

**DESIGNING A SIMULATION APPLICATION FOR MOBILITY  
BASED CLUSTERING ALGORITHM FOR MANETS**

Enrollment Number: 101304

Name: Nishtha Sharma

Project Supervisor: Mr Amol Vasudeva



May 2014

Submitted in partial fulfillment of the Degree of  
Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
AND INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,  
WAKNAGHAT

## CERTIFICATE

This is to certify that the work titled **Designing a Simulation Application for Mobility Based Clustering Algorithm for MANETS** submitted by **Nishtha Sharma** in partial fulfillment for the award of degree of Bachelor of technology of Jaypee University of Information Technology, Wagnaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor:

Name of Supervisor : Mr. Amol Vasudeva

Designation : Assistant Professor

Date :

## ACKNOWLEDGEMENT

I would like to express my gratitude to all those who gave us the possibility to complete this project. I want to thank the Department of CSE & IT in JUIT for giving us the permission to commence this project in the first instance, to do the necessary research work.

I am deeply indebted to my project guide Mr Amol Vasudeva, whose help, stimulating suggestions and encouragement helped me in all the time of research on this project. I feel motivated and encouraged every time I get his encouragement. For his coherent guidance throughout the tenure of the project, I feel fortunate to be taught by him, who gave me his unwavering support.

We are also grateful to **Mr. Amit Singh (CSE Project lab)** for his practical help and guidance

Nishtha Sharma

## Abstract

A **wireless ad-hoc network** is a self-configuring network that does not depend on any infrastructure for communication. Every node is free to move anywhere in the network and data is exchanged independently across the network. Destruction of one node does not affect the communication of other nodes in the network. Every node in the network can act as both host as well as destination. A wireless ad-hoc network does not rely on fixed infrastructure or predetermined connectivity. It is a self organizing multi-hop wireless network in which all of the nodes can be mobile. Data is exchanged between nodes via wireless communication. Aside from the ability to be rapidly deployed, wireless ad-hoc networks have the ability to exist in highly volatile environments. Unlike traditional networks, if one node is destroyed it will not impact the data exchange between the remaining nodes within the network. The selection of the correct network topology given the network characteristics is extremely important to ensure reliable and efficient communication between nodes. The topology of a network can be either hierarchical or flat. In a hierarchical topology nodes are divided into clusters. Within each cluster, a cluster head is selected via a mathematical formulation or heuristic method. Cluster heads are responsible for keeping track of which nodes are maintained in their respective cluster. Furthermore, they are responsible for transmitting data between clusters. Each of the cluster heads maintains a continuously updated routing table. This table contains specific information detailing which cluster each node belongs to. If a node desires to transfer information to another node, the information is sent to the sending node's cluster head. This cluster head scans its routing table to determine which cluster the recipient is in. If the recipient is in the same cluster, the data is immediately forwarded to the receiving node. If not, the cluster head scans its routing table to determine which cluster the recipient is in and forwards the data to the appropriate cluster head where it is again forwarded to the recipient. **Mobility based Clustering algorithm** uses mobility of the nodes as a feature to form the clusters. Each node in the Mobile Ad hoc Network computes the ratio of two successive "Hello" messages from all its neighbors. This gives the relative mobility metric of the nodes with respect to their respective neighbors. Then by calculating the variance of relative mobility values of all the nodes with respect to their neighbors in a distributed manner, the aggregate speed of all the mobile nodes can be estimated.

Finally, the mobile node with lowest variance value in its neighborhood is elected as the cluster head. MOBIC is suitable for MANETs in which a group of mobile nodes moves with almost similar speed and the direction. However, if mobile nodes move randomly and change their speeds from time to time, the performance of this algorithm is degraded.

## **Problem Statement**

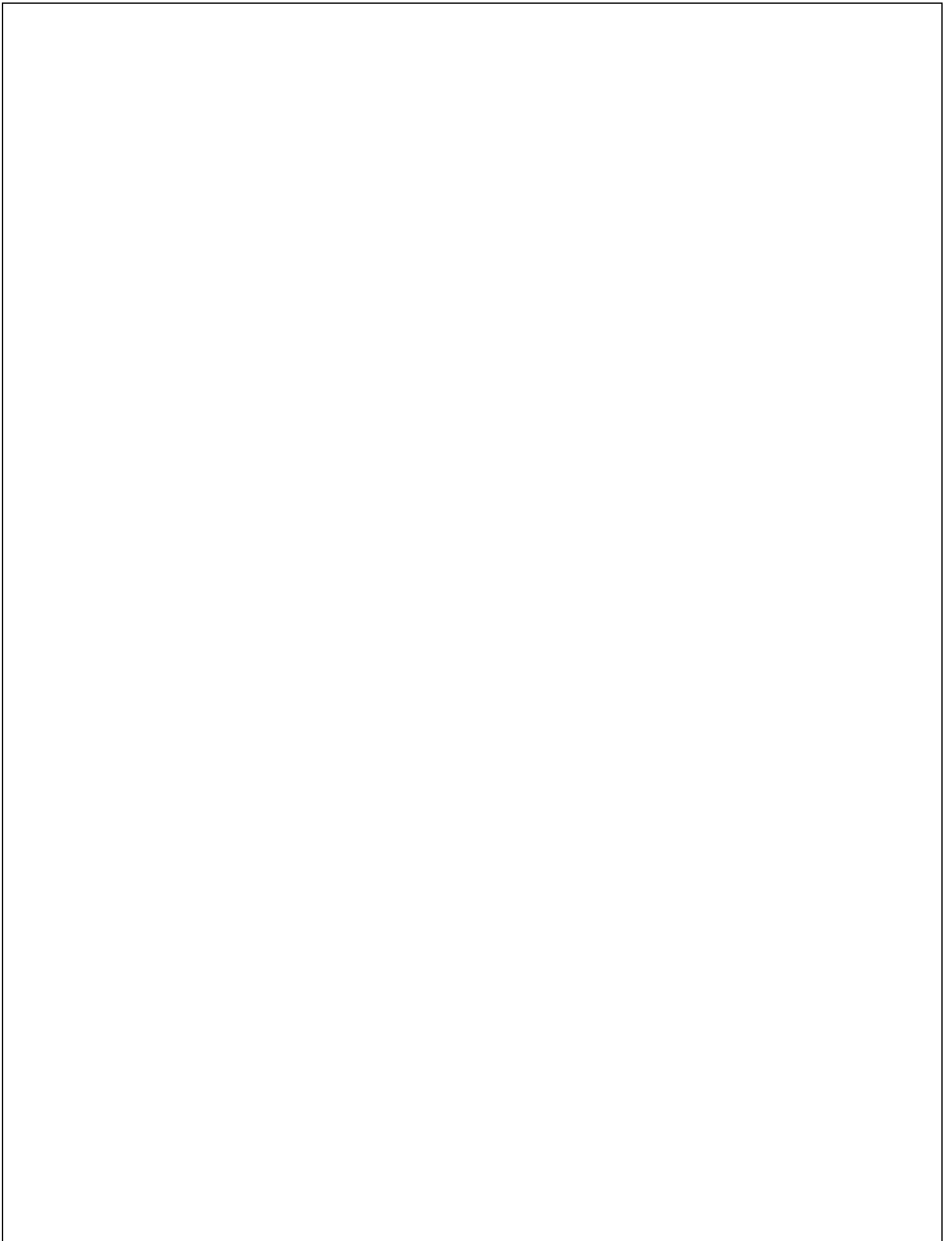
The proposal put forward in this project is to develop efficient Mobility based clustering algorithm (MOBIC) for cluster formation, which can be used by a set of asynchronous, self-organizing nodes to form an ad hoc network, under the above model. When a set of nodes are powered on at the same time, they start executing these algorithms immediately in order to form a connected set of clusters. These topology construction algorithms should ensure the following.

- Minimum Number of Clusters
- Stable environment

## **Motivation**

Mobile Ad-Hoc Network (MANET) has become an increasingly active research area work in ad-hoc routing, media access, and protocols, etc. However, much of the effort so far has been in simulation with only a few systems that have ever been implemented and none that we know have gone beyond field trial to regular use. One of the reason is the high complexity involved in implementing and testing actual ad-hoc networks, and the lack of software tools for doing so.

Our vision is to implement Mobility based clustering algorithm to make MANET easy to develop, easy to deploy, and easy to use. The project includes: reasons for choosing this particular algorithm, implementing the algorithm, and model using which we will implement the algorithm. An application that will provide the means for user input and simulation status while results are exported for analysis. This method works well as long as the user is relatively computer literate.

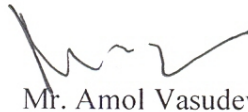




## CERTIFICATE

This is to certify that the work titled **Designing a Simulation Application for Mobility Based Clustering Algorithm for MANETS** submitted by **Nishtha Sharma** in partial fulfillment for the award of degree of Bachelor of technology of Jaypee University of Information Technology, Waznaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor:



Name of Supervisor : Mr. Amol Vasudeva

Designation : Assistant Professor

Date : 15<sup>th</sup> May 2014

## ACKNOWLEDGEMENT

I would like to express my gratitude to all those who gave us the possibility to complete this project. I want to thank the Department of CSE & IT in JUIT for giving us the permission to commence this project in the first instance, to do the necessary research work.

I am deeply indebted to my project guide Mr Amol Vasudeva, whose help, stimulating suggestions and encouragement helped me in all the time of research on this project. I feel motivated and encouraged every time I get his encouragement. For his coherent guidance throughout the tenure of the project, I feel fortunate to be taught by him, who gave me his unwavering support.

We are also grateful to **Mr. Amit Singh (CSE Project lab)** for his practical help and guidance

*Nishtha Sharma*

Nishtha Sharma

## Contents

<b>Chapter 1: Mobile Adhoc Networks (MANETS)</b> .....	<b>1</b>
<b>1.1 Introduction</b> .....	<b>1</b>
<b>1.2 Characteristics</b> .....	<b>1</b>
<b>1.3 Applications</b> .....	<b>2</b>
<b>1.4 Limitations</b> .....	<b>2</b>
<b>1.5 Challenges</b> .....	<b>3</b>
<b>Chapter 2 : Mobility Models</b> .....	<b>5</b>
<b>2.1 Definition</b> .....	<b>5</b>
<b>2.2 Types</b> .....	<b>5</b>
<b>2.2.1 Random Walk Mobility Model</b> .....	<b>5</b>
<b>2.2.2 Random Waypoint Mobility Model</b> .....	<b>6</b>
<b>2.2.3 Random Direction Mobility Model</b> .....	<b>6</b>
<b>Chapter 3: Routing Algorithms</b> .....	<b>7</b>
<b>3.1 Definition</b> .....	<b>7</b>
<b>3.2 Types</b> .....	<b>7</b>
<b>3.2.1 Flat Routing Algorithm</b> .....	<b>7</b>
<b>3.2.2 Destination-Sequenced Distance-Vector Routing (DSDV)</b> .....	<b>8</b>
<b>3.2.4 Hierarchical Routing</b> .....	<b>9</b>
<b>3.2.5 Lowest ID</b> .....	<b>10</b>
<b>3.2.6 Highest Degree</b> .....	<b>11</b>
<b>3.2.7 Mobility Based Clustering (MOBIC):</b> .....	<b>11</b>
<b>Chapter 4: Implementation</b> .....	<b>12</b>
<b>4.1 Simulation Environment</b> .....	<b>12</b>
<b>4.1.1 Comparison Parameters</b> .....	<b>12</b>
<b>4.2 Tools and Technologies</b> .....	<b>13</b>
<b>4.2.1 Java 1.6 Version</b> .....	<b>13</b>
<b>4.2.1.1 Characteristics</b> .....	<b>13</b>
<b>4.3.3.2 JAVA Virtual Machine (JVM)</b> .....	<b>13</b>
<b>4.3.3.3 Applet and Standalone Application</b> .....	<b>13</b>
<b>4.2.2 Eclipse Galileo Version 3.5.1</b> .....	<b>14</b>
<b>4.3 Diagrams</b> .....	<b>14</b>

<b>4.3.1 Data Flow Diagram.....</b>	<b>14</b>
<b>4.3.2 Sequence Diagram .....</b>	<b>15</b>
<b>4.3.3 Class Diagram .....</b>	<b>16</b>
<b>4.4 Algorithm .....</b>	<b>20</b>
<b>4.5 Code .....</b>	<b>25</b>
<b>Chapter 6 Average Cluster Calculation and Stability.....</b>	<b>51</b>
<b>6.1 Definition .....</b>	<b>51</b>
<b>6.2 Code .....</b>	<b>51</b>
<b>6.3 Need For Stability .....</b>	<b>51</b>
<b>6.4Stability.....</b>	<b>52</b>
<b>Chapter 5 Conclusion.....</b>	<b>53</b>
<b>5.1 Conclusion .....</b>	<b>53</b>
<b>Chapter 6 Glossary:.....</b>	<b>54</b>
<b>Chapter 7 References .....</b>	<b>55</b>

# Chapter 1: Mobile Adhoc Networks (MANETS)

## 1.1 Introduction

A mobile ad hoc network (MANET) is a self-configuring infrastructure less network of mobile devices which are connected by wireless. Each device in a MANET is free to move independently in any direction, and will therefore change its links to other devices frequently.

