qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwe

rtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwetyuiopasdfghjklzxcvbnmqwertyuiopadfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm

|  |
| --- |
| DESIGNING A SIMULATION APPLICATION FOR OPTIMIZED LINK STATE ROUTING PROTOCOL IN MOBILE AD HOC NETWORKS  12/16/2014  Project Report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology in  Computer Science & Engineering under the Supervision of  AMOL VASUDEVA  By SHUBHI JAIN 111338   to DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY W AKNAGHAT |

**CERTIFICATE**

This is to certify that the work titled “**Designing a Simulation Application for Optimized Link State Routing Protocol for Mobile Ad Hoc Networks”** submitted by **Shubhi Jain** in the partial fulfilment for the award of degree of Bachelor of Technology in Computer Science & Engineering from Jaypee University of Information Technology, Waknaghat 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

Shubhi Jain

Contents

[Chapter 1: Mobile Adhoc Networks(MANETS) 10](#_Toc387596465)

[1.1 Introduction 10](#_Toc387596466)

[1.2Characteristics 10](#_Toc387596467)

[1.3 Applications 11](#_Toc387596468)

[1.4 Limitations 11](#_Toc387596469)

[1.5 Challenges 12](#_Toc387596470)

[Chapter 2: Routing Algorithms 16](#_Toc387596477)

[2.1 Definition 16](#_Toc387596478)

[2.2 Types 16](#_Toc387596479)

[2.2.1 On-Demand Driven Protoco/Reactive: 16](#_Toc387596480)

[2.2.2 Ad hoc On-Demand Distance Vector Routing (AODV): 17](#_Toc387596481)

[2.2.3 Hybrid Routing Protocol:. 18](#_Toc387596482)

[2.2.4 Table Driven Routing Protocol/Proactive: 18](#_Toc387596483)

[2.2.5 Link and neighbour Sensing: 19](#_Toc387596484)

[2.2.6 Multipoint Relaying: 19](#_Toc387596485)

[2.2.7 Link State messaging and Route Calculation: 20](#_Toc387596486)

[Chapter 3 : Mobility Models 14](#_Toc387596471)

[3.1 Definition 14](#_Toc387596472)

[3.2 Types 14](#_Toc387596473)

[3.2.1 Random Walk Mobility Model 14](#_Toc387596474)

[3.2.2 Random Waypoint Mobility Model 15](#_Toc387596475)

[3.2.3 Random Direction Mobility Model 15](#_Toc387596476)

[Chapter 4: Implementation 21](#_Toc387596487)

[4.1 Simulation Environment 21](#_Toc387596488)

[4.1.1 Comparison Parameters: 21](#_Toc387596489)

[4.2 Tools and Technologies: 21](#_Toc387596490)

[4.2.1 Java 1.6 Version: 22](#_Toc387596491)

[4.2.1.1 Characteristics: 22](#_Toc387596492)

[4.3.3.2 JAVA Virtual Machine (JVM): 22](#_Toc387596493)

[4.3.3.3 Applet and Standalone Application: 23](#_Toc387596494)

[4.2.2 Eclipse Galileo Version 3.5.1 23](#_Toc387596495)

[4.3 Diagrams 23](#_Toc387596496)

[4.3.1 Use Case Diagram 23](#_Toc387596497)

[4.3.2 Data Flow Diagram 24](#_Toc387596498)

[4.4 Algorithm for OLSR: 25](#_Toc387596499)

[4.5 Code 30](#_Toc387596500)

[Chapter 5 Route Optimalityand Stability 57](#_Toc387596501)

[5.1 Definition 57](#_Toc387596502)

[5.2 Benefits 57](#_Toc387596503)

[5.3 Need For Stability: 57](#_Toc387596504)

[5.4 Scalability 58](#_Toc387596505)

[Chapter 6 Conclusion 59](#_Toc387596506)

[6.1 Conclusion 59](#_Toc387596507)

[Chapter 7 Glossary: 60](#_Toc387596508)

[Chapter 8 References, IEEE Format 61](#_Toc387596509)

**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. OLSR protocol is an optimization of the classical link state algorithm tailored to the requirements of a mobile wireless LAN. The key concept used in the protocol is that of multipoint relays (MPRs). MPRs are selected nodes which forward broadcast messages during the flooding process. This technique substantially reduces the message overhead as compared to a classical flooding mechanism, where every node retransmits each message when it receives the first copy of the message. In OLSR, link state information is generated only by nodes elected as MPRs. Thus, a second optimization is achieved by minimizing the number of control messages flooded in the network. As a third optimization, an MPR node may chose to report only links between itself and its MPR selectors. Hence, as contrary to the classic link state algorithm, partial link state information is distributed in the network. This information is then used for route calculation. OLSR provides optimal routes (in terms of number of hops). The protocol is particularly suitable for large and dense networks as the technique of MPRs works well in this context.

**Problem Statement**

A console application can 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. However, it can cause confusion for large-scale projects. As a fix, a graphical user interface could provide a clean interface. Visualization will be incorporated into the software to graphically show node movement, congestion levels etc

**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 reasons 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 Optimized Link State Routing Protocol 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.

# Chapter 1: Mobile Ad hoc 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.

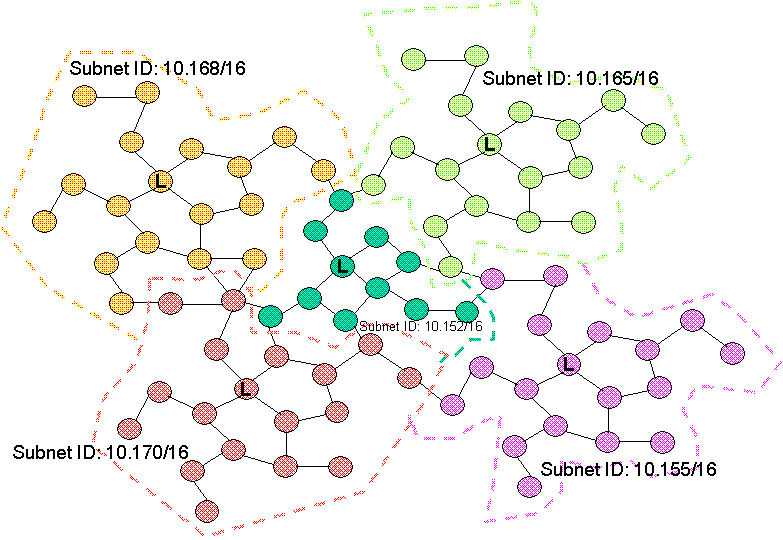


Figure 1.1: Example of MANET [ ]

# 1.2Characteristics

* In MANET, each node acts as both host and router. That is it is autonomous in behaviour.
* 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 behaviour 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.

