DESIGNING A SIMULATION APPLICATION FOR GREEDY PERIMETER STATELESS ROUTING PROTOCOL FOR AD HOC NETWORKS

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

under the Supervision of

Amol Vasudeva

By

Pranshu Srivastava ***111232***

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that the work titled "**Designing a Simulation Application for Greedy Perimeter Stateless Routing Protocol for Ad Hoc Networks"** submitted by **Pranshu Srivastava** 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.

Name of Supervisor : Mr. Amol Vasudeva

Designation : Assistant Professor

Signature of Supervisor:

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

Date: Pranshu Srivastava

Contents

List of Figures

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. Greedy Perimeter Stateless Routing (GPSR), a novel routing protocol for wireless datagram networks that uses the positions of routers and a packet's destination to make packet forwarding decisions. GPSR makes greedy forwarding decisions using only information about a router's immediate neighbours in the network topology. When a packet reaches a region where greedy forwarding is impossible, the algorithm recovers by routing around the perimeter of the region. By keeping state only about the local topology, GPSR scales better in per-router state than shortest-path and ad-hoc routing protocols as the number of network destinations increases. Under mobility's frequent topology changes, GPSR can use local topology information to find correct new routes quickly. We describe the GPSR protocol, and use extensive simulation of mobile wireless networks to compare its performance with that of Dynamic Source Routing. Our simulations demonstrate GPSR's scalability on densely deployed wireless networks.

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 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.

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.

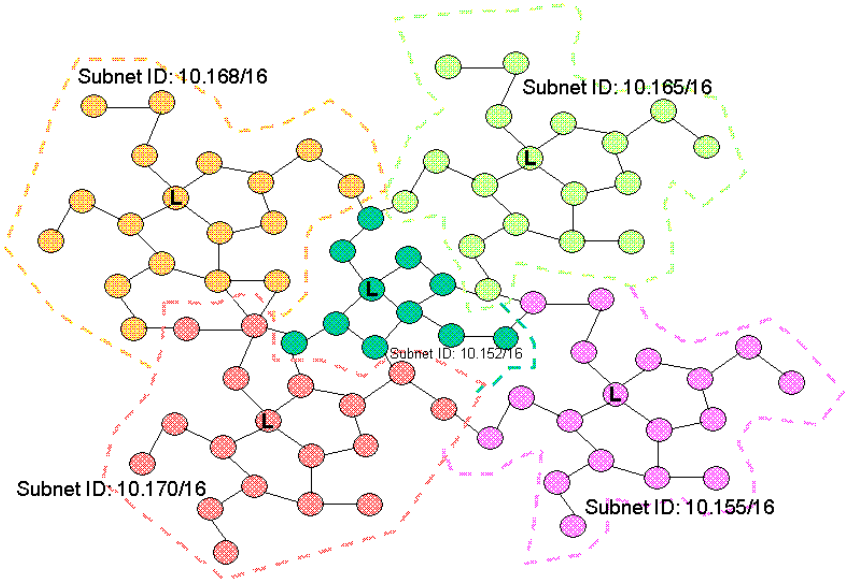


Figure 1: Example of MANET

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, address 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 employee 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; 2π] 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 , 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.

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.

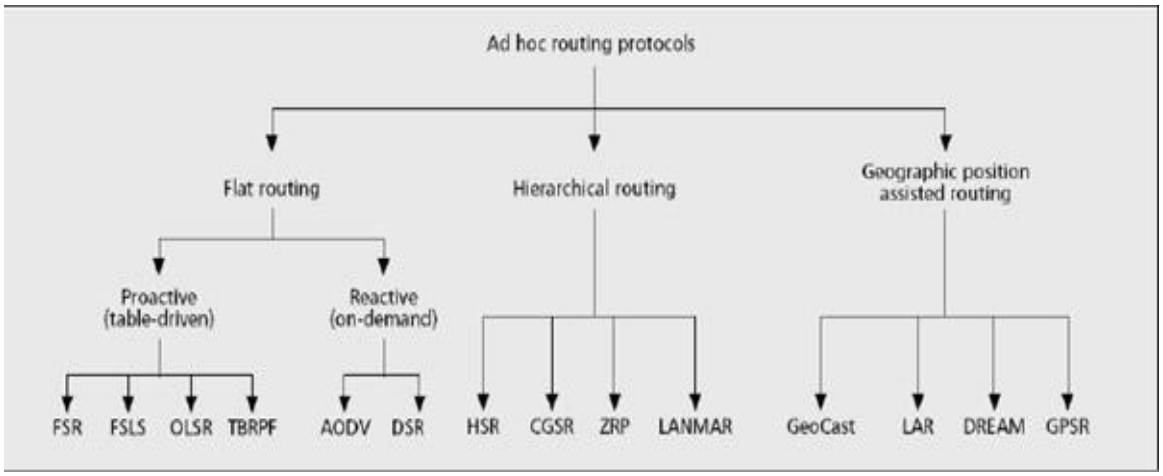


Figure 2: Routing Algorithms

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.

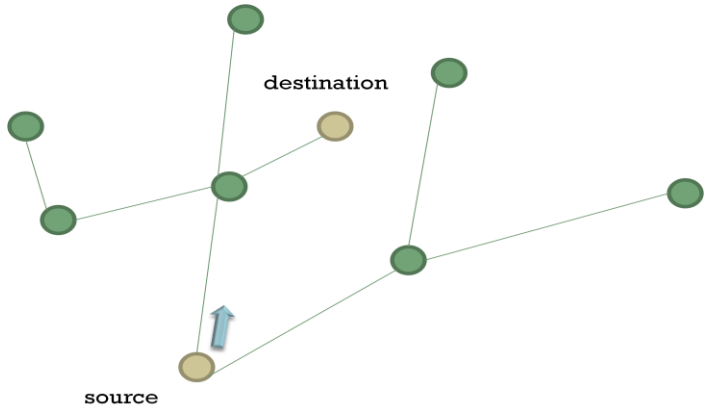


Figure 3: Flat Routing

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. Routing information is distributed between nodes by sending ***full dumps*** infrequently and smaller incremental updates more frequently.