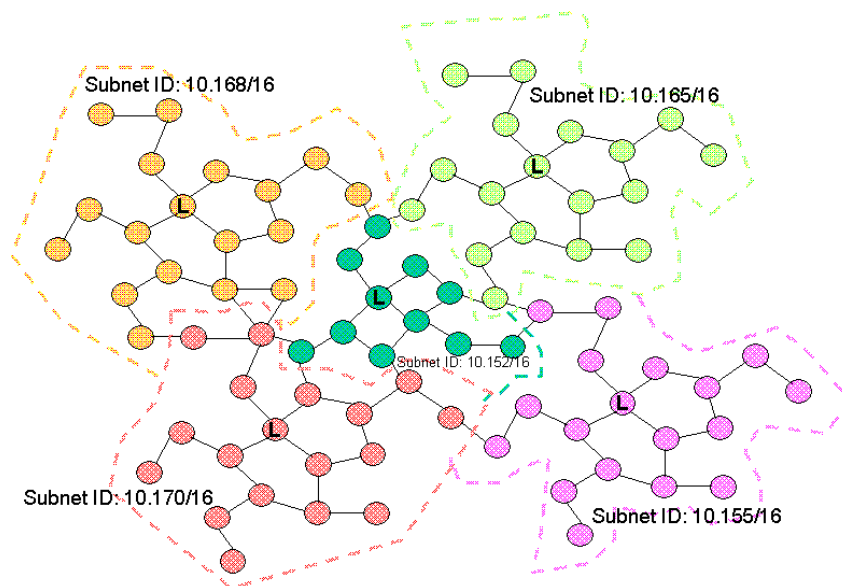


Figure 1.1: Example of MANET [8]

## 1.2 Characteristics

- In MANET, each node acts as both host and router. That is it is autonomous in behavior.
- Multi-hop radio relaying-When a source node and destination node for a message is out of the radio range, the MANETs are capable of multi-hop routing. The nodes can join or leave the network anytime, making the network topology dynamic in nature.

- Network topology dynamic in nature.
- Distributed nature of operation for security, routing and host configuration. A centralized firewall is absent here.
- The nodes can join or leave the network anytime, making the network topology dynamic in nature.
- Nodal connectivity is intermittent.
- Mobile nodes are characterized with less memory, power and light weight features.
- Completely symmetric environment.
- The reliability, efficiency, stability and capacity of wireless links are often inferior when compared with wired links. This shows the fluctuating link bandwidth of wireless links.
- Mobile and spontaneous behavior which demands minimum human intervention to configure the network.
- All nodes have identical features with similar responsibilities and capabilities and hence it forms a completely symmetric environment.
- High user density and large level of user mobility.

### **1.3 Applications**

- Personal area networking (cell phone, laptop).
- Military environments (battle grounds).
- Civilian environments (taxi cab network, meeting rooms, boats, aircraft).  
Emergency operations (search-and-rescue, policing and fire fighting).

### **1.4 Limitations**

- Limitations of the Wireless Network
  - packet loss due to transmission errors

- variable capacity links
- frequent disconnections/partitions
- limited communication bandwidth
- Broadcast nature of the communications
- Limitations Imposed by Mobility
  - dynamically changing topologies/routes
  - lack of mobility awareness by system/applications
- Limitations of the Mobile Computer
  - short battery lifetime
  - limited memory
- Routing efficiency
  - Discovery, maintenance
- Network services
  - Authentication, service discovery, addresses binding, address assignment.

## **1.5 Challenges**

- The reliability of wireless transmission is resisted by different factors-The wireless link characteristics are time-varying in nature. There are transmission impediments like fading, path loss, blockage and interference that adds to the susceptible behavior of wireless channels.

- Limited range of wireless transmission-The limited radio band results in reduced data rates compared to the wireless networks. Hence optimal usage of bandwidth is necessary by keeping low overhead as possible
- Packet losses due to errors in transmission – MANETs experience higher packet loss due to factors such as hidden terminals that results in collisions, wireless channel issues (high bit error rate (BER)), interference, and frequent breakage in paths caused by mobility of nodes, increased collisions due to the presence of hidden terminals and uni-directional links.
- Route changes due to mobility- The dynamic nature of network topology results in frequent path breaks.
- Frequent network partitions- The random movement of nodes often leads to partition of the network. This mostly affects the intermediate nodes.
- The application of this wireless network is limited due to the mobile and ad hoc nature. Similarly, the lack of a centralized operation prevents the use of firewall in MANETs. It also faces a multitude of security threats just like wired networks. It includes spoofing, passive eavesdropping, denial of service and many others. The attacks are usually classified on the basis of employed techniques and the consequences.



## **Chapter 2 : Mobility Models**

### **2.1 Definition**

**Mobility models** represent the movement of mobile users, and how their location, velocity and acceleration change over time. Such models are frequently used for simulation purposes when new communication or navigation techniques are investigated. For mobility modeling, the behavior or activity of a user's movement can be described using both analytical and simulation models. The input to analytical mobility models are simplifying assumptions regarding the movement behaviors of users. Such models can provide performance parameters for simple cases through mathematical calculations. In contrast, simulation models consider more detailed and realistic mobility scenarios. Such models can derive valuable solutions for more complex cases.

### **2.2 Types**

Typical mobility models include

- Random Walk Mobility Model
- Random Waypoint Mobility Model
- Random Direction Mobility Model

#### **2.2.1 Random Walk Mobility Model**

In this mobility model, a node moves from its current location to a new location by randomly choosing a direction and speed in which to travel. The new speed and direction are both chosen from pre defined ranges, [minspeed; maxspeed] and  $[0; 2\pi]$  respectively. Each movement in the Random Walk Mobility Model occurs in either at constant time interval ' $t$ ' or a constant distance traveled ' $d$ ', at the end of which a new direction and speed are calculated. If any node reaches to the simulation boundary, it bounces off the simulation border with an angle determined by the incoming direction. The node then continues along this new path.

### **2.2.2 Random Waypoint Mobility Model**

The random waypoint mobility model contains pause time between changes in direction and/or speed. Once a node begins to move, it stays in one location for a specified pause time. After the specified pause time is elapsed, the randomly selects the next destination in the simulation area and chooses a speed uniformly distributed between the minimum speed and maximum speed and travels with a speed  $v$  whose value is uniformly chosen in the interval  $(0, v_{max})$ .  $v_{max}$  is some parameter that can be set to reflect the degree of mobility. Thereafter, it continues its journey toward the newly selected destination at the chosen speed. As soon as it arrives at the destination, it stays again for the indicated pause time before repeating the process.

### **2.2.3 Random Direction Mobility Model**

In random direction mobility model each node alternates periods of movement (move phase) to periods during which it pauses (pause phase). During the beginning of each move phase, a node independently selects its new direction and speed of movement. Speed and direction are kept constant for the whole duration of the node movement phase.

## Chapter 3: Routing Algorithms

### 3.1 Definition

Routing is the process of selecting best paths in a network. To find and maintain routes between nodes in a dynamic topology with possibly unidirectional /by directional links, using minimum resources.

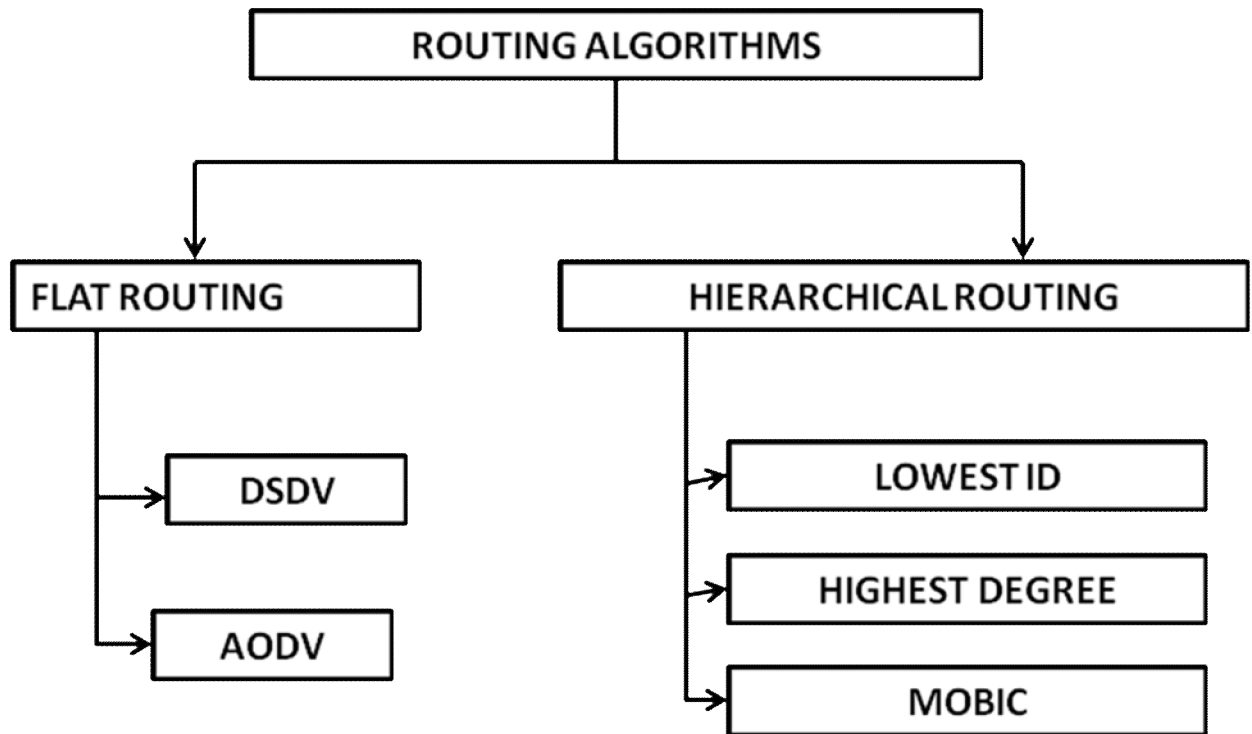


Figure 3.1: Routing Algorithms

### 3.2 Types

#### 3.2.1 Flat Routing Algorithm:

In a Flat Routing technique, the message to be delivered is directly passed from source node to the destination node without passing the message to any cluster head node. The information directly hops from one node to another and finds the shortest path to reach the

destination node. Best example of flat routing is Routing Information Protocol which is a simple hop count protocol where maximum 15 number hops can be taken to reach the destination node.