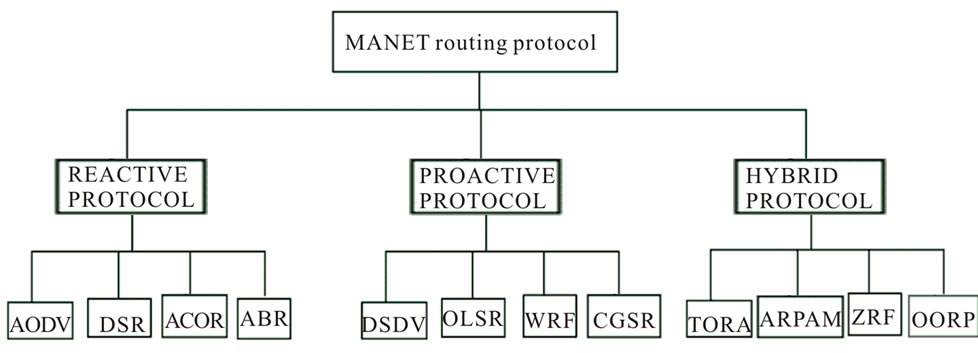
# 1.5 Challenges

* 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 behaviour 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: Routing Algorithms

# 2.1 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.



# 

# 2.2 Types

# 2.2.1 On-Demand Driven Protocol/Reactive:

Every node in this routing protocol maintains information of only active paths to the destination nodes. A route search is needed for every new destination therefore the communication overhead is reduced at the expense of delay to search the route. Rapidly changing wireless network topology may break active route and cause subsequent route search.

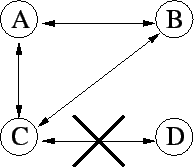


Figure 2.1: Reactive Routing

**2.2.2 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 2.2 (a) Ad hoc On-Demand Distance Vector Routing- source

Figure 2.2 (b) Ad hoc On-Demand Distance Vector Routing- Destination

# 2.2.3 Hybrid Routing Protocol:

These protocols try to incorporate various aspects of proactive and reactive routing protocols. They are generally used to provide hierarchical routing; routing in general can be either flat or hierarchical. The difficulty of all hybrid routing protocols is how to organize the network according to network parameters. The common disadvantage of hybrid routing protocols is that the nodes that have high level topological information maintains more routing information, which leads to more memory and power consumption. Some examples of Hybrid Routing Protocols include CEDAR , ZRP and SRP .

# 2.2.4 Table –Driven Routing protocol/Proactive:

In this type of routing protocol, each node in a network maintains one or more routing tables which are updated regularly. Each node sends a broadcast message to the entire network if there is a change in the network topology. However, it incurs additional overhead cost due to maintaining up-to-date information and as a result; throughput of the network may be affected but it provides the actual information to the availability of the network. Distance vector (DV) protocol, Destination Sequenced Distance Vector (DSDV) protocol, Wireless Routing protocol Fisheye State Routing (FSR) protocol ,Optimized link State Routing Protocol (OLSR) are the examples of Proactive protocols.

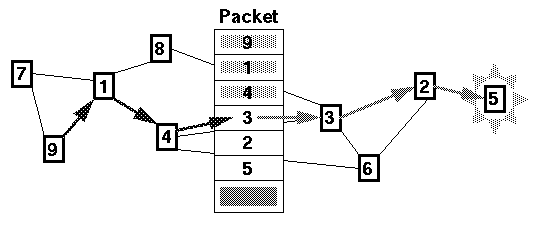


Figure 2.3 Proactive Routing

**2.2.4 Optimized Link State Routing:**

The *Optimized Link State routing* (OLSR) is a table-driven pro-active protocol. As the name suggests, it uses the link-state scheme in an optimized manner to diffuse topology information. In a classic link-state algorithm, link-state information is flooded throughout the network. OLSR uses this approach as well, but since the protocol runs in wireless multi-hop scenarios the message flooding in OLSR is optimized to preserve bandwidth. The optimization is based on a technique called *Multipoint Relaying*. Being a table-driven protocol, OLSR operation mainly consists of updating and maintaining information in a variety of tables. The data in these tables is based on received control traffic, and control traffic is generated based on information retrieved from these tables. The route calculation itself is also driven by the tables.

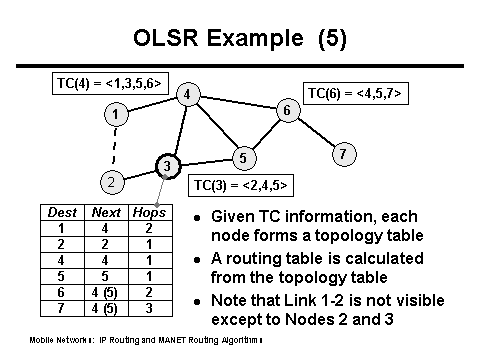
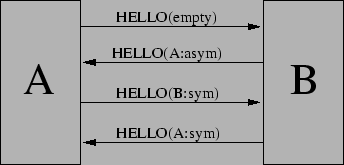


Figure 2.4 OLSR Routing

# 

# 2.2.5 Link and neighbor sensing:

Neighbours and links are detected by HELLO mesages.All nodes transmit HELLO messages on a given interval. These contain all heard-of neighbours grouped by status. A simplified neighbour detection scenario:

1

Figure 2.5 Link and Neighbour sensing

# 

# 2.2.6 Multipoint Relaying:

Reduce the number of duplicate retransmissions while forwarding a broadcast packet. Restricts the set of nodes retransmitting a packet from all nodes(regular flooding) to a subset of all nodes. The size of this subset depends on the topology of the network.

