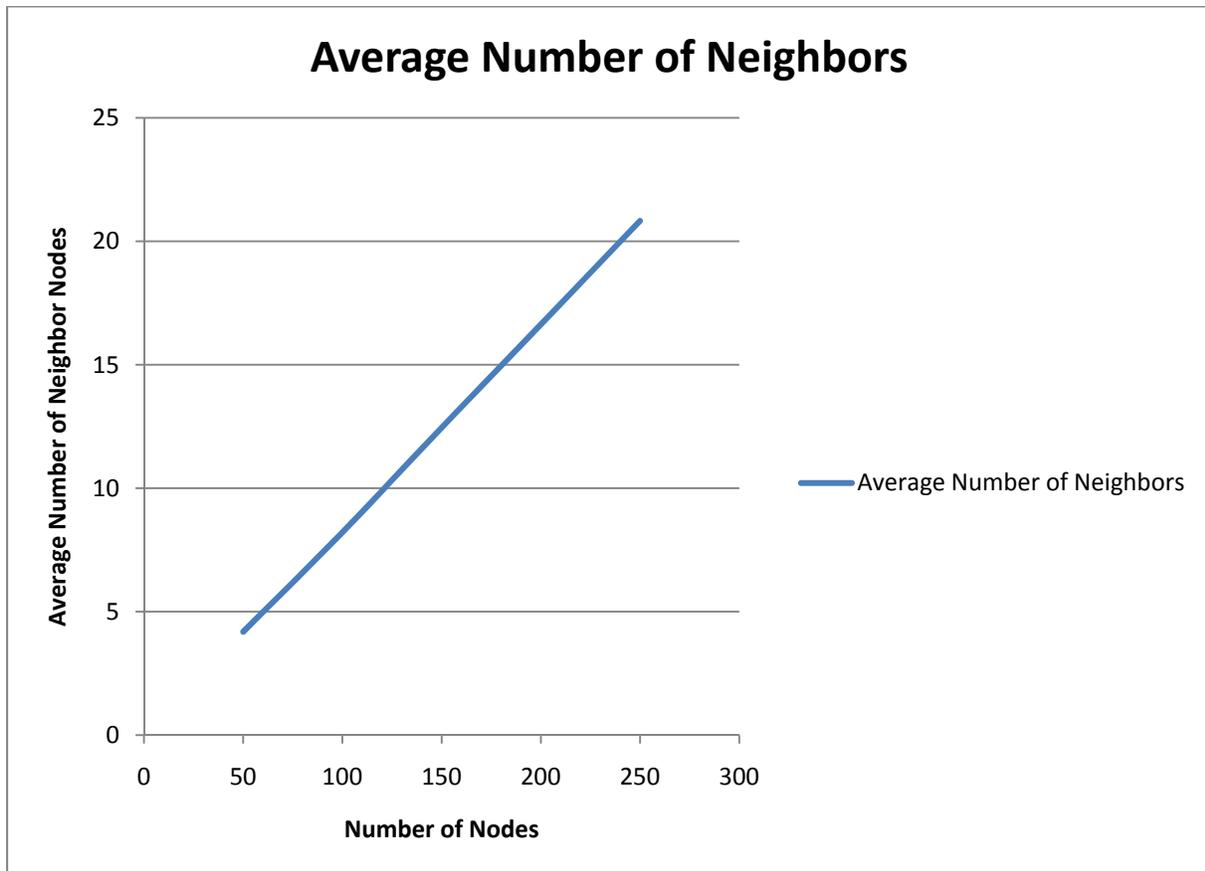


Figure 3.8 shows the average number of neighbors for each node

Figure 3.8 shows the average number of neighbors and how it increases as the number of nodes increases, and the average number of neighbors represents the number of neighbors for any nodes, and these number usually the node need to communicate with to get the information and exchange the date, and since this number increases, the number of messages exchanged will increase ,and the possibility of the collisions increases and also the number of resent messages increases, and this will lead to the increase of the algorithm duration.

And this explains the big jump at 200 for both cases the 3-hop and 2.5 hop for the number of bytes exchanged. And this also will explain the big jump for the algorithm duration in Figure 3.9.

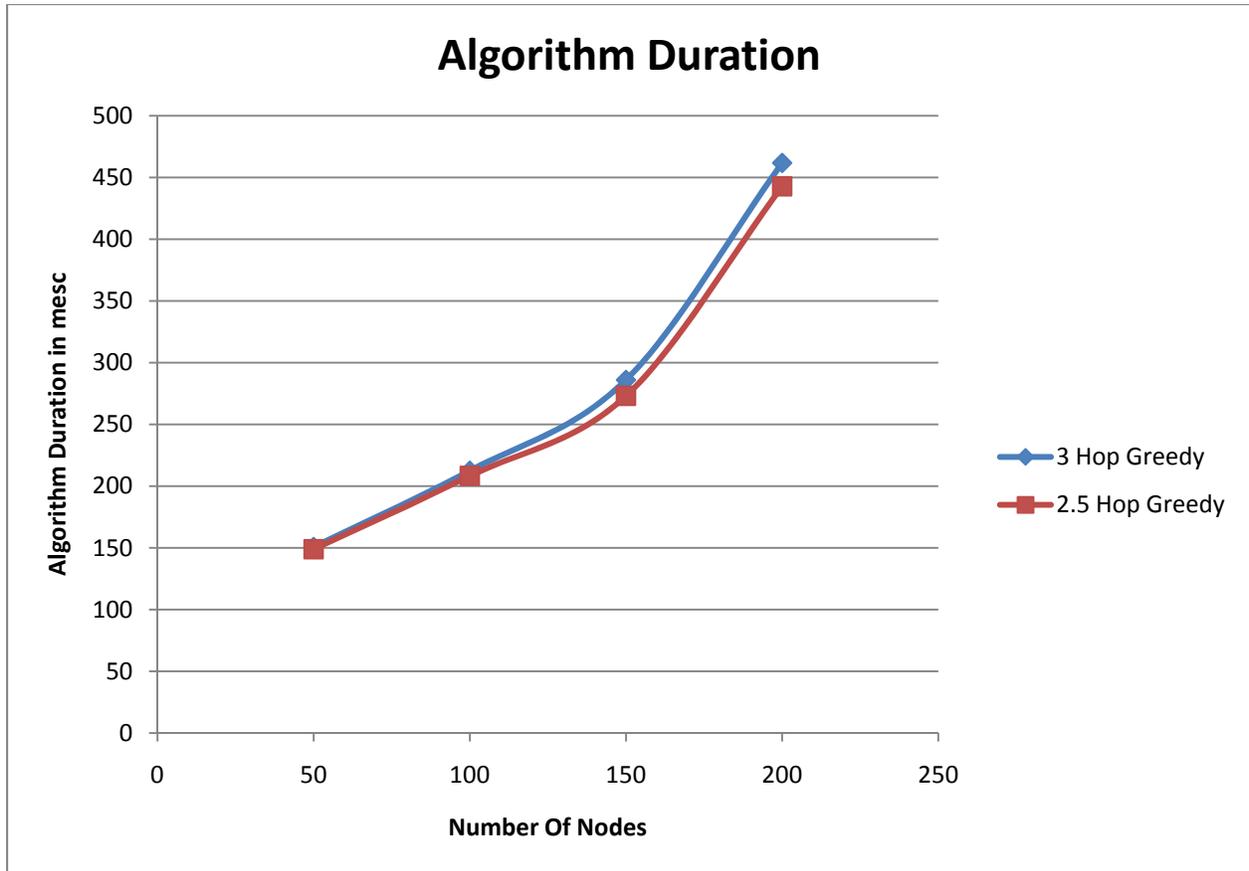


Figure 3.9 shows the Algorithm duration for 3 hop and 2.5 hop algorithms.

Figure 3.9 shows the difference in time spent for both algorithm to end, and shows that the 2.5 hop greedy algorithm is faster than 3 hop coverage set, and the difference is not big.

At the end of this chapter, the 2.5 algorithm was presented and compared to the 3 hop greedy algorithm, most of the figures showed the better performance of 2.5-hop, and the performance was not that big, and the main difference is that 2.5 hop algorithm forms directed graph.

Chapter 4

In this chapter, another approach for determining the coverage set for the clusterheads will be addressed. In Chapter 3 and 4, the 3-hop coverage set and 2.5-hop coverage are addressed, and this chapter, another approach which is 2.25 hop coverage set will be presented. I developed this approach which has less area of 2.25-hop coverage area.

4.1 2.25-Hop Coverage Set

In this coverage set, not all two-hop are connected and not all three-hop connected, for two hop clusterheads will be connected only if there is no any node between the clusterheads belong to one of them, and for 3-hop clusterheads, the clusterheads will be connected only if both of them have members neighbor to the member of the other clusterheads. And for the 3-hop clusterheads there are three cases

4.2 3-Hop Clusterheads Cases

This part will show the three cases that is possible between any two 3-hop clusterheads and explain which case belong to 2.25 hop coverage and which one is not.

Case 1:

This case happens when the two nodes between the two clusterheads are members to these clusterheads.

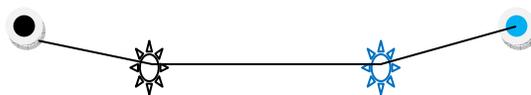


Figure 4.1 shows the Case 1

Figure 4.1 shows the first case, the first case occurs when the two 3-hop clusterheads have members neighbor to each other, and this case the two 3-hop clusterheads need to be connected.

The blue sun is an ordinary node and a member of the blue circle which is clusterhead cluster, and the black sun in figure 4.1 is an ordinary node and is a member of the black circle which is head cluster.

Case 2:

This case occurs when the one of the two ordinary nodes between the two clusterheads belong to the third clusterhead.



Figure 4.2 shows the Case 2

Figure 4.2 shows two clusterheads, one is blue and the other is black, and the one of nodes between these two clusterheads belong to different clusterhead which is the red sun node. According to the definition of the 2.25 hop, these two nodes does not need to be connected, because the blue clusterhead does not have a node which is neighbor of the black sun node or black ordinary node.

Case 3:

This case happens when between any two clusterheads there are no any nodes belong to the one of the clusterheads.



Figure 4.3 shows the Case 3 of 2.25 Hop coverage set

Figure 4.3 shows the third case, and this case happens when the nodes between these two clusterheads does not belong to any of them. And also this case does not need to be connected.

The difference here between 2.25 and 2.5 is the second case where in 2.5 the second case is connected while in 2.25 the second case does not need to be connected. in 3-hop all the cases are connected.

These three cases belong to the 3-hop clusterheads, but for 2.25 there are some cases where 2-hop not connected, and this is not the case for 2.5 hop and 3 hop, where all the 2-hop nodes are connected.

4.3 2-Hop Clusterheads Cases

Case 1:



Figure 4.4 shows the Case 1 for 2-hop clusterhead

In this first case, there are two clusterheads 1 & 2 and one ordinary node Ord1, Ord1 has joined to clusterhead1, the clusterhead1 & clusterhead2 are connected, because they don't have any node in between that does not belong to neither of them.

Case 2: It is the same case as 1, except that Ord1 belongs to clusterheads2 instead of clusterhead1 and the clusterheads also need to be connected.

Case 3: in this case the two-hop clusterheads do not need to be connected.

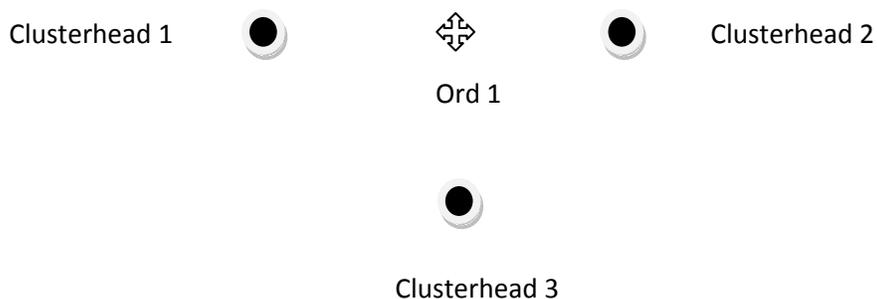


Figure 4.5 shows the Case 3 for 2-hop clusterhead

In this case there are three clusterheads and Ord1 belongs to another clusterhead (head3), so there is no need to have a connection between clusterhead1 & clusterhead2.

Case 4:

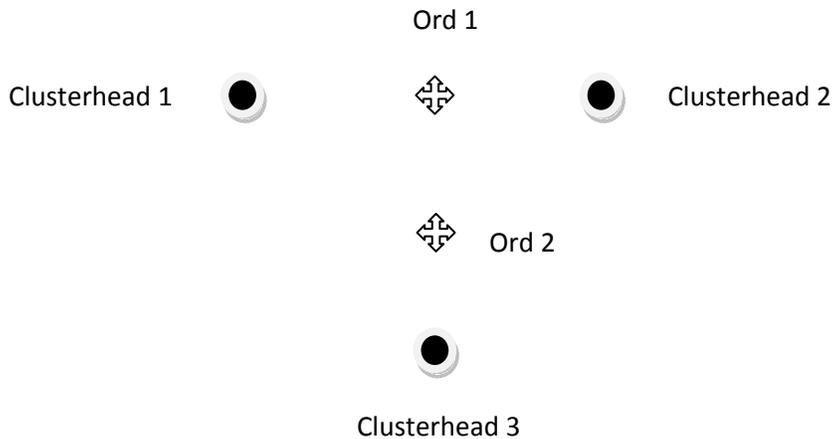


Figure 4.6 shows the Case 4 for 2-hop clusterheads

In figure 4.6 Ord1 belongs to clusterhead1 and Ord2 belongs to clusterhead3, so there are two ways between clusterhead1 and clusterhead2, either through Ord1 or Ord2. According to the definition of 2.25 Hop, there will be no direct connection between clusterhead1 and clusterhead2, even though the way through Ord1 is similar to the case 1 & 2. But there is still one node in between neither belong to clusterhead1 nor clusterhead2 which is Ord2 that belongs to clusterhead3.