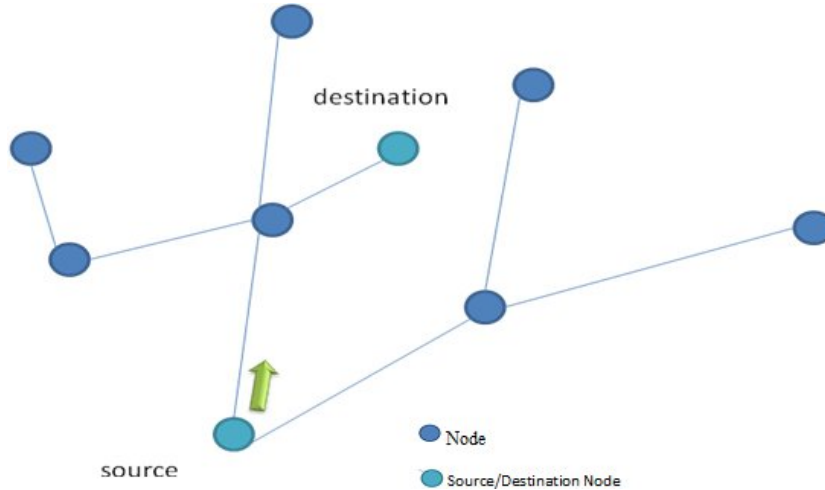


Figure 3.1: Flat Routing

**3.2.2 Destination-Sequenced Distance-Vector Routing (DSDV):**

DSDV is a table-driven routing scheme for ad hoc mobile networks based on the Bellman–Ford algorithm. The main contribution of the algorithm was to solve the routing loop problem. Each entry in the routing table contains a sequence number, the sequence numbers are generally even if a link is present; else, an odd number is used. The number is generated by the destination, and the emitter needs to send out the next update with this number

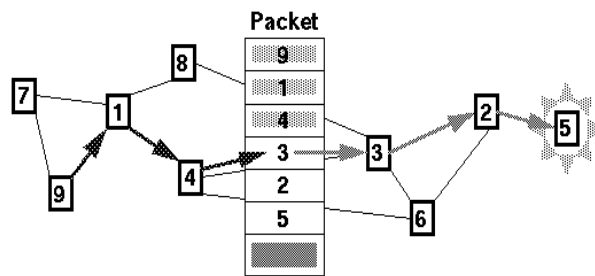


Figure 3.2 Destination-Sequenced Distance-Vector Routing

**3.2.3 Ad hoc On-Demand Distance Vector Routing (AODV):** In this routing algorithm, the path is drawn on demand by the source node when it wants to send the message that is when the source node desires to send a message and does not have any path so its broadcast the route request message across the network. All the nodes in the network after receiving this message will look for the shortest path through which the message can be sent .The source node will receive the message along with the IP address of the destination node and then can send the message through that path/route.

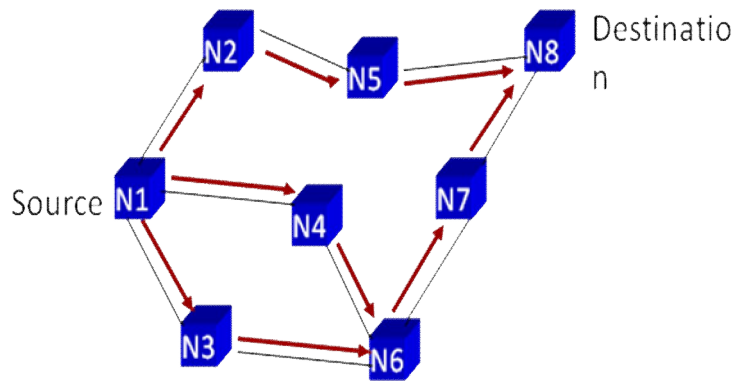


Figure 3.3 (a) Ad hoc On-Demand Distance Vector Routing- source

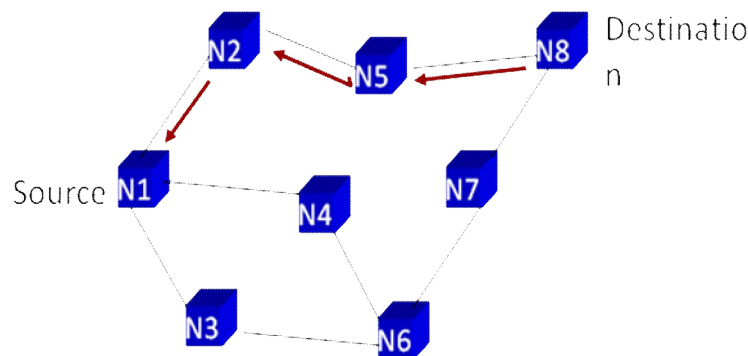


Figure 3.3 (b) Ad hoc On-Demand Distance Vector Routing- Destination

### 3.2.4 Hierarchical Routing:

In Hierarchical Routing nodes are divided into groups on the basis of the categories of nodes. These groups are known as clusters. A cluster can have three categories of nodes: cluster head, gateway node and ordinary or member nodes. In order to send the information from one node to another node, the message is first passed on to the cluster head and then the information is passed to the destination node after looking at the routing table for the

desired cluster. If the node is within the same cluster then the information is directly passed else is first passed to the respective cluster head and then the destination cluster head followed by the destination node. There are numerous methods to form clusters in a simulation environment.

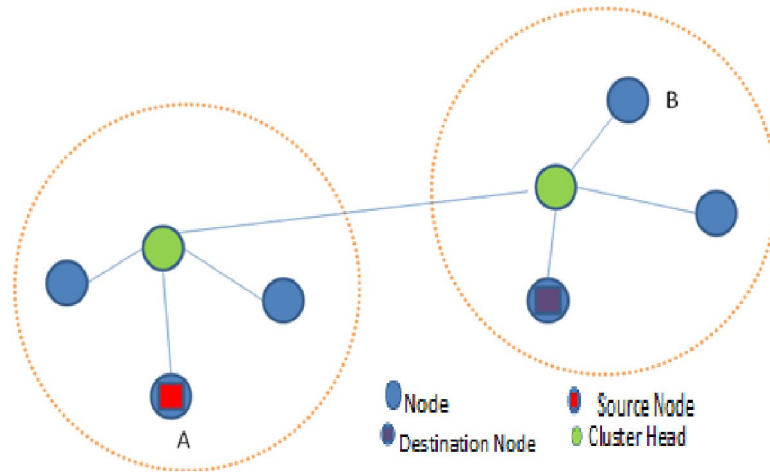


Figure 3.4 Hierarchical Routing

### 3.2.5 Lowest ID:

The Lowest-ID is considered as a simplest clustering scheme algorithm. In this scheme unique identifier (ID) is assigned to each node. All nodes recognize its neighbors ID and CH is chosen according to minimum ID. Thus, the nodes IDs of the neighbors of the CH will be higher than that CH. The main drawback with this scheme is there is no limitation to the number of nodes attached to the same CH. Also, CHs are prone to power drainage due to serving as cluster heads.

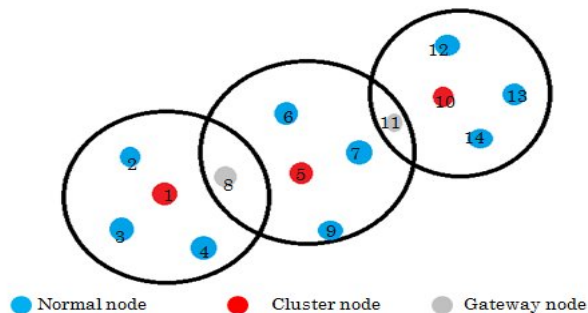


Figure 3.5 Lowest ID

### 3.2.6 Highest Degree:

In comparison with Lowest-ID scheme, the degree of nodes is computed based on its distance from each others. All nodes flood its connectivity value within their transmission range. Thus, a node decides to become a CH or remain as CN by comparing the connectivity value of its neighbors with its own value. Node with highest connectivity value in its vicinity will become CH. Connectivity-based clustering follows the same circumstances of ID-based regarding to cluster size and performance degradation.

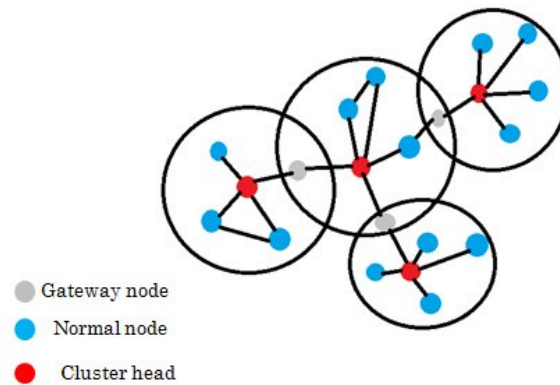


Figure 3.6: Highest Degree

### 3.2.7 Mobility Based Clustering (MOBIC):

MOBIC uses mobility of the nodes as a feature to form the clusters. Each node in the Mobile Ad hoc Network computes the ratio of two successive “Hello” messages from all its neighbors. This gives the relative mobility metric of the nodes with respect to their respective neighbors. Then by calculating the variance of relative mobility values of all the nodes with respect to their neighbors in a distributed manner, the aggregate speed of all the mobile nodes can be estimated. Finally, the mobile node with lowest variance value in its neighborhood is elected as the cluster head.

## **Chapter 4: Implementation**

### **4.1 Simulation Environment**

We will simulate an ad hoc network with  $n$  nodes randomly distributed in a  $100 \times 100$  pixel area. The simulator was implemented in Java due to its multithreading feature and collection of numerous container classes. The network simulator has the ability to generate network with any number of nodes. The mobility is based on the Random Way Point model (RWP) in which a mobile node moves from its current location to a new location by randomly choosing a direction and speed in which to travel. The new speed and direction are both chosen randomly from pre-defined ranges,  $[0, 5 \text{ unit/sec}]$  and  $[0, 2\pi]$  respectively. We have set the election process time to be 2 seconds i.e. after each 2 seconds the cluster head selection mechanism will be initiated. Once the election process is over, new directions and speeds are computed for all the nodes in the same manner, as mentioned above. This process was repeated throughout the simulation causing continuous changes in the topology of the underlying network. Once a node reaches the boundary of edge, it returns back with the same direction and speed. The transmission range of all the nodes has been taken to be 40 units, i.e. the two nodes can communicate with each other if the distance between them is shorter than 40 units. In order to complete these objectives, a network simulator was developed using Java, compute the five metrics as previously discussed, apply each of the clustering techniques, and evaluate congestion.

#### **4.1.1 Comparison Parameters:**

- Number of nodes (scalability)
- Velocity of nodes (uniform or non uniform)
- Transmission power
- Density of nodes



## **4.2 Tools and Technologies:**

### **4.2.1 Java 1.6 Version:**

#### **4.2.1.1 Characteristics:**

JAVA is a programming language, developed by Sun Microsystems and first released in 1995 (release 1.0). Since that time, it gained a large popularity mainly due to two characteristics:

- A JAVA program is hardware and operating system independent. If well written, the same JAVA program, compiled once, will run identically on a SUN/Solaris workstation, a PC/windows computer or a Macintosh computer. Not mentioning other Unix flavors, including Linux, and every Web browser, with some restrictions described below. This universal excitability is made possible because a JAVA program is run through a JAVA Virtual Machine.
- It is an object oriented language. This feature is mainly of interest for software developers.

#### **4.3.3.2 JAVA Virtual Machine (JVM):**

A JAVA program is build by a JAVA compiler which generates its own binary code. This binary code is independent from any hardware and operating system. To be executed, it needs a *virtual machine*, which is a program analyzing this binary code and executing the instructions it contains. Of course, this Java Virtual Machine (JVM) is hardware and operating system dependant. Two types of Virtual Machines exist: those included in every Web Browser, and those running as an independent program, like the Java Run Time Environment (JRE) from Sun Microsystems. These programs need to be downloaded for your particular platform.

#### **4.3.3.3 Applet and Standalone Application:**

A JVM in a web browser runs a JAVA program as an Applet. The applet is embedded in a web page and downloaded from a web server like any other HTML page or image when requested. An independent JVM runs a JAVA program as a Standalone Application.

