EFFICIENT MULTICASTING IN WIRELESS MESH NETWORKS USING STEINER TREES

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

Bachelor of Technology

in

Computer Science & Engineering

under the Supervision of

Dr. Hemraj Saini

by

Ritika Bhatnagar
111248

to

Jaypee University of Information Technology

Waknaghat, Solan – 173234, Himachal Pradesh

CERTIFICATE

This is to certify that project report entitled "**Efficient multicasting in wireless mesh networks using Steiner trees**", submitted by **Ritika Bhatnagar (111248)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: **Dr. Hemraj Saini**

Assistant Professor (Senior Grade)

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 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 Dr. Hemraj Saini, whose help, stimulating suggestions and encouragement helped me in all the aspects of my 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.

I am also grateful to Mr.Amit Singh (CSE Project lab) for his practical help and guidance.

Date: Ritika Bhatnagar (111248)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Wireless Mesh Networks (WMNs) form a new class of networks that has emerged recently. In a WMN, only some of the nodes are connected through a wire to the internet. These nodes or wireless mesh routers are the access points which then wirelessly share their internet with other nearby wireless mesh routers which have been installed on rooftops of buildings or towers. These wireless mesh routers then in turn share their internet with wireless mesh routers that are nearby to them. In this way, the mesh network is set up. Any device that wants to connect with the network can do so by connecting to the device that is nearest to it.

In a WMN, the wireless mesh routers are stationary as opposed to routers in Mobile Ad-hoc Networks (MANETs) that are mobile and hence change their topology continuously. Wireless mesh routers form the wireless mesh backbone, which provides multi-hop connectivity for mobile mesh hosts to communicate with either other mesh hosts or the Internet via Access Points.

WMNs offer low-cost, easy-deployed, location independent and self-configured networks. In WMNs, multicast is an efficient way to distribute data to a group of receivers.

The multicast routing protocols for WMNs have not been developed much and not much work has been done in this area. The multicast routing protocols used by mobile ad-hoc networks or MANETs cannot be used for multicast routing in wireless mesh networks as in WMNs the nodes are static while in MANETs the nodes are mobile. This difference in the behaviour of routers in the two types of networks makes the use of multicast routing protocols for MANETs for multicasting in wireless mesh networks an inefficient choice.

So, protocols need to be designed such that they meet the needs of multicasting in a wireless mesh network. This project aims to propose such a routing algorithm using Steiner trees.

Steiner trees are minimum cost trees and using Steiner trees for multicasting is an efficient way to ensure minimum number of transmissions for multicasting. A Steiner tree is created for every new multicast group.

PROBLEM STATEMENT

Multicasting is a communication technique that allows a source to transmit data to a set of recipients in an efficient manner. Therefore, the primary objective of a multicast routing protocol would be to minimize number of transmissions to conserve bandwidth.

The problem of computing multicast trees with minimal bandwidth consumption is similar to Steiner tree problem and has shown to be NP-complete. So, heuristic based algorithms are suitable to approximate such bandwidth optimal trees. This project aims to propose a multicast routing algorithm using a heuristic approach.

MOTIVATION

Multicasting is an essential technology for wireless networks as it provides efficient communication between mesh nodes in a WMN and is used in many applications and services such as video conferencing, distance education and distributed Internet gaming.

Most of the existing work on Wireless Mesh Networks (WMNs) concentrates on the issues of unicast routing. This project is focusing on multicasting in WMNs.

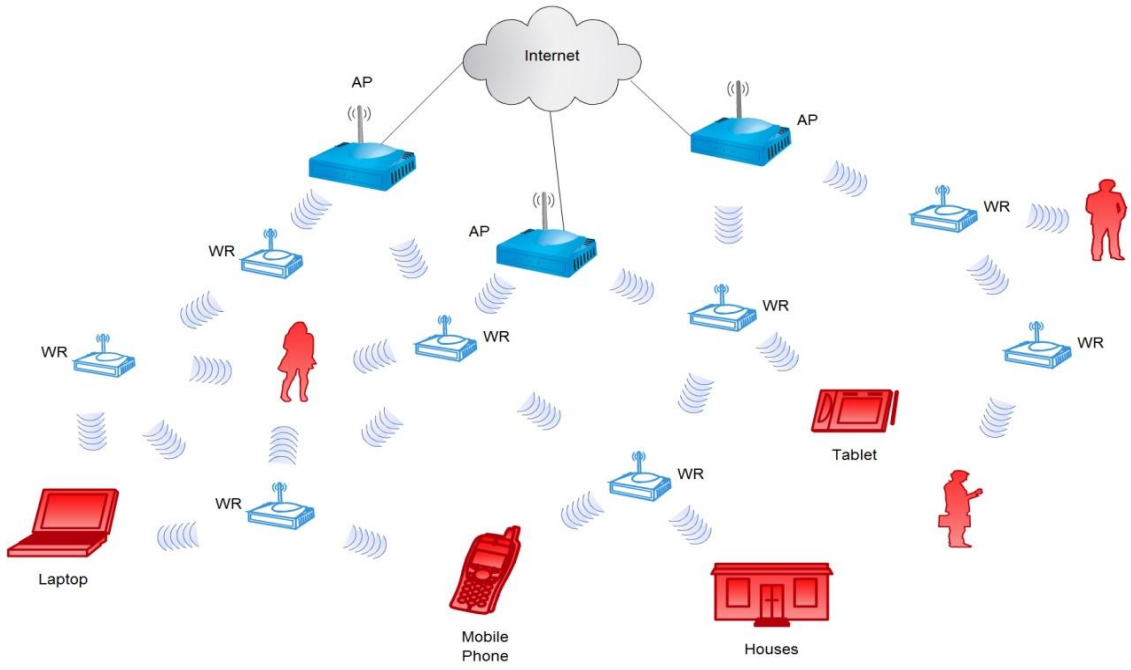
Chapter 1:

Introduction to the Topic

1.1 Introduction:

Wireless Mesh Networks (WMNs) are a communications network consisting of nodes arranged in a mesh topology. It has emerged as the new hot topic in wireless communication. [1]

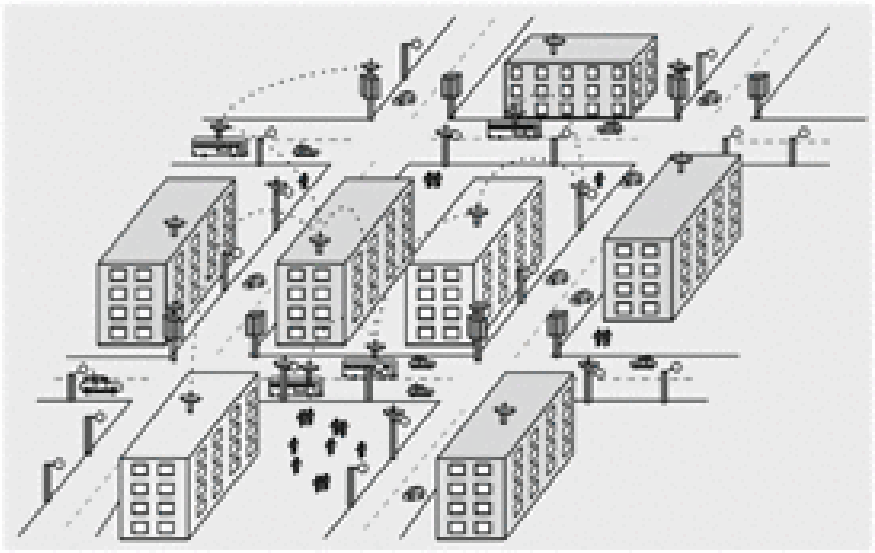
In a WMN, the network connection is spread out among dozens or even hundreds of wireless mesh nodes that communicate with each other to share the network connection across a large area. Mesh nodes are small radio transmitters that function in the same way as a wireless router. Nodes use the common Wi-Fi standards to communicate wirelessly with users and with each other. Only one node needs to be physically wired to a network connection like an Internet modem. That one wired node then shares its Internet connection wirelessly with all other nodes in its vicinity. Those nodes then share the connection wirelessly with the nodes closest to them. The more nodes, the further the connection spreads, creating a wireless "cloud of connectivity" that can serve a small office or a city of millions. [2]



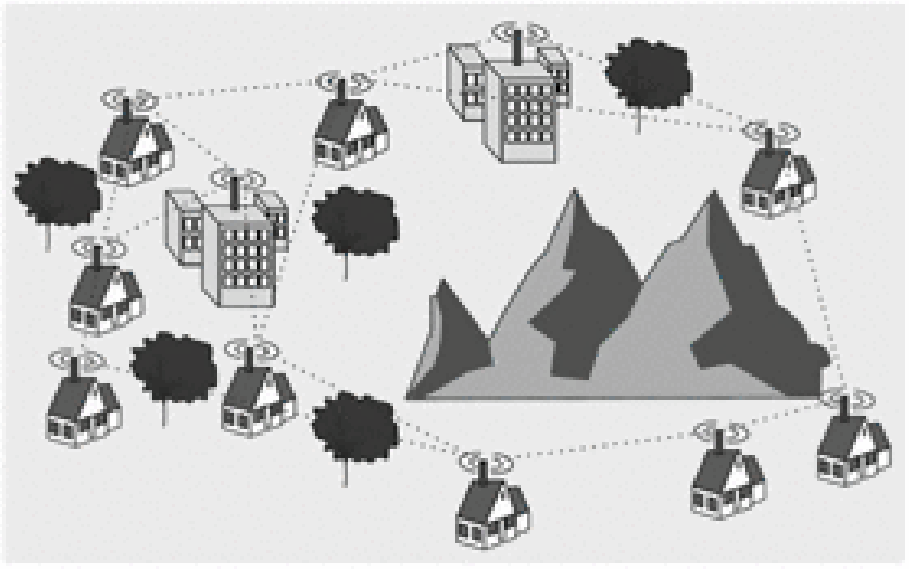
A diagram showing the working of a Wireless Mesh Network (WMN)

AP – Access Point, WR – Wireless Router

WMNs are very useful in providing internet access to remote/isolated areas, rural areas, places with a difficult and rugged terrain like mountains and valleys and temporary venues like construction sites, trade fairs, concerts and rallies, etc.



(a) in a city



(b) in rural areas

1.2 Types of WMN nodes:

WMNs are comprised of two types of nodes: **mesh routers** and **mesh clients**.

- Other than the routing capability in a conventional wireless router, a mesh router contains additional routing functions to support mesh networking. To improve the flexibility of mesh networking, a mesh router is usually equipped with multiple wireless interfaces. Mesh routers have minimal mobility.