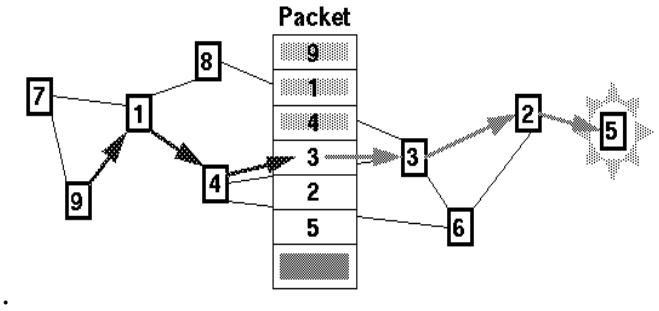


Figure 4: Destination-Sequenced Distance-Vector Routing

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.

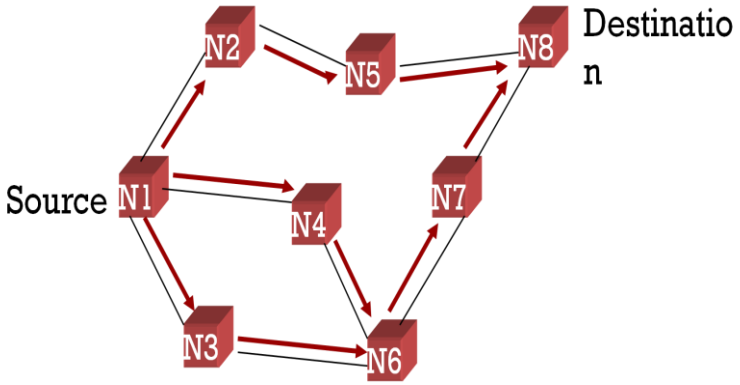


Figure 5 (a) Ad hoc On-Demand Distance Vector Routing- source

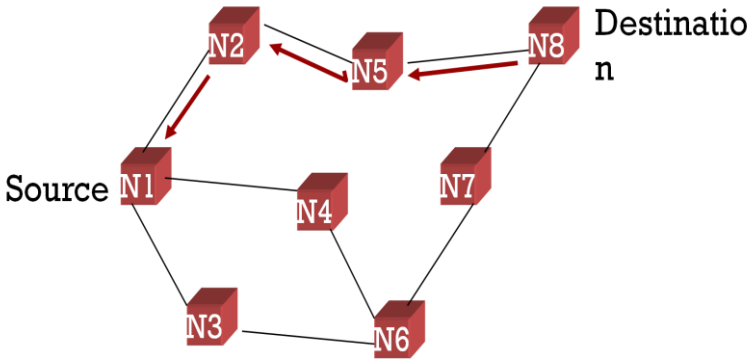


Figure 6 (b) Ad hoc On-Demand Distance Vector Routing- Destination

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 a node is within the same cluster then the information is directly passed else 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.

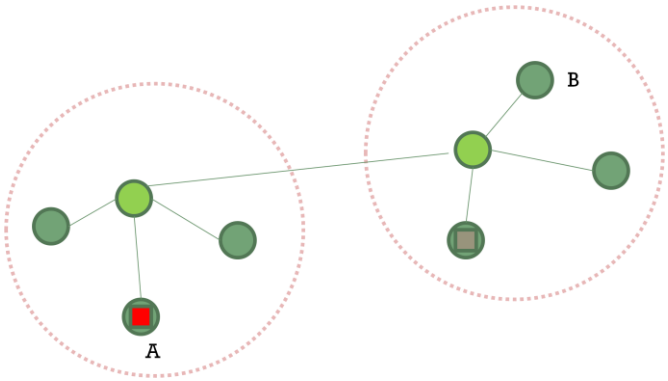


Figure 7 Hierarchical Routing

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.

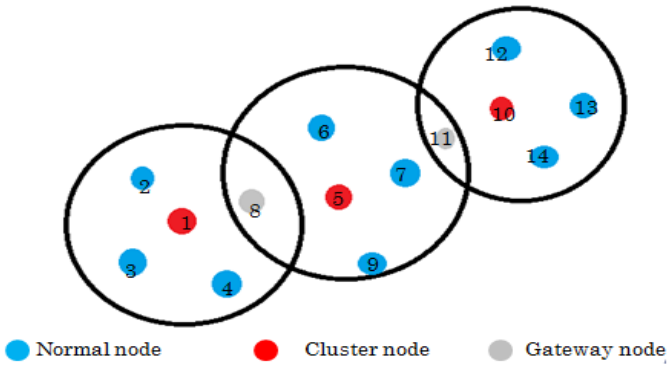


Figure 8 Lowest ID

3.2.6 Highest Degree:

In comparison with Lowest-ID scheme, the degree of nodes is computed based on its distance from each other's . 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.

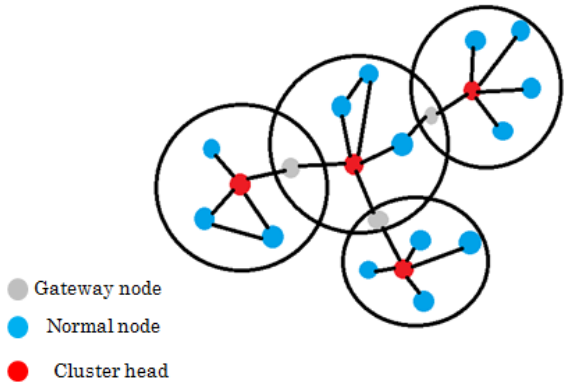


Figure 9: Highest Degree

3.2.7 Greedy Perimeter Stateless Routing (GPSR):

GPSR, a novel routing protocol for wireless datagram networks that uses the positions of routers and a packet's destination to make packet forwarding decisions. GPSR makes greedy forwarding decisions using only information about a router's immediate neighbors in the network topology. When a packet reaches a region where greedy forwarding is impossible, the algorithm recovers by routing around the perimeter of the region.

3.2.7.1 Working of Greedy Algorithm:

1. Nodes are arranged Randomly in a simulation area. All the nodes are static and equipped with some localization mechanism such as GPS. We assume that information i.e. ID, location of all the nodes in the network are maintained by each node. This algorithm is suitable is number of nodes are less i.e. Information table is maintained by every node.

2. Assume Node “s” want to send message to Destination Node “D”. S will find distance between S and D using location of D from information table. If node D is within radio range of S, then it will send packet directly.
3. Else S will find nodes that are within radio range of S using Information Table. Make those nodes neighboring nodes of S.
4. Find distance of every neighboring node from D, using information table. Select node that has smallest distance among S and other neighboring nodes and send message to it.
5. If S is local minimum i.e. closest to D, then apply Perimeter Algorithm.