So the direct connection between clusterhead1 & clusterhead2 is dropped, and there will be only connection between clusterhead1 & clusterhead3 and clusterhead2 & clusterhead3.

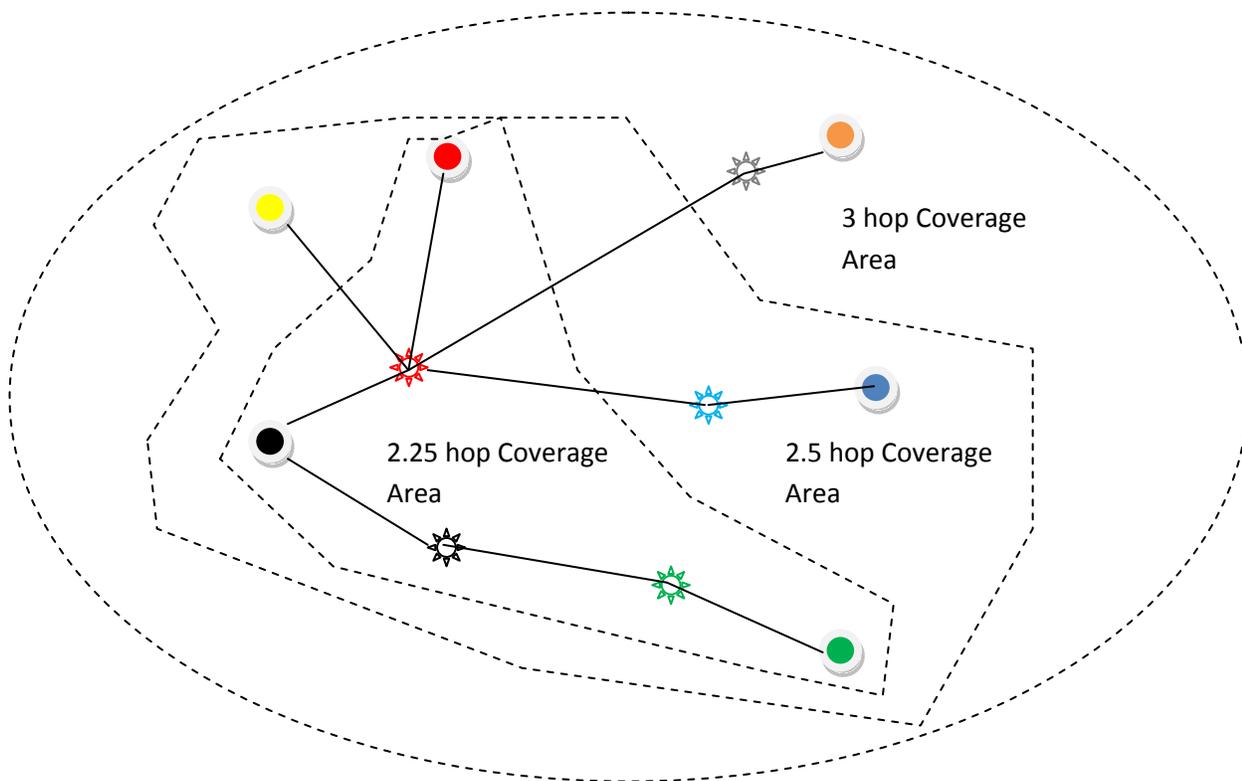


Figure 4.7 shows the Cover set area for 3-hop, 2.5 hop and 2.25 hop

Figure 4.7 shows the coverage set area for the black clusterhead, the Coverage set for 2.25 is smaller than 2.5, and also it is noticeable that in 2.5 and 3 hop coverage set are include all 2-hop clusterheads while in 2.25 is not the case, and also 2.25 does not include all three-hop clusterheads while include only the case where two 3-hop clusterheads has members neighbor to each other.

4.4 Proof of 2.25-Hop Coverage Set

To prove the correctness of 2.25 Hop, It depends on the theory of 3 Hop backbone connectivity [3], according to the lemma that says”To guarantee the backbone connectivity, it is required to make all 3-hop and 2-hop clusterheads connected”.

The proof is divided into two parts, the first part is 3-hop clusterheads, and the second part is 2-hop clusterheads.

4.4.1 Proof of the Connectivity of 3-Hop Clusterheads

In this part, the goal is to prove that even not connecting some of 3-hop clusterheads, the backbone is still connected. and I depended in this proof about making sure that all 3-hop clusterheads still connected, if they are not connected directly, they will be connected indirectly through another clusterhead.

For 3-hop clusterheads, there are only four possible ways to connect any 3-hop clusterheads:

a)-

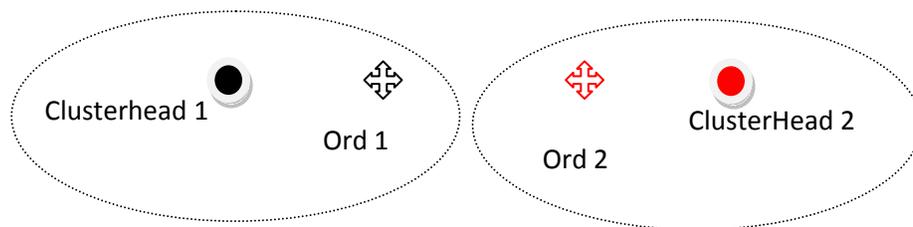


Figure 4.8 shows the first Scenario of 3-hop clusterhead

In Figure 4.8, we have two clusterheads which are 3 hops away, and also there are two ordinary nodes Ord1 & Ord2 and they are neighbor to each other.

Ord1 is a member of clusterhead1 (Ord1 joined to the clusterhead1). And Ord2 is a member of clusterhead2. According to the definition of 2.25 Hop coverage set, these two clusterheads has to be connected. Because clusterhead1 has a member Ord1 which is a neighbor of Ord2. Nothing here needs to be proved, since the connection is made not dropped.

b)-

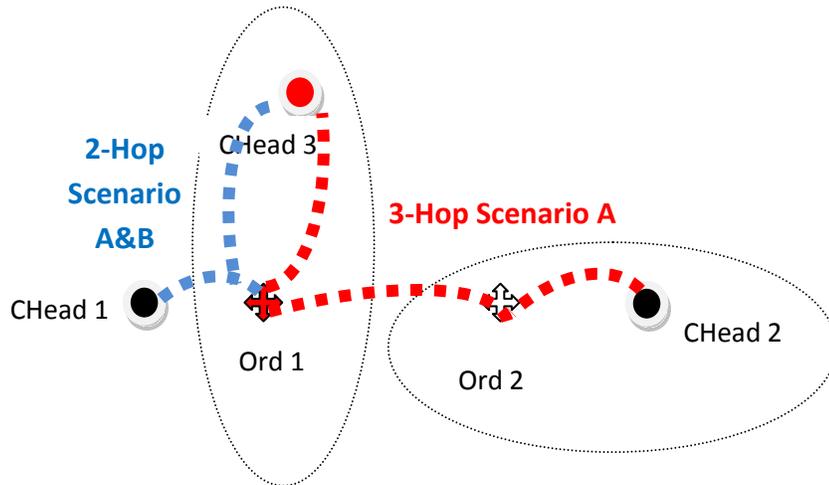


Figure 4.9 shows the 3-Hop Scenario B

In this scenario, there are three clusterheads, Ord1 joined to clusterhead3 and Ord2 joined to clusterhead3. clusterhead1 & clusterhead3 will be connected according to 2-hop scenario A&B, and clusterhead3 and clusterhead2 are connected according to 3-hop scenario A (the previous case). Therefore there will be no need to make connection between clusterhead1 and clusterhead2, because the clusterhead1 & clusterhead2 will be connected through clusterhead3.

c) This scenario is similar to the previous one, but Ord2 instead belongs to clusterhead2 it belongs to clusterhead3 and Ord1 belongs to clusterhead1.

d) This is the last scenario where Ord1 & Ord2 does not belong to clusterhead1 nor clusterhead2, they belong to other clusterheads.

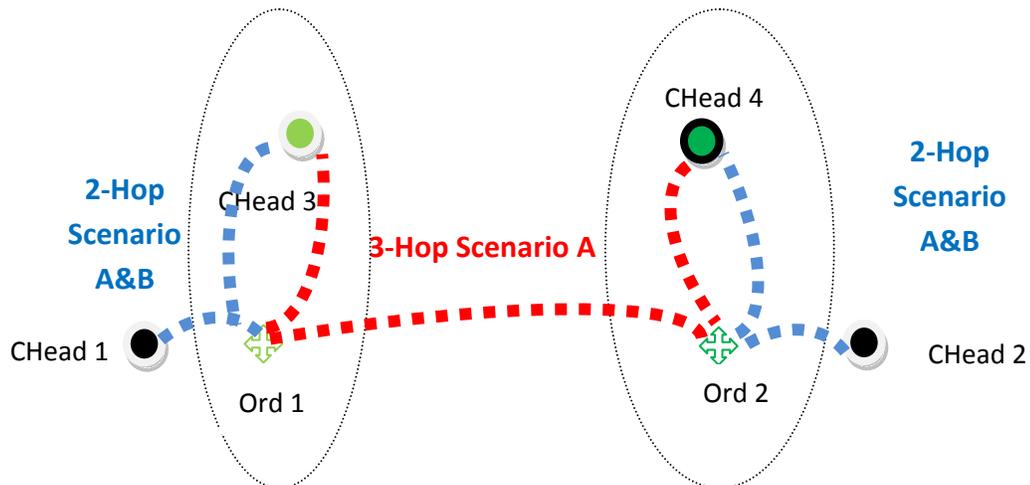


Figure 4.10 shows the 3-hop clusterheads scenario D

In this scenario Ord1 & Ord2 are members of clusterhead3 & clusterhead4 respectively. even though clusterhead1 & clusterhead2 are three-hop clusterheads, there is no need to do direct connection between them. clusterhead1 will be connected to clusterhead3 according to 2-hop scenario A, clusterhead4 and clusterhead2 will be connected according to the same scenario, and clusterhead3 & clusterhead4 will be connected according to 3-hop scenario A. thus clusterhead1 and clusterhead2 are connected through clusterhead3 and clusterhead4.

4.4.2 Proof of the Connectivity of 2-Hop Clusterheads

Two-hop clusterheads are connected only if there is no any ordinary node in between belong to the third clusterhead

To prove this, I need to use the mathematical induction to prove it is valid for all possible situations.

This is when $n = 1$

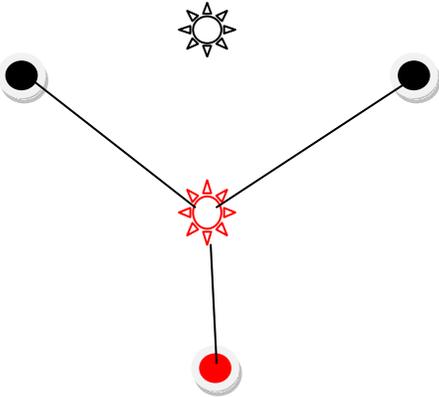


Figure 4.11 shows the simple case for mathematical induction proof

This is the base case, where the red star (ordinary node) does not belong to the red clusterheads, so there will be no connection between the two black clusterheads.

Now when $n = 2$,

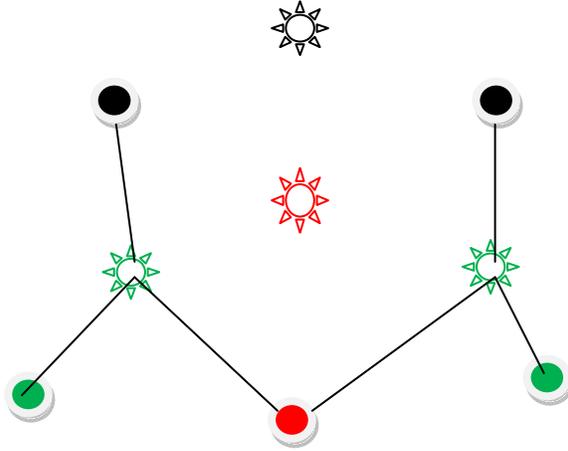


Figure 4.12 shows the case when $n=2$

Now here between the red and black there is intermediate green ordinary node that does not belong to both of them, going back to the base case, so there is no direct connection between the black and red clusterheads through the red star node, but both of them will be connected through the green node. Now the theory still holds for $n = 2$,

Now for $n = 3$,

Let us suppose there is another node between the black and the green clusterheads that does not belong to either of them, and this node is the blue node.