- Although mesh clients can also work as a router for mesh networking, the hardware and software for them can be much simpler than those for mesh routers. For example, communication protocols for mesh clients can be light-weight, routing functions do not exist in mesh clients, and only a single wireless interface is needed in a mesh client. [12]

1.3 Advantages of using wireless mesh networks:

- **WMNs are less expensive than traditional wired networks**

  Using fewer wires means it costs less to set up the wireless mesh network. The wireless mesh network is mainly used for covering large areas. Using wireless mesh networks eliminate the cost of installing wires.

- **WMNs are extremely adaptable and expandable**

  As the size of the area that needs to be covered changes, wireless mesh nodes can be added or removed. WMNs are very useful for areas where the geography and topology is challenging, and where wires or towers are not

easy to lay or build. With wireless mesh networks, if we add more wireless mesh routers, the network will adjust to find a clear signal.

- **WMNs support high demand**

  Public safety and emergency response demand wireless connectivity that supports coverage of large geographic areas, and high quality video surveillance. Wireless Mesh Networks are ideal to deliver high throughput and highly reliable wireless connectivity.

  WMNs are very useful for covering large areas without sacrificing quality of the wireless network. Wireless Mesh Networks are a reliable source of wireless connectivity for a variety of uses such as parking garages, campus grounds, schools, parks, and other large outdoor facilities.

- **WMNs are stronger and faster**

  If your laptop is in the broadcast range of four nodes, you're tapping into four times the bandwidth of one traditional wireless router. This makes the network much faster than traditional networks and also fault tolerant. If one node fails the data can be transmitted using the next nearest wireless mesh router.

  Distance plays a huge role in wireless signal strength. If you reduce the distance between your computer and the nearest wireless node by two, the signal strength is four times as strong.

1.4 Disadvantages of using wireless mesh networks:

The main drawback of the technology is its complexity. The main source of this complexity is a combination between wireless technology - with its flexibility - and the unusual role of each wireless node - as simultaneously router and as well as a host.

1.5 Applications of WMNs:

- **Cities and Municipalities**

  With wireless mesh networks, cities can connect citizens and public services over a widespread high-speed wireless connection. A whole city or municipality can be connected wirelessly.
  People can check their emails and surf the internet while on the train, in their cars, in parks - anywhere and everywhere.

- **Isolated locations and rugged terrain**

  Even in developed countries, there are rugged locations and topologies that make it difficult for Internet service providers to provide high speed internet. Wireless mesh networks are being considered for these areas. A series of nodes would be mounted from the nearest available wired access point out to the hard-to-reach areas.

- **Surveillance**

  The network can provide power to stand-alone devices like surveillance cameras without having to plug the camera into an electrical outlet and transmit the information to surveillance centres in high quality.

- **Education**

  Many colleges and universities are converting their entire campuses to wireless mesh networks. This solution eliminates the need to bury cables in old buildings and across campuses. With dozens of well-placed indoor and outdoor nodes, everyone will be connected all the time. Also, colleges can monitor the surveillance cameras and act in case of any emergencies.

- **Healthcare**

  The ability to connect to the hospital or clinic network is very important as doctors and nurses maintain and update patient information - test results, medical history, and insurance information - on portable electronic devices that can be carried from room to room.

- **Temporary Venues**

  Wireless mesh networks can be used for temporary venues such as construction sites. Architects and engineers can stay wired to the office, and wireless surveillance cameras can decrease theft and vandalism. Mesh nodes can be moved around and added or removed as the construction progresses. Other temporary venues like conferences, etc can also gain from this technology.

1.6 Examples of wireless mesh networks:

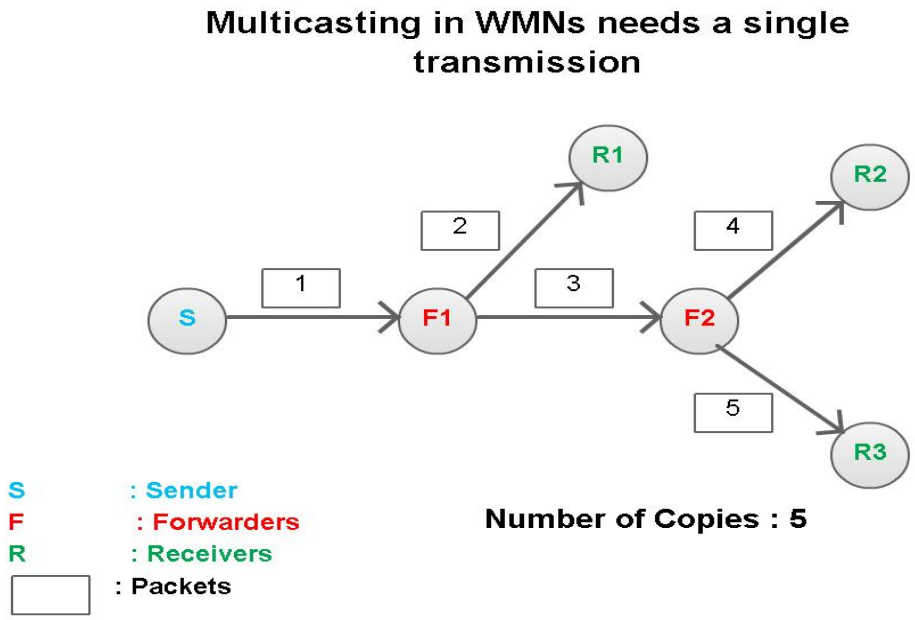
The Cloud in the city of London is one example of a wireless mesh network. The Cloud has been providing Wi-Fi across the City of London since 2006. At the time, few agencies had woken up to the importance of providing internet access on the move – and the City Corporation was keen to provide a service that would reinforce the world class telecommunications offering in the city of London. From the outset, the City Corporation wanted to provide workers and visitors to the City with a seamless communications experience. This meant rolling out outdoor access points placed on street furniture to provide a comprehensive service covering 95% of the City.

However, the smartphone revolution and the inexorable rise of tablets over the past five years meant that the existing infrastructure and bandwidth increasingly came under pressure, and needed reviewing. So, in 2011, The Cloud undertook a full network upgrade and major technology refresh, which quadrupled the network's wireless capacity and created the largest gigabit network in Europe. This was designed in order to ensure the area could easily cope with the continuing rise in demand and to ensure its resilience.

1.7 Importance of multicasting in wireless mesh networks in the current scenario:

Multicasting is an essential technology for wireless networks as it provides efficient communication between mesh nodes in a WMN and is used in many applications and services such as video conferencing, distance education and distributed Internet gaming. It is very useful in wireless communication where bandwidth is scarce, as

transmission is done only once. For WMNs, multicasting can hugely enhance the network capacity by taking advantage of links which can be shared by multiple users to receive the same data, which is transmitted only once.



Not much research on multicast in WMNs has been done. The only existing work is the routing protocol proposed by Ruiz et al. [3]

Chapter 2:

Related work (Literature Survey)

Efficient multicast routing has been studied in depth for wired networks and many protocols such as DVMRP (Distance Vector Multicast Routing Protocol), MOSPF (Multicast Extensions to Open Shortest Path First), PIM (Protocol Independent Multicast) and CBT (Core-Based Tree) are in use.

But wired network protocols do not work for wireless networks. In wireless networks, bandwidth is scarce and wireless links are more error-prone than wired links.

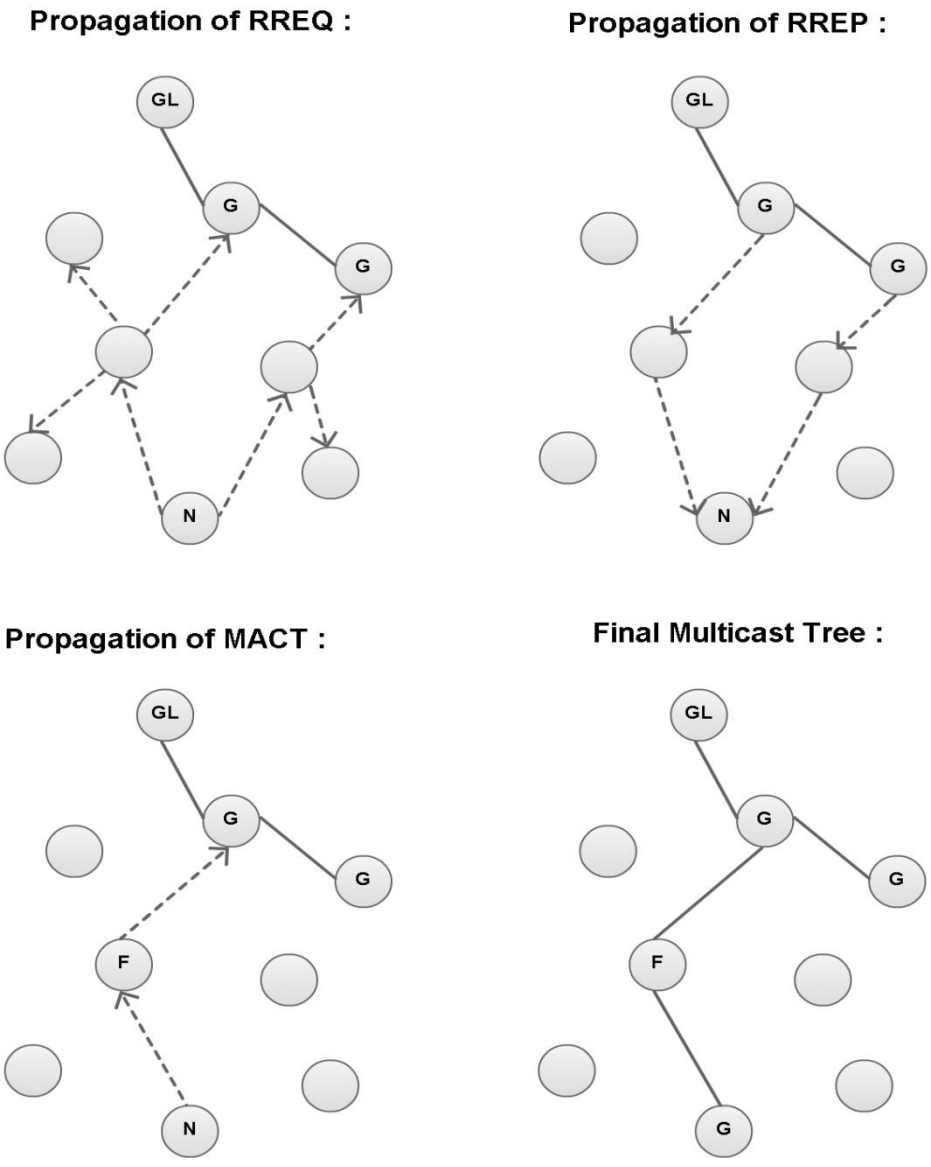
Also, the protocols proposed for mobile ad hoc networks (MANETs) such as MAODV (Multicast Ad hoc On-Demand Distance Vector Routing) , ODMRP (On-Demand Multicast Routing Protocol) and CAMP (Core-Assisted Mesh Protocol) are not efficient for wireless mesh networks .[3]

