P-3607

ANALYSIS OF MINIMAL DELAY PATHS AND DYNAMIC RESOURCE SHARING IN UNSTRUCTURED PEER TO PEER NETWORKS

PROJECT REPORT

Submitted by

GOWTHAM. S **Reg.No: 0710108016**

VENGATESH. P **Reg.No: 0710108055**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University of Technology, Coimbatore)

COIMBATORE – 641049

APRIL 2011

KUMARAGURU COLLEGE OF TECHNOLOGY

BONAFIDE CERTIFICATE

This is to Certify that the project entitled **"ANALYSIS OF MINIMAL DELAY PATHS AND DYNAMIC RESOURCE SHARING IN UNSTRUCTURED PEER TO PEER NETWORKS"** is the bonafide work of **"GOWTHAM.S and VENGATESH. P"** who carried out the project work under my supervision.

[illegible]

SIGNATURE

Prof.P.Devaki,

Head of the Department.

[illegible]

SIGNATURE

Ms.S.Rajini,

Supervisor,

Associate Professor.

Department of Computer science & Engineering,
Kumaraguru College of Technology,
Coimbatore - 641 006.

The candidates with University Register Nos. **0710108016, 0710108055** were examined by us in the project viva-voice examination held on ...20/4............

[illegible]

INTERNAL EXAMINER

[illegible]

EXTERNAL EXAMINER

DECLARATION

We,

GOWTHAM.S	**0710108016**
VENGATESH.P	**0710108055**

hereby declare that the project work titled "**ANALYSIS OF MINIMAL DELAY PATHS AND DYNAMIC RESOURCE SHARING IN UNSTRUCTURED PEER TO PEER NETWORKS**" being submitted in partial fulfillment for the award of **BACHELOR OF ENGINEERING** in **COMPUTER SCIENCE AND ENGINEERING** is the original work carried out by us. It has not formed the part of any other project work submitted for award of any degree or diploma, either in this or any other University.

Place: Coimbatore

Date : 19-04-11

[illegible]

[GOWTHAM S]

[illegible]

[VENGATESH P]

Project Guided by

[illegible]

Ms.S.Rajini., M.S

ACKNOWLEDGEMENT

We are extremely grateful to **Dr.J.Shanmugam**, Director, Kumaraguru College of Technology for having given us this opportunity to embark on this project.

We extend our sincere thanks to our Principal **Dr.S.RamaChandran M.Tech., Ph.D**, Kumaraguru College of Technology for providing us with necessary facilities and has given the opportunities to work on this project.

We express our heartiest thanks to our Head of the Department **Ms.P.Devaki, M.E (Ph.D)**, Department of Computer Science and Engineering, who have helped us to overcome the perplexity while choosing the project.

We thank our guide **Ms.S.Rajini M.S**, Associate Professor, Department of Computer Science and Engineering, for her excellent guidance in each and every step of our project and been with us to complete the project.

We would like to express our sincere gratitude to our Class Advisor **Mrs.C.Ramathilagam, M.E,** Assistant Professor, Department of Computer Science and Engineering for being an initiative for this project and for her constant support and motivation throughout the course of the project.

We thank the **Teaching and Non-Teaching Staff members** of our department for providing us the technical support in the duration of our project. We also thank all our **parents and friends** who helped us to complete this project successfully.

ABSTRACT

This project consists of two simulation works in unstructured peer to peer networks.

Work 1: In an unstructured peer-to-peer (P2P) network, participating peers choose their neighbors randomly for transmission of data, resulting in lengthy communication between the peers. Previous solutions to this problem in the literature have no performance guarantee and it is time consuming. In this analysis, we propose a path delay algorithm based on the delay calculation method, by which the minimal delay path is chosen for transmission from source node to destination node in the network, and several experiments for the constraints like network traffic, and intruder attacks are performed.

Work 2: We facilitate dynamic resource sharing between the peers, that is if a node requests for a piece of data from two other nodes, the requested data is split from each of the nodes and returned as the response.

We verified our solution through extensive simulations and they helped to prove that our proposal considerably outperforms state-of-the-art solutions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1.	P2P	Peer-to-peer Network
2.	DSR	Dynamic source routing protocol
3.	CBR	Constant Bit Rate Traffic

CHAPTER 1

1 INTRODUCTION

1.1 GENERAL

PEER-TO-PEER (P2P) networking is an emerging technique for next-generation network applications. P2P networks (or overlays) are application-level networks built on top of end systems, which provide message routing and delivery.

P2P networks can be largely classified into two categories, namely,

- ✓ Structured P2P networks
- ✓ Unstructured P2P networks

Although P2P networks like CAN, Chord, Pastry, and Tapestry can be structured in a well-organized fashion to guarantee the quality of services, unstructured P2P networks are widely deployed in the mass market.

A critical function offered by an unstructured P2P network is the broadcasting of a message. Since the peers participate in an unstructured overlay, the interconnection with one another is in a random fashion, there by resulting in lengthy communication between the peers so the packet delivery along the paths will be consuming more time.

In this analysis work, a wireless P2P network is considered, with the paths available from source to destination. The delay time is calculated for each of the paths and the path with the minimum delay is chosen as the minimal delay path.

We impose some constraints like network traffic and intruder attack, which tends to packet drop in that path so that an alternative path is provided for transmission.

Similarly in the case of an unknown attack (intruder), the network detects and removes the intruder and restarts the transmission.

1.2 OBJECTIVE

- ✓ To analyze the minimal delay path in an unstructured peer to peer network and perform some experiments imposing some constraints like network traffic, and intruder attack.

- ✓ To facilitate the dynamic resource sharing between the peers

CHAPTER 2

LITERATURE REVIEW

2.1 FEASIBILITY STUDY

2.1.1 CURRENT STATUS OF THE PROBLEM

2.1.1.1 ANALYSING MINIMAL DELAY PATH

In an unstructured peer to peer network there may be any number of nodes, and it may have numerous paths from the fixed source to destination to transmit any data. If we transmit any data randomly it may consume more time since it may be the longest path of travel and it may lead to packet loss also. And so it cannot withstand for any kind of unknown attacks or some other constraints.

2.1.1.2 DYNAMIC RESOURCE SHARING

In an unstructured peer to peer network, resource sharing is done between any two nodes using the flooding techniques, that may lead to inefficient search of data to share.

2.1.2 PROPOSED SYSTEM AND ADVANTAGES

2.1.2.1 ANALYSING MINIMAL DELAY PATH

With reference to the above problem, we propose an algorithm that computes the delay for each of the paths considered, the path with minimum delay is analyzed as the minimal delay path through which the packet transmission can be done quickly and efficiently without any packet loss. Although we impose the constraints like packet loss, intruder attack and show the performance analysis by means of metrics like drop, throughput, bit error rate by our simulation.

2.1.2.2 DYNAMIC RESOURCE SHARING

As discussed earlier, in an unstructured peer to peer network, resource sharing is done between two or more nodes directly. Here we propose a split up and add concept of fetching the required data from two nodes and giving to the requested node. That is a single peer will get the resource from two other peers.