3.2.7.2 Flowchart of Greedy Algorithm:

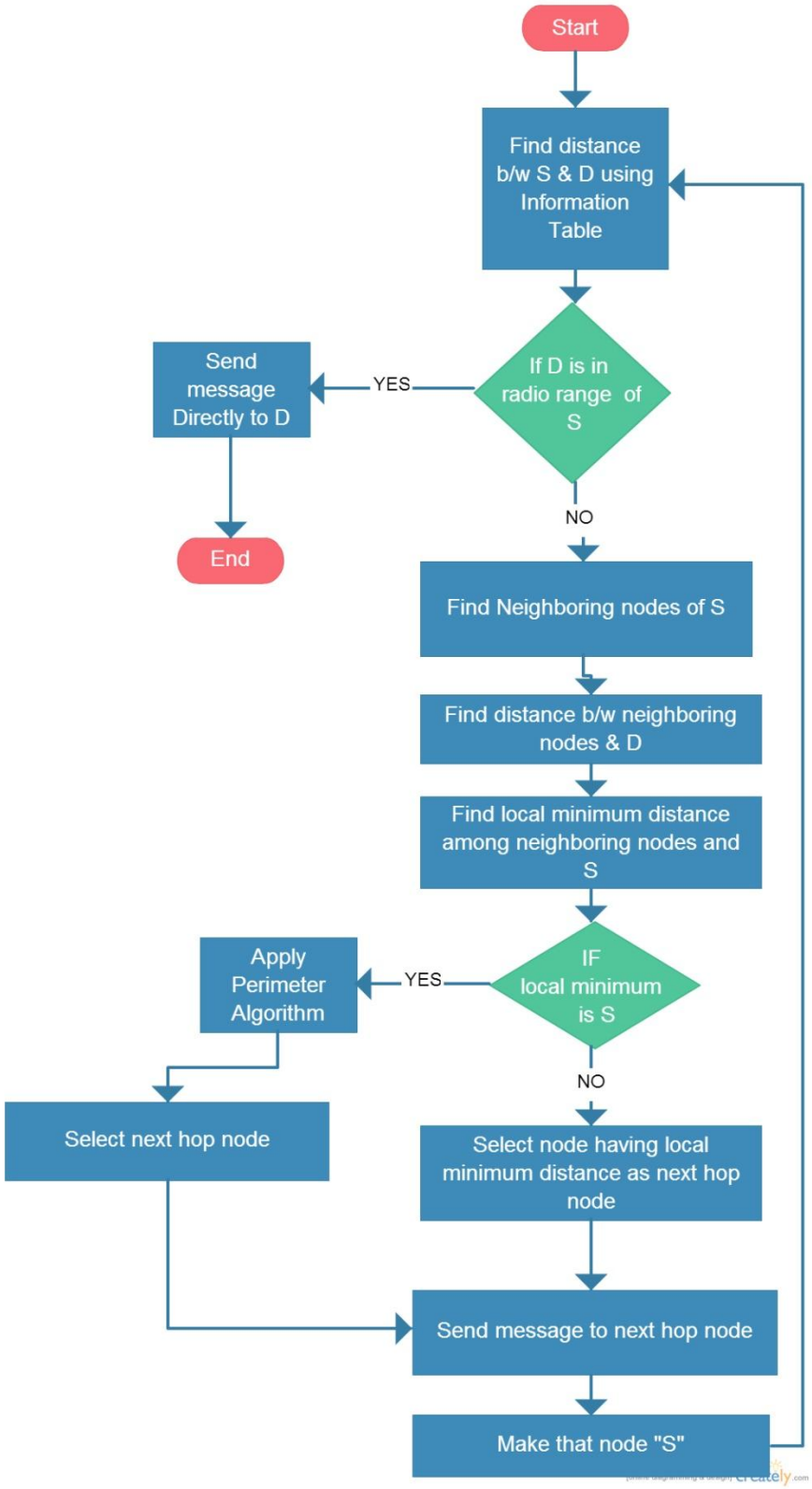


Figure 10: Flowchrt of Greedy Algorithm

3.2.7.3 Working of Perimeter Algorithm

1. After greedy algorithm has been failed, Perimeter algorithm sends data packet through the perimeter of the void created while Greedy algorithm's failure.
2. Select the node as Source "S" on which Greedy algorithm failed.
3. Search for the node that is closest to destination among neighboring nodes which can further forward data packets.
4. If any such node is present make that node as Next Hop node and send packet to that Next Hop node.
5. Else whole GPSR algorithms fails, and packet cannot be send using GPSR algorithm.