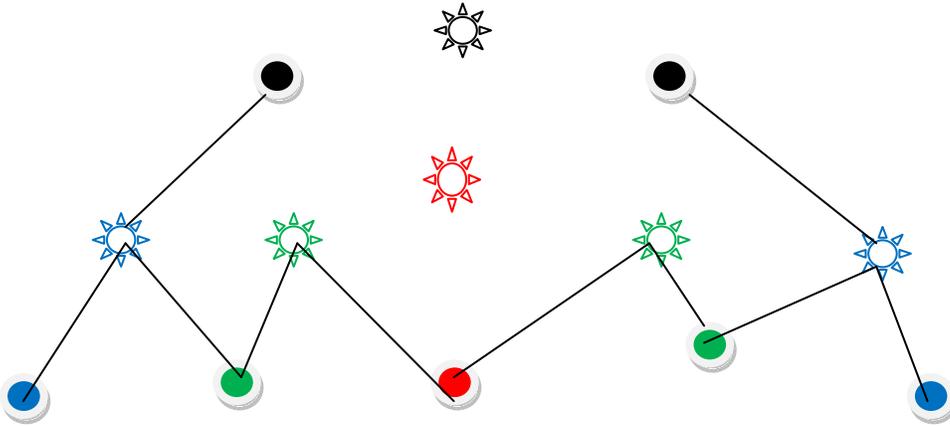


Figure 4.13 shows the case when $n=3$

It is noticeable that the two black nodes still connected, but through different nodes.

If we assume the graph with n number of nodes in the middle is connected,

Case $n = n$

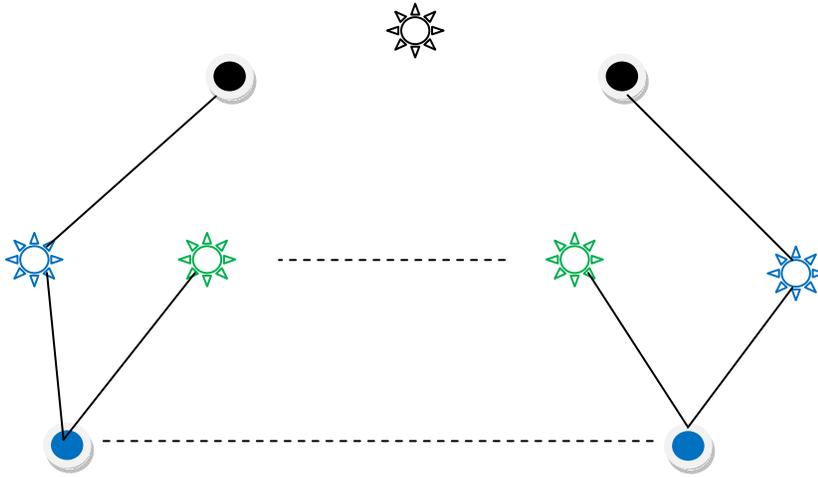


Figure 4.14 shows the case for n clusterheads added

Now if we add another node to the left and another to the right, that neither belong the blue nor the black clusterheads, the topology must be connected too.

Case $n = n+1$

Here another node with color yellow is added on the right and the left.

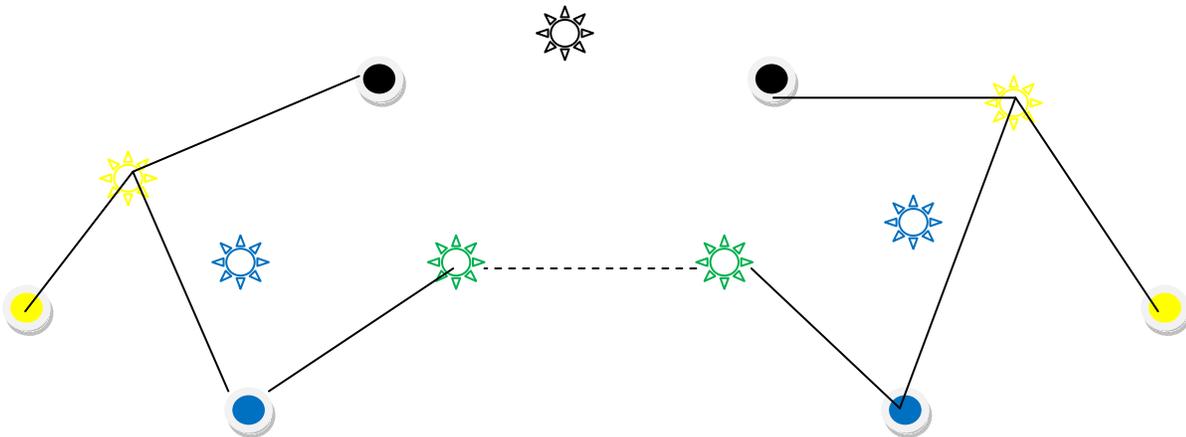


Figure 4.15 shows the case for $n+1$

if we look here, the yellow added node kept the two black connected, since the green in the middle still connected due to the assumption for case in n , so adding and node will not disconnect the graph, now let's assume if we add the node in the middle among the green nodes that does not belong to each one of them with the same case $n+1$

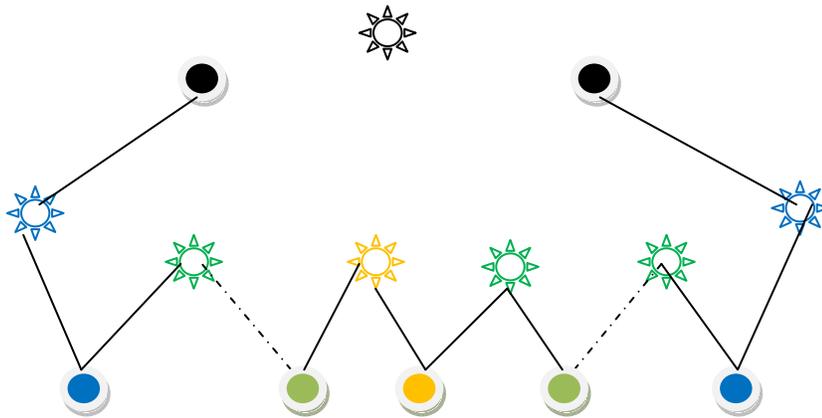


Figure 4.16 shows the case for n , but with details in the middle

Now in the figure 4.16 shows the case for n , but there is no dotted line in the middle, now if the assumption for n is correct, then adding node in the middle will stay correct too.

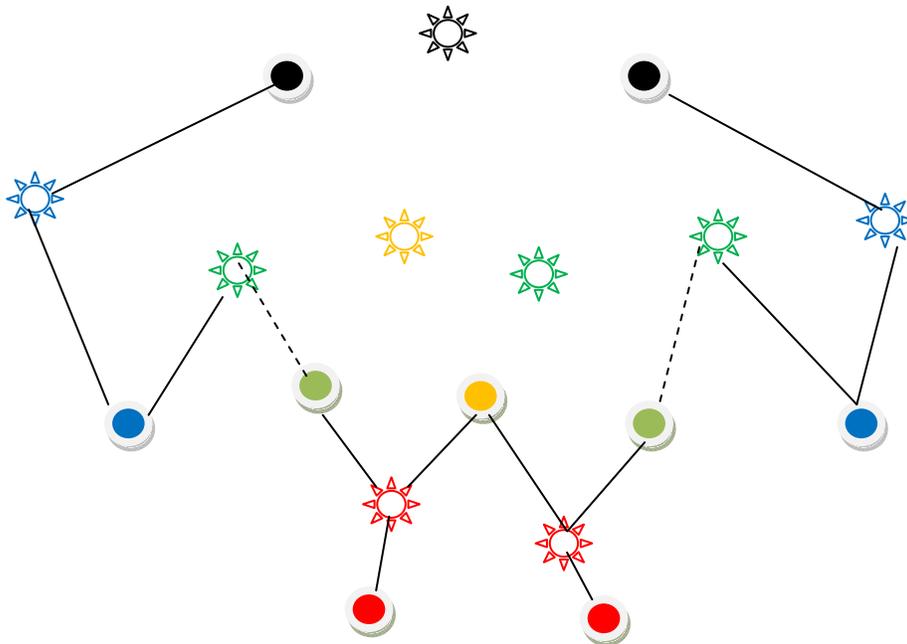


Figure 4.17 shows the case for $n+1$ when add them in the middle

since the red nodes that added does not belong to the yellow or green, then the yellow and green nodes will connect through the red nodes, and since the green and yellow still connected and the other graph still connected due in our assumption for n , therefore our assumption at n is correct.

The induction always stops at the base case. There is no node that does not belong to any clusterhead, so any intermediate nodes between three clusterheads belong only to one of the clusterheads; there are no intermediate nodes neighbors to three clusterheads belong to different clusterheads. Therefore the induction can't be stuck in infinite loop where intermediate nodes belong to both of them. Also adding any new clusterheads and intermediate node will lead to start of induction.

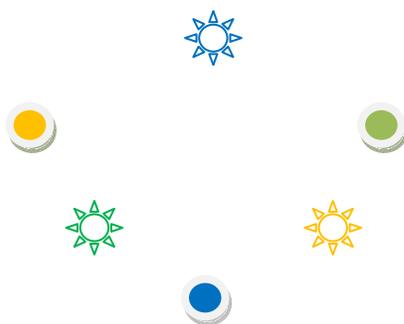


Figure 4.18 shows the case of impossible dead lock.

In Figure 4.18, it is impossible to have this case, if this happens then the algorithm will be stuck in infinite loop, where will be no connection between the three clusterheads.

Why is it impossible to have this case?

Due to DCA algorithm, there are many reasons:

- 1- The nodes only belong to one clusterhead.
- 2- The nodes belong to the biggest clusterhead.

According to these two reasons and back to Figure 4.18, this happens when the three star nodes are neighbor and all of them can see the three clusterheads. Existing of the green nodes means that the green clusterhead has weight bigger than the yellow and blow (the weights are unique and different, could be ID or MAC address or any other metrics), and also existing of the blow and yellow star nodes means the blow and yellow clusterheads have weights bigger than the

others respectively, and this is contradiction with the algorithm where only one clusterhead is the biggest.

Now, it is proved that for all value of n , the connection between two hop clusterheads is still connected by indirectly; therefore the topology is still connected too. 2.25 Hop coverage set still satisfies the condition of that all 3-hop clusterheads and 2-hop clusterheads still connected, it just beaks some circle path in the topology.

4.5 Experimental Results

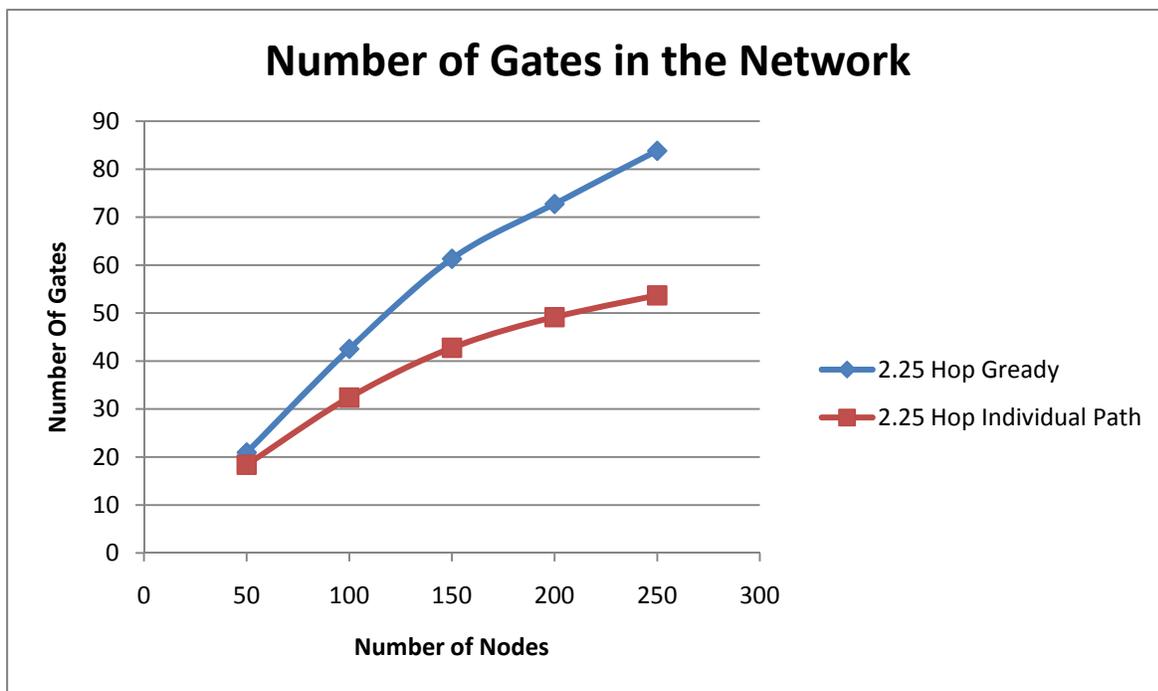


Figure 4.19 shows the number of Gateways for Greedy and Individual Path Algorithms

Figure 4.19 shows that the number of Greedy Algorithm is greater than the number of gateways for the individual path algorithm, and this is because the greedy algorithm work better with two round of exchange information, at which the available information about the neighbor is more and therefore this leads to less number of gates, but with one round of exchange information, the amount of information about the neighbor is small, and this will make greedy is worse.