ADVANTAGES

- ✓ Packet loss is avoided
- ✓ Quicker delivery of packets
- ✓ Efficient resource sharing

2.2 HARDWARE REQUIREMENTS

• Hard disk	: 320 GB
• RAM	: 4 GB
• Processor Speed	: 2.66 GHz
• Processor	: Core2Duo processor

2.3 SOFTWARE REQUIREMENTS

• Operating System	: Fedora8.0
• Language	: C++ and TCL Scripting

2.4 SOFTWARE OVERVIEW

2.4.1 NS-2 Simulator

2.4.1.1 About NS-2

Network simulator 2 is used as the simulation tool in this project. NS was chosen as the simulator partly because of the range of features it provides and partly because it has an open source code that can be modified and extended. There are different versions of NS and the latest version is NS-2.1b9a while NS-2.1b10 is under development

Network simulator (NS) is an object–oriented, discrete event simulator for networking research. NS provides substantial support for simulation of TCP, routing and multicast protocols over wired and wireless networks. The simulator is a result of an ongoing effort of research and development. Even though there is a considerable confidence in NS, it is not a polished product yet and bugs are being discovered and corrected continuously.

NS is written in C++, with an OTcl interpreter as a command and configuration interface. The C++ part, which is fast to run but slower to change, is used for detailed protocol implementation. The OTcl part, on the other hand, which runs much slower but can be changed very quickly, is used for simulation configuration. One of the advantages of this split-language program approach is that it allows for fast generation of large scenarios. To simply use the simulator, it is sufficient to know OTcl. On the other hand, one disadvantage is that modifying and extending the simulator requires programming and debugging in both languages.

NS can simulate the following:

1. **Topology:** Wired, wireless
2. **Scheduling Algorithms:** RED, Drop Tail,
3. **Transport Protocols:** TCP, UDP

4. **Routing:** Static and dynamic routing

5. **Application:** FTP, HTTP, Telnet, Traffic generators

NS-2 is an open source simulation tool running on Unix-like operating systems. And as said above it is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP, SRM, CBR over wired, wireless and satellite networks. It has many advantages that make it a useful tool, such as support for multiple protocols and the capability of graphically detailing network traffic.

Additionally, NS-2 supports several algorithms in routing and queuing. LAN routing and broadcasts are part of routing algorithms. Queuing algorithm includes fair queuing, deficit round robin and FIFO. NS-2 started as a variant of the REAL network simulator IN 1989. REAL is a network simulator originally intended for studying the dynamic behavior of flow and congestion control schemes in packet-switched data network. In 1995 ns development was supported by Defense Advanced Research Projects Agency DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. The wireless codes from the UCB Daedelus and CBU Monarch projects and Sun Microsystems have added the wireless capabilities to NS-2. Valery Naumov proposed a list-based improvement for NS-2 involving maintaining a double linked list to organize mobile nodes based on their X-coordinates.

When sending a packet, only those neighbor nodes are considered, which are within a circle corresponding to the carrier-sense threshold energy level, below which a node cannot hear the packet. Compared to the original version, where all nodes in the topology are considered, its considerable gain in run time, performance goes down by about 4 to 20 times, depending on the size of the topology. The larger the topology with larger number of nodes, yields to greater improvement seen with the list-based implementation. NS-2 also builds and runs windows with Cygwin. Simple scenarios should run on any reasonable machine; however, very large scenarios benefit from large amounts of memory and fast CPU's.

2.4.1.2 NS-2, implementing languages

NS-2 is basically written in C++, with an Otcl (Object Tool Command Language) interpreter as a front-end. It supports a class hierarchy in C++, called compiled hierarchy and a similar one within the OTcl interpreter, called interpreter hierarchy. Some objects are completely implemented in C++, some others in Tcl and OTcl and some others are implemented in combination of these. For them, there is one-to-one correspondence between classes of the two hierarchies. But why should one use two languages? The simulator can be viewed as doing 2 different things. While one on hand detailed simulations of protocols are required, we also need to be able to vary the parameters or configurations and quickly explore the changing scenarios. For the first case we need a system programming language like C++ that effectively handles bytes, packet headers and implements algorithms efficiently. But for the second case iteration time is more important than the run- time of the part of task. A scripting language like Tcl accomplishes this.

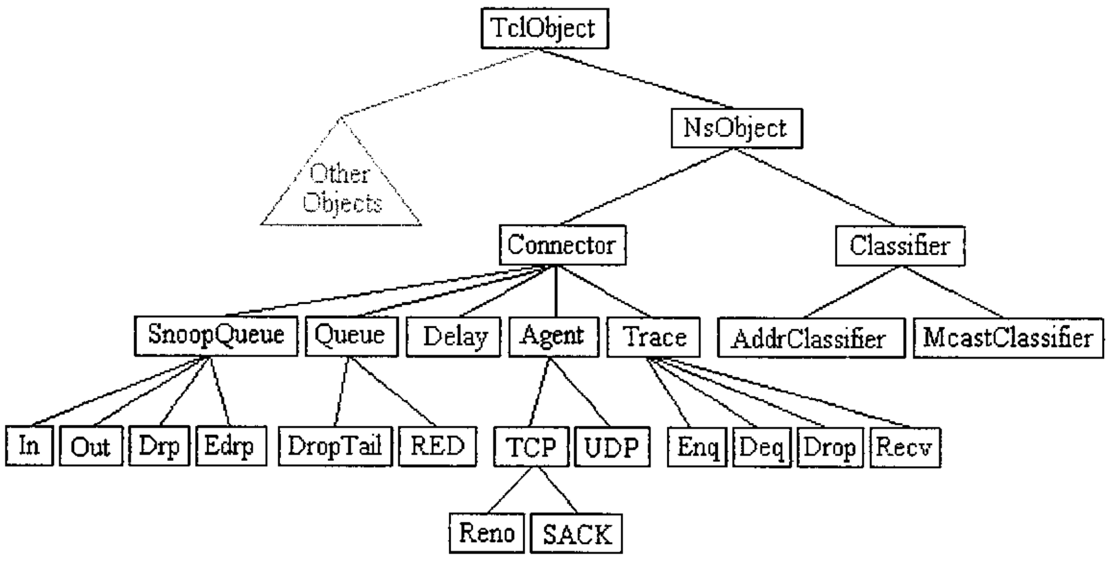


Figure 2.1: OTcl Class Hierarchy

2.4.1.3 Architecture of NS-2

As already mentioned above, NS-2 is an object oriented, discrete event simulator. There are presently five schedulers available in the simulator, each of which is implemented by using a different data structure: a simple linked-list, heap, calendar queue (default) and a special type called 'realtime'. The scheduler runs by selecting the next earliest event , executing it to completion, and returning to execute the next event. The units of time used by the scheduler are seconds. An event is handled by calling the appropriate Handler Class. The most important Handler is NSObject with TclObject as its twin in the OTcl world. They provide all the basic function group is mainly used. For handling OTcl statements in C++ Ns Objects NsObjects provide the so-called command function. NsObject is the parent class for some important classes as the classifier, the Connector and the TraceFile class.