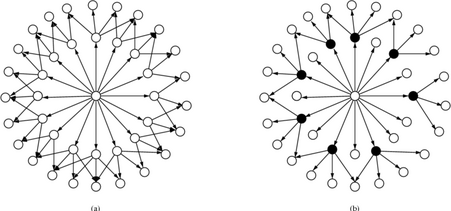


Figure 2.6: Multipoint Relaying

# 2.2.7 Link-State messaging and route calculation:

  
Figure 2.7: Link State Messaging and Route calculation

# 

# Chapter 3 : Mobility Models

# 3.1 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 modelling, the behaviour 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 behaviours 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.

# 3.2 Types

Typical mobility models include

* Random Walk Mobility Model
* Random Waypoint Mobility Model
* Random Direction Mobility Model

# 3.2.1 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; 2π] respectively. Each movement in the Random Walk Mobility Model occurs in either at constant time interval ***‘t’*** or a constant distance travelled ‘***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.

# 3.2.2 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 , 𝑉max). 𝑉max 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.

**3.2.3 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 4: Implementation

# 4.1 Simulation Environment

We will simulate an ad hoc network with n nodes randomly distributed in a 100 ×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 unit/sec] and [0, 2π] respectively. We have set the movement process time to be 2 seconds i.e. after each 2 seconds the nodes will move. Once the movement 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.

.

* + 1. **Comparison Parameters:**
* Number of nodes (scalability)
* Velocity of nodes (uniform or non uniform)
* Transmission power
* Density of nodes
  1. **Tools and Technologies:**
     1. **Java 1.6 Version:**
        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 programme is hardware and operating system independent. If well written (!), the same JAVA programme, compiled once, will run identically on a SUN/Solaris workstation, a PC/windows computer or a Macintosh computer. Not mentioning other Unix flavours, including Linux, and every Web browser, with some restrictions described below. This universal executability is made possible because a JAVA programme 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 programme 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 programme 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 programme, like the Java Runtime Environment (JRE) from Sun Microsystems. These programmes need to be downloaded for your particular platform. As seen in the next paragraph, these two types of Virtual Machines do not behave exactly the same.

# 4.3.3.3 Applet and Standalone Application:

A JVM in a web browser runs a JAVA programme 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 programme as a Standalone Application.

* + 1. **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.