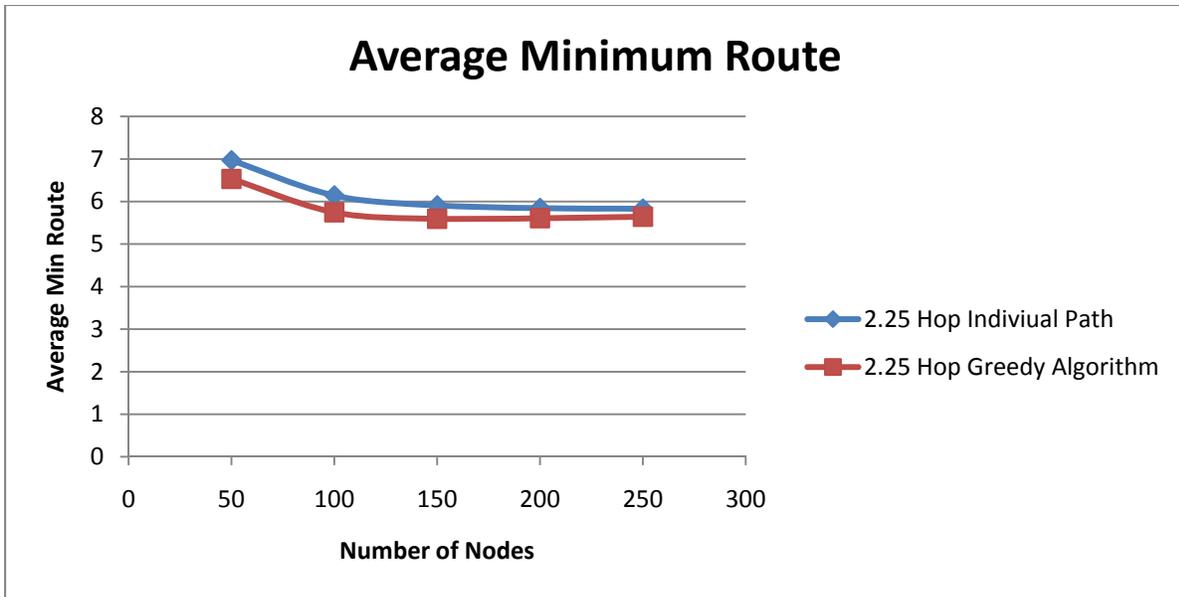


Figure 4.20 shows the Average minimum route for Greedy and Individual Path Algorithms

In figure 4.20, since the number of gateways is more for Greedy algorithm, therefore the Average minimum route is less.

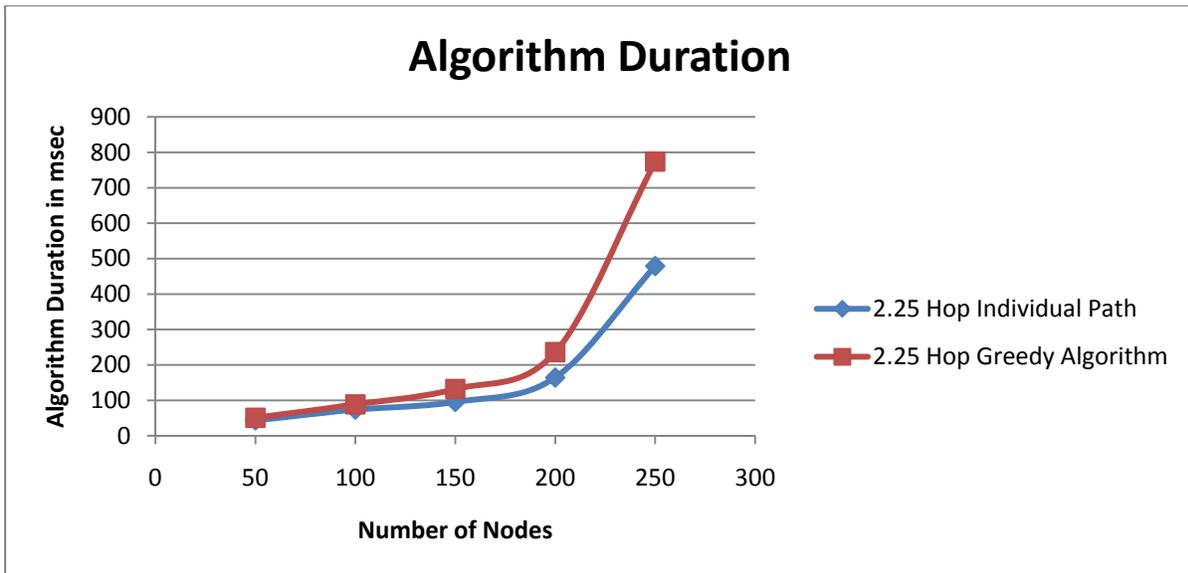


Figure 4.21 shows the Algorithm duration for Greedy and Individual Path Algorithms

Figure 4.21 shows that the algorithm duration in Individual path is less than in greedy algorithm, and this is expected because the announcing gateways is done into two stages for greedy while in one stage in individual path algorithm.

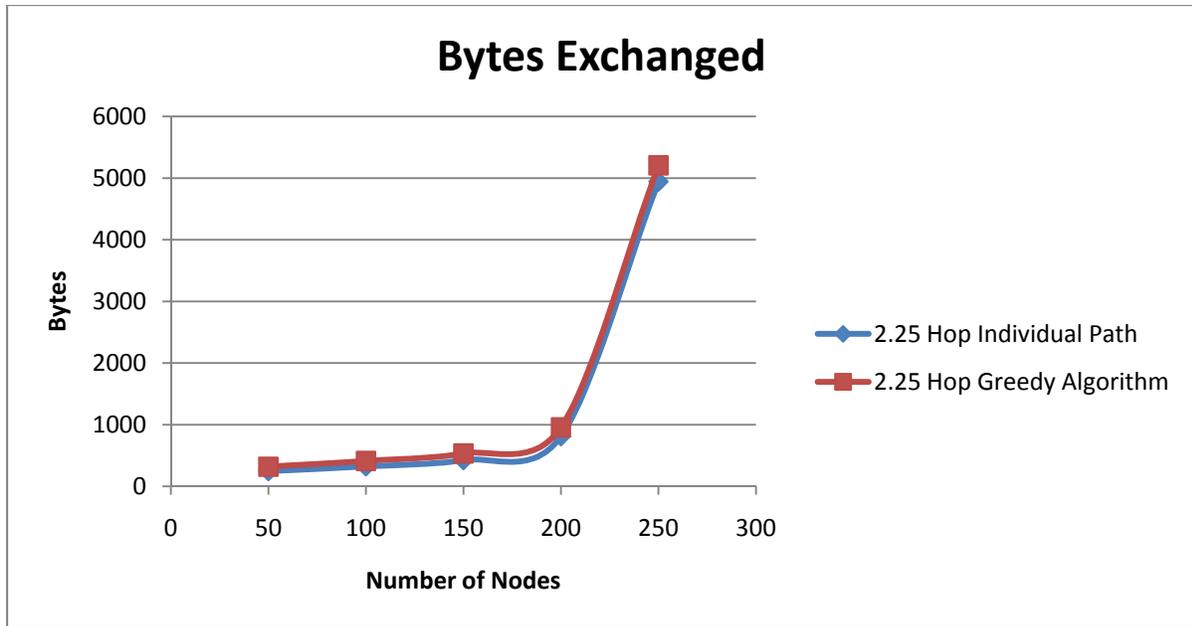


Figure 4.22 shows the Algorithm duration for Greedy and Individual Path Algorithms

Figure 4.22 shows that the bytes exchanged by greedy is little more than the bytes exchanged by the greedy algorithm, and this is also because of the announcing of the gateways in greedy is done in two stages.

At the end of this chapter, greedy algorithm gives bad results with 2.25 hop coverage because building the backbone is done in one round of exchange information, and the good choice is individual path algorithm which does not depend too much on the available information.

Chapter 5

5.1 3-Hop, 2.5-Hop, and 2.25-Hop Comparison

In this chapter, the comparison between 3-hop, 2.5-hop, and 2.25-hop coverage sets will be addressed, since the greedy algorithm does not produce good results with 3-hop and 2.25-hop, therefore they will not be included.

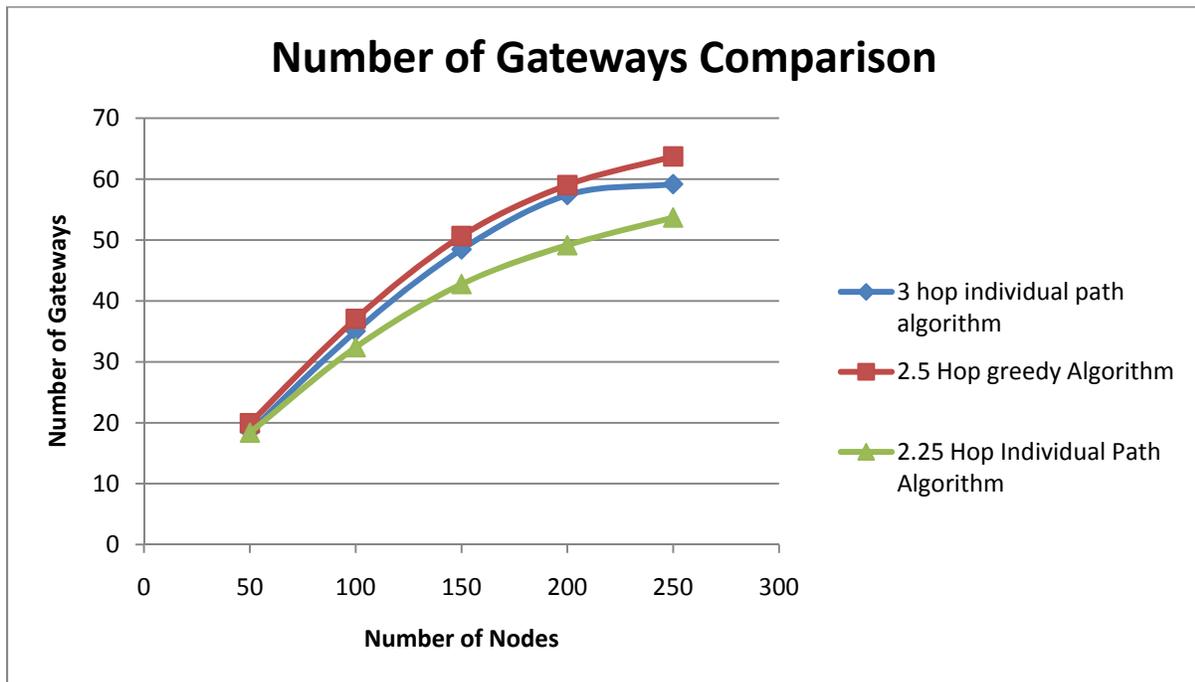
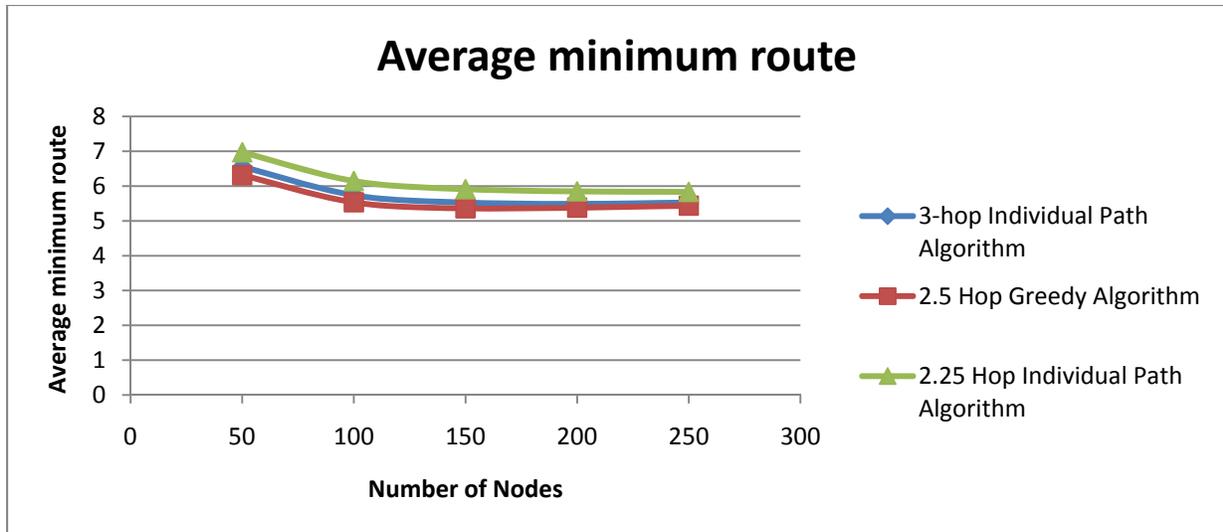