2.1 Existing methodologies for multicasting in wireless mesh networks:

Many protocols have been suggested for efficient multicasting in Mobile Ad-hoc Networks (MANETs) like:

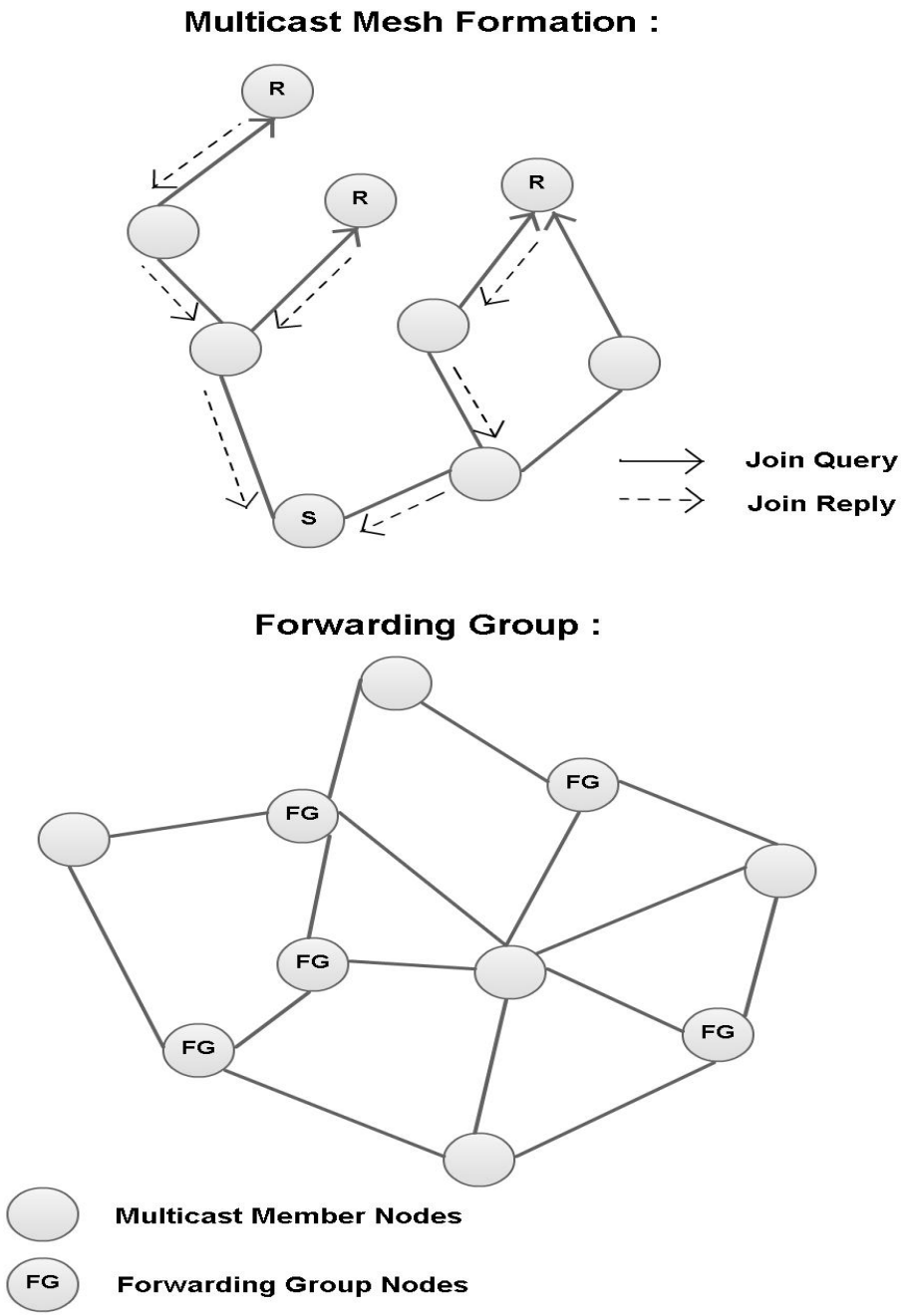
2.1.1 Multicast Ad hoc On-Demand Distance Vector (MAODV) routing protocol – it enables dynamic routing between mobile nodes which want to join a multicast group within an ad hoc network. In MAODV, a multicast tree is established when a sender wants to send data to multiple receivers. The membership of these multicast groups can change anytime during the lifetime of the network. MAODV enables mobile nodes to establish a tree

connecting multicast group members. Tree construction is done with the help of group leaders and group sequence numbers. In the event of a network partition, multicast trees are established independently in each partition, and trees for the same multicast group are quickly connected if the network components merge. [4][5][8]



2.1.2 On Demand Multicast Routing Protocol (ODMRP) – it is mesh based rather than tree based. A mesh is formed on demand by a group of nodes

called *forwarding nodes*. These nodes forward data packets from source to destination. ODMRP dynamically creates routes on demand and also maintains multicast group membership. It forwards the multicast packets via flooding providing path redundancy. [7][8]



2.1.3 Core Assisted Mesh Protocol (CAMP) - CAMP builds a multicast mesh within each multicast group. It provides at least one path from each source to each receiver in the multicast group. CAMP ensures that the shortest paths from receivers to sources are part of a group's mesh. Packets are forwarded on the shortest paths from sources to receivers within the mesh. A router keeps a cache of the identifiers of the packets it has forwarded recently, and forwards a multicast packet received from a neighbour if the packet identifier is not in its cache. [6]

2.2 Problems with the existing methodologies:

2.2.1 These multicast routing protocols were made keeping MANETs in mind. But WMNs and MANETs differ in the way that in WMNs, the mesh routers are static, while in MANETs, the nodes are mobile.

2.2.2 When a network is static, there is no need for creating a mesh. Overheads can be reduced by creating cost effective trees. So tree construction is better suited to wireless mesh networks, for example, Steiner trees.

Chapter 3:

Steiner Trees-An Overview

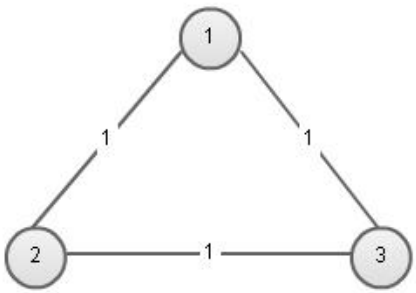
3.1 Usefulness of Steiner trees:

There are two fundamental approaches to multicast routing: **shortest path trees** (SPTs) and **minimum cost trees** (MCTs). The SPT algorithms minimize the distance from the sender to each receiver, while the MCT algorithms such as **minimum Steiner trees** (MSTs) minimize the overall edge cost of the multicast tree. In wireless multi-hop networks, a minimum cost tree is one which connects sources and receivers by issuing a minimum number of transmissions (MNT). [9]

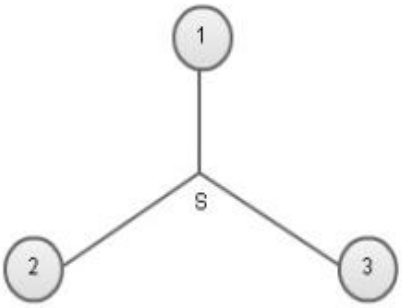
The minimum Steiner tree problem is different from the minimum spanning tree problem in the way that in minimum Steiner tree, we are allowed to create extra intermediate nodes (other than the nodes of the tree) so as to achieve the goal of minimum cost whereas in minimum spanning tree, we must connect the existing nodes via the shortest path.

For example, for a set of 3 points:

Minimum Spanning Tree :



Minimum Steiner Tree:



S is the Steiner point – the intermediate point

that needed to be created to construct the minimum cost tree. S is the Fermat point of the triangle formed by joining the nodes of the graph.

The total cost of Steiner trees will be less than the total cost of the corresponding SPT for a given set of nodes. So, MCTs are used for laying down telephone lines, building sewer systems, etc.

3.2 Minimum Steiner Tree problem:

The MST problem is to find the smallest tree connecting all the vertices of a tree. For instance, let us say we are given a set of sites. The minimum Steiner tree problem will be to connect these sites with wires or pipes as cheaply as possible.

More formally put,
Given an undirected graph (network) G = (V,E,c) where c : E → R is an edge length function, and a non-empty set N, N ⊆ V, of terminals; we have to find a sub-network $T_G(N)$ of G such that:

- there is a path between every pair of terminals,
- total length $|T_G(N)| = \sum_{e_i \, \epsilon \, T_G(N)} c(e_i)$ is minimised.

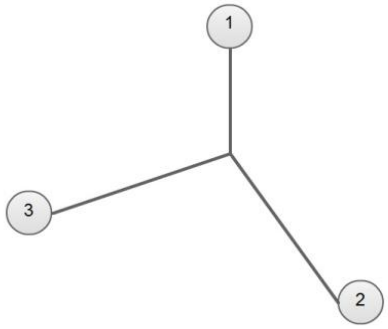
The Euclidean minimum Steiner tree problem is an NP-hard problem and there is no optimal way of solving it. So heuristic algorithms are used to find a near optimal solution of the minimum Steiner tree problem.

3.3 Types of Steiner Trees:

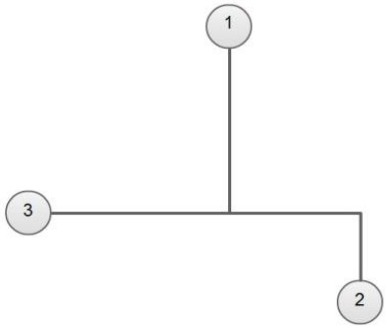
The Steiner tree problem can be divided into 3 main areas: [10]

3.3.1 **Euclidean Steiner tree:** it is the original Steiner tree problem, i.e. to find the minimum tree connecting ‘n’ nodes with the addition of some intermediate nodes (Steiner points).