3.2.7.4 Flowchart of Perimeter Algorithm

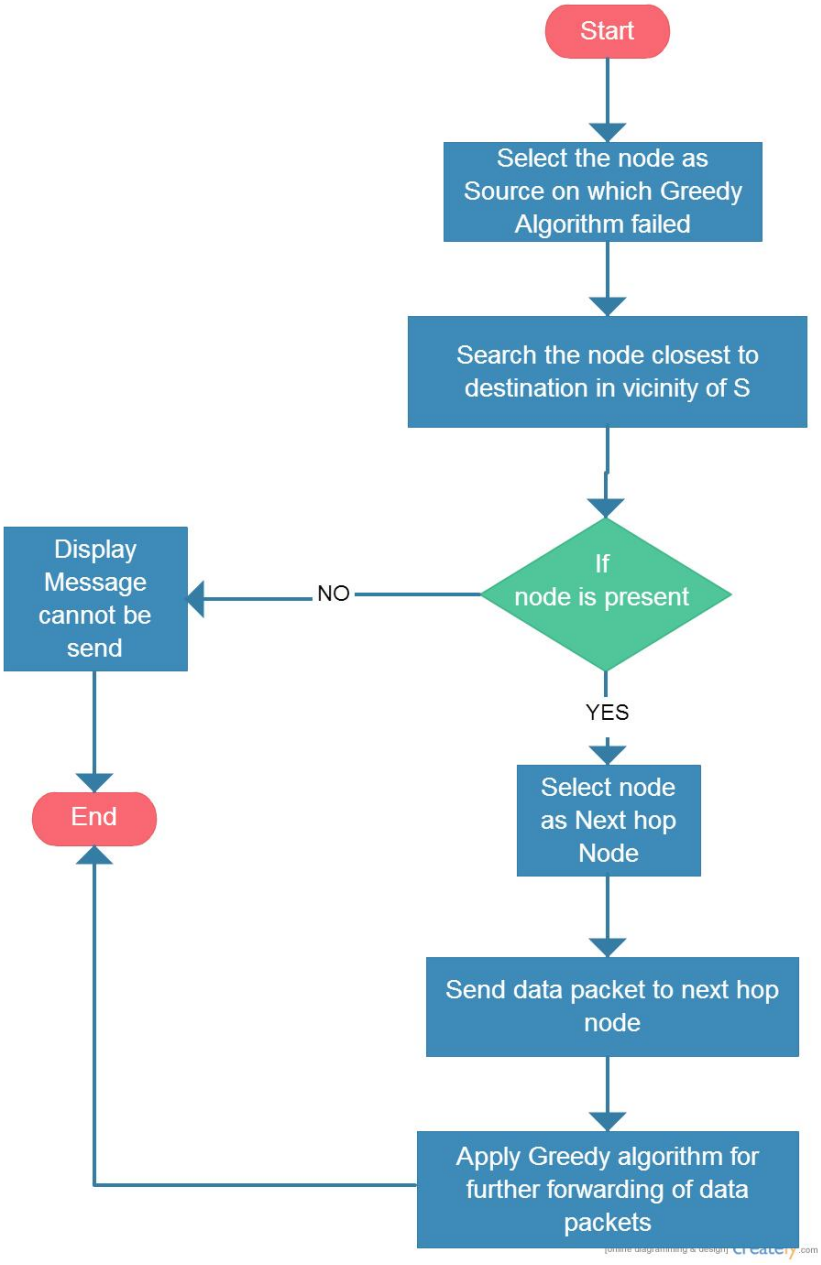


Figure 11: Flowchart of Perimeter Algorithm

3.2.7.5 Flowchart of GPSR

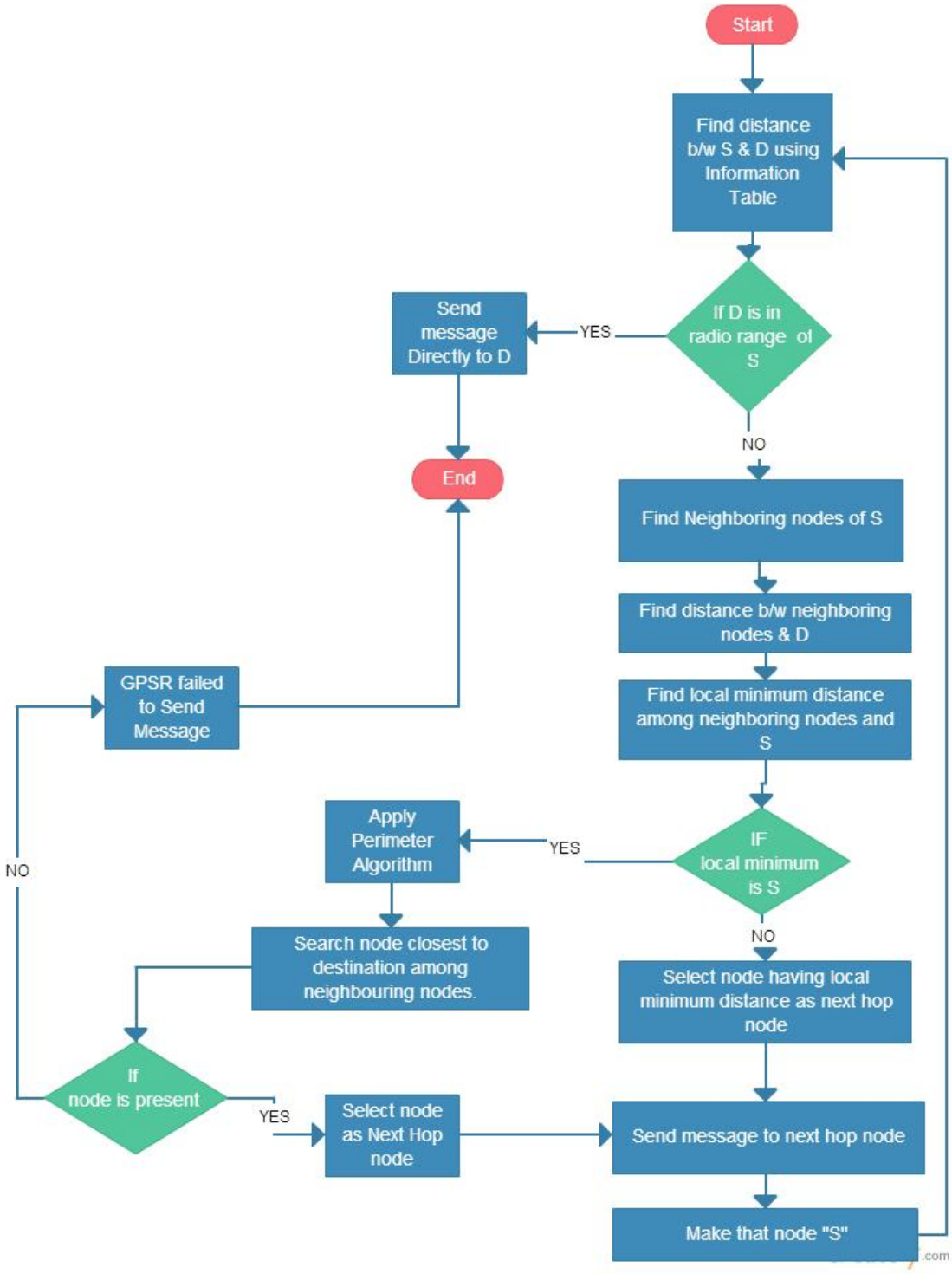


Figure 12: Flowchart of GPSR

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 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 programme is hardware and operating system independant. 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 flavors, 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.

4.2.1.2 JAVA Virtual Machine (JVM):

A JAVA programme is build by a JAVA compiler which generates its own binary code. This binary code is independant 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.

4.2.1.3 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.