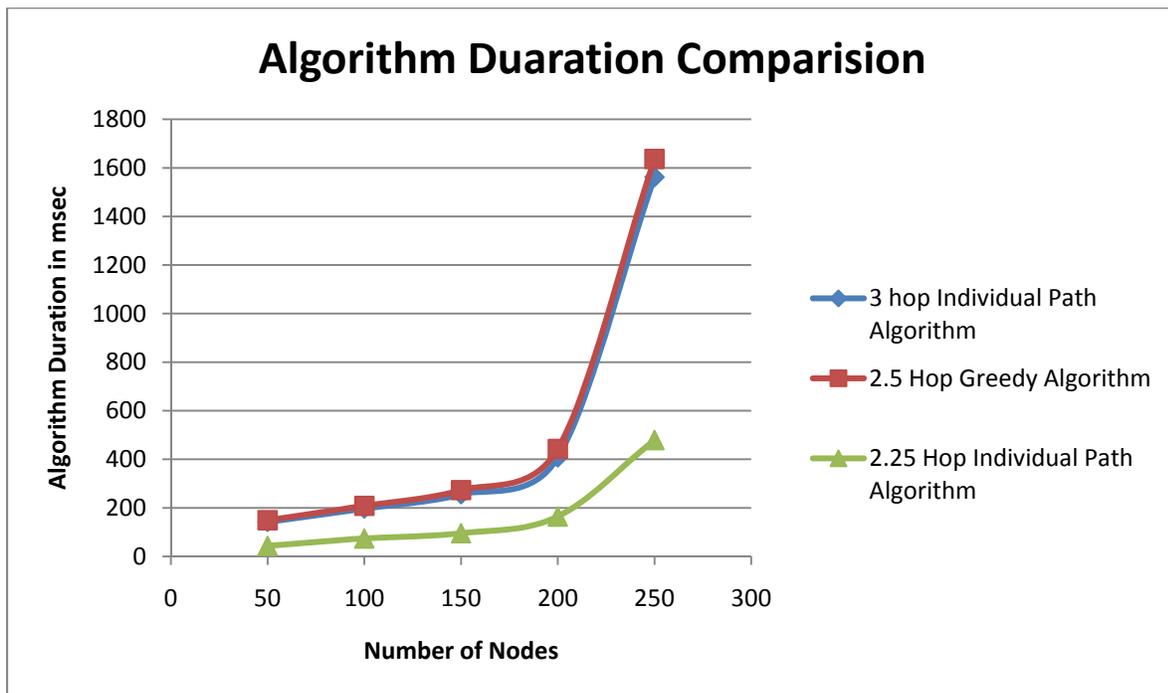
Figure 5.1 shows the number of Gateways for 3-hop, 2.5 hop and 2.25 hop coverage set

Figure 5.1 indicates that the number of gateways for 2.25 hop is less than the other two algorithms, and this due to the smaller coverage-set area of 2.25 hop.



5.2 shows the average min route for 3-hop, 2.5 hop and 2.25 hop coverage set

In figure 5.2, it is obvious that the average minimum route for 2.25 is the highest one, and this is because it has the least number of gateways.



5.3 shows the Algorithm duration for 3-hop, 2.5 hop and 2.25 hop coverage set

figure 5.3 shows that the algorithm duration for 2.25 hop is less than the other two, this is because 2.25 hop information exchange is done in one round of exchange information, and this will lead to less time to finish the algorithm.

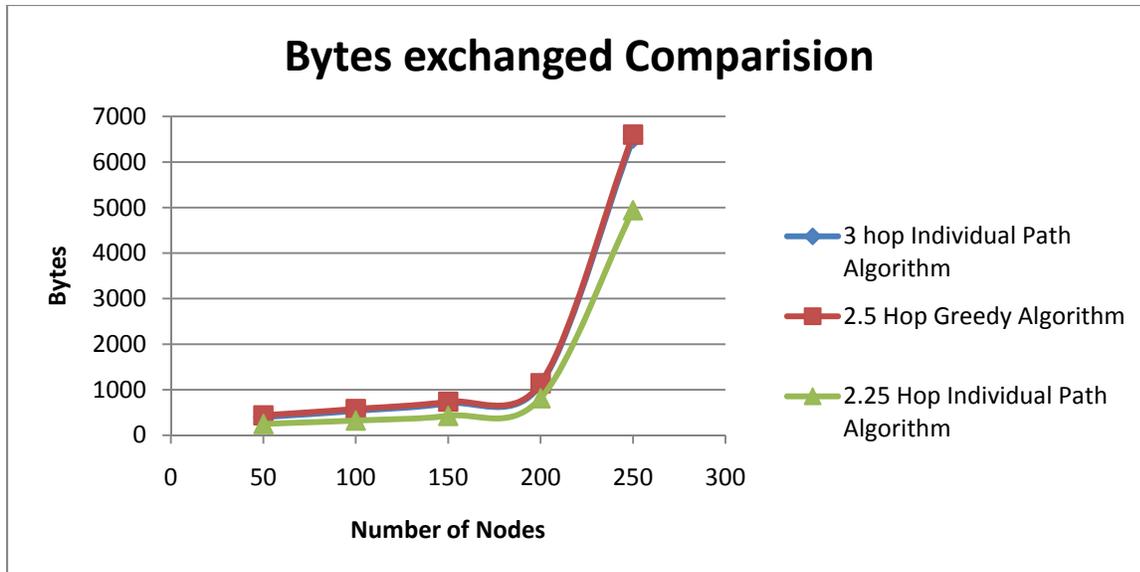


Figure 5.4 shows the bytes exchanged for 3-hop, 2.5 hop and 2.25 hop coverage set

In figure 5.4, the bytes exchanged in 2.25 hop is less than the bytes exchanged in the other two algorithms, and this is because of two reasons:

2.25 hop take one round of exchange information, while in 3 hop and 2.5 hop, it takes two round of exchange information.

In the 2.25 round of exchange information, the message sent from the ordinary nodes to only their clusterhead, while in 3 hop and 2.5 hop, the messages sent from all nodes to all neighbor nodes, and this will lead to more collision and resend the message again, and this is in turns increase the bytes exchanged.

At the end of this chapter, 2.25-hop usually outperforms 2.5 hop and 3 hop coverage sets, and gives better duration and less number of message exchange, and also produces less number of gateways, in spite of greater average minimum route.

Chapter 6

6.1 Heterogeneous Network

Homogeneous network was discussed in the previous chapters, while in this chapter the heterogeneous network will be addressed.

In a heterogeneous network, not all nodes are the same, there are some nodes different from the others. The nodes that will be clusterheads of the cluster will be denoted as red nodes, while the ordinary nodes will be denoted as white nodes. The red nodes will be pre-determined, not like in the homogeneous network where the clusterhead nodes were nominated. The purpose of this algorithm is to reduce the cost of the nodes, since each node has only one role, so that may lead to less cost, and also the flexibility of this algorithm can specify the recovery requirement for each node. The white nodes can be programmed to have two clusterheads or three clusterheads.

The algorithm which is implemented in this experiment is the Politburo algorithm, and this algorithm is introduced in [6].

6.2 Problem Description

Group of ad hoc network nodes in which nodes can be either red or white and each white node has an associated recovery requirement r_u . A Politburo is a set of red nodes such that each white node is a neighbor to at least r_u nodes. The goal is to find a Politburo with a small number of red nodes.

6.3 Algorithm

Before starting explaining the algorithm, some terminologies and notations need to be defined.

$u \in X \rightarrow$ this notation means u is an un-covered white node and it is a neighbor of red node X .

Covered nodes: at any steps of the algorithm, if a white node has r_u red nodes in Politburo, it is called a Covered node.

Un-Covered Node: at any steps in the algorithm, if a white node does not have r_u red nodes in Politburo, it is called un-covered node.

$\text{deg}(X)$: it means , the number of uncovered white nodes which are neighbor of the Red Node X.

$$\text{value}(u) = \min_{X \in u} \frac{1}{\text{deg}(X)}$$

The algorithm consists of simple basic steps, and these basic steps are repeated till all the white nodes are covered.

The Basic step of the Algorithm consists of the following sub steps:

1. Each Red Nodes send a message called degree, this message has the number of uncovered white nodes which are neighbor to the red node. $\text{deg}(x)$ will be sent to all neighbor uncovered white nodes.
2. Once the uncovered white nodes receives the degree message from all red neighbor nodes, then the white node will compute $\text{value}(u)$ and then send it to all red neighbor nodes.
3. Once each red node receive $\text{value}(u)$ from all uncovered white neighbor nodes, it checks whether it is candidate or not

Red Node X is candidate if

$$\sum_{u \in X} \text{value}(u) \geq \frac{1}{2}$$

Then the red nodes inform all uncovered white nodes about its candidacy.

4. Once the uncovered white nodes receives the candidacy information from all red neighbor nodes. The white node will insert them in τ array, and shuffle them randomly, then each white node will vote for the first r_u nodes in τ .
5. Once each red nodes receives all votes from the voting white nodes, then X red nodes will check

$$\sum_{u \in X \text{ and } u \text{ voted for } X} \text{value}(u) \geq \frac{1}{32}$$

if yes, the X red node will enter Politburo and inform all white nodes.

6. Each uncovered white nodes will update its r_u , r_u will be decremented by one for each red neighbor node has entered the Politburo.
7. Each white node will check if it is covered or not, and then inform all red nodes that has not enter the Politburo yet.
8. Repeat the steps from 1 – 7 till all white nodes are covered.

The algorithm will stop in two cases:

Red nodes: will stop if it enters the Politburo or all its white neighbor nodes are discovered.

White nodes: the algorithm will stop if it is covered by r_u red nodes.

6.4 Politburo Algorithm Message Types

Deg message: this message contains the degree of the red nodes, and it is sent by the red node.

| | | |
|-----------|--------------|--------|
| Source ID | Message Type | Degree |
|-----------|--------------|--------|

Figure 6.1 shows the degree message format

Figure 6.1 shows the message format of Deg message, and it has the following fields.

Source ID: it is the MAC address of the sender.

Message Type: it is the Message type, it has value zero for Degree message.

Degree: it is the number of uncovered white neighbor nodes.

Value Message: it is a message sent by the white nodes, and it has the value of the node.

| | | |
|-----------|--------------|-------|
| Source ID | Message Type | Value |
|-----------|--------------|-------|

Figure 6.2 shows the Value message format

Figure 6.2 shows the message format of Value message, and it has the following fields.

Source ID: it is the MAC address of the sender.

Message Type: it is the Message type, it has value one for Degree message.

Value: it has the Value of the white node.

Candidacy Message: it is message sent by red nodes, and it informs the white nodes whether it is a candidate or not.

| | | |
|-----------|--------------|-----------|
| Source ID | Message Type | Candidacy |
|-----------|--------------|-----------|

Figure 6.3 shows the Candidacy message format

Figure 6.3 shows the Candidacy message format, and it consists of the following fields

Source ID: it is the MAC address of the sender.

Message Type: it is the Message type, it has value two for Degree message.

Candidacy: it indicates whether the red node is candidate or not, and it has only two values, one is candidate and zero is not candidate.

Vote Message: this message is sent by the white nodes to the red nodes which are voted by the white node.

| | | |
|-----------|--------------|-----------------------------|
| Source ID | Message Type | List of the red voted nodes |
|-----------|--------------|-----------------------------|

Figure 6.4 shows the Vote message format

Source ID: it is the MAC address of the sender.

Message Type: it is the Message type; it has value two for Degree message.

List of the red voted nodes: it includes the red nodes that the white nodes voted for.

Politburo Message: this message indicates whether the red node has entered the politburo or not.

| | | |
|-----------|--------------|---------|
| Source ID | Message Type | Pulitro |
|-----------|--------------|---------|

Figure 6.5 shows the Politburo message format

Figure 6.5 shows the Politburo message format, and it consists of the following field

Politburo: it indicates whether the red node has entered the Politburo or not and it has two possible values, 1 for entering the Politburo and zero for not.

Cover Message: this message is sent by the white message to indicate whether the node has been covered or not.

| | | |
|-----------|--------------|---------|
| Source ID | Message Type | Covered |
|-----------|--------------|---------|

Figure 6.6 shows the Cover message format

Figure 6.6 shows the cover message format and the new field introduce is defined as follows

Covered: it indicates whether the white node has been covered or not.

6.5 Difficulty of the Algorithm

The algorithm consists of repeating many sub steps, and each sub step depends on the previous one, and each node can't start the next sub step till all other neighbor node finish this sub step, this kind of synchronization is difficult to achieve. and to solve this problem, each node need not to start the next basic step till all neighbor nodes finished this step, and this is done by sensing the media to check whether there is still message exchanged by other nodes.

6.6 Determining the Number of Red Nodes

To determine the number of reds nodes that guarantee the feasible solution for the problem, the following approach was done.

Since the Area that the nodes were distributed in are fixed, the coverage radius of the node is fixed too. in general the number of red nodes required is the number that cover the whole area.

In this algorithm there are three r_u tested, $r_u = 1$, $r_u = 2$ and $r_u = 3$.

The number of red nodes when $r_u = 1$:

Since the area is fixed, therefore the number of red nodes does not depend on total number of nodes, the only requirement is to choose number of red nodes that will cover the whole area A.

Ideally the number of Nodes to cover the Whole Area is : $\frac{A}{\pi r^2}$.

Since A is square, and coverage area of the node is circle, therefore this number will not cover the whole square, because the circles can't fit the square completely.