3.3.2 **Rectilinear Steiner tree:** in this type of Steiner tree, to join all the nodes we only consider paths that consist of horizontal or vertical line segments. The Euclidean distance is replaced by the rectilinear distance.



Euclidean Steiner tree for 3 nodes



Rectilinear Steiner tree for 3 nodes

3.3.3 **Steiner trees in networks:** it is the field that has emerged recently and the one in which the most research is going on. This area focuses on use of Steiner trees in designing cost effective algorithms for routing in networks.

3.4 About Steiner Trees:

3.4.1 Properties of Minimal Steiner Tree:

3.4.1.1 Angle condition: no two edges must meet at an angle less than 120°.

3.4.1.2 Each Steiner point has degree 3.

3.4.1.3 There are no intersecting edges.

3.4.1.4 There can be at most (n-2) Steiner points, n being the total number of
nodes in the network. [11]

3.4.2 Applications of Steiner trees:

3.4.2.1 Network design

3.4.2.2 Wiring layout design on circuits

3.4.2.3 Drainage networks

3.4.2.4 Wire routing phase in physical VLSI design.

3.4.3 Topologies of Steiner trees:

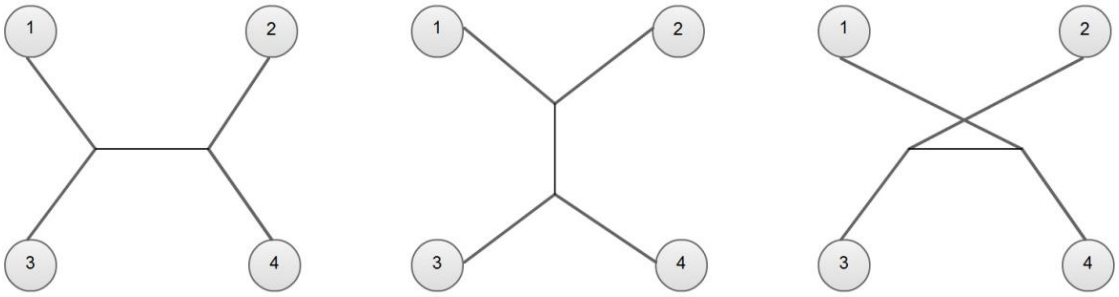
Steiner topologies show in how many ways the terminals (nodes) and Steiner points can be connected to form the minimal Steiner tree. A full Steiner topology has n-2 Steiner points.

The number of full Steiner topologies can be given by: [11]

$$f(n) = \frac{(2n-4)!}{[2^{n-2}(n-2)!]}$$

For example,

for n = 4 $\Rightarrow$ f(n) = 3. So, the 3 topologies are :



3.5 Sum up of problem statement:

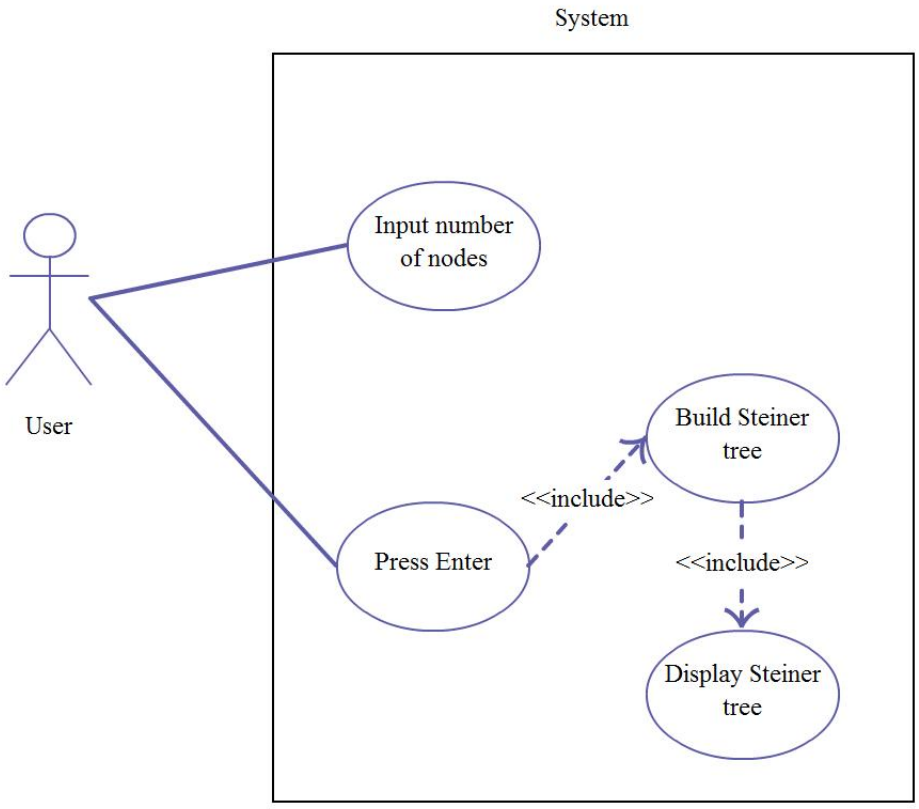
The existing multicasting methodologies for Mobile Ad-hoc Networks (MANETs) are not useful enough to use as multicasting algorithms in wireless mesh networks. Steiner trees, by creating a minimum cost network, can prove useful in implementation of multicasting in WMNs.

Chapter 4:

Implementation

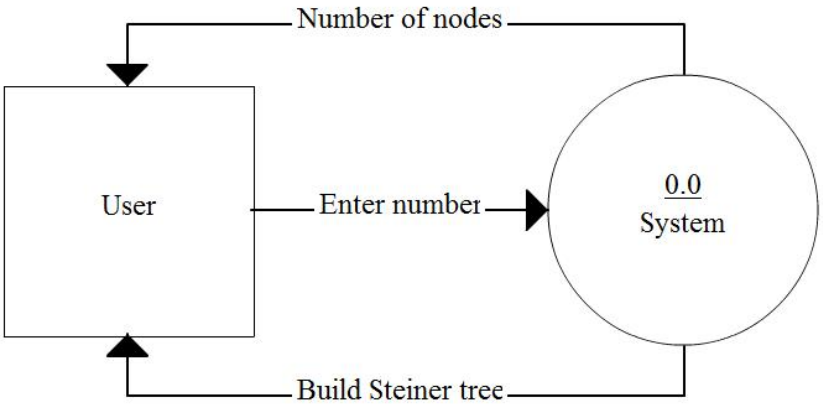
4.1 Diagrams:

4.1.1 Use case diagram:

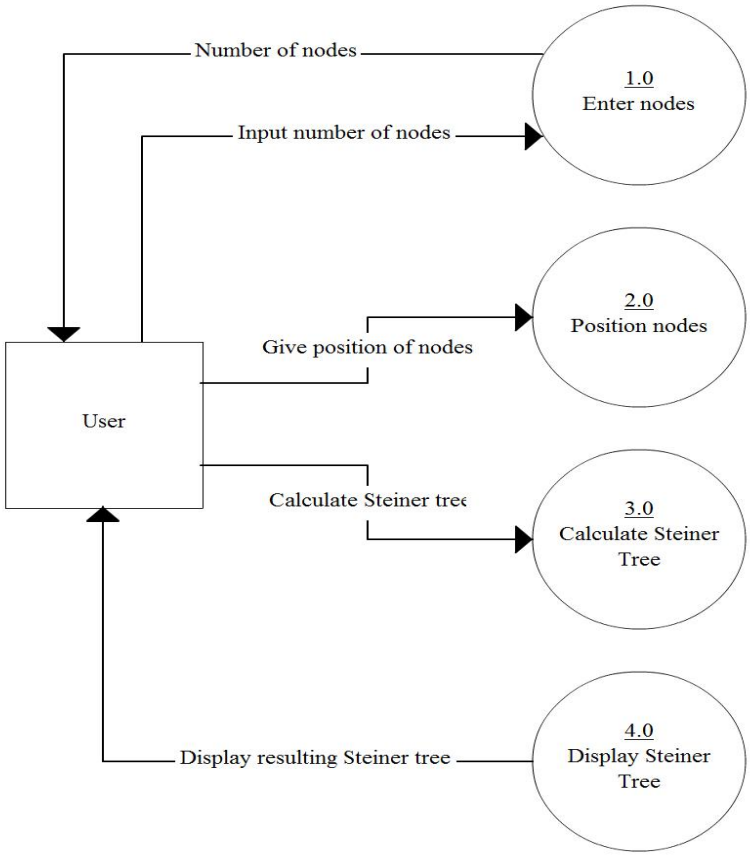


4.1.2 Data Flow Diagram:

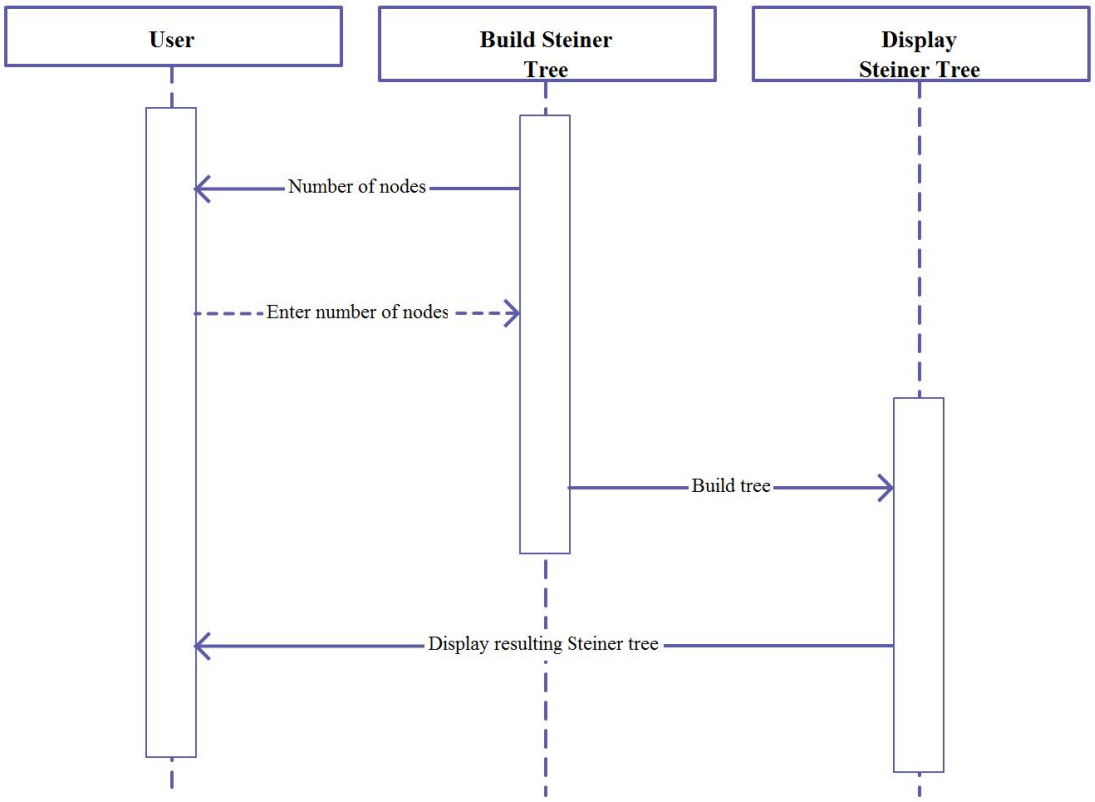
4.1.2.1 Context Level Dfd:



4.1.2.2 Level 0 Dfd:



4.1.3 Sequence Diagram:



4.2 Algorithm:

Input: an undirected distance graph G = (V, E, d) and a set of Steiner points S ⊆ V.