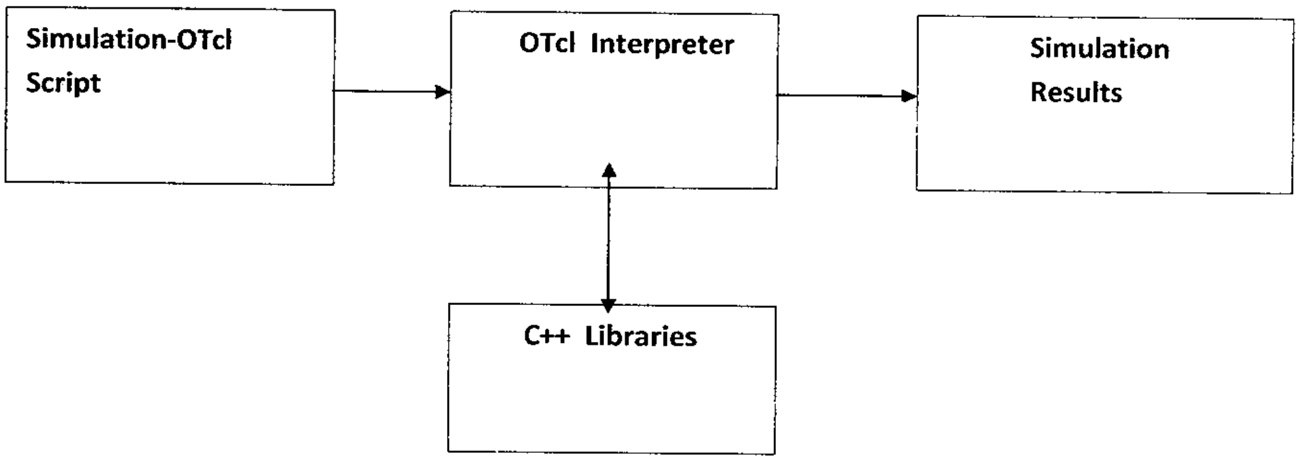


Figure 2.2: Block diagram of Architecture of NS-2

2.4.1.4 Usage of NS-2

An NS-2 simulation is controlled by a TCL scripts, which contains all necessary parameters and configurations. Additionally the 'opt' parameters within the TCL script can be modified from the command line as shown below.

Ns script.tcl –nn 100 –x 5000 –y 5000 –stop 800

-tr out.tr –sc mov –cp traffic

The TCL script specifies the path of movement and connection files to be loaded as well as the path to the trace files, usually a nam and a tr file, which are the product of a simulation.

2.4.1.5 TCL Simulation scripts

As already mentioned a TCL script is used for configuring a simulation with parameters. It consists of several important parts:

- ✓ Physical and protocol specifications
- ✓ Node creation and movement (mostly imported from a separate file, the so called movement, scene or scenario file)
- ✓ Node communication (mostly imported from a separate file, the so called traffic, connection or communication file)

2.4.1.6 Trace Files

One gets two trace files as a result of a simulation: a normal trace file (created by $ns_trace-all commands) and a nam trace file ($ns_nam-trace all). The nam trace file is a subset of a normal trace file with the suffix ".nam". It contains information for visualizing packets flow and node movement for use with the homonymous nam tool. Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet trace data. The design theory behind nam was to create an animator that is able to read large animation data sets and extensible enough, so that it could be used in different network visualization situations. For studying protocol behavior one has to refer to the normal trace file with the suffix ".tr" .It contains all requested trace data produced by a simulation. In the TCL configuration script one can tell the simulator which kind of traces information shall be printed out: agent, route and mac trace. They can separately be turned on or off for every node

$ns_node-config -agentTrace ON/OFF

$ns_node-config -routerTrace ON/OFF

$ns_node-config -macTrace ON/OFF

2.4.1.7 Mobile networking in NS

The wireless model essentially consists of the mobile node at the core with additional supporting features that allows simulations of multi-hop ad-hoc networks, wireless LANs etc. The mobile node object is a split object. The C++ class Mobile Node is derived from parent class Node. A mobile node thus is the basic Node object with added functionalities of a wireless and mobile node like ability to move within a given topology, ability to receive and transmit signals to and from a wireless channel etc. A major difference between them is that a mobile node is not connected by means of Links to other nodes or mobile nodes.

Mobile node is the basic nsnode object with added functionalities like movement, ability to transmit and receive on a channel that allows it to be used to create mobile, wireless simulation environments. The class Mobile Node is derived from the base class Node. The four ad-hoc routing protocols that are currently supported are, Dynamic Source Routing (DSR), Temporally ordered Routing Algorithm (TORA) and Adhoc On-demand Distance Vector (AODV).

The general structure for defining a mobile node in ns2 is described as follows:

```
$ns node-config -adhocRouting $opt (adhocRouting)

-IIType $opt (II)

-macType $opt (mac)

-ifqType $opt (ifq) -ifqLen $opt (ifqlen)

-antType $opt (ant)

-propInstance [new $opt (prop) -phyType $opt (netif)

-channel [new $opt (chan)]

-topoInstance $topo
```

-wiredRouting OFF

-agentTrace ON

-routerTrace OFF

-macTrace OFF

The above API configures for a mobile node with all the given values of ad hoc-routing protocol, network stack, channel, topography, propagation model, with wired routing turned on or off (required for wired-cum-wireless scenarios) and tracing turned on or off at different levels (router, mac, agent).

2.4.1.8 Limitations of NS-2

A simulator model of a real-world system is necessarily a simplification of the real-world system itself. Especially for WLAN simulation one has to be very generous, because there are tremendously many quick changing parameters, which cannot be handled up to now and in the near future. There are also some practical limitations. We found out that the maximum number of mobile nodes, ns-2 allows is 16250(nn_max). Processor speed is also a problem. For example, on a 2 GHz machine a 500 node simulation, with the random way point movement model, in an area of 5x1 km2 with connections over 600 seconds, easily takes several days and requires easily a GB of RAM.

CHAPTER 3

METHODOLOGY USED

3.1 PROTOCOL USED IN THE ANALYSIS

Here we use Dynamic Source Routing (DSR) protocol for the transmission of data in the desired path. A wireless unstructured peer to peer network is just similar to wireless ad hoc network, so the mechanism is implemented here as well. Detailed description of DSR is given below.

a) DYNAMIC SOURCE ROUTING PROTOCOL (DSR)

The Dynamic Source Routing protocol (DSR) is a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. With DSR, the network is completely self-organizing and self-configuring, requiring no existing network infrastructure or administration. Network nodes (computers) co-operate each other in forwarding packets to allow communication over multiple "hops" between nodes not directly within wireless transmission range of one another. As the nodes in the network move about or join or leave the network, the wireless transmission conditions such as sources of interference change and others, in routing is automatically determined and maintained by the DSR routing protocol. Since the number or sequence of intermediate hops needed to reach any destination may change at any time, the resulting network topology may be quite rich and rapidly changing.

The DSR protocol allows nodes to dynamically discover a source route across multiple hops to any destination in the ad hoc network. Each data packet carries the information in its header, allowing packet routing to be trivially loop-free and avoiding the need for updating routing information in the intermediate nodes through which the packet is forwarded. By including this source route in the header of each data packet, other nodes can forward the message in the similar way.

b) PHASES OF DSR

There are two phases in Dynamic source routing they are

- ✓ Route Discovery
- ✓ Route Maintenance

ROUTE DISCOVERY

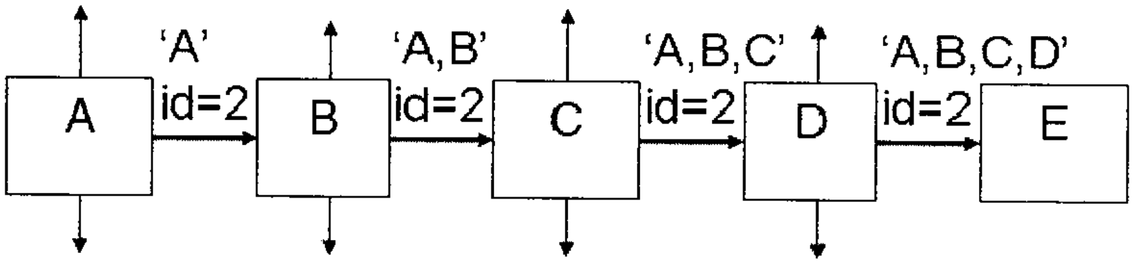


Figure: 2.3 Illustration of route discovery

If node A has in his Route Cache a route to the destination E, this route is immediately used. If not, the Route Discovery protocol is started:

- ✓ Node A (initiator) sends a Route request packet by flooding the network.
- ✓ If node B has recently seen another Route request from the same target or if the address of node B is already listed in the Route record, Then node B discards the request!
- ✓ If node B is the target of the Route Discovery, it returns a Route reply to the initiator. The Route reply contains a list of the "best" path from the initiator to the target. When the initiator receives this RouteReply, it caches this route in its Route Cache for use in sending subsequent packets to this destination.
- ✓ Otherwise node B isn't the target and it forwards the Route Request to his neighbors (except to the initiator).