4.2.1.4 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

4.3 Diagrams

4.3.1 Use Case Diagram

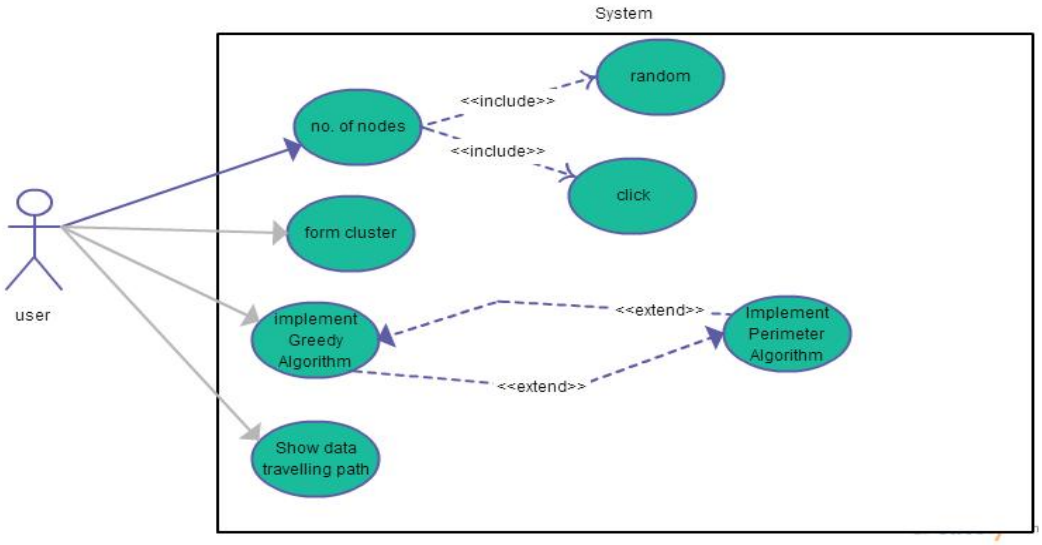


Figure 13: Use case Diagram

4.3.2 Data Flow Diagram

4.3.2.1 Zero Level DFD

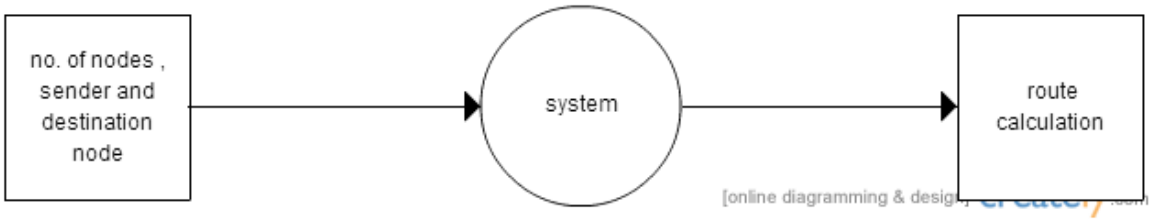


Figure 14 (a) Zero Level DFD

4.3.2.2 First Level DFD

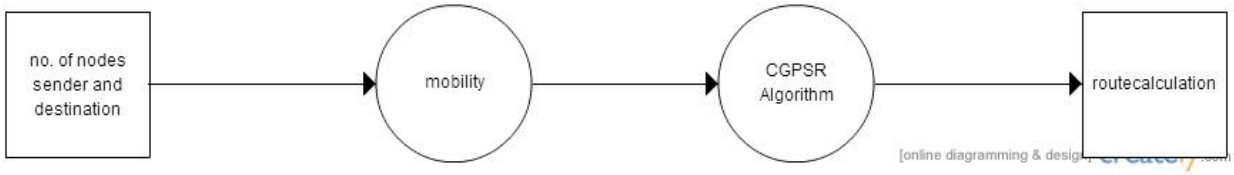


Figure 15 (b) First Level DFD

4.3.2.3 Second Level DFD

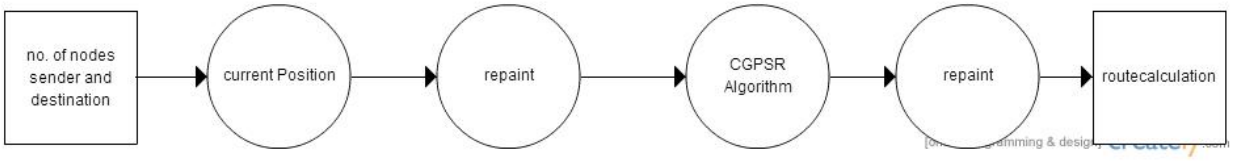


Figure 16 (c) Second Level DFD

4.4Code

```
package prototype1gpsr;

import java.util.*;
import java.applet.*;
import java.awt.Button;
import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.awt.Label;
import java.awt.Point;
import java.awt.TextField;
import java.awt.event.*;
import java.awt.*;
import java.awt.geom.*;

public class Prototype1GPSR1 extends Applet implements Runnable,
ActionListener,MouseListener
{
//int[] result;
int pc=0,visitd=0,faild=0,faildNode;
double phi;
private Graphics globalGraphics=null;
int barb;
Point sw,ne;
Node[] p;
```

```
Thread th;
int click=0;
ArrayList<Node> nodeList = new ArrayList<Node>();
int numNodes;
int srcNode,destNode;
int visited[];
int path[];
int path1[];
int pathIndex=0;
int adj[][];
double dist=0;
int SimSize=500;
String greedy,msg;
        double range = 200.0;
        TextField source,destination,nodes,simsize;
        Label l2,l3,l1,l4,l5,l6;
        Button resume, button, button1, button2, button3,  pause;
        int flag = 0,flag1=1, suspend = 0;


public void init() {
    resize(1000,1000);
        this.addMouseListener(this);
        p = new Node[25];

        phi = Math.toRadians(40);

        nodes = new TextField(8);
        source = new TextField(8);
        destination = new TextField(8);
        simsize = new TextField(8);
```

```
l1 = new Label("                    Number of Nodes:   ");
l2 = new Label("       Enter Source Node Id:   ");
l3 = new Label("                    Enter Destination Node ID:   ");
l4= new Label("Enter Size of Simulator");
greedy="Greedy Algorithm failed at  ";
msg="Message can't be Send  ";


l2.setForeground(Color.red);

add(l2);

add(source);

      l3.setForeground(Color.green);
add(l3);


add(destination);
       add(l1);


     add(nodes);
button = new Button("Submit");

button.addActionListener(this);

add(button);
```

```
    button2=new Button("Random");

   button2.addActionListener(this);

   add(button2);

   l3.setForeground(Color.black);
   add(l4);
   add(simsize);

  p=new Node[30];
   }

public void run() {

   while (true) {

          if (suspend == 0)
      {
         for (int i = 0; i < numNodes; i++)
        {
                  repaint();


        }
      }

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