Output: a Steiner tree, T_H, for G and S.

Algorithm:

Step 1: Construct the complete undirected distance graph $G_1 = (V_1, E_1, d_1)$ from G and S.

Step 2: Find the minimal spanning tree, T_1, of G_1. (If there are several minimal spanning trees, pick an arbitrary one.)

Step 3: Construct the subgraph, G_S, of G by replacing each edge in T_1 by its corresponding shortest path in G. (If there are several shortest paths, pick an arbitrary one.)

Step 4: Find the minimal spanning tree, T_S, of G_S. (If there are several minimal spanning trees, pick an arbitrary one.)

Step 5: Construct a Steiner tree, T_H, from T_S by deleting edges in T_S, if necessary, so that all the leaves in T_H are Steiner points.

4.3 Code:

Prims:

```
package steiner;

import java.util.InputMismatchException;

import java.util.Scanner;
```

```
public class Prims

{

    private boolean unsettled[];

    private boolean settled[];

    private int numberofvertices;

    private int adjacencyMatrix[][];

    private int key[];

    public static final int INFINITE = 999;

    private int parent[];

    public Prims(int numberofvertices)

    {
```

```
    this.numberofvertices = numberofvertices;

    unsettled = new boolean[numberofvertices + 1];

    settled = new boolean[numberofvertices + 1];

    adjacencyMatrix = new int[numberofvertices + 1][numberofvertices + 1];

    key = new int[numberofvertices + 1];

    parent = new int[numberofvertices + 1];

}


public int getUnsettledCount(boolean unsettled[])

{

    int count = 0;

    for (int index = 0; index < unsettled.length; index++)
```

```
    {

        if (unsettled[index])

        {

            count++;

        }

    }

    return count;

}

public void primsAlgorithm(int adjacencyMatrix[][])

{

    int evaluationVertex;
```

```
        for (int source = 1; source <= numberofvertices; source++)

        {

            for (int destination = 1; destination <= numberofvertices; destination++)

            {

                this.adjacencyMatrix[source][destination] =
adjacencyMatrix[source][destination];

            }

        }

        for (int index = 1; index <= numberofvertices; index++)

        {

            key[index] = INFINITE;

        }
```

```
    key[1] = 0;

    unsettled[1] = true;

    parent[1] = 1;



    while (getUnsettledCount(unsettled) != 0)

    {

        evaluationVertex = getMimumKeyVertexFromUnsettled(unsettled);

        unsettled[evaluationVertex] = false;

        settled[evaluationVertex] = true;

        evaluateNeighbours(evaluationVertex);

    }

}
```

```
private int getMimumKeyVertexFromUnsettled(boolean[] unsettled2)

{

   int min = Integer.MAX_VALUE;

   int node = 0;

   for (int vertex = 1; vertex <= numberofvertices; vertex++)

   {

      if (unsettled[vertex] == true && key[vertex] < min)

      {

         node = vertex;

         min = key[vertex];

      }
```

```
        }

        return node;

    }

    public void evaluateNeighbours(int evaluationVertex)

    {

        for (int destinationvertex = 1; destinationvertex <= numberofvertices; destinationvertex++)

        {

            if (settled[destinationvertex] == false)

            {

                if (adjacencyMatrix[evaluationVertex][destinationvertex] != INFINITE)

                {
```

```
            if (adjacencyMatrix[evaluationVertex][destinationvertex] < key[destinationvertex])

            {

               key[destinationvertex] = adjacencyMatrix[evaluationVertex][destinationvertex];

               parent[destinationvertex] = evaluationVertex;

            }

            unsettled[destinationvertex] = true;

         }

      }

   }

}
```

```
    public void printMST()

    {

        System.out.println("SOURCE  : DESTINATION =   WEIGHT");

        for (int vertex = 2; vertex <= numberofvertices; vertex++)

        {

            System.out.println(parent[vertex]  +  "\t:\t"  +  vertex  +"\t=\t"+
adjacencyMatrix[parent[vertex]][vertex]);

        }

    }


    public static void main(String... arg)

    {

        int adjacency_matrix[][];
```

```
int number_of_vertices;

Scanner scan = new Scanner(System.in);

try

{

   System.out.println("Enter the number of vertices");

   number_of_vertices = scan.nextInt();

   adjacency_matrix = new int[number_of_vertices + 1][number_of_vertices +
1];

   System.out.println("Enter the Weighted Matrix for the graph");

   for (int i = 1; i <= number_of_vertices; i++)

   {

      for (int j = 1; j <= number_of_vertices; j++)
```

```
{
    adjacency_matrix[i][j] = scan.nextInt();
    if (i == j)
    {
        adjacency_matrix[i][j] = 0;
        continue;
    }
    if (adjacency_matrix[i][j] == 0)
    {
        adjacency_matrix[i][j] = INFINITE;
    }
```

```
            }

        }

        Prims prims = new Prims(number_of_vertices);

        prims.primsAlgorithm(adjacency_matrix);

        prims.printMST();

      } catch (InputMismatchException inputMismatch)

      {

        System.out.println("Wrong Input Format");

      }

      scan.close();

    }

}
```

Applet code:

```
package steiner;


import java.lang.*;
import java.awt.*;


public class Tree extends java.applet.Applet {

private final int       maxN = 30;    // the max number of terminals
private int             n = 10;       // the current number of terminals
private final int       r = 4;        // the radius of a terminal
private Point           p[];          // the terminals
private Point           current;      // the current one and its old location
private boolean                 m[][];          // the minimum spanning tree edges
private Rectangle       border, inner; // applet borders
private Scrollbar       sb;           // scrollbar for changing n
private Image           buffer;       // buffer for double-buffering
private Graphics        bufg;         // buffer's Graphics
```

```
public void init()

{

p = new Point[maxN];

current = null;

m = new boolean[maxN][maxN];


border = new Rectangle(0, 0, size().width - 1, size().height - 1);

inner = new Rectangle(r + 1, r + 1,size().width - 2 * r - 3, size().height - 2 * r - 23);


// initialize the terminals to random locations

for (int i = 0; i < maxN; i++)

{

p[i] = new Point((int) Math.round(Math.random() * (size().width - 2 * r - 2) + r + 1),(int) Math.round(Math.random() * (size().height - 20 - 2 * r - 2) + r + 1));

for (int j = 0; j < maxN; j++)

{

m[i][j] = false;

}

}
```

```
mst();


setBackground(Color.white);


setLayout(new BorderLayout());
sb = new Scrollbar(Scrollbar.HORIZONTAL, n, 5, 2, maxN);
add("South", sb);


buffer = createImage(size().width, size().height);
bufg = buffer.getGraphics();
bufg.setFont(getFont());
} // init()




public void update(Graphics g) {
bufg.setColor(getBackground());
bufg.fillRect(border.x, border.y, border.width, border.height);
bufg.setColor(Color.black);
bufg.drawRect(border.x, border.y, border.width, border.height);


// first do the MST edges
for (int i = 0; i < n; i++) {
```

```
for (int j = (i + 1); j < n; j++) {
if (m[i][j]) {
bufg.setColor(Color.red);
bufg.drawLine(p[i].x, p[i].y, p[j].x, p[j].y);
}
}
}
```

```
// redraw the terminals
for (int i = 0; i < n; i++) {
bufg.setColor(Color.green);
bufg.fillOval(p[i].x - r, p[i].y - r, 2 * r, 2 * r);
bufg.setColor(Color.black);
bufg.drawOval(p[i].x - r, p[i].y - r, 2 * r, 2 * r);
}

// draw the current one in cyan
if (current != null) {
bufg.setColor(Color.cyan);
bufg.fillOval(current.x - r, current.y - r, 2 * r, 2 * r);
bufg.setColor(Color.black);
```

```
bufg.drawOval(current.x - r, current.y - r, 2 * r, 2 * r);
}

g.drawImage(buffer, 0, 0, null);
} // update(Graphics)



public void paint(Graphics g) {
update(g);
} // paint(Graphics)



public boolean handleEvent(Event evt) {
switch (evt.id) {

case Event.MOUSE_DOWN: {
Rectangle rect = new Rectangle();

current = null;
for (int i = 0; (i < n) && (current == null); i++) {
rect.reshape(p[i].x - r, p[i].y - r, 2 * r, 2 * r);
```

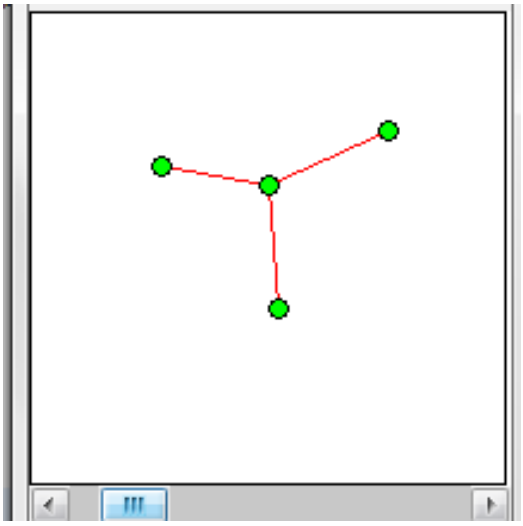
```
if (rect.inside(evt.x, evt.y)) {
current = p[i];
}
}
break;
}

case Event.MOUSE_UP: {
current = null;
repaint();
break;
}

case Event.MOUSE_DRAG: {
if (current != null) {
if (inner.inside(evt.x, evt.y)) {
current.move(evt.x, evt.y);
}
else {
current.move(Math.max(Math.min(evt.x, inner.x + inner.width), inner.x),
Math.max(Math.min(evt.y, inner.y + inner.height), inner.y));
}
mst();
```

```
repaint();
}
break;
}

case Event.SCROLL_LINE_UP: case Event.SCROLL_LINE_DOWN:
case Event.SCROLL_PAGE_UP: case Event.SCROLL_PAGE_DOWN:
case Event.SCROLL_ABSOLUTE: {
n = sb.getValue();
mst();
repaint();
break;
}

default: {
break;
}

} // switch

return(true);
} // handleEvent(Event)
```