Path-finding-process: Route Request & Route Reply:

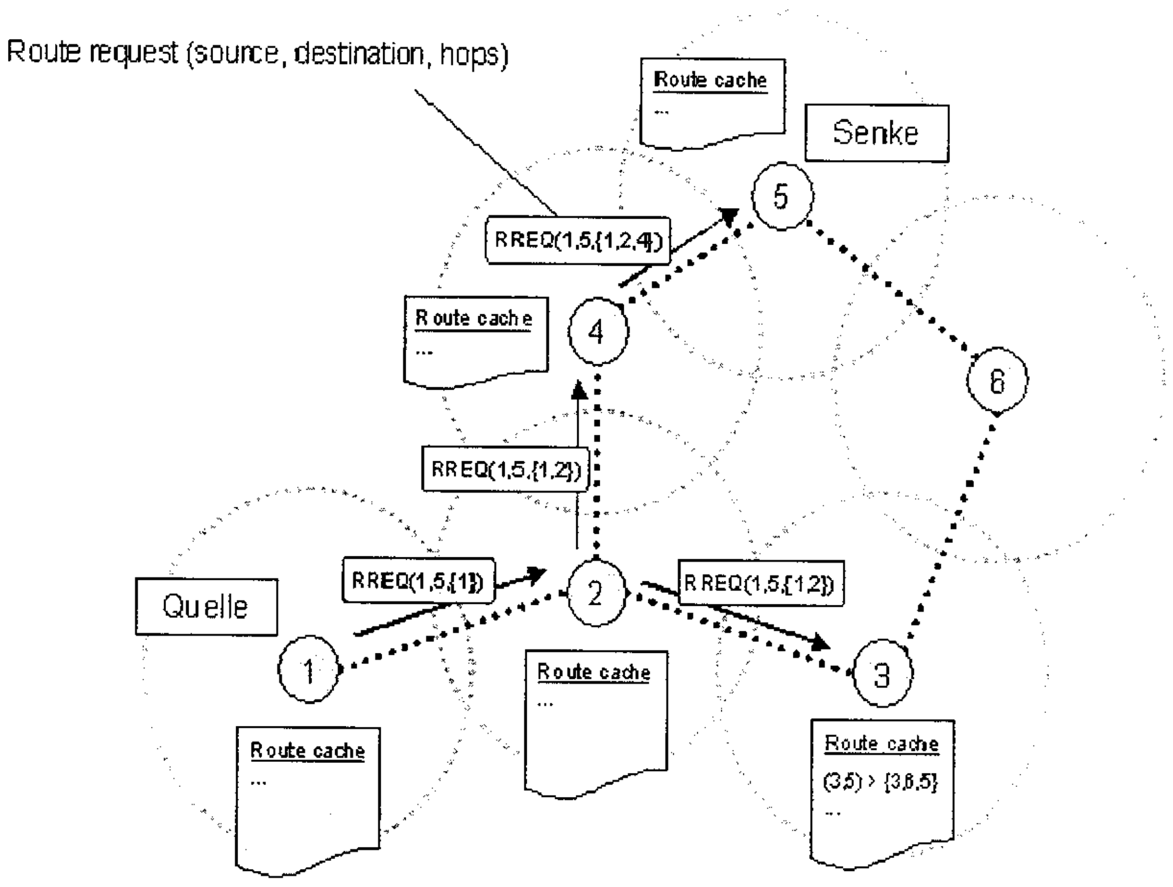


Figure 2.4 : Illustration of route request

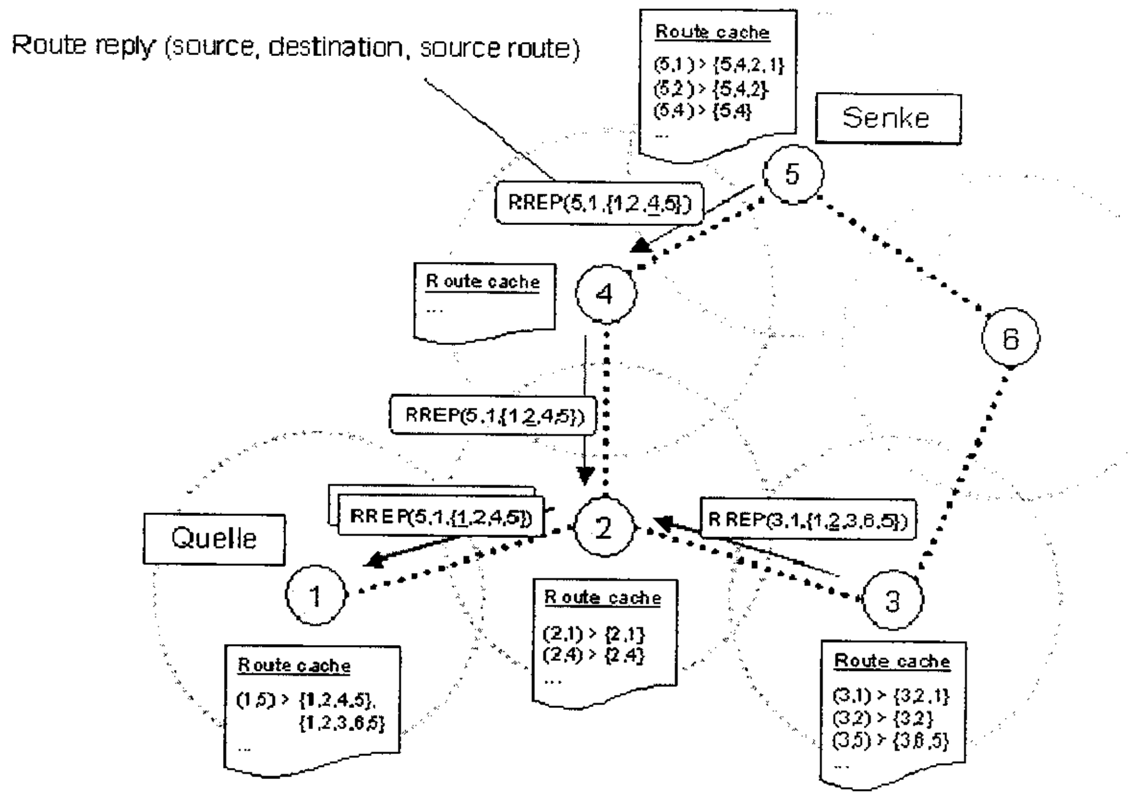


Figure 2.5: Illustration of route reply

ROUTE MAINTENANCE

In DSR every node is responsible for confirming that the next hop in the Source Route receives the packet. Also each packet is only forwarded once by a node (hop-by-hop routing). If a packet can't be received by a node, it is retransmitted up to some maximum number of times until a confirmation is received from the next hop.