## 4.2.2 Eclipse Galileo Version 3.5.1

**Eclipse** is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications in Java.

## 4.3 Diagrams

### 4.3.1 Data Flow Diagram

#### Zero Level DFD

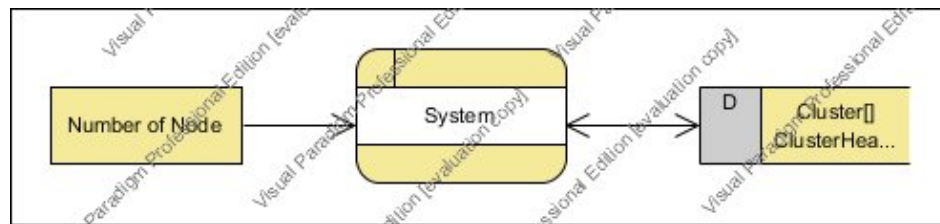


Figure 4.1 (a) Zero Level DFD

#### First Level DFD

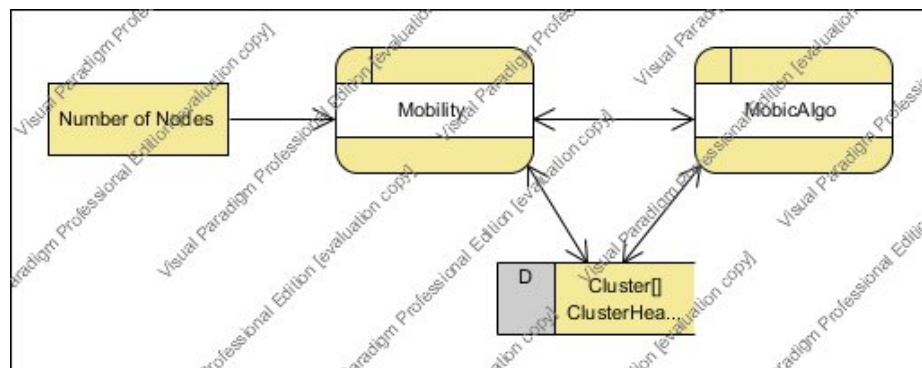


Figure 4.1 (b) First Level DFD

#### Second Level DFD

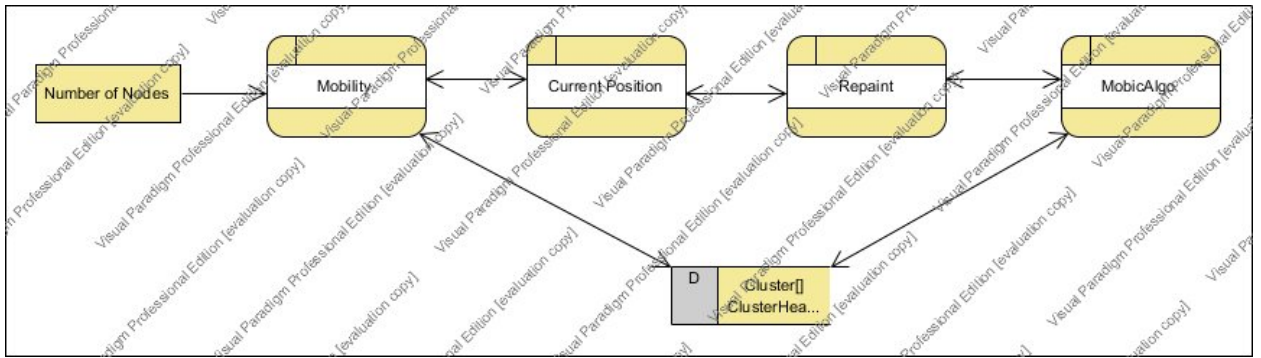


Figure 4.1(c) Second Level DFD

### 4.3.2 Sequence Diagram

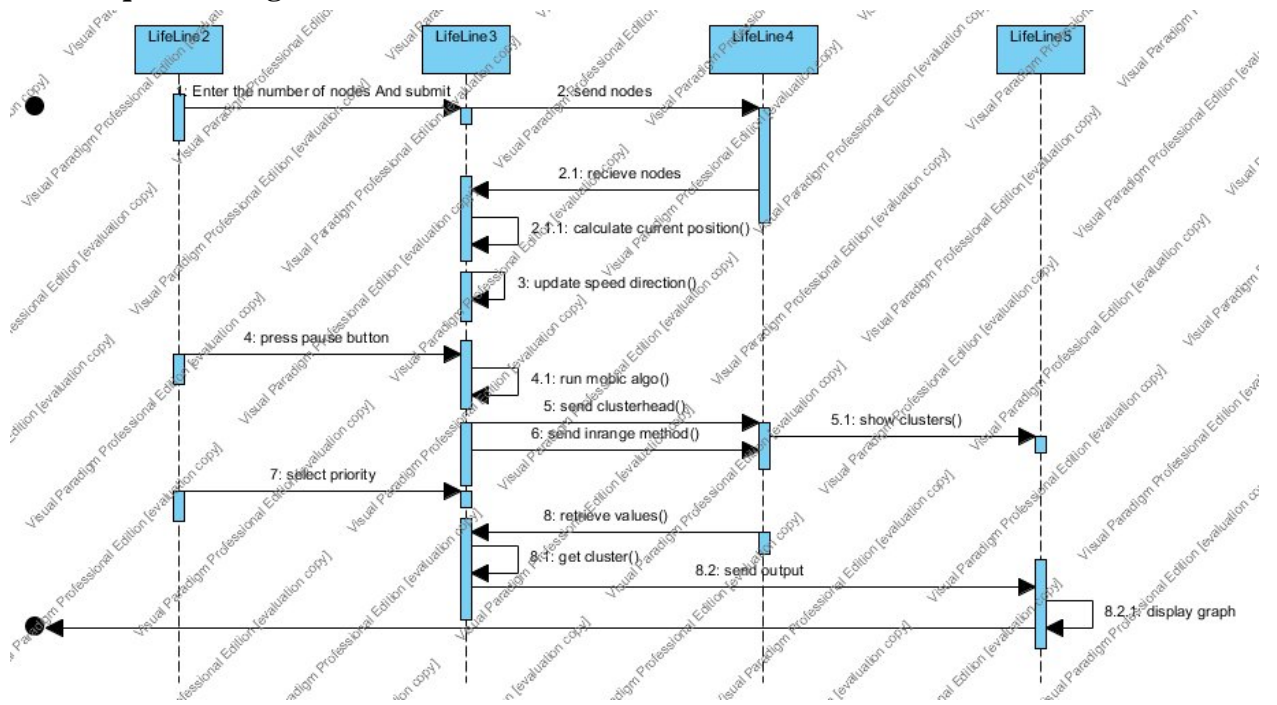


Figure 4.2 Sequence Diagram

### 4.3.3 Class Diagram

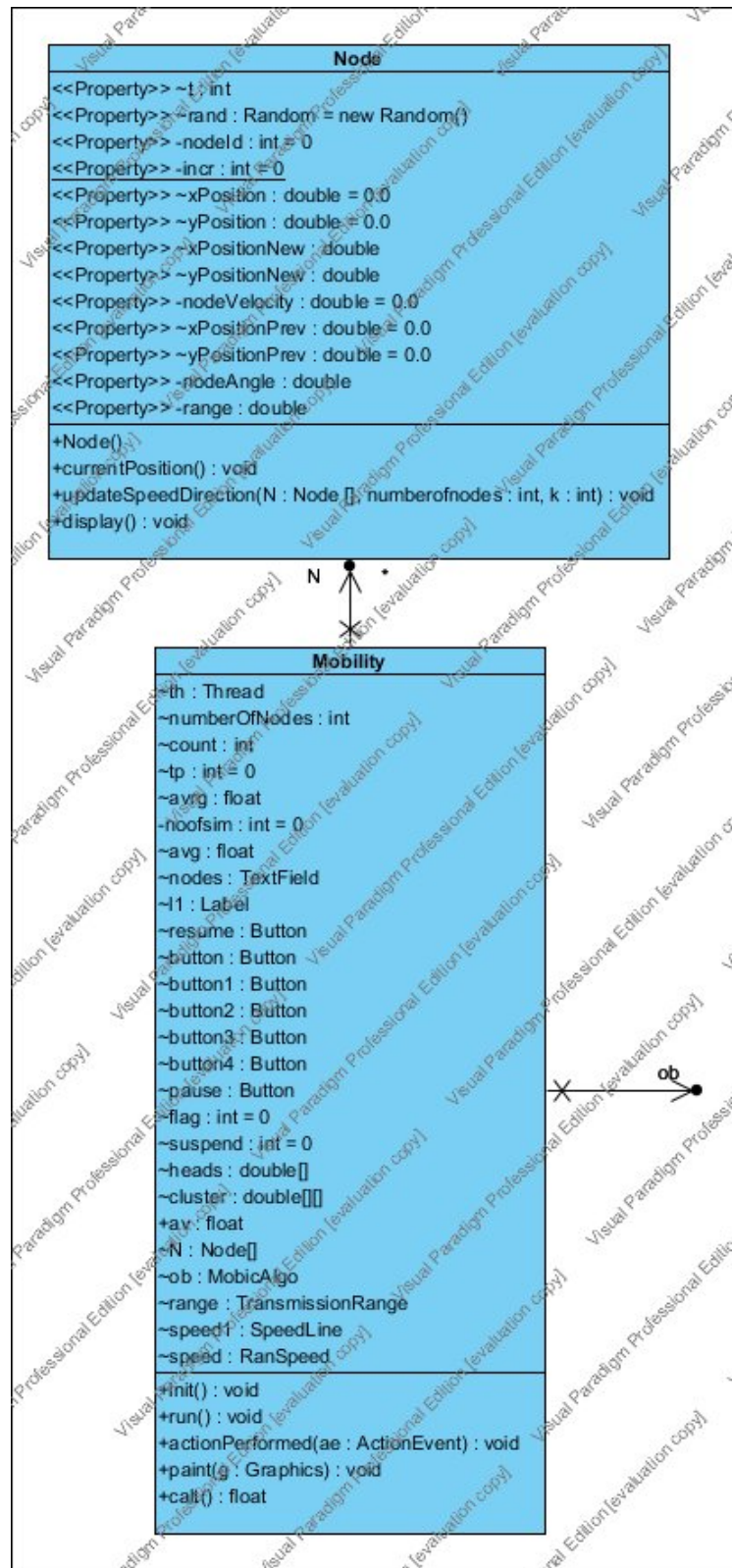


Figure 4.3(a) Class Diagram (Part1)

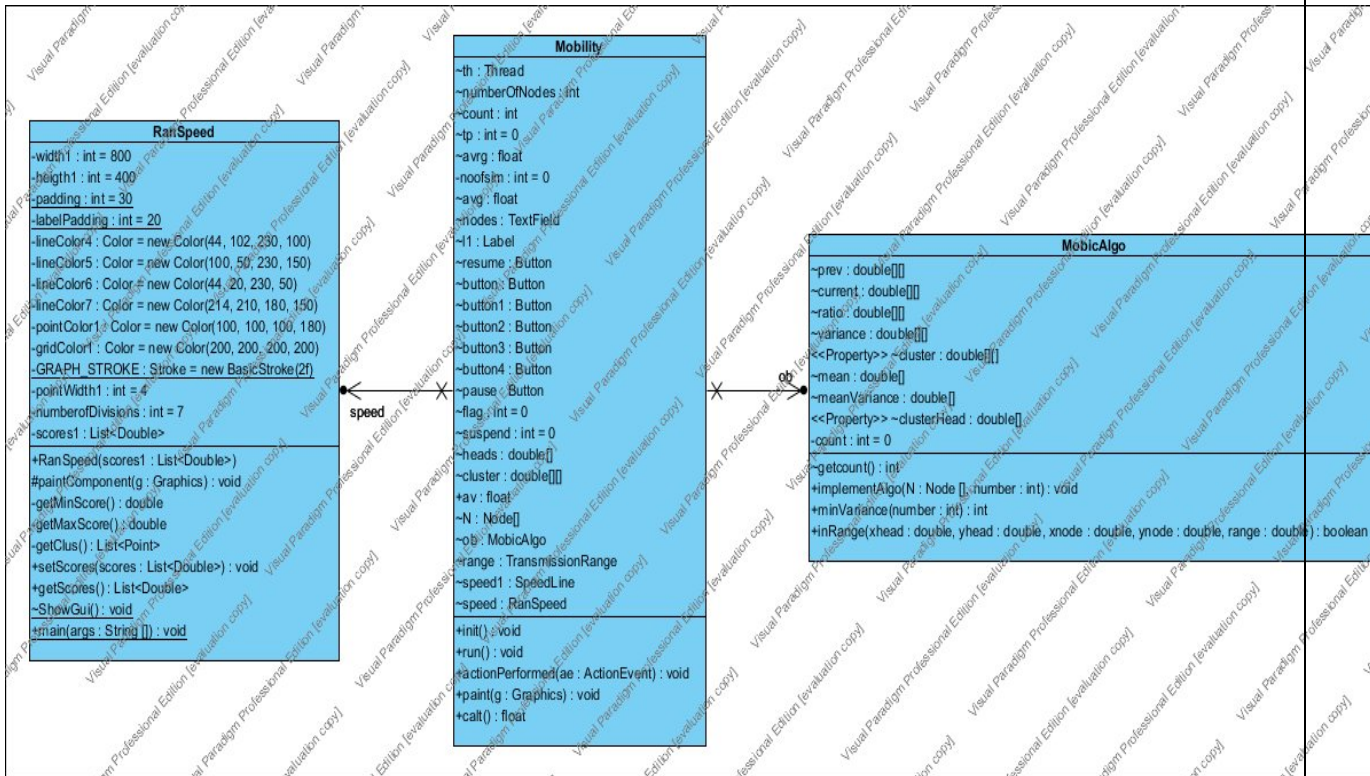


Figure 4.3 (b) Class Diagram (Part2)