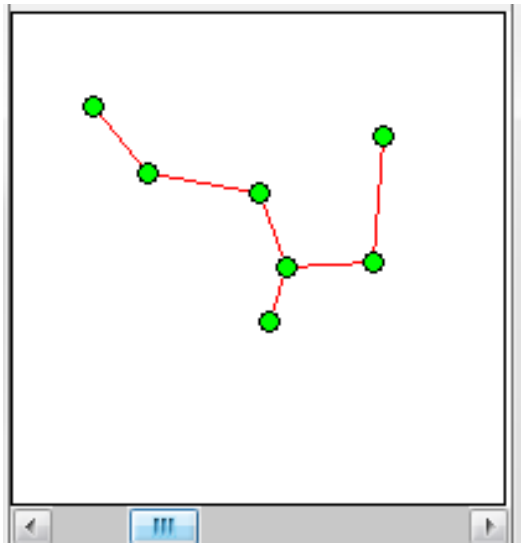
```
//Euclidean distance between two points (x1,y1) and (x2,y2)
private int distance(int x1, int y1, int x2, int y2) {
return((int) Math.round(Math.sqrt(
(double) (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2))));
} // distance(int,int,int,int)


//mst - compute a minimum spanning tree using Prim's algorithm with "dumb"
//heaps.  This is the fastest graph algorithm for complete graphs, though
//we could do better geometrically - but with 10 terminals, why bother?
private void mst() {
int dist[], neigh[], closest, minDist, d;

dist = new int[n];
neigh = new int[n];

// initialize data structures
for (int i = 0; i < n; i++) {
dist[i] = distance(p[0].x, p[0].y, p[i].x, p[i].y);
neigh[i] = 0;
for (int j = 0; j < n; j++) {
m[i][j] = false;
```

```
}

}


// find terminal closest to current partial tree

for (int i = 1; i < n; i++)

{

closest = -1;

minDist = Integer.MAX_VALUE;

for (int j = 1; j < n; j++)

{

if ((dist[j] != 0) && (dist[j] < minDist))

{

closest = j;

minDist = dist[j];

}

}


// set an edge from it to its nearest neighbor

m[neigh[closest]][closest] = true;

m[closest][neigh[closest]] = true;


// update nearest distances to current partial tree

for (int j = 1; j < n; j++)
```

```
{
d = distance(p[j].x, p[j].y, p[closest].x, p[closest].y);
if (d < dist[j])
{
dist[j] = d;
neigh[j] = closest;
}
}
}
} // mst()


} // class Tree extends java.applet.Applet
```



Applet (a)



Applet (b)

Network code:

```
# Define options
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 9 ;# number of mobilenodes
```

```
set val(rp) DSDV ;# routing protocol

set val(x) 1000 ;# X dimension of topography

set val(y) 1000 ;# Y dimension of topography

set val(stop) 150 ;# time of simulation end



set ns [new Simulator]

set tracefd [open simple.tr w]

set namtrace [open simwrls.nam w]

$ns trace-all $tracefd

$ns namtrace-all-wireless $namtrace $val(x) $val(y)



# set up topography object

set topo [new Topography]


$topo load_flatgrid $val(x) $val(y)


create-god $val(nn)



# configure the nodes

$ns node-config -adhocRouting $val(rp) \

-llType $val(ll) \

-macType $val(mac) \
```

```
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

for {set i 0} {$i < $val(nn) } { incr i } {
set n($i) [$ns node]
}


# Provide initial location of mobilenodes
$n(0) set X_ 347.0
$n(0) set Y_ -20.0
$n(0) set Z_ 0.0

$n(1) set X_ 345.0
$n(1) set Y_ 40.0
$n(1) set Z_ 0.0

$n(2) set X_ 330.0
```

```
$n(2) set Y_ 150.0

$n(2) set Z_ 0.0


$n(3) set X_ 316.0

$n(3) set Y_ 200.0

$n(3) set Z_ 0.0


$n(4) set X_ 246.0

$n(4) set Y_ 90.0

$n(4) set Z_ 0.0


$n(5) set X_ 400.0

$n(5) set Y_ 6.0

$n(5) set Z_ 0.0


# Set a TCP connection between n(1) and n(5)

set tcp [new Agent/TCP/Newreno]

$tcp set class_ 2

set sink [new Agent/TCPSink]

$ns attach-agent $n(1) $tcp

$ns attach-agent $n(5) $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 10.0 "$ftp start"


# Set a TCP connection between n(1) and n(0)
```

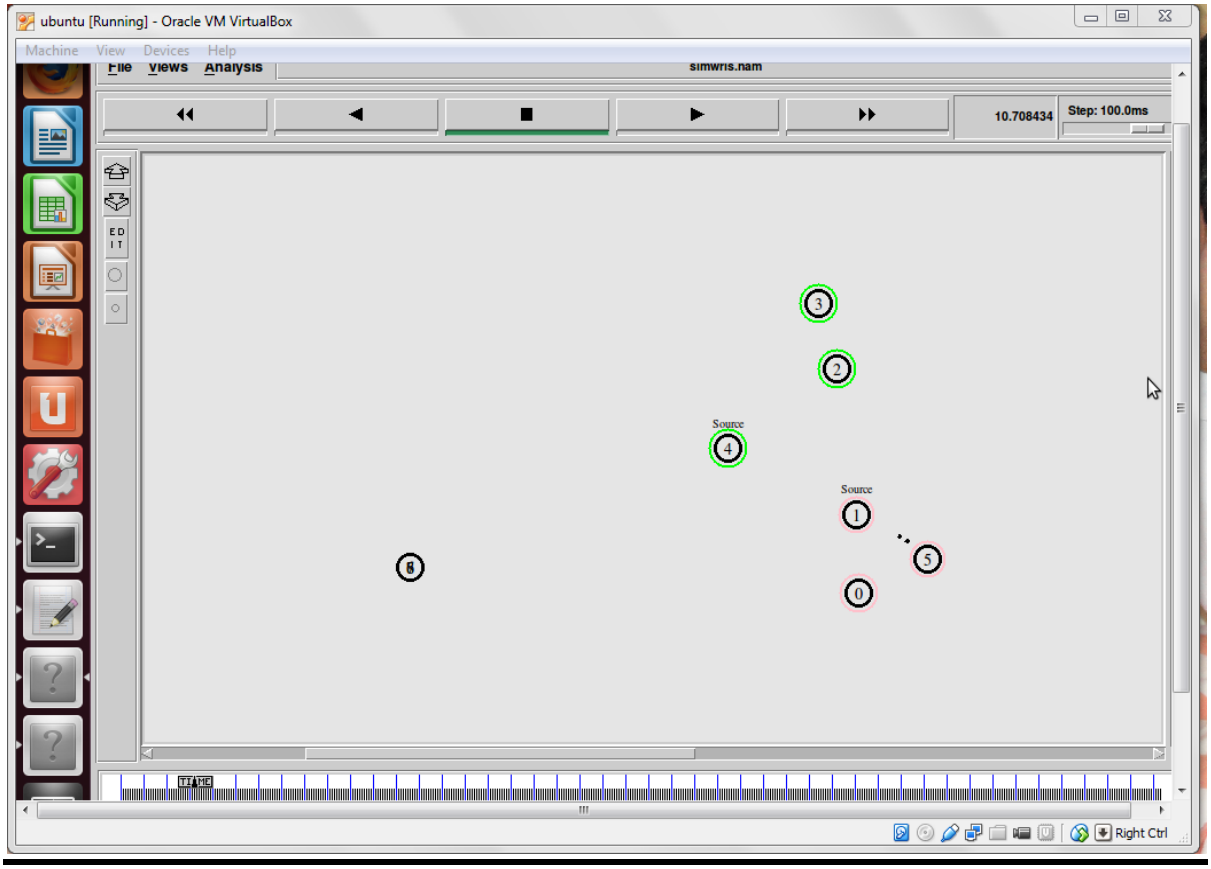
```
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(1) $tcp
$ns attach-agent $n(0) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

# Set a TCP connection between n(1) and n(7)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(1) $tcp
$ns attach-agent $n(7) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 60.0 "$ftp start"
$ns at 110.0 "$ftp stop"

# Set a TCP connection between n(4) and n(8)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(4) $tcp
```

```
$ns attach-agent $n(8) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 45.0 "$ftp start"

# Set a TCP connection between n(4) and n(2)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(4) $tcp
$ns attach-agent $n(2) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

# Set a TCP connection between n(4) and n(3)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $n(4) $tcp
$ns attach-agent $n(3) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"
```