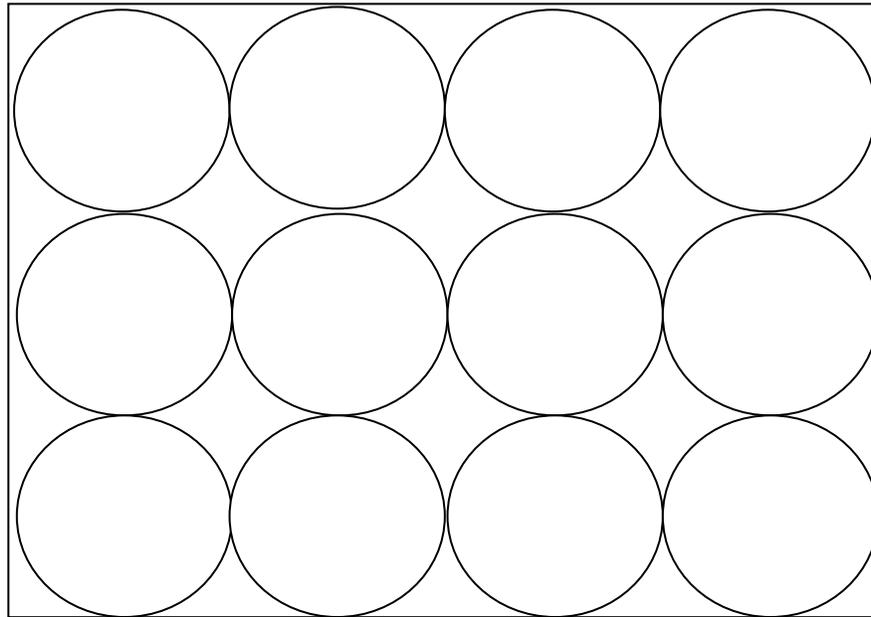


Figure 6.7 shows how the circle forms can't fit the square

Figure 6.7 shows that if the red nodes was distributed in such way as in the Figure, it is still can't cover the whole area, and to guarantee to cover the whole area, these uncovered areas need to be fit by another nodes, and this will lead to the number of nodes be $(2\frac{A}{\pi r^2})$.

This number is considered good enough to be as a start number. And since this number is generated from an ideal case where is rare for the nodes to be distributed in such a way, therefore bigger number is expected. So the topology will generate this number of red nodes and tried for 400 times, and then increase it till the feasible solution is obtained.

And for the $r_u=2$, and $r_u=3$, the number of red nodes will be $(4\frac{A}{\pi r^2})$ and $(6\frac{A}{\pi r^2})$ respectively. It is obvious that the number does not depend on the number of the nodes. However it is expected that this number will increase little bit as N increases till will be fixed.

6.7 Feasible Network Topology

To make the network topology feasible, two conditions has to be satisfied:

1. The whole topology has to be connected.
2. Each white node has to have at least r_u red neighbor nodes.

6.8 Experimental Results

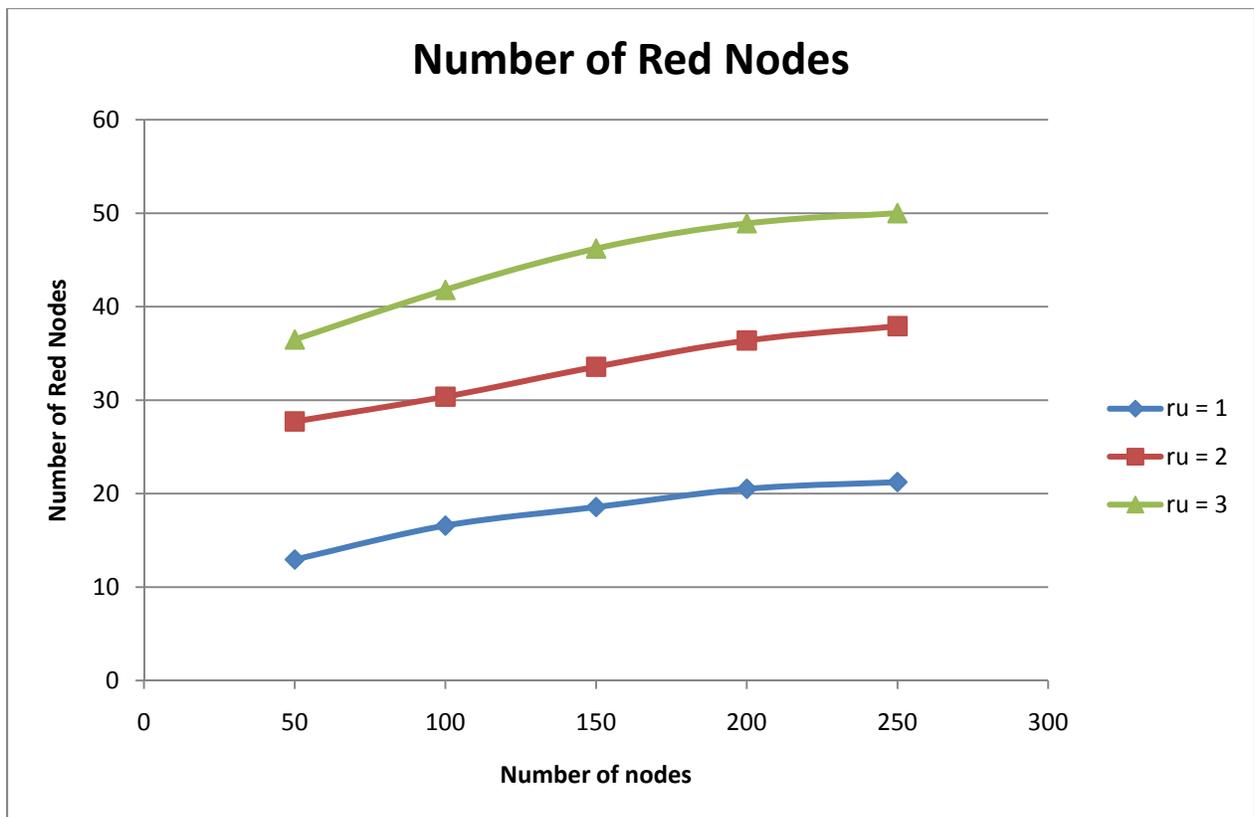


Figure 6.8 Shows the Number of Red Nodes

Figure 6.8 shows the number of red nodes, the figure shows that the number of red nodes increase gradually and goes to be constant, and this is because at certain number of red nodes, the whole area is always covered, there is no need to add any more red nodes, even if the white

nodes increases, and the reason why is the number of red nodes is less in small number of nodes, this is because the white nodes does not cover the whole area completely, so less number of red nodes are enough. It is noticeable that as r_u increases, the number of red nodes increases, and this clear because each white node needs two or three nodes instead of one.

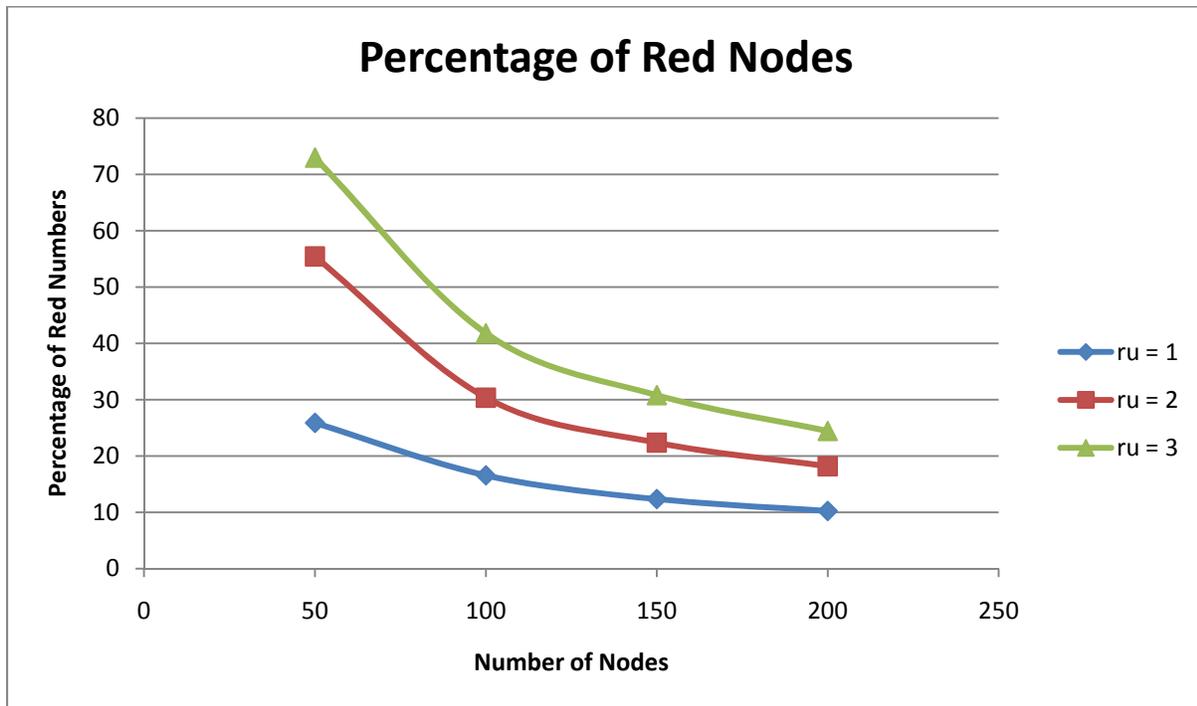


Figure 6.9 shows the percentage of the red nodes to the total number of Nodes

Figure 6.9 shows the percentage of the red nodes to the total number of nodes, and the curve goes down as the number of node increases, and this is due to that the number of red nodes is almost constant and therefore the percentage decreases as the number of nodes increases.

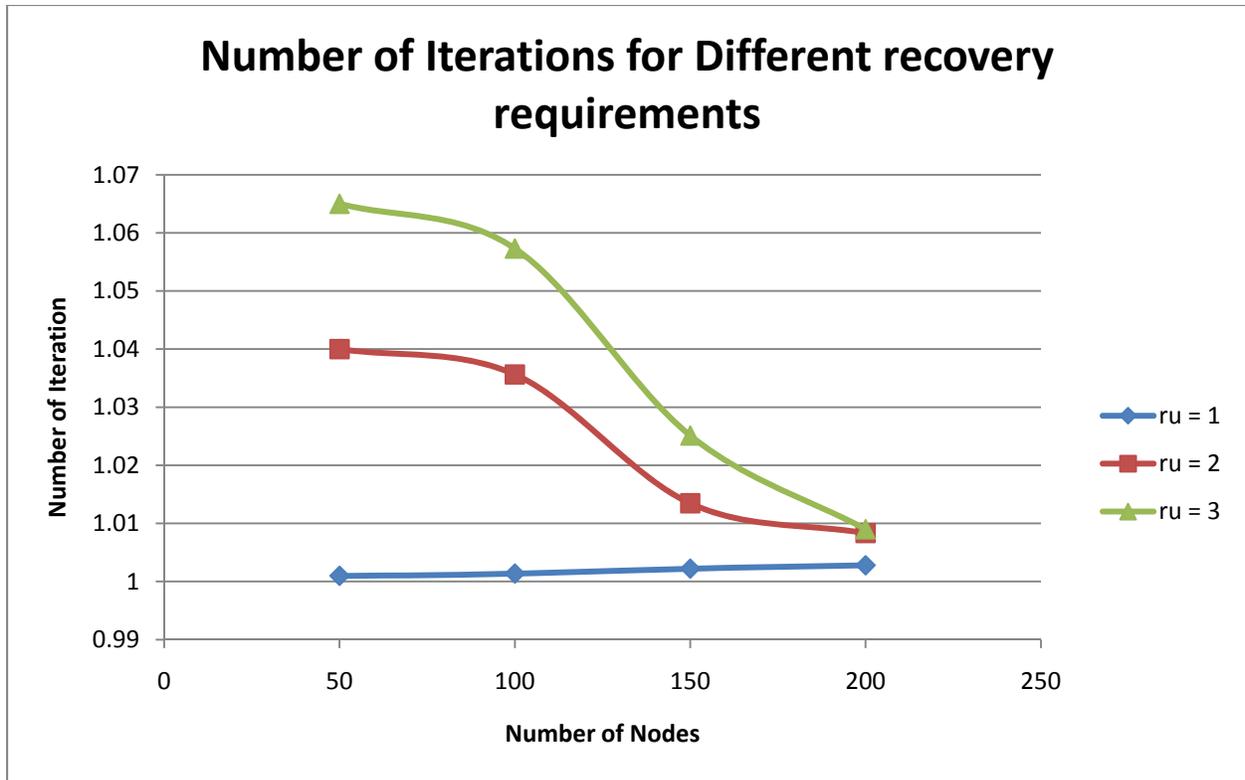


Figure 6.10 shows the number of Iteration for each node to finish

Figure 6.10 shows the decrease in the number of Iteration as the number of nodes increase, and this decrease due to the larger number of red nodes at 50 and 100 than the white nodes need, and since there are more number of red nodes, then the algorithm has a lot of choices and candidate to choose from, and this will make the algorithm set after more number of Iterations.