Only if retransmission results then in a failure, a Route error message is sent to the initiator that can remove that Source Route from its Route Cache. So the initiator can check his Route Cache for another route to the target. If there is no route in the cache, a Route request packet is broadcasted.

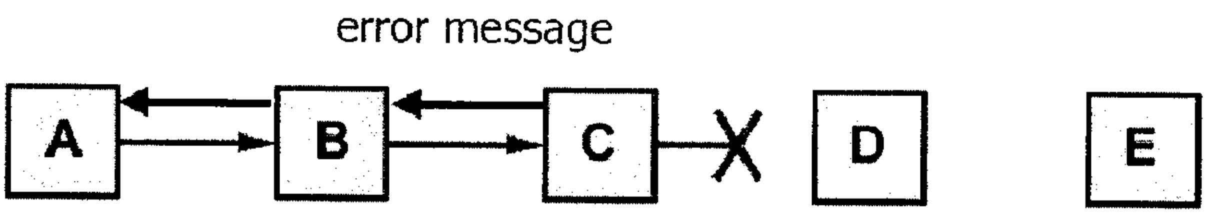


Figure 2.6: Depicting the link failure

- ✓ If node C does not receive an acknowledgement from node D after some number of requests, it returns a Route error to the initiator A.
- ✓ As soon as node receives the Route error message, it deletes the broken-link-route from its cache. If A has another route to E, it sends the packet immediately using this new route.
- ✓ Otherwise the initiator A is starting the Route Discovery process again.

c) ADVANTAGES

Reactive routing protocols have no need to periodically flood the network for updating the routing tables like table-driven routing protocols do. Intermediate nodes are able to utilize the Route Cache information efficiently to reduce the control overhead. The initiator only tries to find a route (path) if actually no route is known (in cache). Current and bandwidth saving because there are no hello messages needed (beacon-less).

d) DISADVANTAGES

The Route Maintenance protocol does not locally repair a broken link. The broken link is only communicated to the initiator. The DSR protocol is only efficient in MANETs with less than 200 nodes. Problems appear by fast moving of more hosts, so that the nodes can only move around in this case with a moderate speed. Flooding the network can cause collusions between the packets. Also there is always a small time delay at the begin of a new connection because the initiator must first find the route to the target.

3.2 TRANSMISSION OF DATA PACKETS

To facilitate the transmission of data packets from source node to the destination node through the network, protocols like TCP,UDP and others are used. Here in this wireless network transmission is done by constant bit rate traffic(CBR) without combining with any other protocols.

3.3 RESOURCE SHARING

To facilitate the resource sharing, the information of each nodes are stored separately in their respective files. **Split Up and Add** methodology is used to fetch the requested data from two different nodes and give response to the requested node.

3.4 AWK SCRIPTING

AWK is a general-purpose programming language designed for processing of text files. AWK refers to each line in a file as a record. Each record consists of fields, each of which is separated by one or more spaces or tabs. Generally, AWK reads data from a file consisting of fields of records, processes those fields with certain arithmetic or string operations, and outputs the results to a file as a formatted report. To process an input file, AWK follows an instruction specified in an AWK script. An AWK script can be specified at the command prompt or in a file.
While the strength of the former is the simplicity, that of the latter is the functionality. In the latter, the programming functionalities such as variables, loop, and conditions can be included into an AWK script to perform desired actions.

CHAPTER 4

SIMULATION AND PERFORMANCE EVALUATION

4.1 MODULE DESCRIPTION

4.1.1 Path Analysis

Generally in an unstructured peer to peer network, the packet transmission from source to destination can be done through the peers in all possible directions ,that may result in lengthy communication.

In order to resolve this, a simple algorithm is proposed to find the best path from the available alternatives for transmission.

First it checks for the paths (5 paths is chosen) available, then it finds the delay for each of the paths,

Delay = (Time at which the first packet sent) - (Time at which the last packet is received)

After the delay calculation is made for each of the paths, the minimal delay path is analyzed.

4.1.2 Selection of back up route

The same procedure is repeated for the modified network, to ensure that the algorithm works for any kind of network.

Here all of the paths are varied and the delay is also varied. Minimal delay path is analyzed for this network also.

Considering this modified network, the network traffic constraint is imposed. In this scenario path 5 is the minimal delay path.

Due to network congestion in path 5, there occurs heavy packet drop leading to improper delivery, in such case

A simple comparison procedure is done to compare the delays of each paths and to allocate the next minimal delay path for transmission. (path 2 is chosen as the second minimal delay path).

4.1.3 Intruder Attack

In the previous module path 2 is chosen as the second minimal delay path. And if suppose an external node on its entering to the network affects the packet transmission. It is actually an unknown node to the network (Intruder). A simple mechanism is employed to detect and remove the intruder from the network. Then the usual transmission is restarted in path 2.

4.1.4 Dynamic resource sharing

In this module, a simple network is constructed in a ringed fashion. Here we implement split up and add concept of unstructured peer to peer network, to do dynamic resource sharing between the peers.

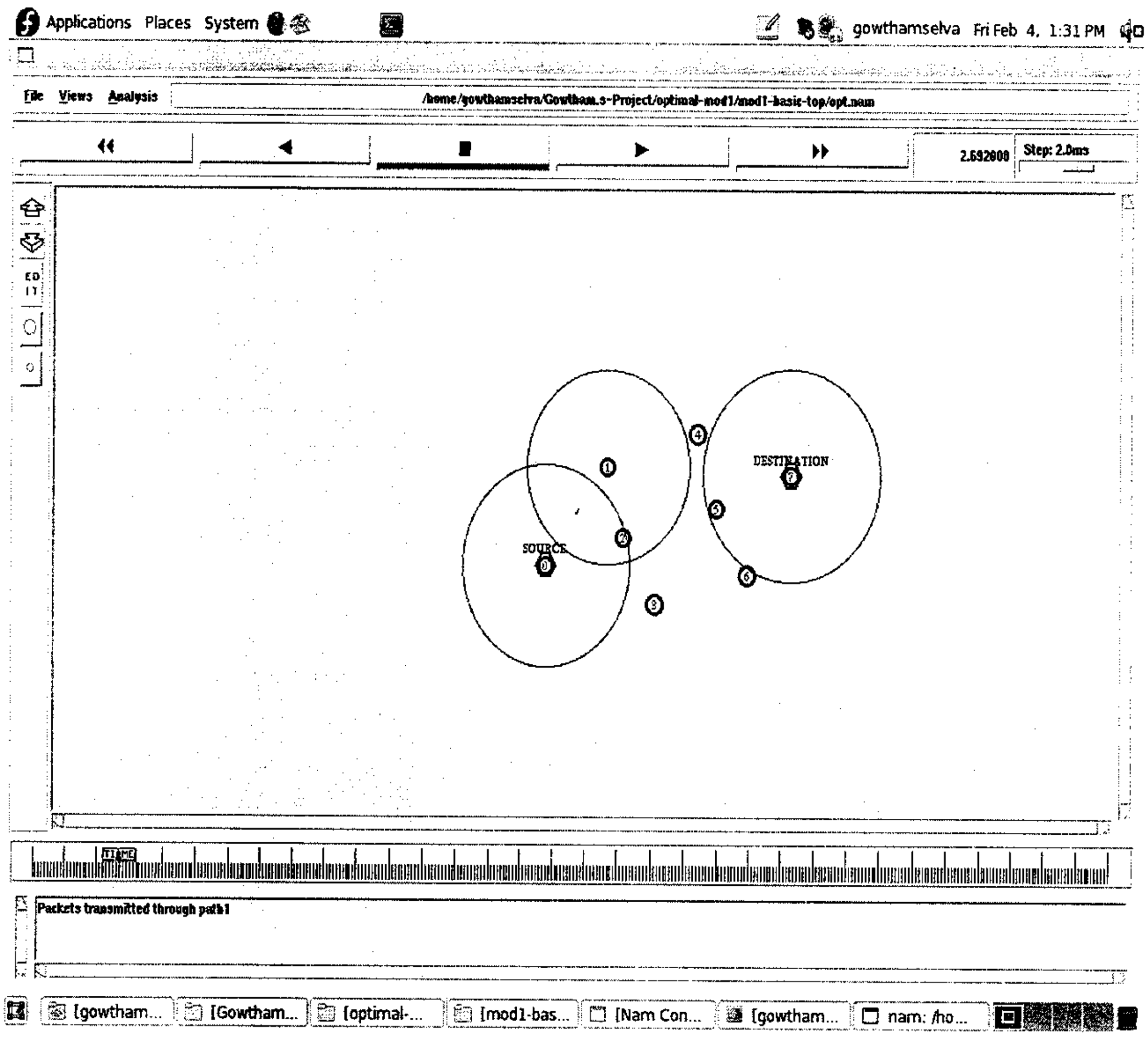
Each node should maintain a database to keep its information stored in it. And each of the nodes should have the database of same size.