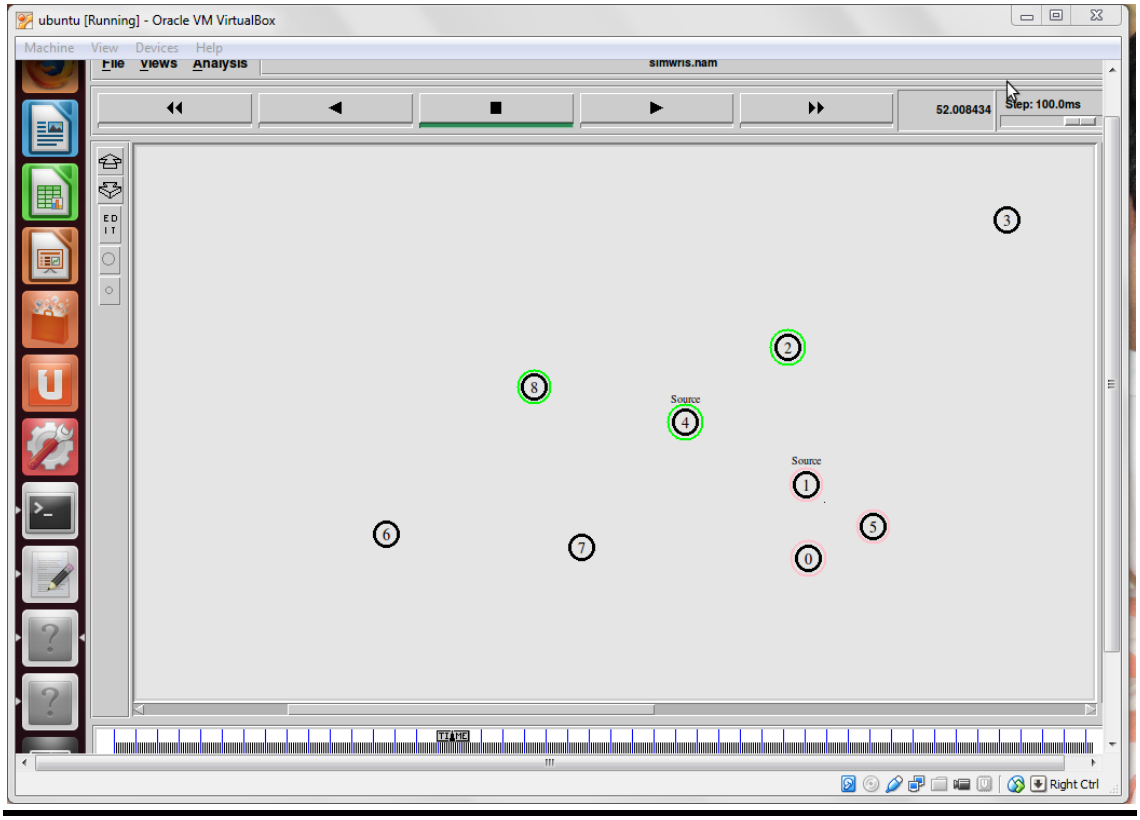
```
$ns at 50.0 "$ftp stop"

#defining heads
$ns at 0.0 "$n(4) label Source"
$ns at 0.0 "$n(1) label Source"



#mobility to the nodes
$ns at 10.0 "$n(3) setdest 785.0 228.0 0.0"
$ns at 13.0 "$n(7) setdest 700.0 20.0 0.0"
$ns at 15.0 "$n(8) setdest 115.0 85.0 0.0"



#Color change while moving from one group to another

$ns at 0.0 "$n(0) delete-mark N0"
$ns at 0.0 "$n(0) add-mark N0 pink circle"
$ns at 0.0 "$n(1) delete-mark N1"
$ns at 0.0 "$n(1) add-mark N1 pink circle"
$ns at 0.0 "$n(5) delete-mark N5"
$ns at 0.0 "$n(5) add-mark N5 pink circle"
$ns at 0.0 "$n(2) delete-mark N2"
$ns at 0.0 "$n(2) add-mark N2 green circle"
$ns at 0.0 "$n(3) delete-mark N3"
$ns at 0.0 "$n(3) add-mark N3 green circle"
$ns at 0.0 "$n(4) delete-mark N4"
```

```
$ns at 0.0 "$n(4) add-mark N4 green circle"
$ns at 45.0 "$n(8) delete-mark N8"
$ns at 45.0 "$n(8) add-mark N8 green circle"
$ns at 50.0 "$n(3) delete-mark N3"
$ns at 60.0 "$n(7) delete-mark N7"
$ns at 60.0 "$n(7) add-mark N7 pink circle"
$ns at 110.0 "$n(7) delete-mark N7"



# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 20 defines the node size for nam
$ns initial_node_pos $n($i) 20
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
$ns at $val(stop) "$n($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.00 "puts \"end simulation\" ; $ns halt"
proc stop {} {
global ns tracefd namtrace
$ns flush-trace
```

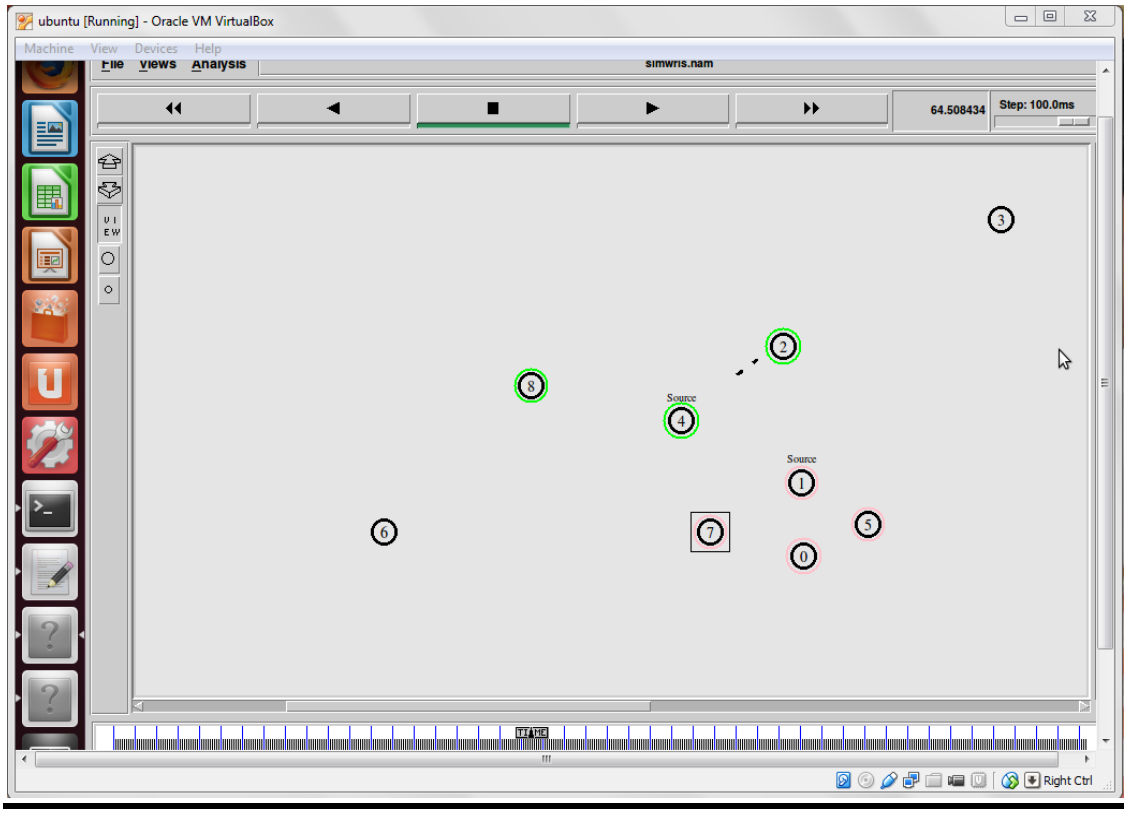
```
close $tracefd

close $namtrace

exec nam simwrls.nam &

}


$ns run
```



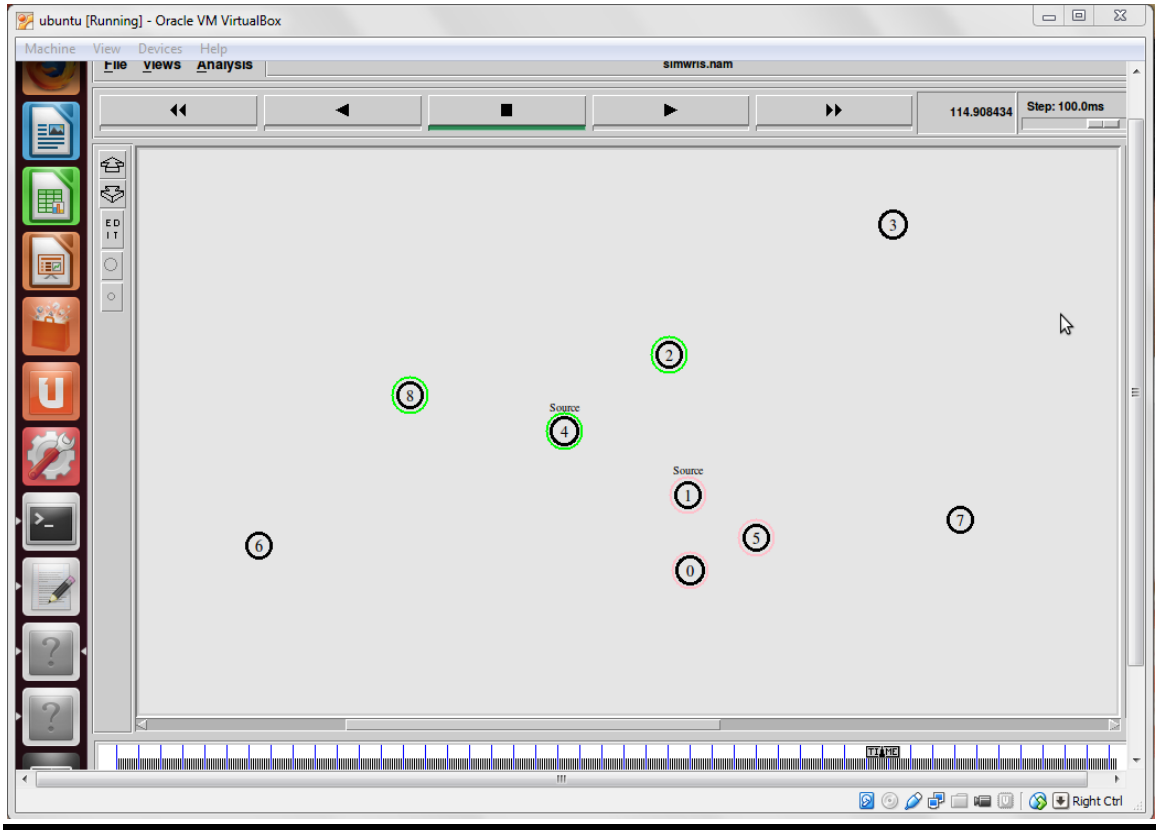
Network (a)



Network (b)