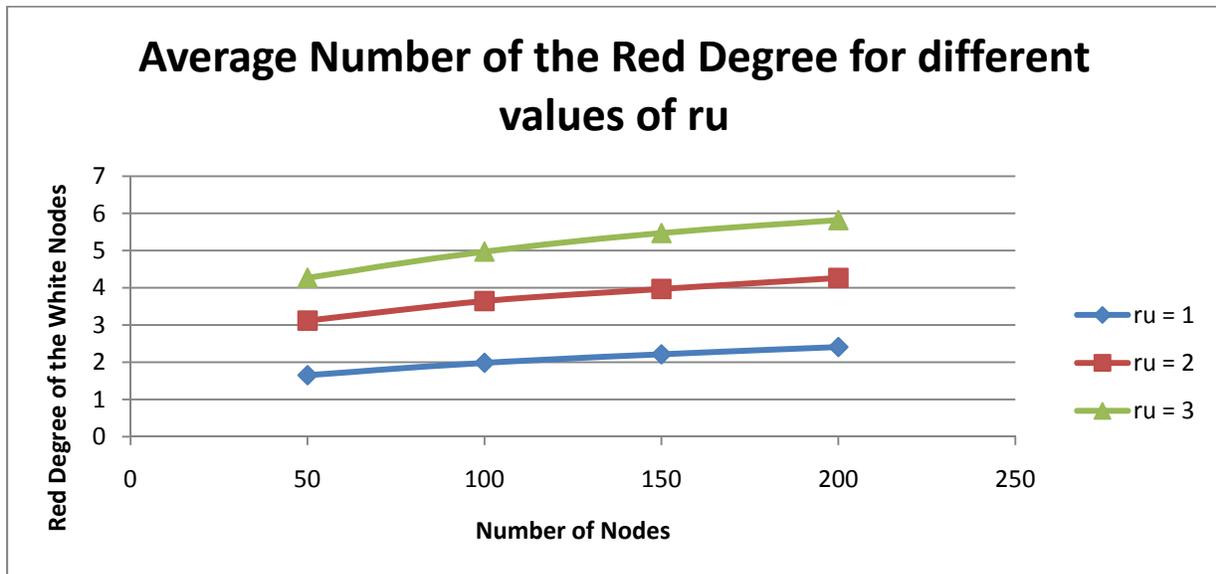


Figure 6.11 shows the red degree

The red degree is the number of red nodes in Politburo which are neighbor to a white node, and as the red degree approaches to r_u is better. Figure 6.11 shows the red degree increases gradually, and as number of nodes increase the increase the red degree is smaller.

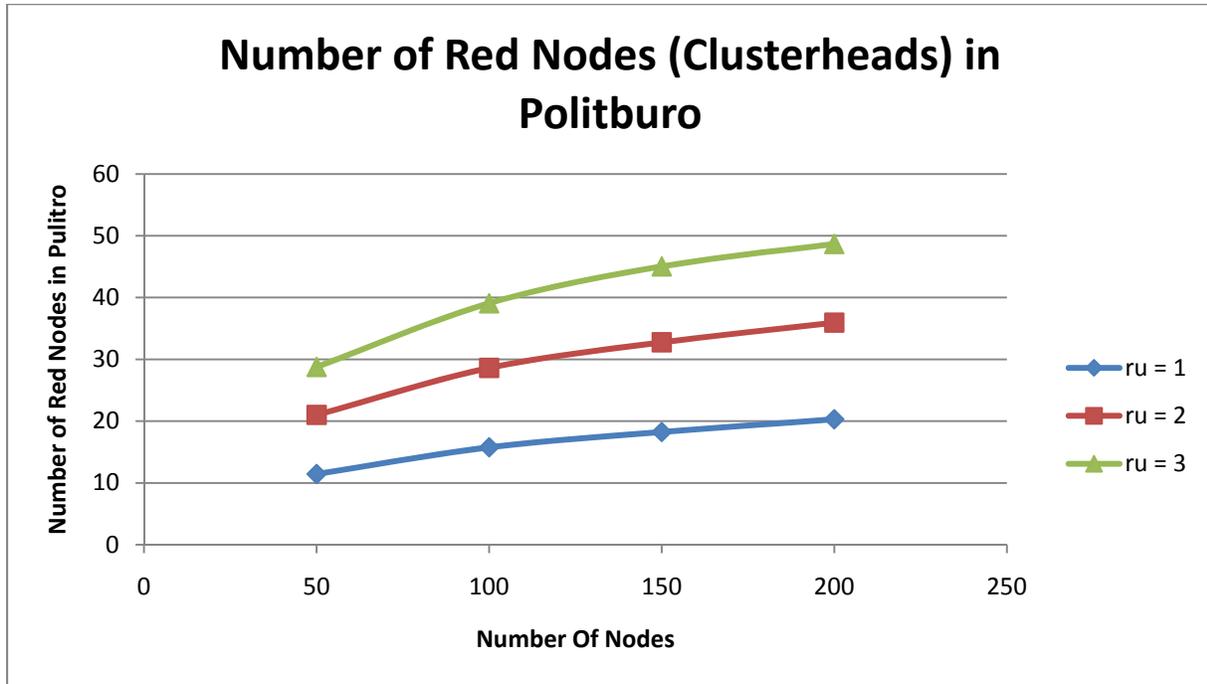


Figure 6.12 shows the number of red nodes in Politburo

Figure 6.12 shows that the number of red nodes in Politburo increases as number of nodes increases, and also the increase in the number of red nodes in Politburo becomes smaller and smaller, and this is because the number of red nodes converges to fixed number at large number of nodes.

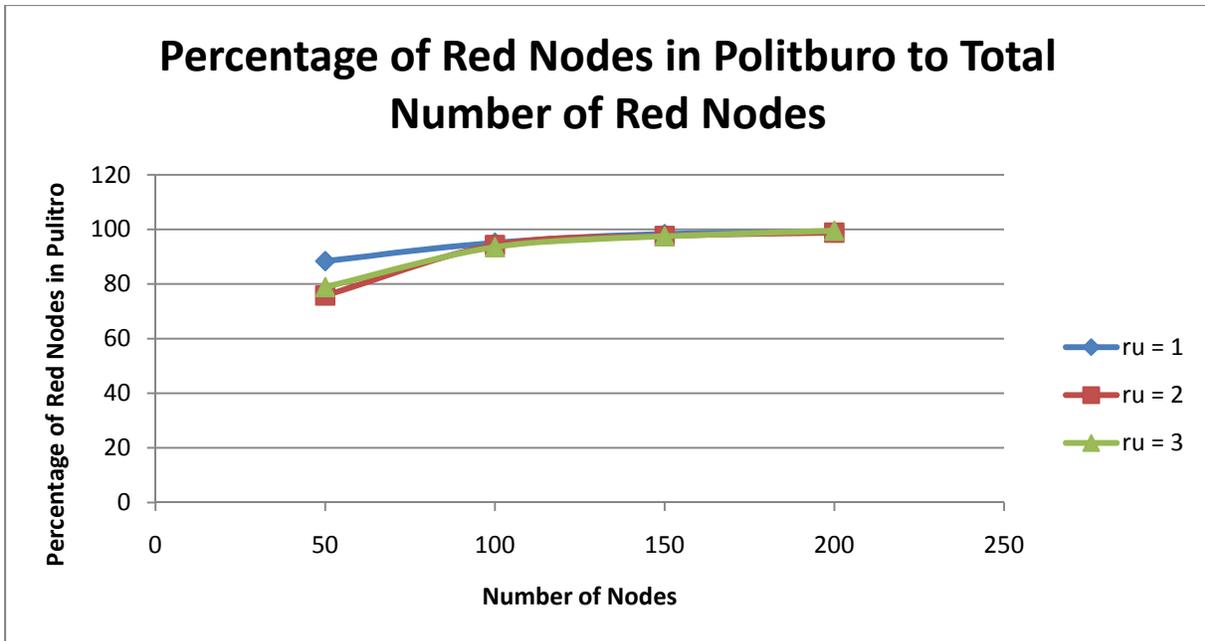


Figure 6.13 shows the percentage of the red node in Politburo to the Total number of Red nodes

Figure 6.13 shows that the percentage at small number of nodes is less, and this is because the number of red nodes in this region is more than the recovery requirement for each node. As number of nodes increases the percentage approach 100, and this is because of two reasons:

The number of red nodes is not big, so the algorithm can't get much improvement.

The condition in the sub step 5, which is $\sum_{u \in X \text{ and } u \text{ voted for } x} \text{value}(u) \geq \frac{1}{32}$, is easy to satisfy, because 1/32 is small number which most of the red nodes even if they get one vote will enter the Politburo. And the number is made small to make the algorithm finish faster and to avoid the more number of Iterations.

At the end of this chapter, the Heterogeneous network has different approach from the homogeneous network, and is good in such a way the cost of the nodes becomes less, and this is because each node will have predefined role, and Politburo algorithm is one of the algorithms designed to make the number of red nodes smaller, and because of the relaxes in some conditions that make the algorithm work faster made the algorithm is less optimal but it is still reasonable.

Chapter 7

In this chapter, a brief comparison between the homogenous and heterogeneous networks are presented, since both of them has different approaches, the applicability of each one depends on the application that used.

7.1 Homogeneous and Heterogeneous Networks Comparisons

This table shows some of the main differences between each kind of network

| Comparisons | Homogeneous Network | Heterogeneous Network |
|-----------------------------------|---|---|
| Type of Nodes | In Homogeneous network, there is only one type of nodes, in other word s all the node are similar. | There are two types of Nodes, one is red node and the other is white node. |
| Determine the clusterhead | in this Network, the clusterheads are selected automatically, where the algorithm itself will elect the nodes to be the clusterheads. | The role of the nodes are predetermined, but an algorithm can be applied to make this number is smaller. |
| Algorithm used in the experiments | DCA, algorithm which select the nodes that biggest weight in such way there is no two clusterheads neighbor. | Politburo Algorithm used, this algorithm is extension to the greedy algorithm but with relaxing some constraints to make it run faster and keep the performance reasonable. |
| Recovery requirement | Since the DCA algorithm is used in this experiments, the algorithm only work when the recovery requirement is one | Politburo algorithm can work for any number of recovery requirement |
| Time consuming | This algorithm if compared with $r_u = 1$ for Politburo, it takes less time, and this is because the algorithm finishes in one iteration. | This algorithm takes longer time and this is because the algorithm may not converge in one iteration and has more sub-steps than DCA |

| | | |
|-------------------------------|--|--|
| | | algorithm. |
| Number of clusterheads | DCA algorithm in general gives less number of clusterheads compared to Politburo $r_u=1$ | This algorithm due to the relax of some constraints produces more number of red nodes |
| Position of clusterhead Nodes | There is no two clusterheads neighbor to each other. | Two red nodes or even three can be neighbor, and this is give chance to have more number of red nodes in fixed area than in DCA. |
| Application | It is good in area where determine the clusterheads needs to be automatically | It is good in fields where red nodes are pre-determined |
| Cost | since each node can play two roles, therefore this may lead to higher cost of the node manufacturing | Since each node has only one role, therefore the node manufacture may be less. |

Table 7.1 shows the general comparisons between the Homogeneous and Heterogeneous network.

Number of clusterheads Comparison in Both Types of Network:

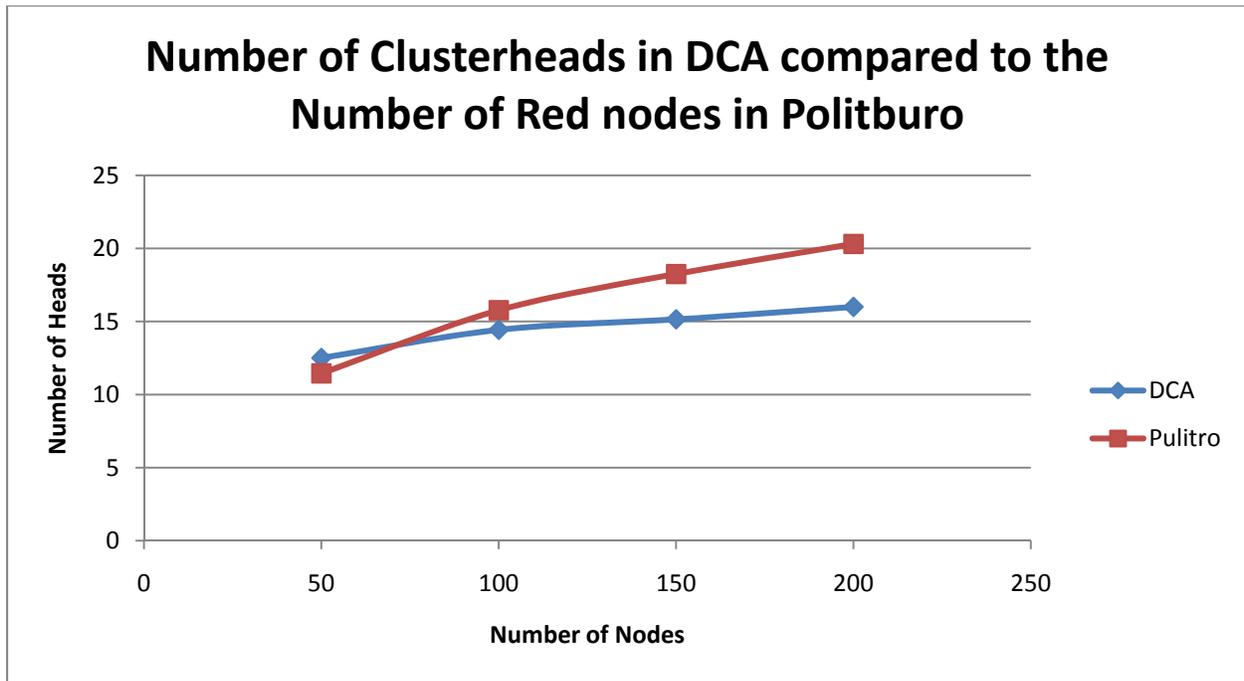


Figure 7.1 shows the number of red nodes and clusterheads in both type of network

Figure 7.1 shows the number of clusterheads in DCA algorithm compared to the Number of red nodes in Politburo, and from the figure, we can say DCA algorithm outperforms Politburo algorithm especially when number of nodes increases, the comparison made for $r_u = 1$ for Politburo.