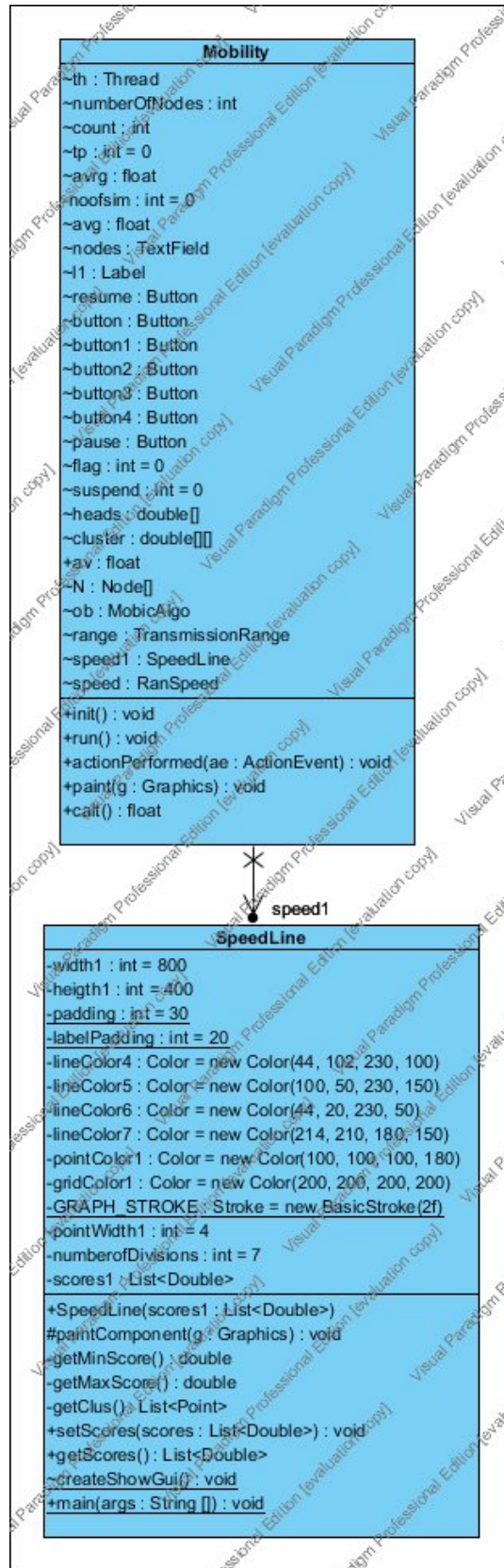


Figure 4.3 (c) Class Diagram (Part3)

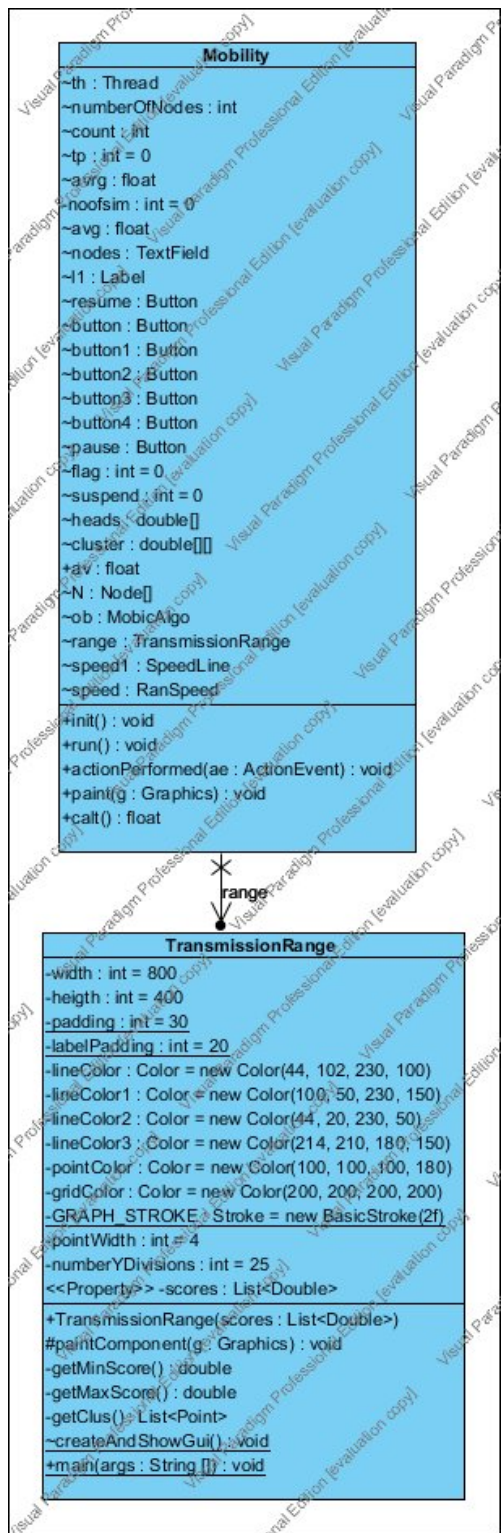


Figure 4.3 (d) Class Diagram (Part4)

#### 4.4 Algorithm :

Find the current distance matrix, `current[][]` .

Find the previous distance matrix, `previous[][]`.

Calculate the ratio: current matrix/ previous matrix and store in `ratio[][]`.

Calculate the `mean[]` of `ratio[][]`.

Calculate `variance=(ratio-mean)^2`, `variance[][]`.