Figure 4.1 Eclipse Galileo

* 1. **Diagrams**

**4.3.1 Use Case Diagram**

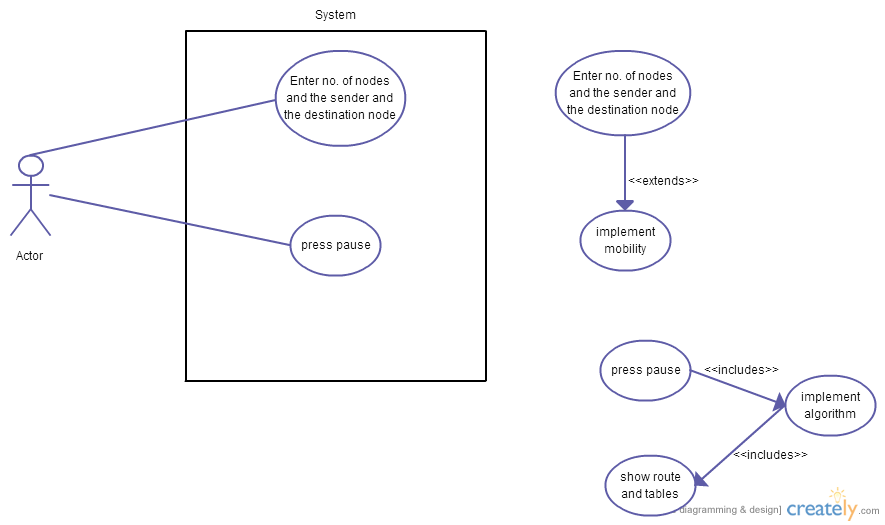


Figure 4.2 Use Case Diagram

# 4.3.2 Data Flow Diagram

**Zero Level DFD**

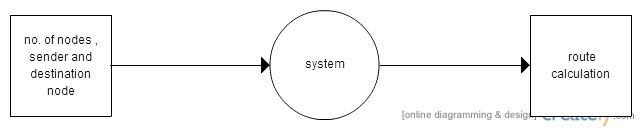
****

Figure 4.3 (a) Zero Level DFD

**First Level DFD**

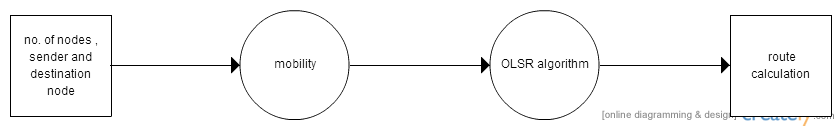
****

Figure 4.3 (b) First Level DFD

**Second Level DFD**

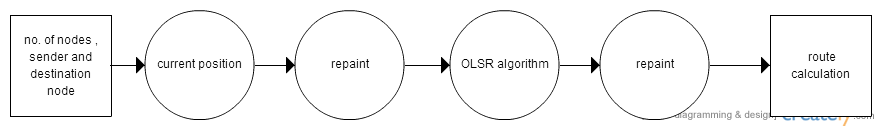
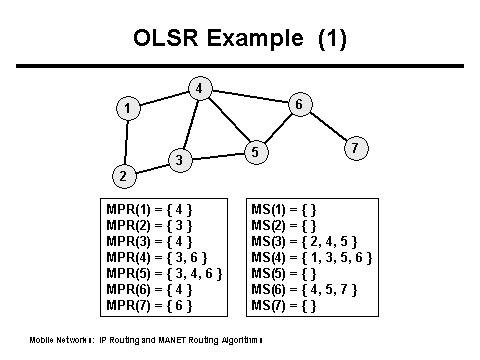
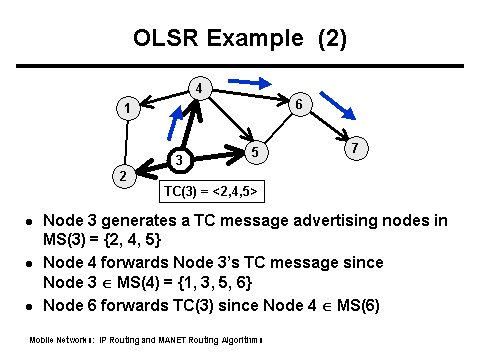
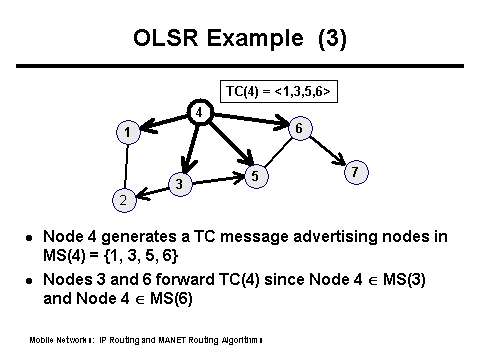
****