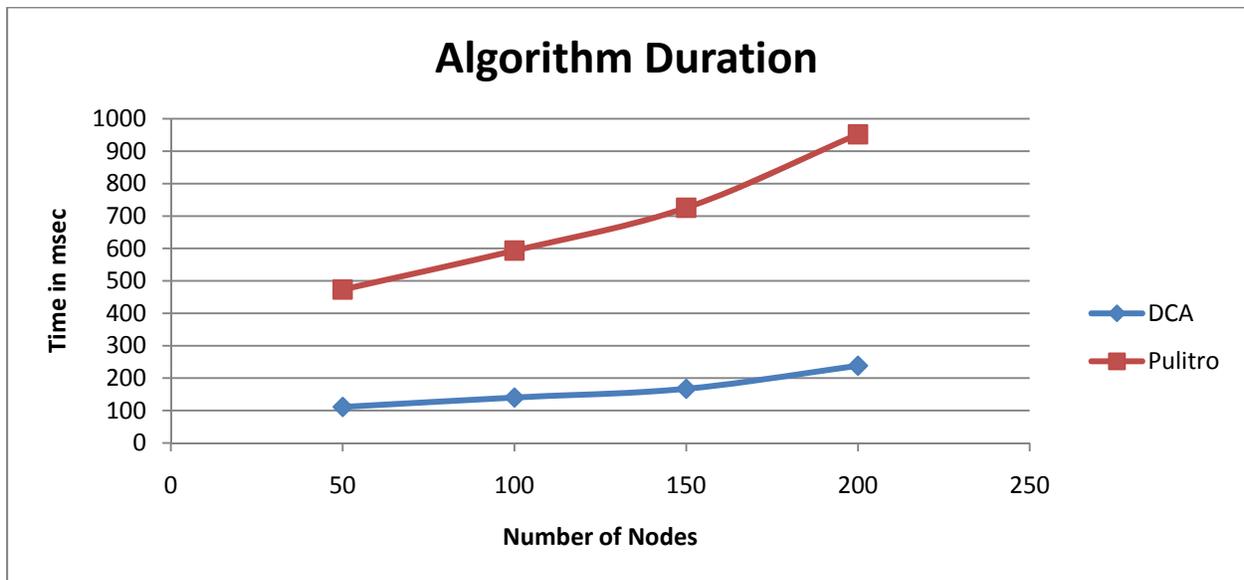


Figure 7.2 shows the time consumed for each algorithm

Figure 7.2 indicates that the algorithm duration for DCA is less than in Politburo and this is because the Politburo algorithm consist of repeating many basic steps and each steps consists of 6 basic sub steps, while DCA algorithm consist of only one step where the nodes with higher weight will send CH message and other nodes will reply with Join message.

7.2 Conclusion

Homogeneous network and Heterogeneous network both may be required in the application of wireless sensor network. The algorithm applied for Homogeneous network may outperform the algorithms implemented for heterogeneous network. However due to the predefined role of the nodes in Politburo Algorithm, this may help to lower the cost of the manufacturing the nodes.

Chapter 8

8.1 NS3 Simulator

NS3 Simulator is an open source simulator for networking. The experiments were implemented by version 3 of the tool. NS-3 is not an extension for NS-2, it is new simulator which does not support API of ns-2. Since it is still new, there are some models still not available in NS-3. However they are adding during the project being built.

8.2 Implementation

Wi-Fi Model:

The NS-3 has a model for Wi-Fi model (IEEE 802.11a); this model is used to implement the nodes. And the default settings were used.

8.3 Collisions

Collision is considered one of the main problems of wireless network. the collision usually happens when two nodes or more send the data at the same time, then these data collide at the destination or in middle. The destination gets in this case ruined data.

Although the Wi-Fi model supports collision sensing, so if the node senses data being transmitted, it stops sending and resend after a while, still collision may occur for the hidden terminals.

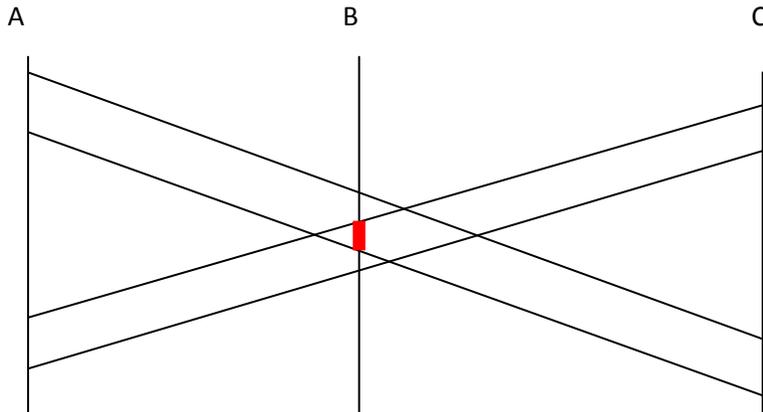


Figure 8.1 shows the collision, Collision in Multiple Access by Adrian Conway

In Figure 8.1, A and C try to send data, and they don't see each other. The data has been sent after checking the media, and since they don't see each other, so they starting sending. Once the data arrive close to B or at node B, the data (packets) coming from both nodes collide.

8.4 Back-off Algorithm

To solve the problem of collision, a simple random Back-off algorithm was implemented for each node. After the packets collided, each node starts to send the same packet again, to avoid sending at the same time to prevent collision again, the two nodes don't send immediately, they wait random amount of time and resend the packets again. this amount of time is calculated according to this formula:

$$T = \text{Rand1}(0, \text{Max1}) * \text{TravellingTime} + \text{TravellingTime}/\text{Max2} * \text{Rand2}(0, \text{Max2}).$$

Travelling time: it is the time that the packet takes to go from one node to another, and since this time depends on the distance between these two nodes, therefore the time was computed according to the maximum distance between two nodes within the same coverage area.

The travelling time was divided into slots; the number of slots is Max1. The value of the Max1 can be any value, but the larger the value is the longer time will be for resending. This value needs to set in such a way that done not take very long time and at the same time to avoid most of the collisions.

The other part of the formula is intended to send the packet within this slot, in other words, the slot is divided into Max2 sub-slots.

Each node chooses random number for Max1, this number determines which slot will be used the send the packet, then choose another random number for Rand2 to determine which sub-slot within the Rand1th slot the packets will be resent.

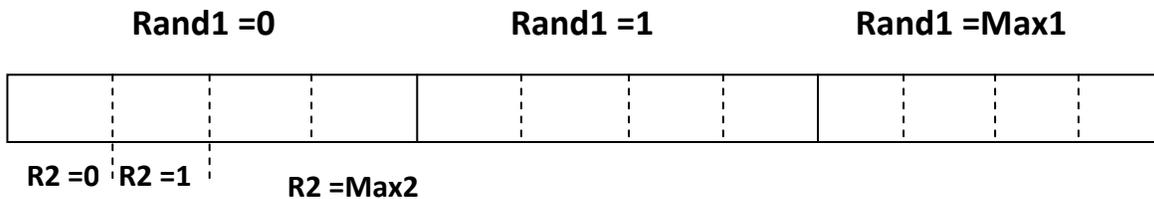


Figure 8.2 shows the Back-off time slots

8.5 Broadcasting

In the algorithm of DCA and Politburo, each nodes send packets to all neighbor nodes, and all they send at the same time. Since they all start to send at the same time, some of the packets are lost. There are two approaches are used to secure the correct arriving for these packets to their destination:

8.5.1 Broadcasting Approach

In this approach, the source node sends the packet into its neighbor and then each neighbor replies back by an acknowledgment. If the source does not get an acknowledgment from all neighbors, then it starts the packet again until all acknowledgment received.

The problem with this approach is that when the source get all the acknowledgment except for one or two, then the source will send the message again, and all neighbors will reply with Ack even the ones whose Ack already received by the source. This may increase the possibility of collision and in turns increase the time to convey all the packets.

8.5.2 Unicasting Approach

In this approach the source sends a message to one destination, and waits for Ack from the destination, if the Ack is not received after certain amount of time, the source sends it again, and so on. Once the Ack is received, the source node starts to send it to next neighbor node.

The problem with this approach is that sending the packets to large number of neighbors takes long time. Even though the message might be received by all nodes at every time sent to different individual node.

8.5.3 Customized Broadcast Approach

I implemented different approach to solve this problem; this new approach is done by sending the packets from the source to all neighbors and waiting for the Acks. The destinations then send the Acks to the source. If the source does not get the whole Acks, then resend the message again attached with the addresses of the neighbors whose Acks not received. this will prevent the all nodes from sending the Acks especially the ones whose Acks are received. In case of large number of Acks not received, the source can resend the message again without attaching any addresses; in this case the all nodes will resend Ack again. And this helps to reduce the size of resending message. E.g. suppose a node has 25 neighbor nodes, and it sends packets to all these nodes, and for some reason, the source obtained the Acks from only 23 nodes. Since 23 is large number compared to number of neighbors, then instead of resending the message with 23 address attached (size will be $23*6$ plus the size of the message), the source will resend only the message without any addresses.

8.6 Topology Generating

Generating the topology is the start point of the algorithm implementation, since the deploying of the nodes has to be in such a way that forms a connected graph.

Distributing the nodes was done over a square area $L \times L$, where L is the length of the square. And the position of the nodes was uniformly and randomly distributed. Since the nodes were randomly distributed, so there is possibility that these nodes are not connected. Therefore connectivity checking algorithm is needed. there are many ways to test whether the graph is

connected or not, and this can be either done using shortest path from one nodes to all other nodes, if there is path has cost of infinity, then the graph is not connected.

Another approach was used in this implementation which is **DFS**, and this algorithm search deeper in the graph from the source node , and start exploring the edges from the most recently discovered vertex, till all edges was searched and go back to the vertex that still does not have edges explored. And keep going till all vertexes are visited. If the number of visited nodes is less than the distributed nodes, then the graph is not connected. This algorithm has complexity of $O(|V| + |E|)$. [11]

Algorithm GenerateTopology

```
While (!IsConnected(G,N))
  Begin
    GenerateNewTopology()
  End
End
```

IsConnected(G,N)

```
Begin
  VisitedNodes = DFS(G)
  if (VisitedNodes==N) return true
  else return false
end
```

* Algorithm used was implemented by Prof. Basaign in C++ in 2008.

8.7 Average Route Length Computation

The average route length is average of the short distances between all different couples of nodes in the topology, and this also can be done using the shortest path algorithm from one source to all

nodes and repeating the same for all other nodes and then taking the average of all shortest distances between all possible combinations of different couples.

Another approach is implemented here which is BFS, Breadth first search. The algorithm explores all the edges to discover the vertices that are reachable from s and find the path that has minimum number of edges to each vertex. And this algorithm is good for undirected and un-weighted graph as in this implementation where the path between all the nodes is unidirectional and all the weights are the same [11].

And by repeating the BFS for all nodes, the minimum distance between any two nodes is obtained.

```
nBFS(G,n)
begin
  for I = 1 to N
    BFS(G,I,A)
    X=X + Average(A)
  end
return X/N
End
```

the variable I represents the source node, A represents the computed array for shortest distance between the Node 'I' and all other nodes. the return value of the algorithm is the average of all shortest distance between all possible combination of every different couples.

References

- [1] S. Basagni, “Distributed Clustering for Ad Hoc Networks”, Erik Jonsson School of Engineering and Computer Science The University of Texas at Dallas, 1999.
- [2] S. Basagni, M. Mastrogiovanni, A. Panconesi and C. Petrioli, “Localized Protocols for Ad Hoc Clustering and Backbone Formation: A Performance Comparison”, IEEE, April 2006.
- [3] I. Chlamtac and A. Farago , “A New Approach to the Design and Analysis of Peep-to-Peer Mobile Networks”, Wireless Network 5, vol.3, pp.149-156, May 1999.
- [4] W. Lou and J. Wu, “An Enhanced Message Exchange Mechanism in Cluster-Based Mobile Ad Hoc Networks”, December 2004.
- [5] J. Wu and W. Lou, “Forward-Node-Set-Based Broadcast in Clustered Mobile Ad Hoc Networks”, Department of Computer Science and Engineering, Florida Atlantic University, Mar 2003.
- [6] S. Rajagopalan and V. V. Vazirani, “Primal-dual RNC approximation algorithms for set cover and covering integer programs”, SIAM Journal on Computing, 28(2):525-540, 1998.
- [7] The Network Simulator - ns-3. Available online at <http://www.nsnam.org/>
- [8] P.J. Deitel and H. M. Deitel, “C++ How to Program, sixth edition”, Prentice Hall. , July 2007.
- [9] I. F. Akyildiz, W. Su, Y. S., and E. Caircy, “A Survey on Sensor Networks”, Georgia Institute of Technology , August 2002.
- [10] L. L. Peterson and B. S. Davie, “Computer Networks,” Fourth Edition, Morgan Kaufmann, 2007.
- [11] T. H. Cormen, C E. Leiserson, R L. Rivest and C. Stein, “Introduction to Algorithms”, 2001.