One of the peer requests for the resource from two other peers. Both the peers respond by splitting the data from each of itself and adds up in a separate file for the users view.

4.2 SIMULATION RESULTS

4.2.1 PATH ANALYSIS EXAMINING AVAILABLE PATHS

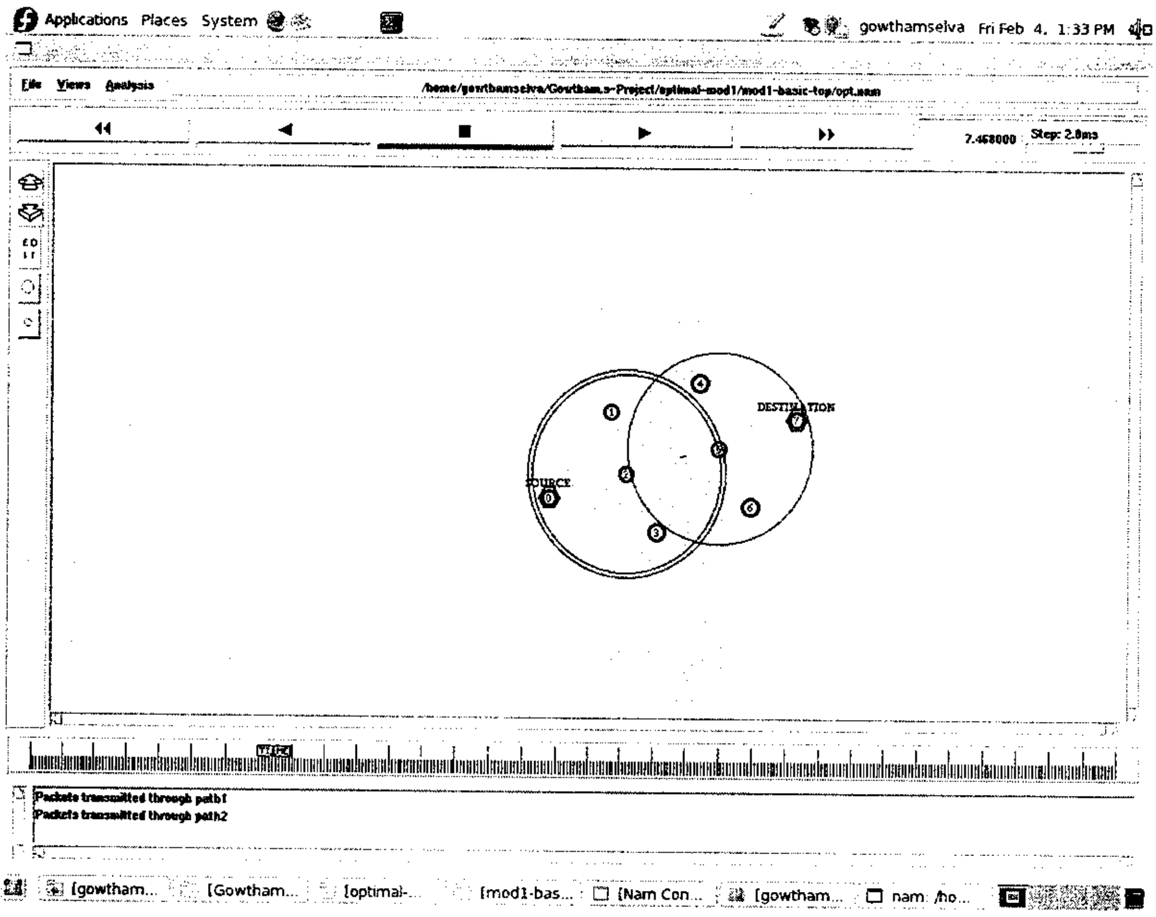
The following are the screen shots of the analysis of minimal delay path from the available five paths.