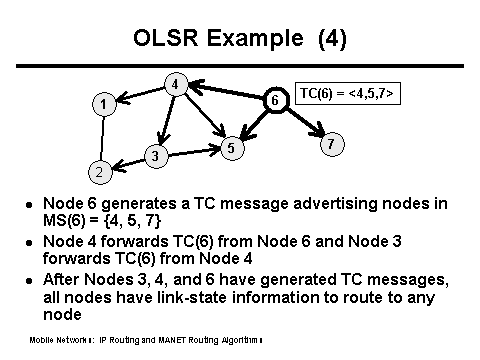
Figure 4.3 (c) Second Level DFD

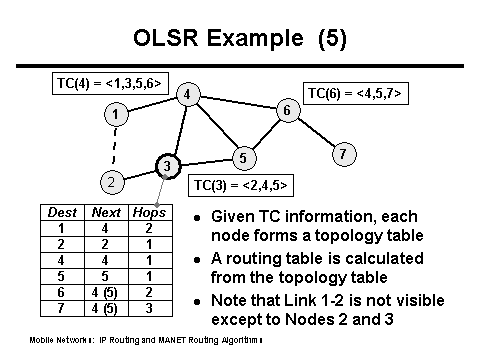
**4.4 Algorithm for OLSR:**





****

****

****

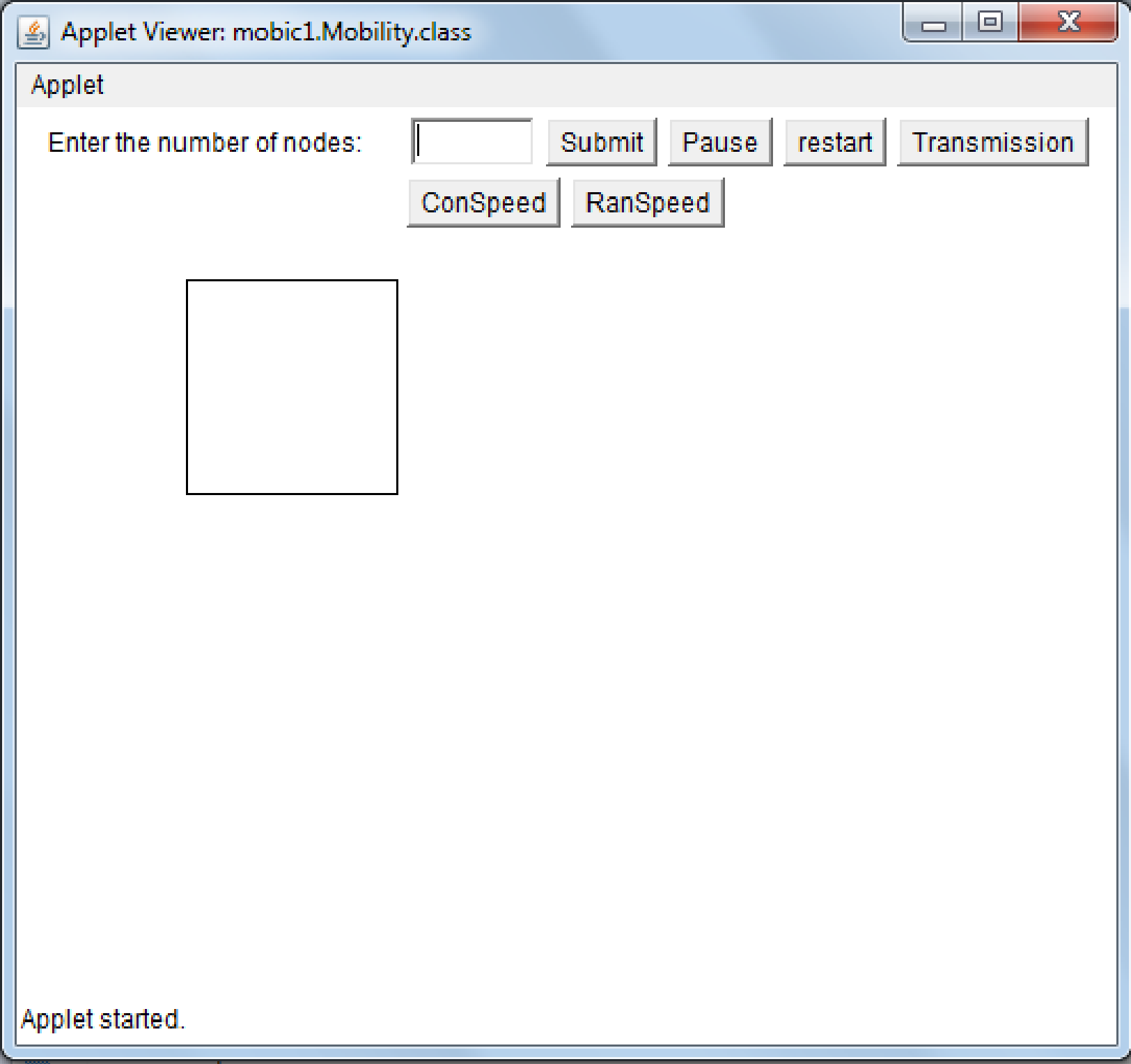


Figure 4.4 Initial Applet

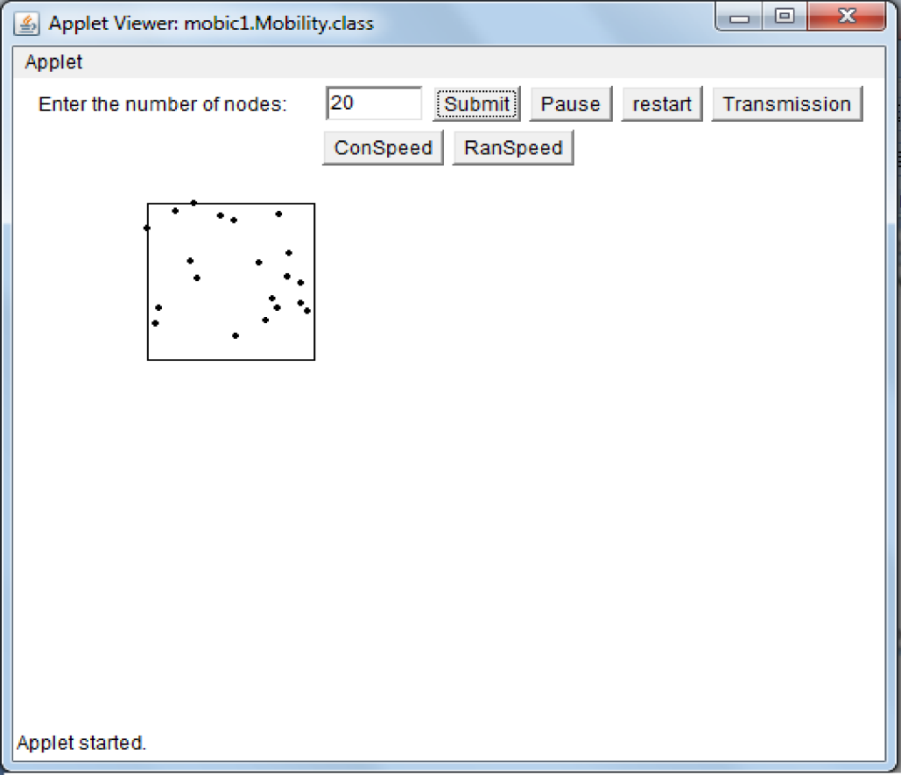


Figure 4.5 Nodes entered

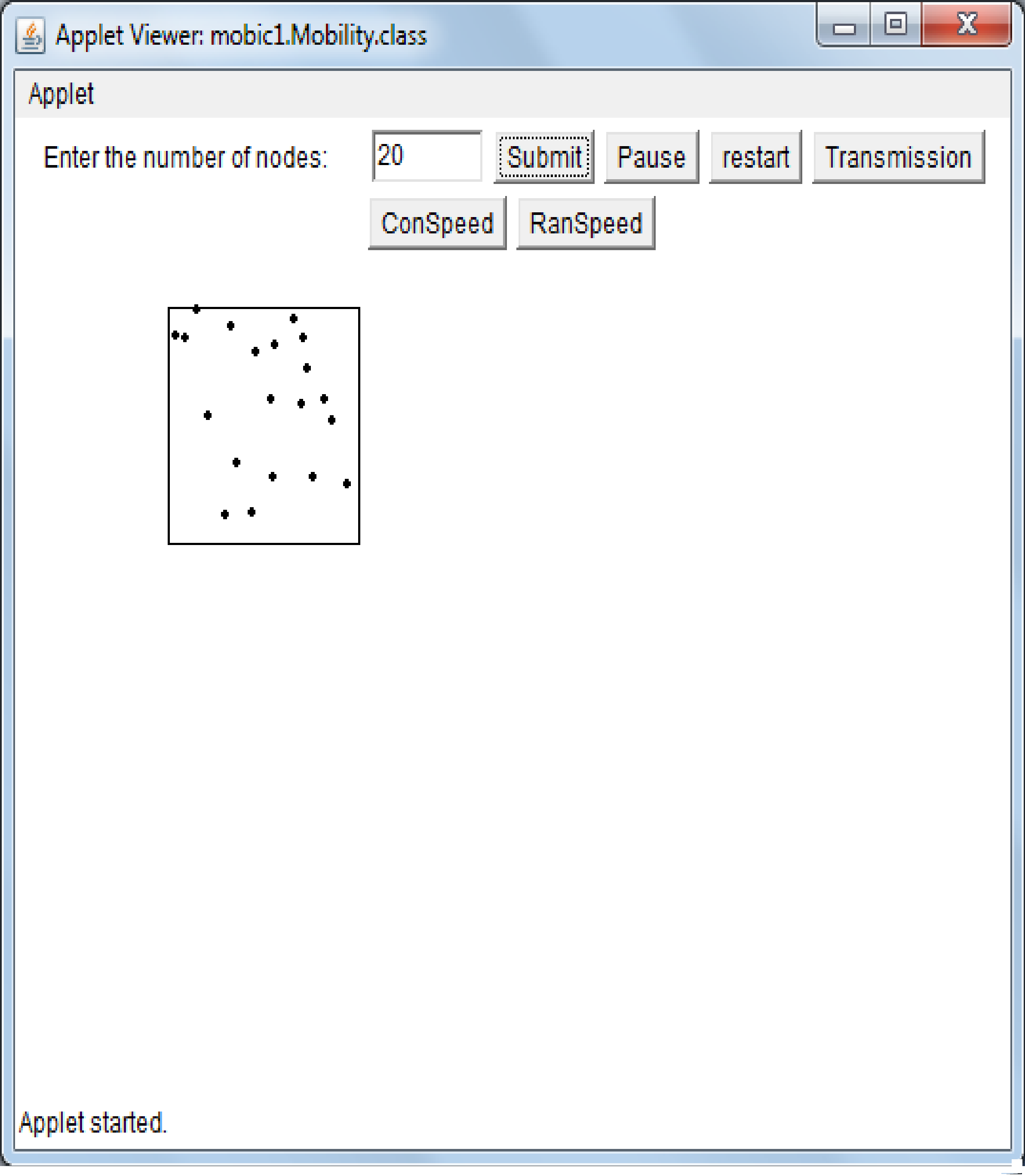


Figure 4.6 Press Pause

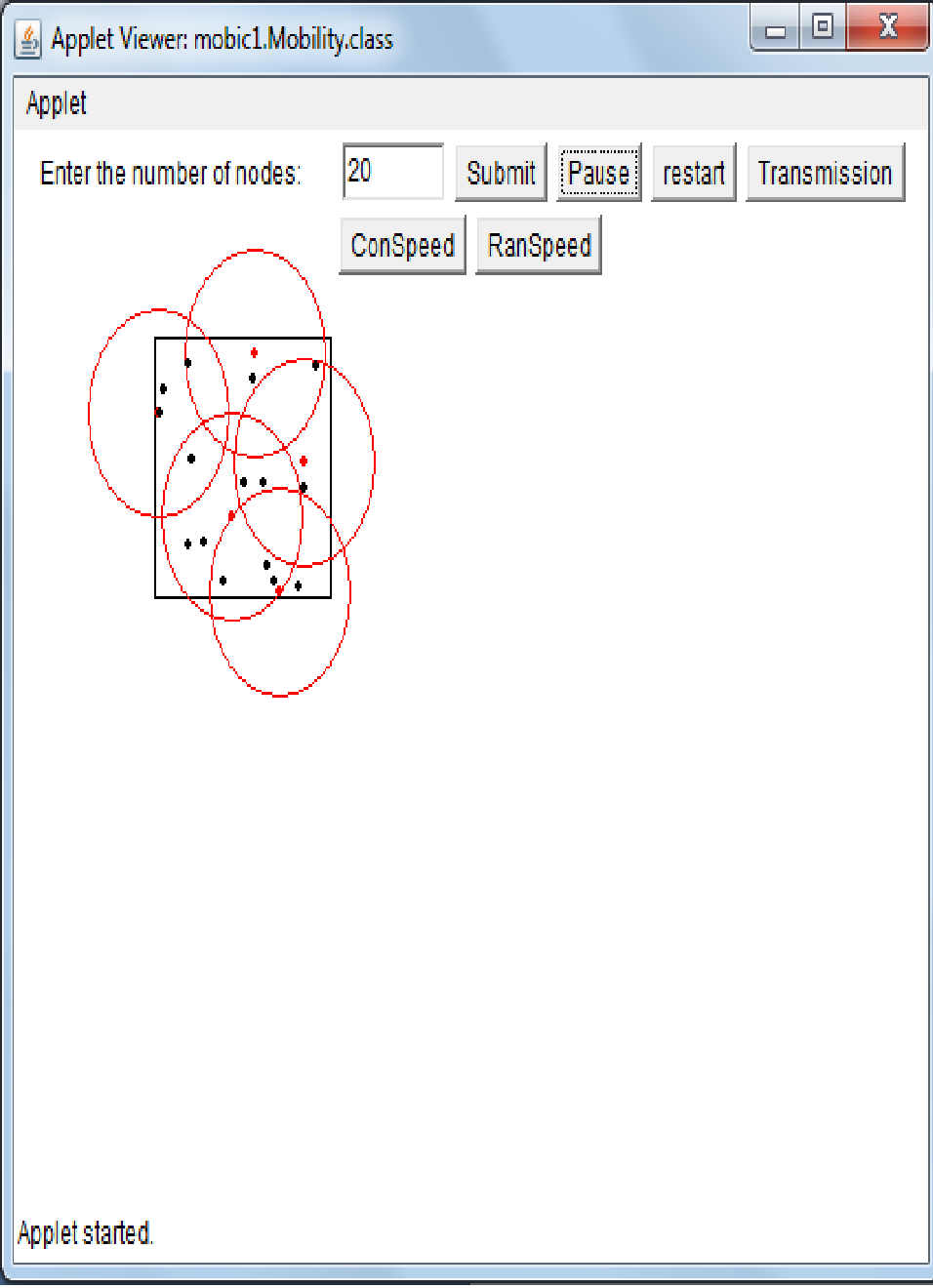


Figure 4.7 Implementation of Clustering

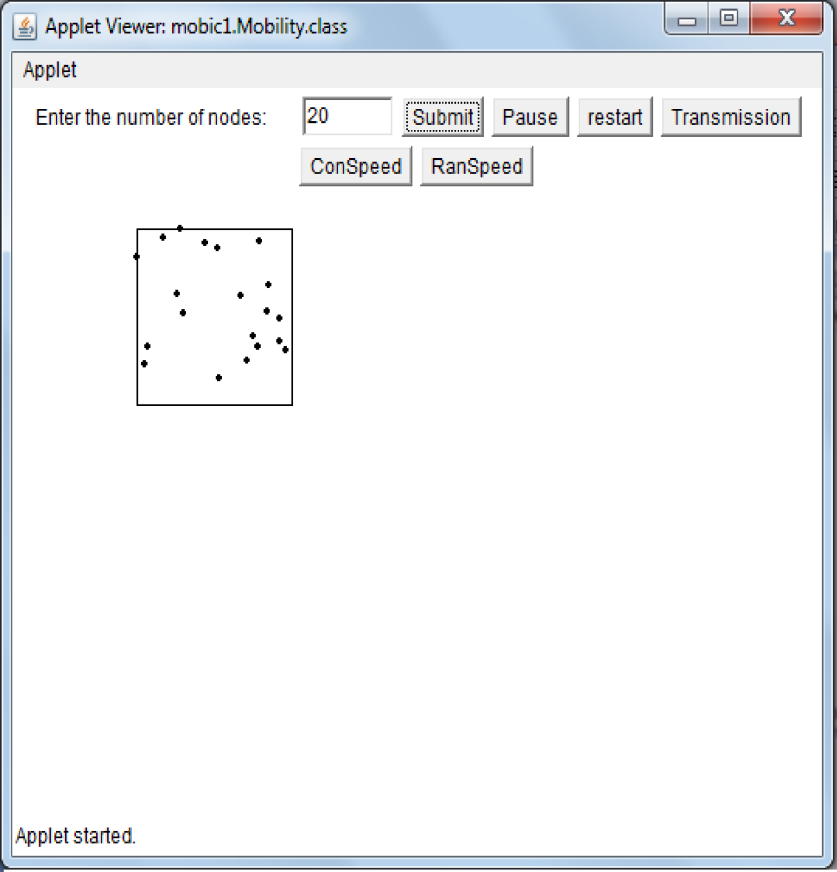


Figure 4.8 Restart

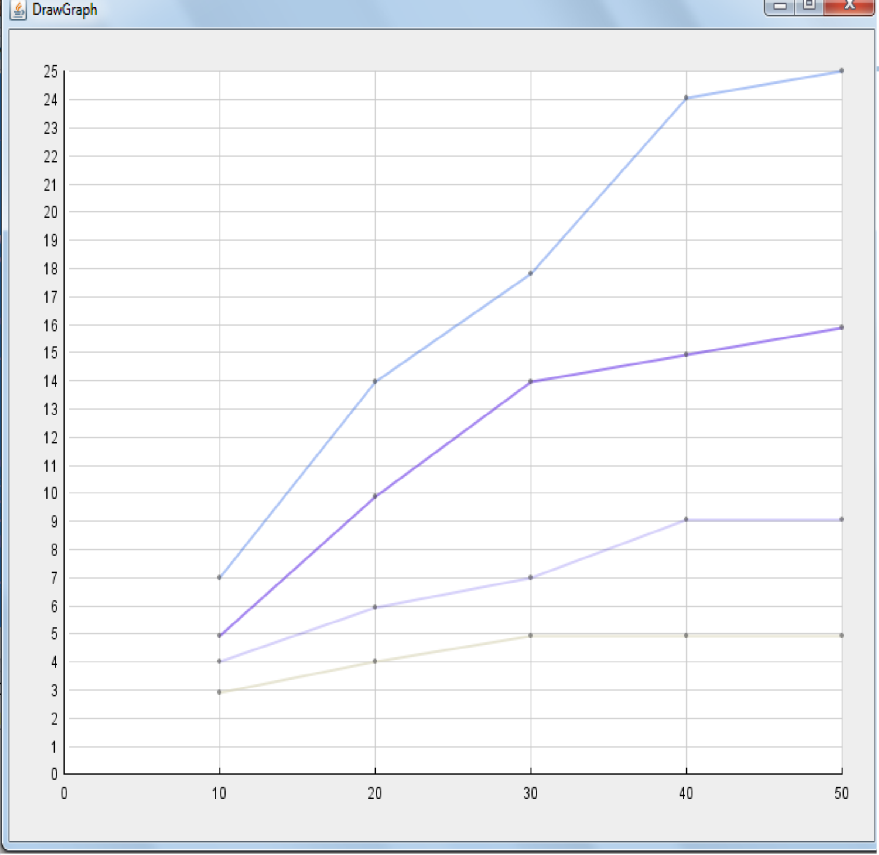


Figure 4.9 Transmission Range Graph

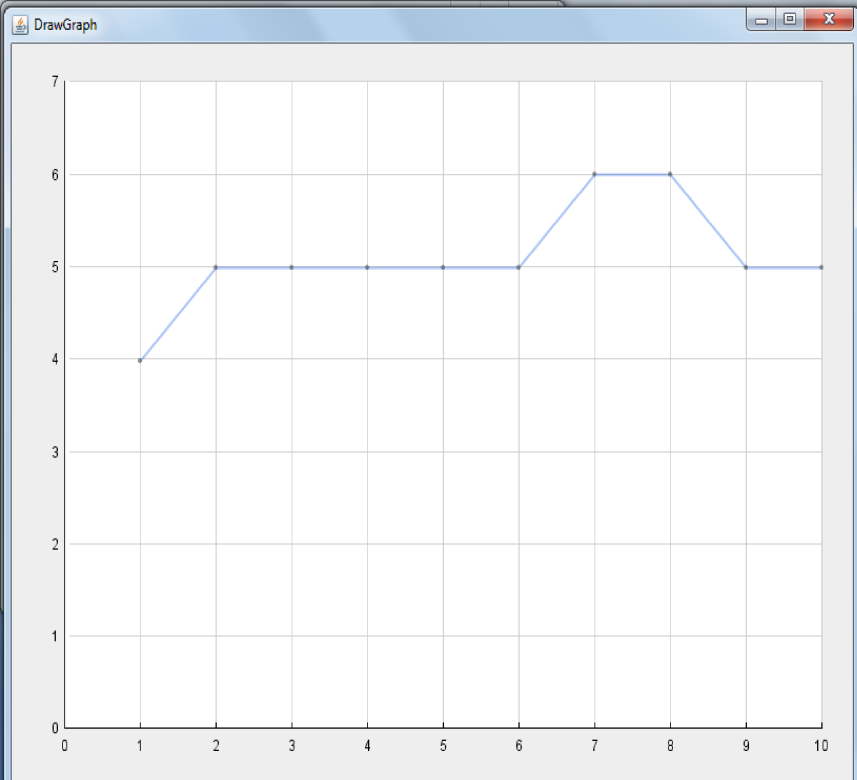


Figure 4.10 Constant Range Graph

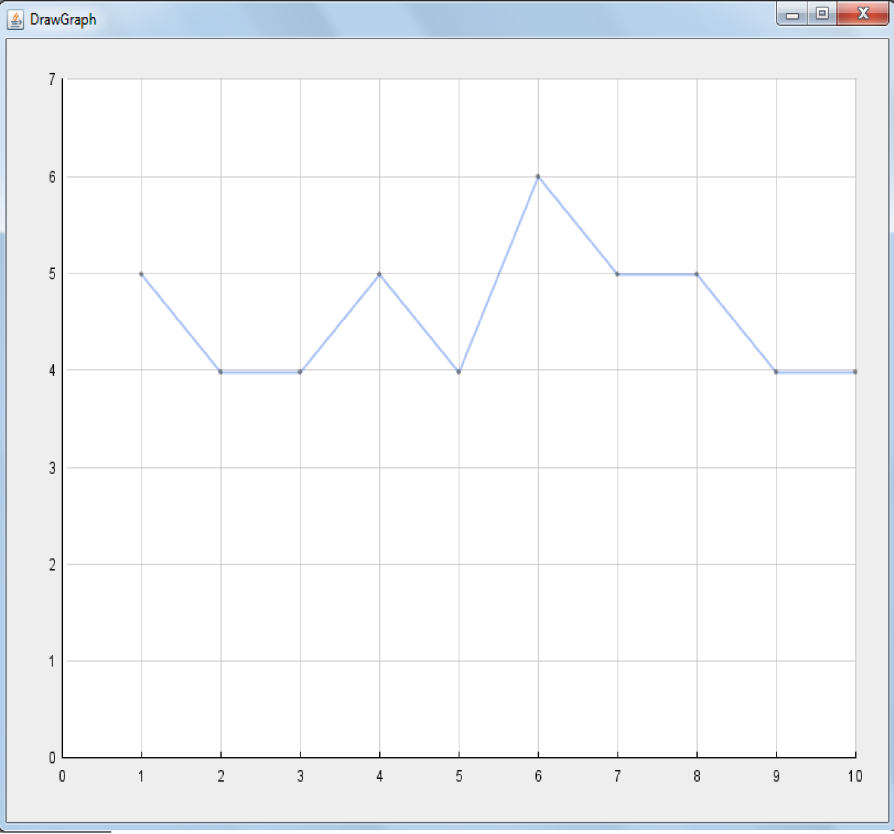


Figure 4.11 Random Range Graph

**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].getxPosition()) + 78,

((int) N[i].getyPosition()) + 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].getxPosition()) + 40,

((int) N[i].getyPosition()) + 40,

80, 80);

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

((int) N[i].getyPosition()) + 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].getxPosition()) + 78,

(int) N[j].getyPosition()) + 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 heigth = 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 5 Route Optimality and Stability