Do

`n=minVariance()`

`head[n]=1`

`meanVariance[n]=1000`

    for every node i

        do

            If `inRange (n,i)`

            Then

`cluster[n][i]=1`

`meanVariance[i]=1000`

            end if

    end for

while `meanVariance` of any node `<1000`



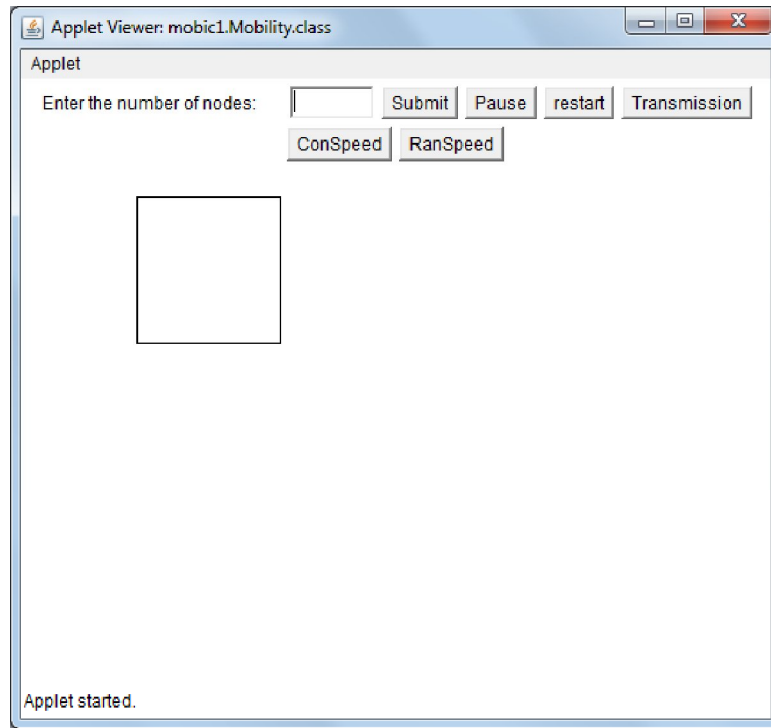


Figure 4.4 Initial Applet

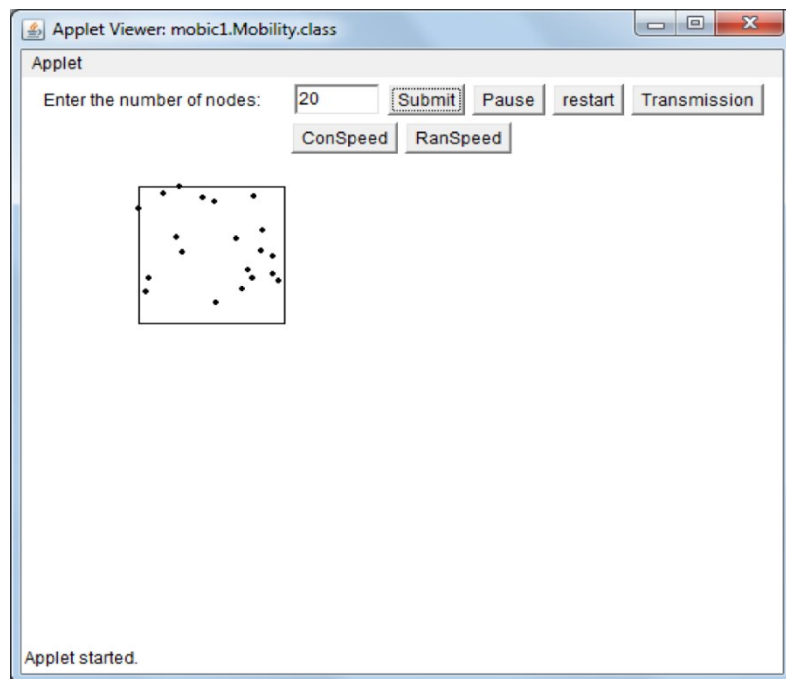


Figure 4.5 Nodes entered

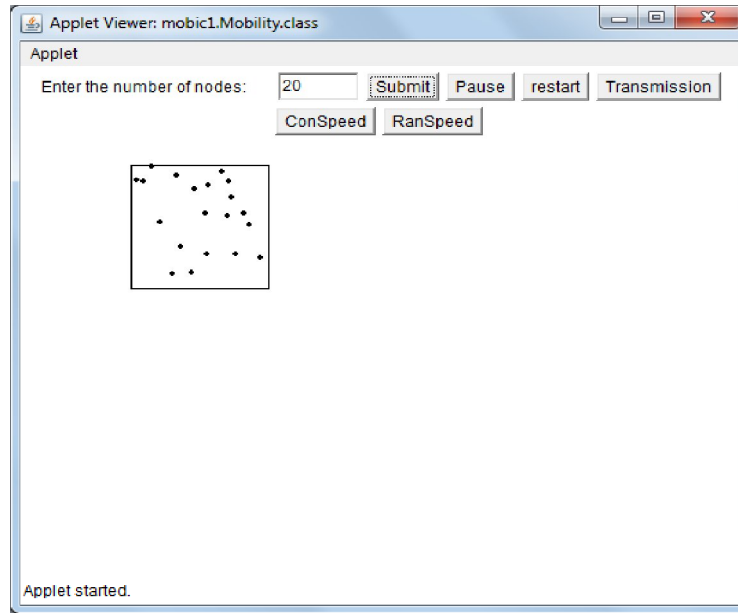


Figure 4.6 Press Pause

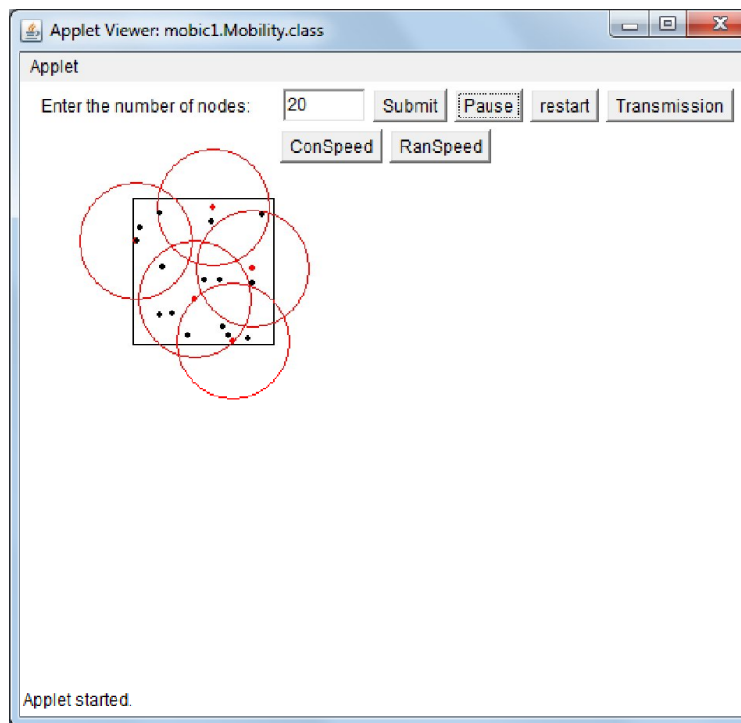


Figure 4.7 Implementation of Clustering

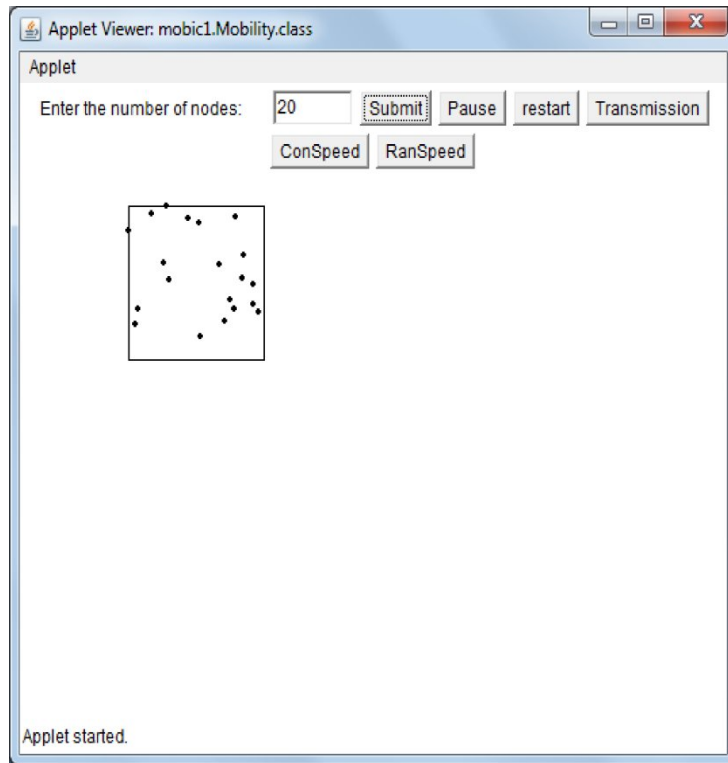


Figure 4.8 Restart

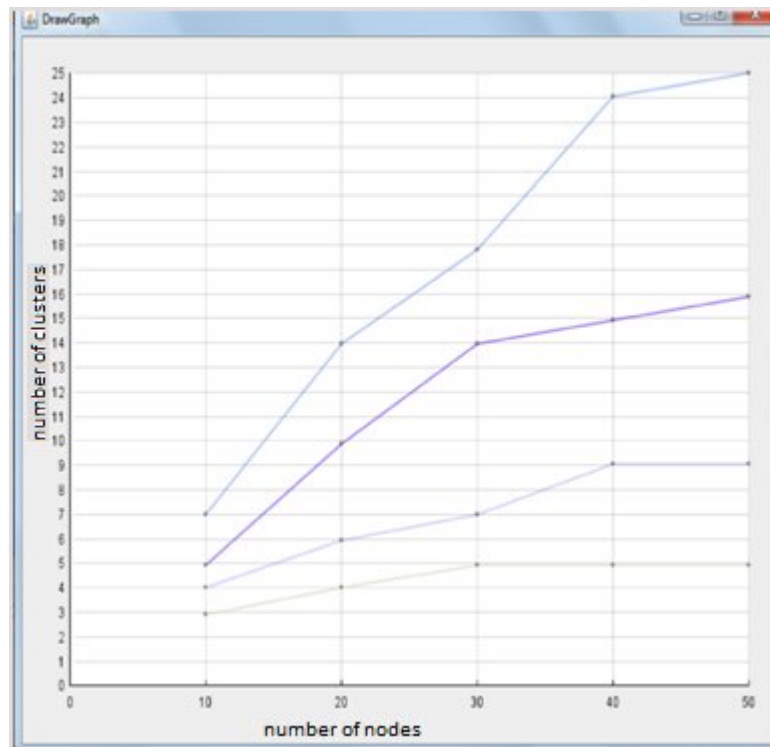


Figure 4.9 Transmission Range Graph

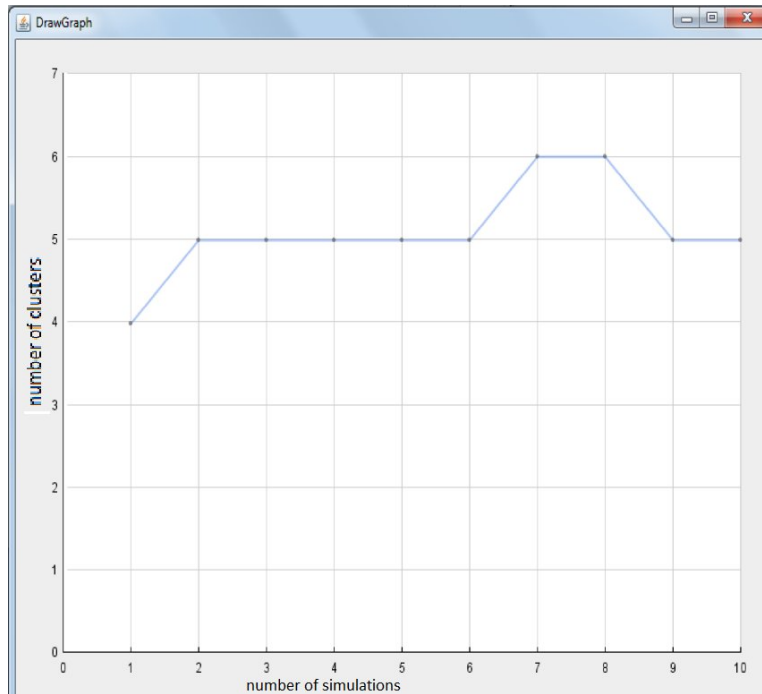


Figure 4.10 Constant Range Graph

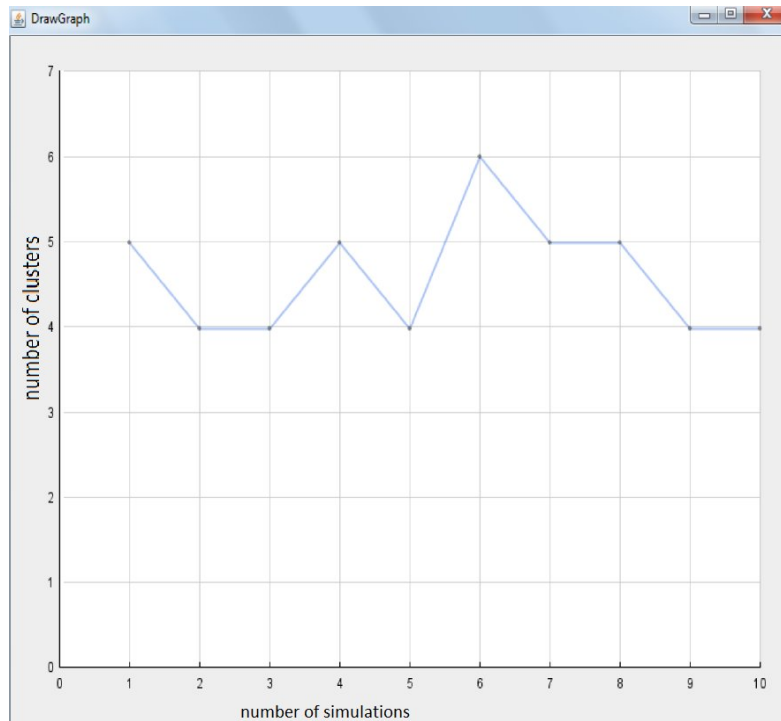


Figure 4.11 Random Range Graph

## 4.5 Code

➤ Node.java

```
package mobic1;

import java.util.Random;

class Node {

    int t;// time interval
    Random rand = new Random();
    private int nodeId = 0;
    private static int incr = 0;
    double xPosition = 0.0; // x coordinate
    double yPosition = 0.0; // y coordinate
    double xPositionNew, yPositionNew;
    private double nodeVelocity = 0.0;
    double xPositionPrev = 0.0, yPositionPrev = 0.0;
    private double nodeAngle; // movement direction
    private double range;

    //setters and getters
    public int getT() {
        return t;
    }

    public Random getRand() {
        return rand;
    }

    public int getNodeId() {
        return nodeId;
    }
}
```

```
    }

    public static int getIncr() {
        return incr;
    }

    public double getXPosition() {
        return xPosition;
    }

    public double getYPosition() {
        return yPosition;
    }

    public double getXPositionNew() {
        return xPositionNew;
    }

    public double getYPositionNew() {
        return yPositionNew;
    }

    public double getNodeVelocity() {
        return nodeVelocity;
    }

    public double getNodeAngle() {
        return nodeAngle;
    }

    public double getRange() {
```

```

        return range;
    }

    public double getXPositionPrev() {
        return xPositionPrev;
    }

    public void setXPositionPrev(double xPositionPrev) {
        this.xPositionPrev = xPositionPrev;
    }

    public double getYPositionPrev() {
        return yPositionPrev;
    }

    public void setYPositionPrev(double yPositionPrev) {
        this.yPositionPrev = yPositionPrev;
    }

//constructor
    public Node() {

        nodeId = incr;
        xPosition = rand.nextInt(100);
        yPosition = rand.nextInt(100);
        t = 2;
        nodeVelocity = rand.nextInt(6);
        nodeAngle = rand.nextInt(360);
        range = 40;
        incr++;
    }

```

```

//calculating position of node as it moves with a predefined speed
public void currentPosition() {

    double rad = 3.14159 / 180.0;

    if (nodeAngle >= 0 && nodeAngle < 90) {
        xPositionNew = xPosition + (nodeVelocity * t)
            * Math.cos(nodeAngle * rad);
        yPositionNew = yPosition + (nodeVelocity * t)
            * Math.sin(nodeAngle * rad);

        if (xPositionNew > 100 && yPositionNew < 100) {
            xPositionNew = 100;
            yPositionNew = yPosition + Math.tan(nodeAngle *
rad)
                * (100 - xPosition);
            nodeAngle = nodeAngle + 180;
        }
        if (yPositionNew > 100 && xPositionNew < 100) {
            yPositionNew = 100;
            xPositionNew = xPosition + (1.0 /
Math.tan(nodeAngle * rad))
                * (100 - yPosition);
            nodeAngle = nodeAngle + 180;
        }
        if (xPositionNew > 100 && yPositionNew > 100) {
            xPositionNew = 100;
            yPositionNew = 100;
            nodeAngle = nodeAngle + 180;
        }
    }
}