```
    }

}


public void actionPerformed(ActionEvent ae)
{

   try {

      if (ae.getSource() == button)
       {

   flag = 1;

   srcNode=Integer.parseInt(source.getText());
   destNode=Integer.parseInt(destination.getText());
    numNodes = click;

    nodes.setText(String.valueOf(click));
    System.out.println("number of nodes   :   "+numNodes);
    visited=new int[numNodes];
    for(int i=0;i<numNodes;i++)
          visited[i]=0;
    visited[srcNode]=1;



         nodeList = new ArrayList<>();
         if( (srcNode < numNodes) && (destNode < numNodes) )
         {
```

```
            //adj=new ArrayList<Integer[]>();
            adj = new int[numNodes][];
            for (int i = 0; i < numNodes; i++)
            {
               adj[i] = new int[numNodes];

            }
            path = new int[numNodes];


            th = new Thread(this);
            th.start();
         }


     }
    if((ae.getSource() == button2))
    {

    Random rn=new Random();
       numNodes=Integer.parseInt(nodes.getText());

flag = 1;


srcNode=rn.nextInt(numNodes);

System.out.println("Source"+ srcNode);

System.out.println("Dest"+ destNode);
source.setText(String.valueOf(srcNode));
```

```
SimSize=Integer.parseInt(simsize.getText());
range=0.4*SimSize;
System.out.println("SimSize"+ SimSize);

destNode=rn.nextInt((numNodes));
while(destNode==srcNode)
{
   destNode=rn.nextInt((numNodes));
}
 destination.setText(String.valueOf(destNode));
  for(int i=0;i<numNodes;i++)
 {
   int x,y;
   Random r=new Random();
   x=80+r.nextInt(SimSize);
   y=80+r.nextInt(SimSize);
   p[click]=new Node(x,y);
   click++;
 }
  nodeList = new ArrayList<>();
       if( (srcNode < numNodes) && (destNode < numNodes) )
       {

          adj = new int[numNodes][];
          for (int i = 0; i < numNodes; i++)
          {
             adj[i] = new int[numNodes];

          }
          path = new int[numNodes];
```

```
                th = new Thread(this);
               th.start();
              }

          }

              }
        catch (Exception e) {

        }

       }



public void paint(Graphics g)
{

   numNodes=click;
 globalGraphics=g.create();
  Graphics2D g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALU
E_ANTIALIAS_ON);
  g.drawRect(80, 80, SimSize, SimSize);
  String no="";

 if (flag > 0)
```

```
{
   if (suspend == 0)
            {

           g.setColor(Color.black);
           for (int i = 0; i < numNodes; i++)
               {
                   no=""+i;
               if (i==srcNode)
                        g.setColor(Color.red);

               else if(i==destNode)
                        g.setColor(Color.GREEN);
               else
                        g.setColor(Color.black);


                    nodeList.add(i,p[i]);
                    //System.out.print("x "+nodeList.get(i).getx());
                    //System.out.println("y "+nodeList.get(i).gety());

                    g.fillOval(p[i].getx(),p[i].gety(), 8, 8);
                    g.drawString(no, p[i].getx()-2, p[i].gety());
               }


    if(flag1==1)
    {
       flag1=0;
       adjacencyMatrix();
```

```
        greedy(srcNode,destNode);
    }
     if(faild!=1)
    { for(int k=visitd-1;k>0;k--)
      {

        Point sw;
          sw = new Point((nodeList.get(path1[k]).getx())+4,(nodeList.get(path1[k]).gety())+4);
        Point ne;
          ne = new Point((nodeList.get(path1[k-1]).getx())+4,(nodeList.get(path1[k-1]).gety())+4);
        g2.draw(new Line2D.Double(sw,ne));

      }
    }
    else
    {
     for(int k=visitd-1;k>0;k--)
      {
        if(path1[k]==faildNode)
          break;
          Point sw;
          sw = new Point((nodeList.get(path1[k]).getx())+4,(nodeList.get(path1[k]).gety())+4);
        Point ne;
          ne = new Point((nodeList.get(path1[k-1]).getx())+4,(nodeList.get(path1[k-1]).gety())+4);
        g2.draw(new Line2D.Double(sw,ne));

      }
```

```
        th.stop();
    }

    if(faild!=1)
    { for(int k=visitd-1;k>0;k--)
      {

        Point sw;
         sw = new Point((nodeList.get(path1[k]).getx())+4,(nodeList.get(path1[k]).gety())+4);
        Point ne;
         ne = new Point((nodeList.get(path1[k-1]).getx())+4,(nodeList.get(path1[k-1]).gety())+4);
        g2.draw(new Line2D.Double(sw,ne));
      }
    }
    else
    {
     for(int k=visitd-1;k>0;k--)
      {
        if(path1[k]==faildNode)
          break;
          Point sw;
         sw = new Point((nodeList.get(path1[k]).getx())+4,(nodeList.get(path1[k]).gety())+4);
        Point ne;
         ne = new Point((nodeList.get(path1[k-1]).getx())+4,(nodeList.get(path1[k-1]).gety())+4);
        g2.draw(new Line2D.Double(sw,ne));

      }
```

```
        th.stop();
      }
   }}}



public void adjacencyMatrix()
{
       System.out.print("AdjacencyMatrix:      ");
       int a;
       for( a=0;a<numNodes;a++)
          System.out.print(a+" ");
       System.out.println("");
       for(int i=0;i<numNodes;i++){

          for(int j=0;j<numNodes;j++){
             adj[i][j]=0;
          }
          for(int j=0;j<numNodes;j++){
             if( (i!=j) && (nodeList.get(i).distance(nodeList.get(j)))<range){
                adj[i][j]=1;
             }
          }

       }
      for(int i=0;i<numNodes;i++)
      {
         System.out.print("Adjacency for node "+i+": ");
      for(int b=0;b<numNodes;b++)
                System.out.print(adj[i][b]+" ");
                System.out.println();
```

```
    }

  }


public void greedy(int strtNode, int desNode)
  {

    int presNode=strtNode,nxtNode=strtNode,prevNode=0,flag=0;
    double dist1=0;
    int node1;
    path1=new int[numNodes+1];
    path1[visitd++]=presNode;

    while(nxtNode!=desNode)
    {
      presNode=nxtNode;
      dist=nodeList.get(presNode).distance(nodeList.get(desNode));
      System.out.print("The PRESENT NODE is "+presNode+"\n");
      System.out.println("The DISTANCE of PRESENT NODE "+presNode+" is
"+dist);
      if(flag==0)
      {
        for(int i=0;i<numNodes;i++)
        {

          if(adj[presNode][desNode]!=1)//dest node not in range
          {
            if(presNode!=i)
            {
```