# 5.1 Definition

We define for any given node the set of multi-point relays of rank 0 as the node itself and the set fo multi-point relays of rank 1 as the multipoint relay set itself. Let us define the set of multipoint relays on rank k+1 for k integer, as the union of multipoint relay of set of all nodes element of the multipoint relay set of rank k.In other words each element Mk of the multipoint relay set of rank k of node X can be reached via a path XMi.....XMk where Mi is multipoint relay of X and Mi+1 is multipoint relay of Mi. .

# 5.2 Benefits

Being a proactive protocol, routes to all destinations within the network are known and maintained before use. Having the routes available within the standard routing table can be useful for some systems and network applications as there is no route discovery delay associated with finding a new route.

The routing overhead generated, while generally greater than that of a reactive protocol, does not increase with the number of routes being created.

Default and network routes can be injected into the system by HNA messages allowing for connection to the internet or other networks within the OLSR MANET cloud. Network routes are something reactive protocols do not currently execute well.

Timeout values and validity information is contained within the messages conveying information allowing for differing timer values to be used at differing nodes.

# 5.3 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.

# 5.4 Scalability

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 routes have been 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 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 6 Conclusion and Future Work:

# 6.1 Conclusion

The MANETS AND OLSR algorithms were thoroughly studied taking a number of parameters varying the conditions and variables. All the work mentioned above involved real time data. OLSR algorithm was implemented and was a success. The overall success rate of implementing the algorithm was 75%.

# 

# Chapter 7 Glossary:

**7.1 Acronyms**

**MANET ---------------------------------------**Mobile Ad hoc Networks

**OLSR** ----------------------------------------Optimized Link State Routing

**ID** ----------------------------------------------Identity

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

**MPR --------------------------------------------**Multipoint Relay

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

**TC ---------------------------------------------**Topology Control

**MS ---------------------------------------------**Multipoint Relay Selector Set

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

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

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

**JRE -------------------------------------------**JAVA Runtime Environment

# 

# Chapter 8 References, IEEE Format

Research Papers

* Jeroen Hoebeke, Ingrid Moerman, Bart Dhoedt and Piet Demeester, “*An Overview of Mobile Ad Hoc Networks: Applications and Challenges*”, Department of Information Technology (INTEC),Ghent University – IME vzw,Sint Pietersnieuwstraat 41, B-9000 Ghent, Belgium.
* Prithwish Basu, Naved Khan, Thomas D. C. Little, “*Optimized Link State Routing Protocol*”, Department of Electrical and Computer Engineering Boston University, Boston MA.
* Siddhu Warrier, “*Characterisation of MANET Routing Algorithms in Wireless Sensor Networks*” School of Informatics, University of Edinburgh,2007.
* Andreas Tønnesen, “*Implementing Optimized Link State Routing Protocol*”, Department of Informatics, University of Oslo.

Software

[8] Creately, for diagrams.