/home/gowthamselva/Gowtham.s-Project/optimal-mod1/mod1-basic-top/opt.nam

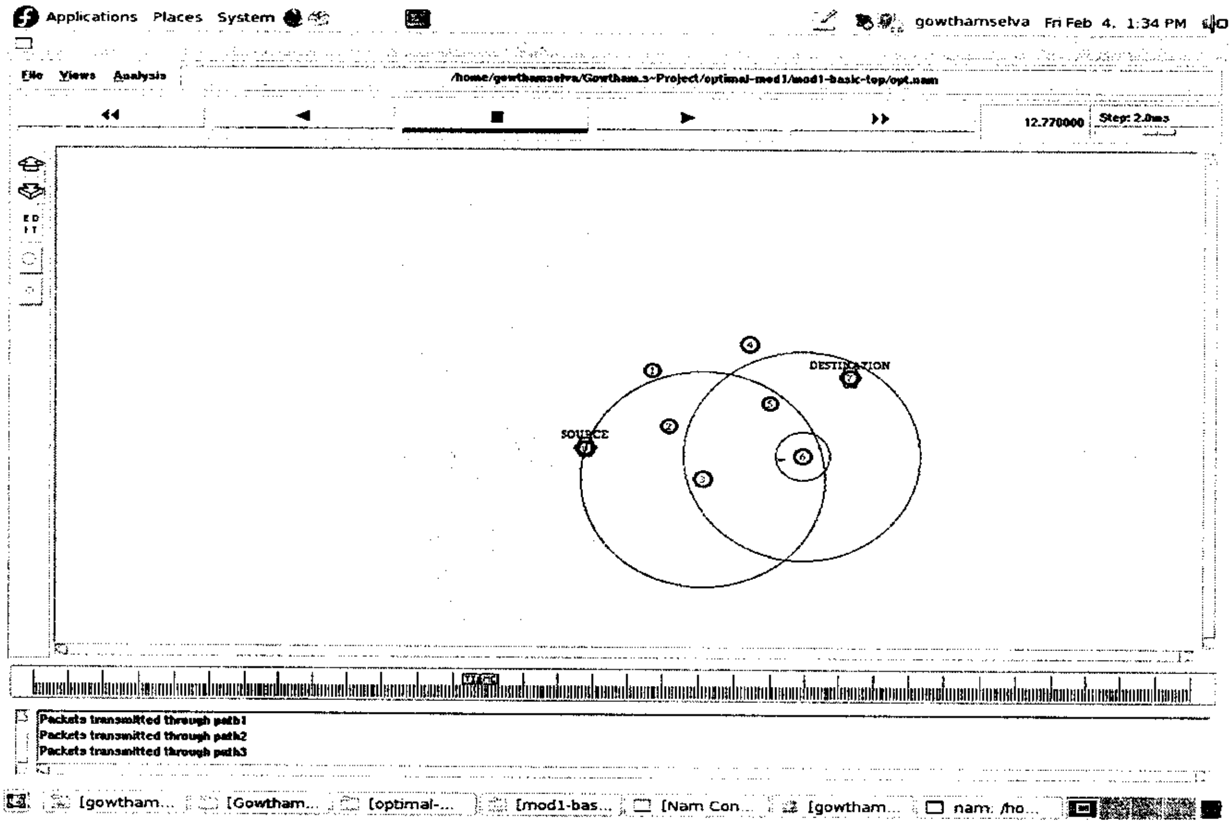
Packets transmitted through path1

Figure 2.7: Packets transmitted through path1 for testing



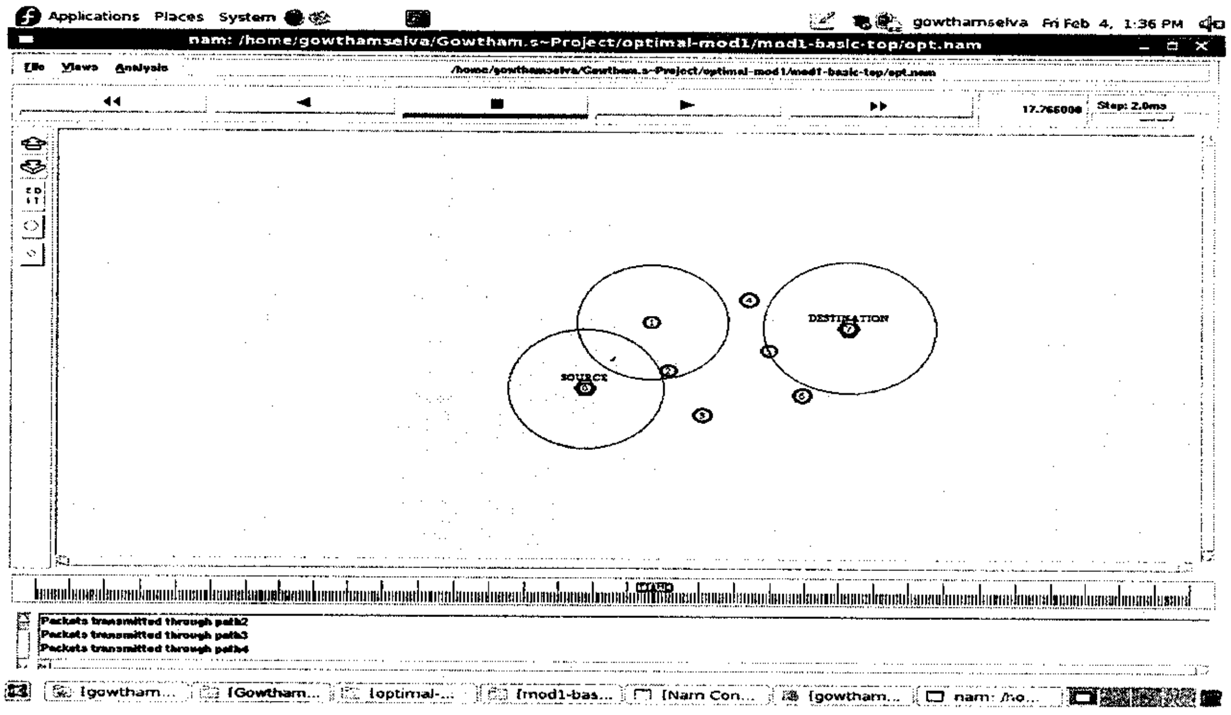
```
Packets transmitted through path1
Packets transmitted through path2
```

Figure 2.8: Packets transmitted through path2 for testing


Applications Places System
gowthamselva Fri Feb 4, 1:34 PM
File Views Analysis
/home/gowthamselva/Gowtham.s~Project/optimal-mod1/mod1-basic-top/opt.nam
12.770000
Step: 2.0ms
DESTINATION
SOURCE
[gowtham... [Gowtham... [optimal-... [mod1-bas... [Nam Con... [gowtham... nam: /ho...

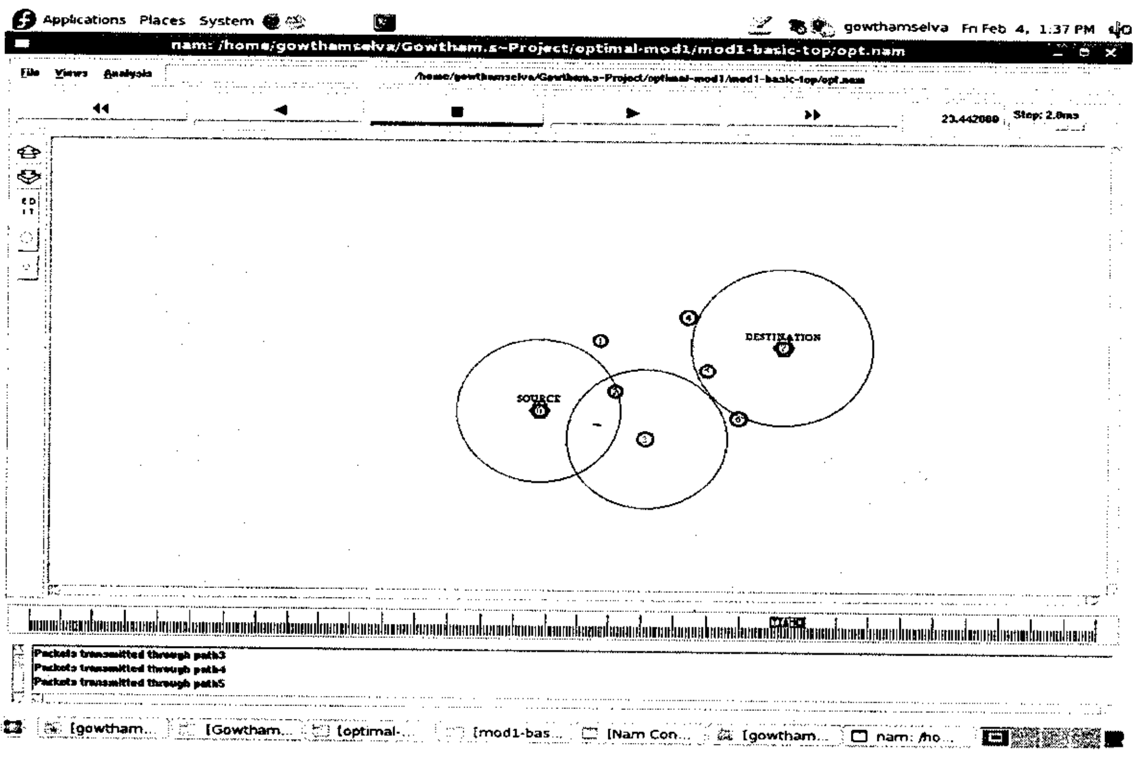
```
Packets transmitted through path1
Packets transmitted through path2
Packets transmitted through path3
```