```
                    if(adj[presNode][i]==1)
                    {

                       dist1=nodeList.get(i).distance(nodeList.get(desNode));
                       System.out.println("The DISTANCE of nxtnode "+i+"
is"+dist1+"\n");
                       if(dist1<dist)
                       {
                          dist=dist1;
                          nxtNode=i;
                          System.out.println("The TEMPORARY NEXTNODE is
"+i+"\n");
                       }

                    }
                 }
              }
              else
              {
                 nxtNode=desNode;
              }

           }
        }
        else       //              for perimeter use
        {
           for(int i=0;i<numNodes;i++)
           {
                if(adj[presNode][i]==1 && i!=prevNode)
                {
                   dist1=nodeList.get(i).distance(nodeList.get(desNode));
```

```
                    System.out.println("The DISTANCE of nxtnode "+i+" is
"+dist1+"\n");
                    if(dist1<dist)
                    {
                       dist=dist1;
                       nxtNode=i;
                       System.out.println("the TEMPORARY NEXTNODE IS
"+i+"\n");
                    }
                 }
            }
            flag=0;
         }
         if(presNode==nxtNode)
         {
            flag=1;
            prevNode=presNode;
            System.out.println("GREEDY ALGORITHM HAS FAILED");
            greedy=greedy+ presNode +" " ;
            grdymsg();
            nxtNode=perimeter(presNode,desNode);
            if(nxtNode==presNode)
            {
              // exit(0);
            }
         }
         path1[visitd++]=nxtNode;
         System.out.print("The PERMANENT NEXTNODE IS "+nxtNode+"\n");
      }
        System.out.print("The Path is ");
```

```
        int j;
         for( j=0;j<visitd-1;j++)
        {

           System.out.print(path1[j]+"->");
        }
        System.out.println(path1[visitd-1]);



    }

//       for perimeter use
public int perimeter(int strtNode,int desNode)
    {
        int nxtNode=strtNode,a=0;
        double dist1=0,dist2;
        for(a=0;a<numNodes;a++)
        {
            if(adj[strtNode][a]==1)
            {
                dist1=nodeList.get(a).distance(nodeList.get(desNode));
                nxtNode=a;
                break;
            }
        }
        // wrong a might not be neighbour
        for(int i=a+1;i<numNodes;i++)
        {
            if(adj[strtNode][i]==1)
            {
                if(strtNode!=i)
```

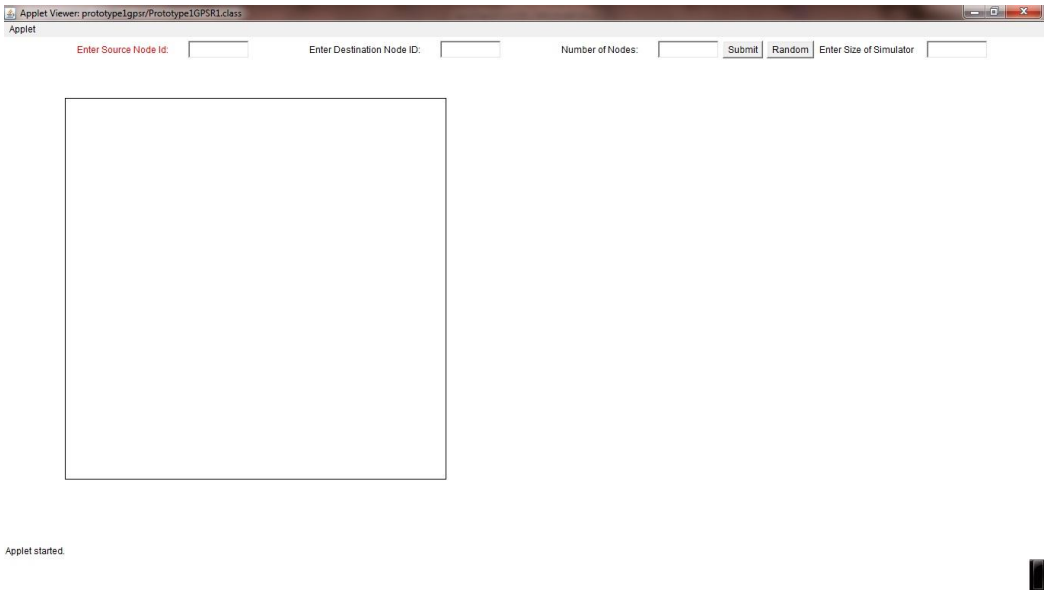
```
            {
                dist2=nodeList.get(i).distance(nodeList.get(desNode));
                if(dist1>dist2)
                {
                    dist1=dist2;
                    nxtNode=i;
                }
            }
          }
        }
        for(int i=0;i<numNodes;i++)
        {

          {
            adj[i][strtNode]=2;
          }
        }
    if(nxtNode==strtNode)
    {
        System.out.println("MESSAGE CAN NOT BE DELIVERED");
        msg=msg+nxtNode +" ";
        msgcnt();
        faild=1;
        faildNode=strtNode;

    }
    return nxtNode;
    }
public void grdymsg()
{
```

```
    l5=new Label(greedy+" ");
    l5.setForeground(Color.red);
    add(l5);
}
public void msgcnt()
{

    l6=new Label(msg+" ");
    l6.setForeground(Color.red);
    add(l6);
}
public void drawDot(int x,int y)
{
  String clic=""+click;
    Color blackColor = new Color(0,0,0);
  globalGraphics.setColor(blackColor);
  globalGraphics.fillOval(x,y,8,8);
  globalGraphics.drawString(clic, x, y);
}

@Override
public void mouseClicked(MouseEvent e)
{}
@Override
public void mouseEntered(MouseEvent arg0)
{}
@Override
public void mouseExited(MouseEvent arg0)
{}
@Override
public void mousePressed(MouseEvent e)
```

```
{
    int mouseX=e.getX();
  int mouseY = e.getY();
  if(mouseY>80 && mouseY < (500+80) && mouseX >80 && mouseX <(500+80))
{

 drawDot(mouseX,mouseY);
p[click]=new Node(mouseX,mouseY);

 click++;
}
}

@Override
public void mouseReleased(MouseEvent e)
{}
}
 class Node {
    int x;
    int y;
    Node()
    {
        x=0;
        y=0;
    }
    Node(int x,int y)
    {
        this.y=y;
        this.x=x;