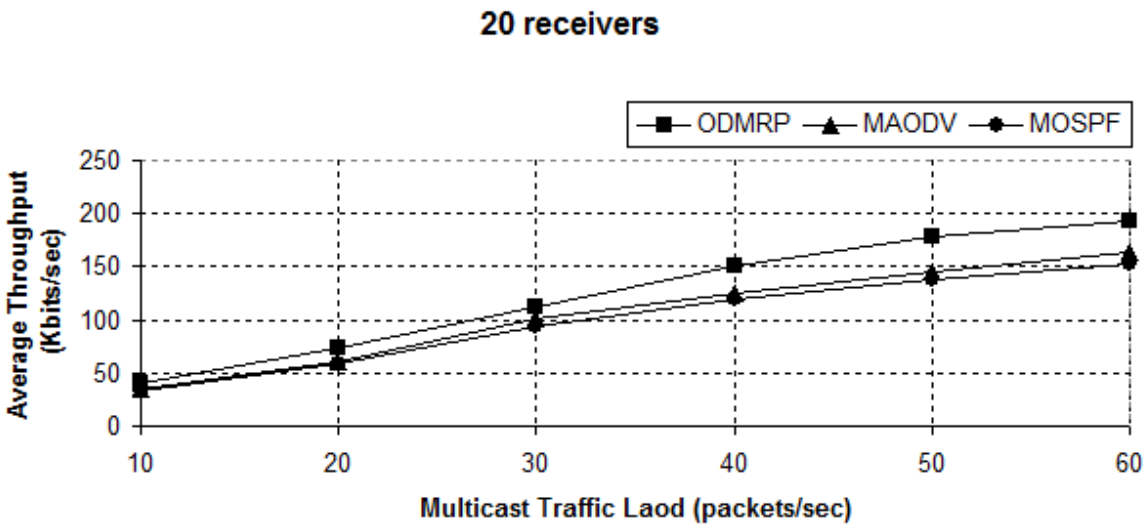
Network (c)

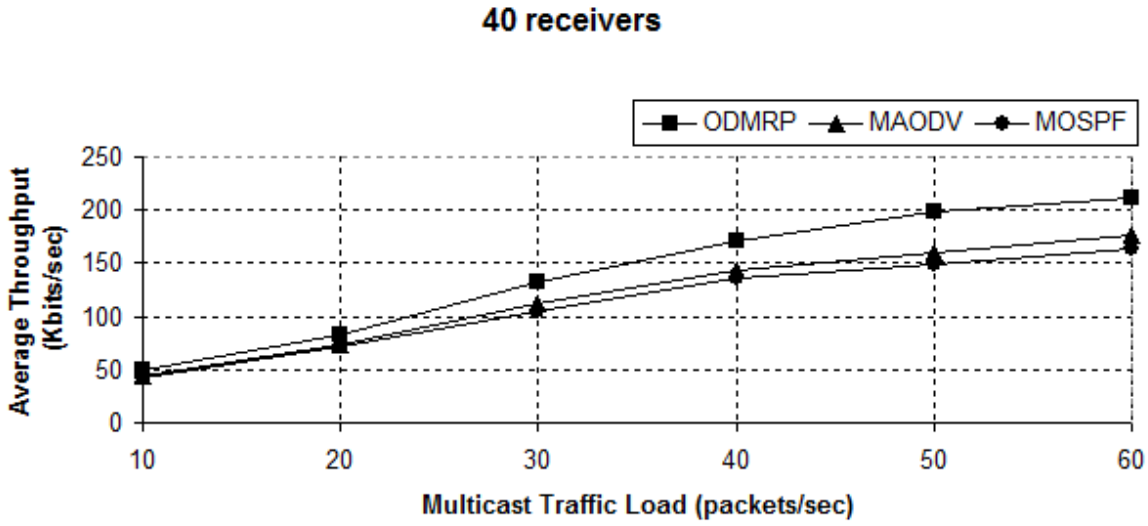

ubuntu [Running] - Oracle VM VirtualBox
Machine View Devices Help
File Views Analysis
simwrls.nam
114.908434
Step: 100.0ms
Source
Source
Right Ctrl

Network (d)

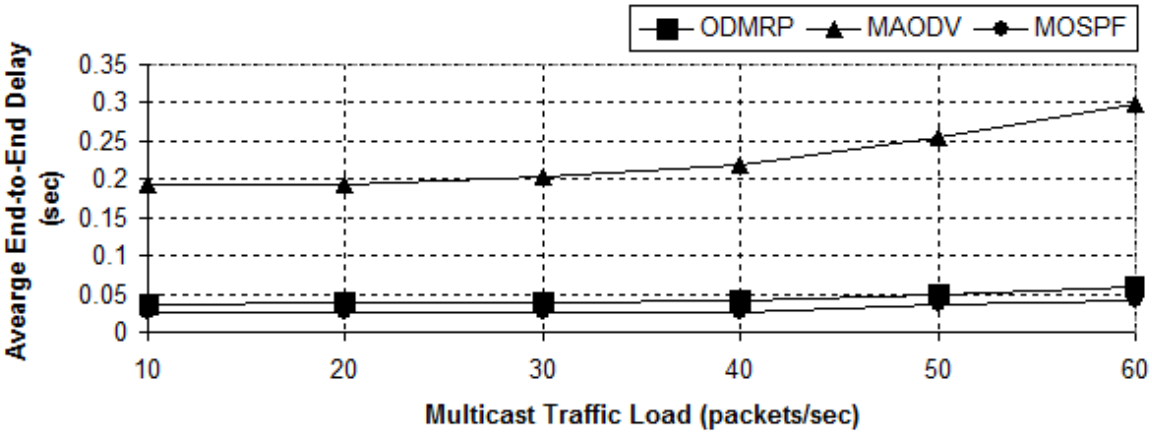
RESULTS

Comparison of different multicasting routing protocols for wireless networks:

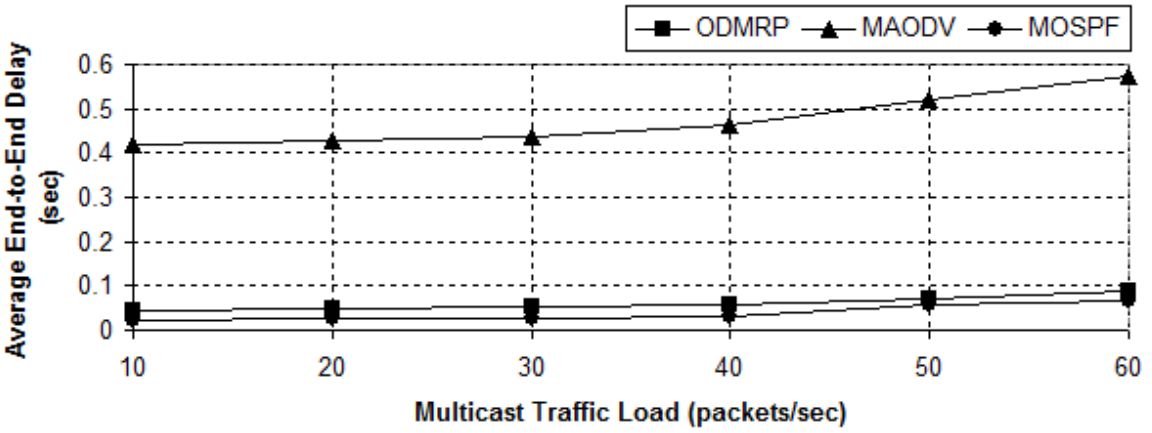




20 receivers



40 receivers



Comparing Shortest Path Trees (SPTs) and Minimum Steiner Trees (MSTs) in a network:

> When the number of multicast nodes in the networks is small or moderate (10 to 30 receiver nodes), the SPT and the MST have similar effects on the average Packet Delivery Ratio (PDR) of the unicast flows. When the multicast group is large (e.g., 5 senders and 50 receivers) and the total multicast traffic load is high (150 packets/sec or more), the SPT causes more packet losses to the unicast flows than the MST, from 1% to 3.5% more. This is due to the fact that the SPT involves more nodes in the data forwarding task than the MST, and thus causes more packet collisions and more congestion to the unicast flows when the multicast sending rate is high.

CONCLUSIONS

Wireless mesh networks are a promising technology for the future of wireless technologies and multicasting is an essential part of this technology. In this project, some of the existing methodologies for multicasting were reviewed and analysed. Many protocols for multicasting in wired networks have been suggested such as DVMRP, MOSPF, CBT, etc. and there are some existing protocols for multicasting in wireless networks too such as MAODV, ODMRP, CAMP, etc. But these protocols do not work well and efficiently for wireless mesh networks as these protocols have been designed for ad-hoc networks in which nodes are mobile, whereas in wireless mesh networks, the nodes are stationary.

So, heuristic based protocols need to be designed for efficient multicasting in wireless mesh networks. Also, we analysed the difference between shortest path trees and minimum cost trees and how MCTs or Steiner trees give a better performance than SPTs and give lower bandwidth consumption, so better throughput.

REFERENCES

[1] Mihail L. Sichitiu, "Wireless Mesh Networks: Opportunities and Challenges" in the 10th IEEE Symposium on Computers and Communications (ISCC), 2005.

[2] http://computer.howstuffworks.com/how-wireless-mesh-networks-work.htm

[3] Pedro M. Ruiz and Francisco J. Galera, Christophe Jelger and Thomas Noel, "Efficient Multicast Routing in Wireless Mesh Networks Connected to Internet" published in InterSense '06 Proceedings of the first International conference on Integrated internet ad hoc and sensor networks, article 26, 2006.

[4] https://tools.ietf.org/id/draft-ietf-manet-maodv-00.txt

[5] Ben-Jye Chang and Jyh-Wei Wang. "Efficient Unicast-based MultiHop Local Repair for Wireless Multicast MANET" in AIT 2007, March 2007.

[6] Ewerton L. Madruga and J.J.Garcia-Luna-Aceves, "Scalable Multicasting: The Core-Assisted Mesh Protocol" in conference on ACM/BALTZER mobile networks and applications, special issue on management of mobility, 1999.

[7] http://www.ietf.org/proceedings/48/I-D/manet-odmrp-02.txt

[8] Karthika A. Nair, Greeshma Vidyadharan T.,P. Revathi and Usha Devi G., "Analyzing the performance of MAODV, ODMRP, MOSPF and PIM in Mobile Ad hoc Networks" in the International Journal of Computer Science and Telecommunications, Vol 4, Issue 2, February 2013.

[9] Uyen Trang Nguyen, "On Multicast Routing in Wireless Mesh Networks" published in Computer Communications Journal, Vol 31, Issue May7, 2008.

[10] "The Steiner Tree Problem (annals of discrete mathematics)" by F. K. Hwang, D. S. Richards, P. Winters, Elsevier Science Publishers B. V.

[11] Germander Soothill, "The Euclidean Steiner Tree Problem", 16th February, 2010.

[12] Ian F. Akyildiz and Xudong Wang, “A Survey on Wireless Mesh Networks” in Journal Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol 47, Issue 4, March 2005.