```



```

    }

}

if (nodeAngle > 180 && nodeAngle < 270) {
    xPositionNew = xPosition + (nodeVelocity * t)
        * Math.cos(nodeAngle * rad);

    yPositionNew = yPosition + (nodeVelocity * t)
        * Math.sin(nodeAngle * rad);

    if (xPositionNew < 0 && yPositionNew > 0) {
        xPositionNew = 0;
        yPositionNew = yPosition + Math.tan(nodeAngle *
rad)
            * (0 - xPosition);
        nodeAngle = nodeAngle - 180;
    }
    if (yPositionNew < 0 && xPositionNew > 0) {
        yPositionNew = 0;
        xPositionNew = xPosition + (1.0 /
Math.tan(nodeAngle * rad))
            * (0 - yPosition);
        nodeAngle = nodeAngle - 180;
    }
    if (xPositionNew < 0 && yPositionNew < 0) {
        xPositionNew = 0;
        yPositionNew = 0;
        nodeAngle = nodeAngle - 180;
    }
}
}

```

```

        if (nodeAngle > 270 && nodeAngle <= 360) {

            xPositionNew = xPosition + (nodeVelocity * t)
                * Math.cos(nodeAngle * rad);

            yPositionNew = yPosition + (nodeVelocity * t)
                * Math.sin(nodeAngle * rad);
            if (xPositionNew > 100 && yPositionNew > 0) {
                xPositionNew = 100;
                yPositionNew = yPosition + Math.tan(nodeAngle *
rad)
                    * (100 - xPosition);
                nodeAngle = nodeAngle - 180;
            }
        }

    public void updateSpeedDirection(Node N[], int numberofnodes, int k) {
        for (int i = 0; i < numberofnodes - 5; i++) {
            nodeVelocity = rand.nextInt(6);
            nodeAngle = rand.nextInt(360);
        }
    }

    public void display() {
        System.out.println(nodeId + " (" + xPosition + "," + yPosition
            + ") vel=" + nodeVelocity);
    }
}
➤ Mobility.java

```

```
package mobic1;

import java.applet.Applet;

import java.awt.Button;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Label;

import java.awt.TextField;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Mobility extends Applet implements Runnable, ActionListener {

    Node[] N;

    Thread th;

    int numberOfNodes;

    TextField nodes;

    Label l1;

    Button resume, button, pause;

    int flag = 0, suspend = 0;

    MobicAlgo ob;

    double[] heads;
```

```
double[][] cluster;

public void init() {
    ob = new MobicAlgo();
    nodes = new TextField(5);
    l1 = new Label("Enter the number of nodes: ");
    add(l1);
    add(nodes);
    button = new Button("Submit");
    button.addActionListener(this);
    add(button);
    pause = new Button("Pause");
    pause.addActionListener(this);
    add(pause);
    resume = new Button("restart");
    resume.addActionListener(this);
    add(resume);
}

public void run() {
    while (true) {
```

```
        if (suspend == 0)
            for (int i = 0; i < numberOfNodes; i++) {
                N[i].currentPosition();
                repaint();
            }
        else {
            ob.implementAlgo(N, numberOfNodes);
            repaint();
        }

        try {
            Thread.sleep(2000);
        } catch (Exception e) {
        }

    }
}
```

```
public void actionPerformed(ActionEvent ae) {

    try {
```

```
        if (ae.getSource() == button) {  
            flag = 1;  
  
            numberOfNodes = Integer.parseInt(nodes.getText());  
            N = new Node[numberOfNodes];  
            heads = new double[numberOfNodes];  
            cluster = new double[numberOfNodes][numberOfNodes];  
            for (int i = 0; i < N.length; i++)  
                N[i] = new Node();  
            th = new Thread(this);  
            th.start();  
        }  
    } catch (Exception e) {  
    }  
  
    if (ae.getSource() == pause) {  
        suspend = 1;  
    }  
  
    if (ae.getSource() == resume) {  
        suspend = 0;
```

```

    }
}

public void paint(Graphics g) {

    // double[] heads=new double[N.length];

    g.drawRect(80, 80, 100, 100);

    if (flag > 0) {

        if (suspend == 0) {

            g.setColor(Color.black);

            for (int i = 0; i < N.length; i++) {

                g.fillOval(((int) N[i].getPosition()) + 78,

                    ((int) N[i].getPosition()) + 78, 4, 4);

            }

        }

        // make clusters here

    else {

        heads = ob.getClusterHead();

        cluster = ob.getCluster();

        for (int i = 0; i < N.length; i++) {

            if (heads[i] == 1) {

```

```

g.setColor(Color.red);

g.drawOval(((int) N[i].getPosition()) + 40,
           ((int) N[i].getPosition()) +
40,
           80, 80);

g.fillOval(((int) N[i].getPosition()) + 78,
           ((int) N[i].getPosition()) +
78, 4, 4);
    }
}

g.setColor(Color.black);
for (int i = 0; i < N.length; i++) {
    for (int j = 0; j < N.length; j++) {
        if (cluster[i][j] == 1) {
            if (i != j) {

                g.fillOval(((int) N[j].getPosition()) +
78,
                (int) N[j].getPosition()) + 78, 4, 4);
            }
        }
    }
}

```



```
    }  
    }  
}
```

➤ MobicAlgo.java

```
package mobic1;  
public class MobicAlgo {  
  
    double[][] prev, current, ratio, variance, cluster;  
    double[] mean, meanVariance, clusterHead;  
  
    public double[] getClusterHead()  
    {  
        return clusterHead;  
    }  
    public double [][] getCluster()  
    {  
        return cluster;  
    }  
    public void implementAlgo(Node[] N, int number) {  
  
        prev = new double[number][number];  
        current = new double[number][number];  
        ratio = new double[number][number];  
        variance = new double[number][number];  
        mean = new double[number];  
        meanVariance = new double[number];  
        clusterHead = new double[number];  
        cluster = new double[number][number];  
    }  
}
```

```

int nodeNumber, flag = 1;

for (int i = 0; i < number; i++) {
    for (int j = 0; j < number; j++) {
        prev[i][j] = 0;
        current[i][j] = 0;
        ratio[i][j] = 0;
        variance[i][j] = 0;
        cluster[i][j] = 0;
    }
    meanVariance[i] = 0;
    mean[i] = 0;
    clusterHead[i] = 0;
}

for (int i = 0; i < number; i++) {
    for (int j = 0; j < number; j++) {
        prev[i][j]
Math.sqrt(Math.pow(N[i].getXPositionPrev()
- N[j].getXPositionPrev(), 2)
+ Math.pow((N[i].getYPositionPrev() -
N[j]
.getYPositionPrev()),
2));
        current[i][j]
Math.sqrt(Math.pow(N[i].getXPosition()
- N[j].getXPosition(), 2)
+ Math.pow(N[i].getXPosition()-
N[j].getXPosition(),2));
    }
}

```

```

    }

    // RSSI is directly proportional to the distance between nodes
    // ratio of distances
    for (int i = 0; i < number; i++) {
        for (int j = 0; j < number; j++) {
            if (i != j)
                ratio[i][j] = current[i][j] / prev[i][j];
            else
                ratio[i][j] = 0;
        }
    }

    for (int i = 0; i < number; i++) {
        for (int j = 0; j < number; j++) {
            mean[i] += ratio[i][j];
        }
        mean[i] = mean[i] / (number - 1);
    }

    // finding mean variance...
    for (int i = 0; i < number; i++) {
        for (int j = 0; j < number; j++) {
            meanVariance[i] += variance[i][j];
        }
        meanVariance[i] = meanVariance[i] / (number - 1);
    }

    while (flag == 1) {

```

```

nodeNumber = minVariance(number); // find node with min
mean
// Variance
clusterHead[nodeNumber] = 1;
meanVariance[nodeNumber] = 1000;
// get position of that node

// N[nodeNumber].getRange();
for (int i = 0; i < number; i++) {
    if
(inRange(N[nodeNumber].getXPosition(),N[nodeNumber].getYPosition(),
N[i].getXPosition(), N[i].getYPosition(), N[nodeNumber].getRange())) {
        cluster[nodeNumber][i] = 1;
        meanVariance[i] = 1000;
    }
}
flag = 0;
for (int i = 0; i < number; i++) {
    if (meanVariance[i] < 1000) {
        flag = 1;
        break;
    }
}
}
System.out.println("\nHeads");
for(int i=0;i<N.length;i++)
    System.out.print(clusterHead[i]+" ");
System.out.println("\nCluster");
for (int i = 0; i < number; i++) {
    System.out.println("");
    for(int j=0;j<N.length;j++)

```

```

        System.out.print(cluster[i][j] + " ");

    }

}

public int minVariance(int number) {
    int min = 0;
    for (int i = 0; i < number; i++) {
        if (meanVariance[i] < meanVariance[min])
            min = i;
    }
    return min;
}

public boolean inRange(double xhead, double yhead, double xnode,
    double ynode, double range) {
    double rad = 3.14159 / 180.0;
    double x, y;
    if ((xnode >= (xhead - range) && xnode <= (xhead + range) && ynode ==
0)
    || (xnode == 0 && ynode >= (yhead - range) && ynode <= (yhead +
range)))
        return true;
    for (int i = 1; i < 90; i++) {
        x = range * Math.cos(i * rad);
        y = range * Math.sin(i * rad);
        if (xnode >= xhead && xnode <= x && ynode >= yhead && ynode
<= y)
            return true;
    }
}

```

```

    }
    for (int i = 91; i < 180; i++) {
        x = range * Math.cos(i * rad);
        y = range * Math.sin(i * rad);
        if (xnode <= xhead && xnode >= (xhead+x) && ynode >= yhead && ynode
        <= y)
            return true;
    }
    for (int i = 181; i < 270; i++) {
        x = range * Math.cos(i * rad);
        y = range * Math.sin(i * rad);
        if (xnode <= xhead && xnode >= (xhead+x) && ynode <= yhead && ynode
        >=(yhead+y))
            return true;
    }
    for (int i = 271; i < 360; i++) {
        x = range * Math.cos(i * rad);
        y = range * Math.sin(i * rad);
        if (xnode >= xhead && xnode <= x && ynode <= yhead && ynode >=
        (yhead+y))
            return true;
    }

    return false;
}
}

```

➤ TransmissionRange.java  
package mobic1;

```

import java.awt.BasicStroke;

public class TransmissionRange extends JPanel {

    private int width = 800;

```

```

private int height = 400;
private static int padding = 30;
private static int labelPadding = 20;
private Color lineColor = new Color(44, 102, 230, 100);
private Color lineColor1 = new Color(100, 50, 230, 150);
private Color lineColor2 = new Color(44, 20, 230, 50);
private Color lineColor3 = new Color(214, 210, 180, 150);
private Color pointColor = new Color(100, 100, 100, 180);
private Color gridColor = new Color(200, 200, 200, 200);
private static final Stroke GRAPH_STROKE = new BasicStroke(2f);
private int pointWidth = 4;
private int numberYDivisions = 25;
private List<Double> scores;

public TransmissionRange(List<Double> scores) {
    this.scores = scores;
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    Graphics2D g2 = (Graphics2D) g;

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);