    }
```

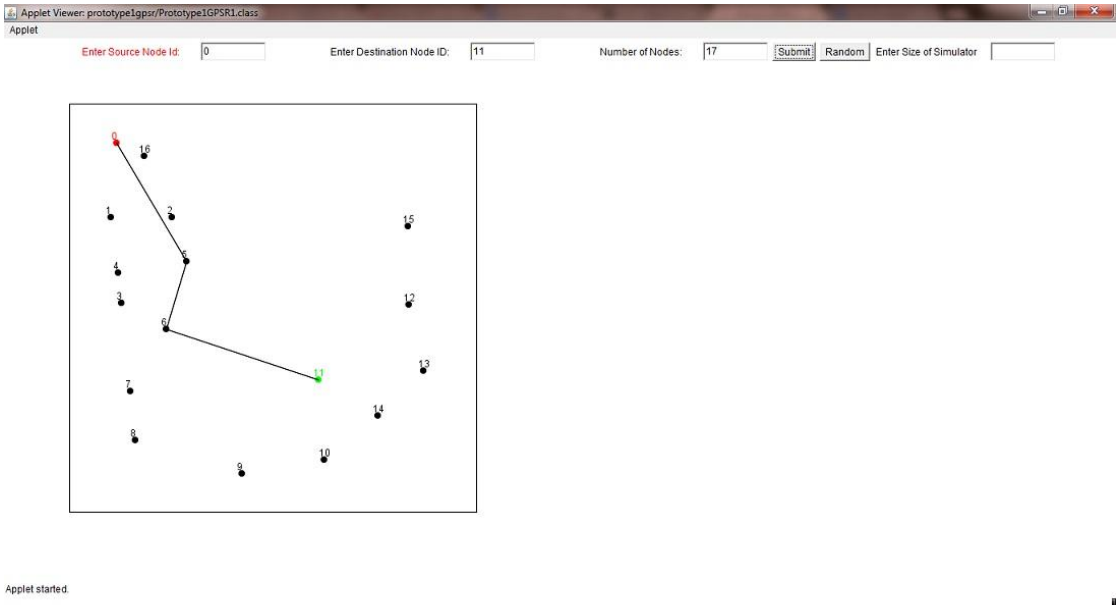
```
    public int getx(){  return x;    }
    public int gety(){  return y;    }
    public double distance(Node a)
    {
       double dist;
       int x2=a.getx();
       int y2=a.gety();
       int x3,y3;
       // shifting nodes coordinate to center of node
       x2=x2+4;
       y2=y2+4;
       x3=x+4;
       y3=y+4;
       dist=Math.sqrt( ((y2-y3)*(y2-y3))+((x2-x3)*(x2-x3)) );
       return dist;
    }
}
```

4.5 ScreenShots

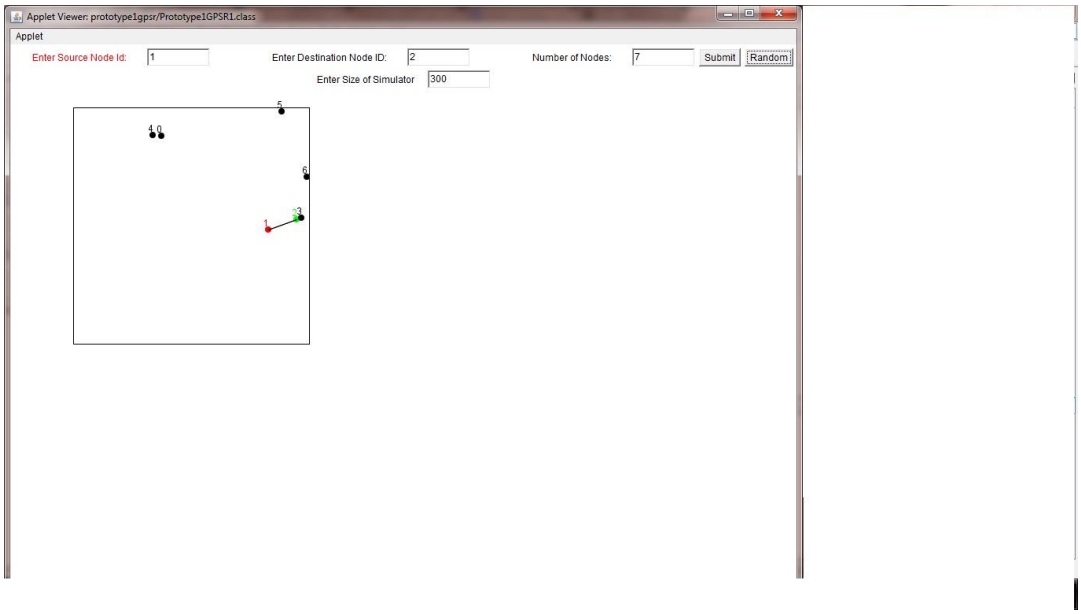


UI Interface of Simulator

Manually Entering Of Nodes



Path of Packet Transfer



Random Allocation of Nodes along with fixing simulation area

4.6 Working of Simulator

4.6.1 Manual Simulation

- Allocate the nodes on simulation area by clicking.

- Enter Source Node description.
- Enter Destination Node description.
- Click on Submit button.

4.6.2 Random Simulation

- Enter number of nodes required for simulation.
- Enter size of Simulation Area (default size 500*500).
- Click on Random button.

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 for 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 GPSR ad-hoc 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 ad-hoc networks 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)
- 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. 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

A great deal of time and effort was put forth in developing the network simulation software discussed. Simulator can be used for further research work of transmission of data packets in wireless ad-hoc network. This is a versatile tool that can be utilized for purposes outside of this work. The program was coded in a manner that makes it scalable for other network analysis. Additional network metrics and clustering methods can be integrated with little modification to the existing code. The UI provides the means for user input and simulation results are displayed on simulator itself. Further little modification in existing code can also be done for using this simulator software for MANETS.

Chapter 7 Glossary:

7.1 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

GPSR----------Greedy Perimeter Stateless Routing

Chapter 8 References, IEEE Format

Book

[1] Herbert Schildt, Complete Reference JAVA, 5th 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.

[8] http://www0.cs.ucl.ac.uk/staff/B.Karp/gpsr/gpsr.html

[9] Karp Brad, and Hsiang-Tsung Kung. "GPSR: Greedy perimeter stateless routing for wireless networks." In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 243-254. ACM, 2000.

Software

[10] Creately, for diagrams.