```

```

    double xScale = ((double) getWidth() - (2 * padding) - labelPadding) / (scores.size() -
1);

    double yScale = ((double) getHeight() - 2 * padding - labelPadding) / (getMaxScore()
- getMinScore());

    List<Point> graphPoints = new ArrayList<>();

    List<Point> graphPoints1 = new ArrayList<>();

    List<Point> graphPoints2 = new ArrayList<>();

    List<Point> graphPoints3 = new ArrayList<>();

//   for (int i = 0; i < scores.size(); i++) {
        //int x1 = (int) (i * xScale + padding + labelPadding);

        // int y1 = (int) ((getMaxScore() - ycs.get(i)) * yScale + padding);

        // graphPoints.add(new Point(x1, y1));

//}

int xa = (int)(xScale + padding + labelPadding);
int xb = (int)(2*xScale + padding + labelPadding);
int xc = (int)(3*xScale + padding + labelPadding);
int xd = (int)(4*xScale + padding + labelPadding);
int xe = (int)(5*xScale + padding + labelPadding);

//int ya = (int) (550- 3 * yScale + padding);

graphPoints.add(new Point(xe , 30));//When Transmission range is 10

graphPoints.add(new Point(xd, 50));

graphPoints.add(new Point(xc, 180));

graphPoints.add(new Point(xb, 260));

graphPoints.add(new Point(xa, 405));

graphPoints1.add(new Point(xe, 220));//When Transmission range is 20

```



```

graphPoints1.add(new Point(xd, 240));
graphPoints1.add(new Point(xc, 260));
graphPoints1.add(new Point(xb, 345));
graphPoints1.add(new Point(xa, 448));
graphPoints3.add(new Point(xe, 448));//When Transmission range is 40
graphPoints3.add(new Point(xd, 448));
graphPoints3.add(new Point(xc, 448));
graphPoints3.add(new Point(xb, 467));
graphPoints3.add(new Point(xa, 490));
graphPoints2.add(new Point(xe, 362));//When Transmission Range is 30
graphPoints2.add(new Point(xd, 362));
graphPoints2.add(new Point(xc, 405));
graphPoints2.add(new Point(xb,427));
graphPoints2.add(new Point(xa,467));

// draw white background
g2.setColor(Color.WHITE);

g2.fillRect(padding + labelPadding, padding, getWidth() - (2 * padding) -
labelPadding, getHeight() - 2 * padding - labelPadding);

g2.setColor(Color.BLACK);

// create hatch marks and grid lines for y axis.
for (int i = 0; i < numberYDivisions + 1; i++) {
    int x0 = padding + labelPadding;

    int x1 = pointWidth + padding + labelPadding;

    int y0 = getHeight() - ((i * (getHeight() - padding * 2 - labelPadding)) /
numberYDivisions + padding + labelPadding);

```

```

int y1 = y0;
/*if (scores.size() > 0) {
    g2.setColor(gridColor);
    g2.drawLine(padding + labelPadding + 1 + pointWidth, y0, getWidth() -
padding, y1);
    g2.setColor(Color.BLACK);
    String yLabel = ((int) ((getMinScore() + (getMaxScore() - getMinScore()) * ((i *
10) / numberYDivisions)) * 100)) / 100 + "";
    FontMetrics metrics = g2.getFontMetrics();
    int labelWidth = metrics.stringWidth(yLabel);
    g2.drawString(yLabel, x0 - labelWidth - 5, y0 + (metrics.getHeight() / 2) - 3);
}
g2.drawLine(x0, y0, x1, y1);
*/ //int y1 = y0 - pointWidth;
if ((i % ((int) ((scores.size() / 20.0)) + 1)) == 0) {
    g2.setColor(gridColor);
    g2.drawLine(padding + labelPadding + 1 + pointWidth, y0, getWidth() - padding,
y1);
    g2.setColor(Color.BLACK);
    String yLabel = i + "";
    //String yLabel = ((int) ((getMinScore() + (getMaxScore() - getMinScore()) * ((i *
10) / numberYDivisions)) * 100)) / 100 + "";
    FontMetrics metrics = g2.getFontMetrics();
    int labelWidth = metrics.stringWidth(yLabel);
    g2.drawString(yLabel, x0 - labelWidth - 5, y0 + (metrics.getHeight()
/ 2) - 3);

```

```

    }
    }

    // and for x axis
    for (int i = 0; i < scores.size(); i++) {
        if (scores.size() > 1) {
            int x0 = i * (getWidth() - padding * 2 - labelPadding) / (scores.size() - 1) +
padding + labelPadding;
            int x1 = x0;
            int y0 = getHeight() - padding - labelPadding;
            int y1 = y0 - pointWidth;
            if ((i % ((int) ((scores.size() / 20.0)) + 1)) == 0) {
                g2.setColor(gridColor);
                g2.drawLine(x0, getHeight() - padding - labelPadding - 1 - pointWidth, x1,
padding);
                g2.setColor(Color.BLACK);
                String xLabel = i*10 + "";
                FontMetrics metrics = g2.getFontMetrics();
                int labelWidth = metrics.stringWidth(xLabel);
                g2.drawString(xLabel, x0 - labelWidth / 2, y0 + metrics.getHeight() + 3);
            }
            g2.drawLine(x0, y0, x1, y1);
        }
    }

    // create x and y axes

```

```
g2.drawLine(padding + labelPadding, getHeight() - padding - labelPadding, padding
+ labelPadding, padding);
```

```
g2.drawLine(padding + labelPadding, getHeight() - padding - labelPadding,
getWidth() - padding, getHeight() - padding - labelPadding);
```

```
Stroke oldStroke = g2.getStroke();
g2.setColor(lineColor);
g2.setStroke(GRAPH_STROKE);
for (int i = 0; i < graphPoints.size() - 1; i++) {
    int x1 = graphPoints.get(i).x;
    int y1 = graphPoints.get(i).y;
    int x2 = graphPoints.get(i + 1).x;
    int y2 = graphPoints.get(i + 1).y;
    g2.drawLine(x1, y1, x2, y2);
}
```

```
g2.setStroke(oldStroke);
g2.setColor(pointColor);
for (int i = 0; i < graphPoints.size(); i++) {
    int x = graphPoints.get(i).x - pointWidth / 2;
    int y = graphPoints.get(i).y - pointWidth / 2;
    int ovalW = pointWidth;
    int ovalH = pointWidth;
    g2.fillOval(x, y, ovalW, ovalH);
}
```

```
g2.setColor(lineColor1);
```

```
g2.setStroke(GRAPH_STROKE);
for (int i = 0; i < graphPoints1.size() - 1; i++) {
    int x1 = graphPoints1.get(i).x;
    int y1 = graphPoints1.get(i).y;
    int x2 = graphPoints1.get(i + 1).x;
    int y2 = graphPoints1.get(i + 1).y;
    g2.drawLine(x1, y1, x2, y2);
}
```

```
g2.setStroke(oldStroke);
g2.setColor(pointColor);
for (int i = 0; i < graphPoints1.size(); i++) {
    int x = graphPoints1.get(i).x - pointWidth / 2;
    int y = graphPoints1.get(i).y - pointWidth / 2;
    int ovalW = pointWidth;
    int ovalH = pointWidth;
    g2.fillOval(x, y, ovalW, ovalH);
}
```

```
static void createAndShowGui() {
    List<Double> scores = new ArrayList<>();
    Random random = new Random();
    int maxDataPoints = 6; int maxScore = 10;
    for (int i = 0; i < maxDataPoints; i++) {
        double graphPoints = 0;
```

```

scores.add((double) random.nextDouble() *graphPoints;
}

TransmissionRange mainPanel = new TransmissionRange(scores);

    mainPanel.setPreferredSize(new Dimension(800, 600));

    JFrame frame = new JFrame("DrawGraph");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.getContentPane().add(mainPanel);

    frame.pack();

    frame.setLocationRelativeTo(null);

    frame.setVisible(true);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGui();
        }
    });
}
}

```

## Chapter 6 Average Cluster Calculation and Stability

### 6.1 Definition

By calculating the average the number of clusters in different simulation gives the amount of stability in the environment. To calculate the stability we need to know the number clusters being made.

### 6.2 Code



```
for(int i=0; i<N.Length; i++)  
{  
    if(clusterHead[i]==1)  
        count++;  
}  
  
System.out.println("\nno of clusters " +count);
```

If the number of clusters comes out to be more than the environment is less stable but if the number of clusters are less than the environment is more stable since the environment is less mobile.

### 6.3 Need For Stability:

- **Stable Routes:** To maximize throughput and reduce traffic latency, it is essential to ensure reliable source-destination connections over time. A route should therefore be elected based on some knowledge of the nodes motion and on a probability model of the path future availability.
- **Efficient Route Repair:** If an estimate of the path duration is available, service disruption due to route failure can be avoided by creating an alternative path before the current one breaks. Note that having some information on the path duration avoids waste of radio resources due to pre-allocation of backup paths.
- **Network Connectivity:** Connectivity and topology characteristics of a MANET are determined by the link dynamics. These are fundamental issues to network

design, since they determine the system capability to support user communications and their reliability level.

- **Performance Evaluation:** The performances achieved by high-layer protocols, such as transport and application protocols, heavily depend on the quality of service metrics obtained at the network layer. As an example, the duration and frequency of route disruptions have a significant impact on TCP behavior, as well as on video streaming and VoIP services. Thus, characterizing route stability is the basis to evaluate the quality of service perceived by the users.

## 6.4 Stability

Analyzing the result on the following parameters

- Number of nodes (scalability)
- Velocity of nodes (uniform or non uniform)
- Transmission power

By changing the number of nodes and calculating the average number of clusters determines the stability of the environment. In this project analysis have being done by changing number of nodes from 10 to 20,30,40,50 and following average clusters have being calculated at varying transmission power. Furthermore keeping the number of nodes constant, velocity has being changed thereby plotting the graph for the same which gave no generalised output. Whereas when the velocity is also constant then after certain amount of time number of clusters become stable that means the simulation environment becomes stable. Thus by changing various parameters graphs has been plotted which indicates that when the environment is stable and when it is not that is for constant velocity and for a particular range the environment becomes stable whereas in other cases the environment is less stable.



## **Chapter 5 Conclusion**

### **5.1 Conclusion**

The MANETS AND MOBIC algorithms were thoroughly studied taking a number of parameters varying the conditions and variables. All the work mentioned above involved real time data. MOBIC algorithm was implemented and was a success. The overall success rate of implementing the algorithm was 95%.

## **Chapter 6 Glossary:**

### **6.1 Acronyms**

**MANET** -----Mobile Adhoc Networks

**MOBIC** -----Mobility Based Clustering

**ID** -----Identity

**DSDV** ----- Destination-Sequenced Distance-Vector  
Routing

**AODV** ----- Ad hoc On-Demand Distance Vector Routing

**MM** -----Mobility Metric

**MN** -----Mobility Node

**RWP** -----Random Way Point Model

**CH** -----Cluster Head

**CN** -----Cluster Node

**RIP** -----Routing Information Protocol

**JVM** -----JAVA Virtual Machine

**IDE** -----Integrated Development Environment

**JRE** -----JAVA RunTime Environment

## **Chapter 7 References, IEEE Format**

### **Book**

[1] Herbert Schildt, Complete Reference JAVA, 5<sup>th</sup> edition, Tata McGraw Hill.

[2] Charles E.Perkins, AD HOC Networking, Perason Education, 08-Jan-2001.

### **Research Paper**

[3] Prithwish Basu, Naved Khan, Thomas D. C. Little, “A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks”, Department of Electrical and Computer Engineering Boston University, Boston MA.

### **Technical Report**

[4] Frank Mufalli, Rakesh Nagi, Jim Llinas, Sumita Mishra, “Investigation of Means of Mitigating Congestion in Complex, Distributed Network Systems by Optimization Means and Information Theoretic Procedures”, Paine College, 1235 15th Street, Augusta GA.

### **Web References**

[5] MANET Research Summary, <http://.hulk.bu.edu/projects/adhoc/summary.html>.

[6]Mobility Based metric for Clustering in mobile and adhoc networks, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=918738&queryText%3Dmanets+for+mobic>.

[7] [www.academia.edu](http://www.academia.edu)

[8] [http://google.co.in/?gfe\\_rd=cr&ei=SLRxU8GuAo\\_V8gf7nIDQCQ#q=manets](http://google.co.in/?gfe_rd=cr&ei=SLRxU8GuAo_V8gf7nIDQCQ#q=manets), Google Images.

### **Software**

[9] Visual Paradigm, for diagrams.

[10] Creately, for diagrams.