Figure 2.9: Packets transmitted through path3 for testing


Applications Places System
gowthamselva Fri Feb 4, 1:36 PM
nam: /home/gowthamselva/Gowtham.s~Project/optimal-mod1/mod1-basic-top/opt.nam
File Views Analysis
/home/gowthamselva/Gowtham.s~Project/optimal-mod1/mod1-basic-top/opt.nam
17.766000
Step: 2.0ms
DESTINATION
SOURCE
[gowtham... [Gowtham... [optimal-... [mod1-bas... [Nam Con... [gowtham... nam: /ho...

```
Packets transmitted through path2
Packets transmitted through path3
Packets transmitted through path4
```

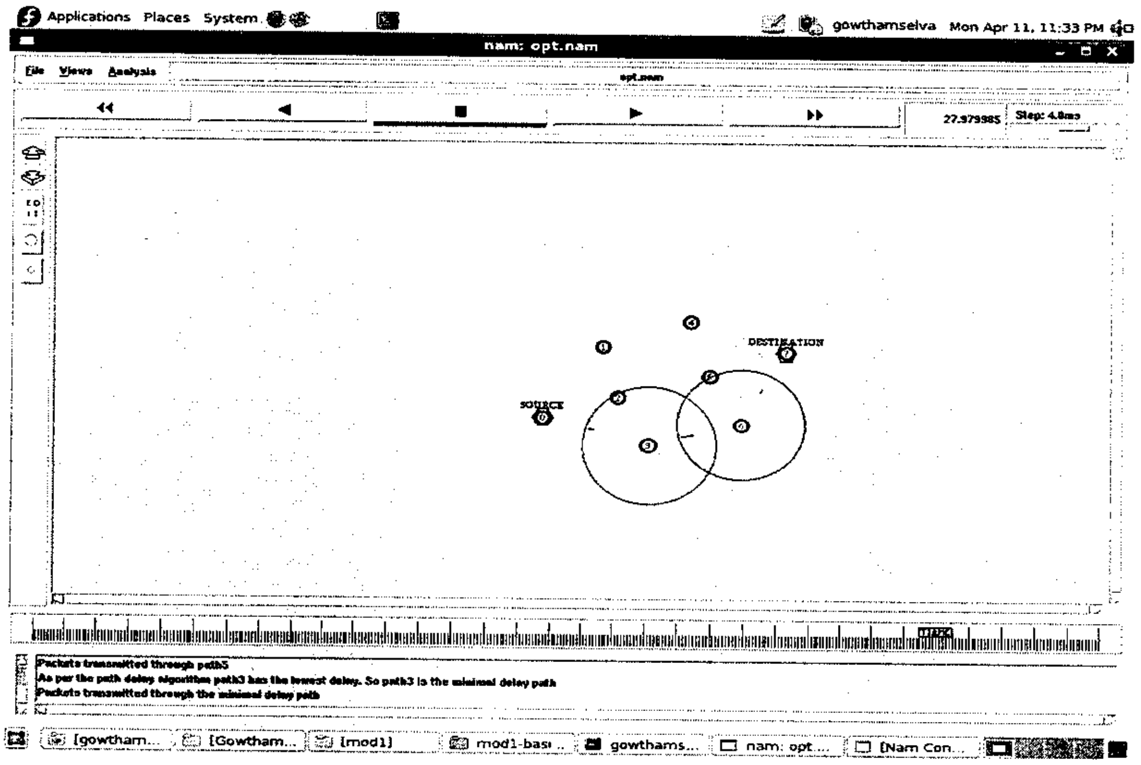
Figure 3.0: Packets transmitted through path4 for testing



Applications Places System
gowthamselva Fri Feb 4, 1:37 PM
nam: /home/gowthamselva/Gowtham.s~Project/optimal-mod1/mod1-basic-top/opt.nam
File Views Analysis
/home/gowthamselva/Gowtham.s~Project/optimal-mod1/mod1-basic-top/opt.nam
23.442080
Step: 2.0ms
DESTINATION
SOURCE
[gowtham...
[Gowtham...
[optimal-...
[mod1-bas...
[Nam Con...
[gowtham...
nam: /ho...

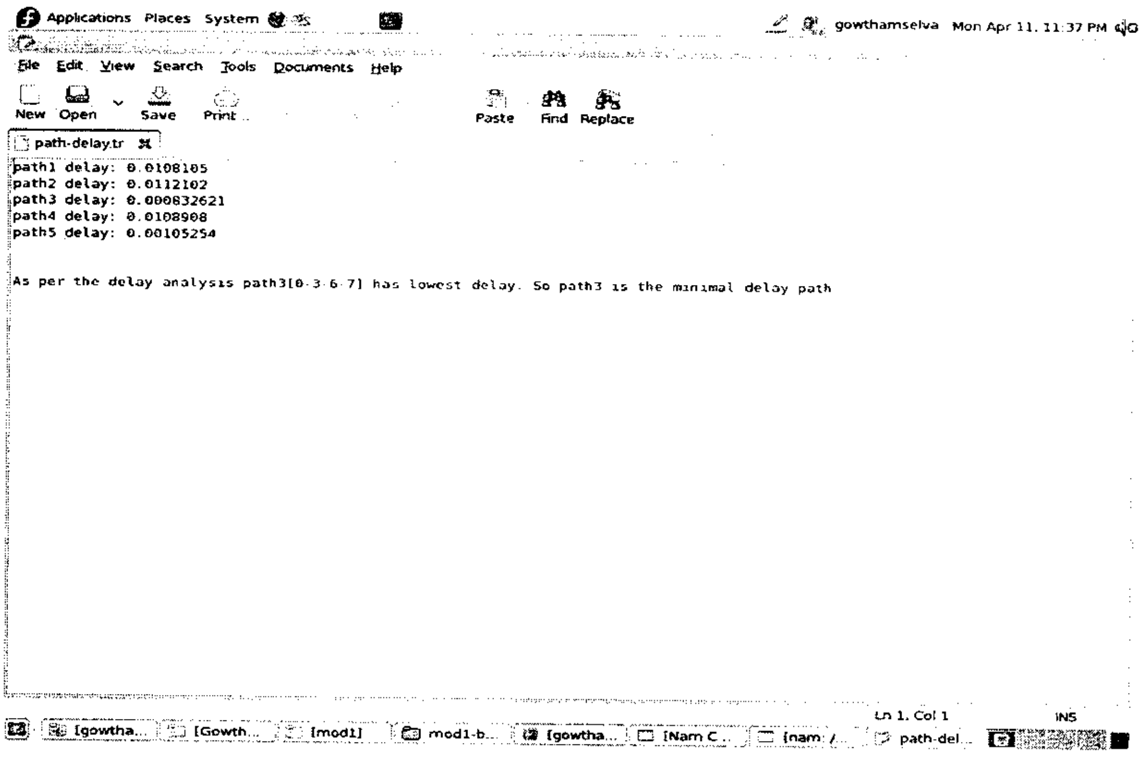
```
Packets transmitted through path3
Packets transmitted through path4
Packets transmitted through path5
```

Figure 3.1: Packets transmitted through path5 for testing



```
Packets transmitted through path5
As per the path delay algorithm path3 has the lowest delay. So path3 is the minimal delay path
Packets transmitted through the minimal delay path
```

Figure 3.2: Path analysis for existing network

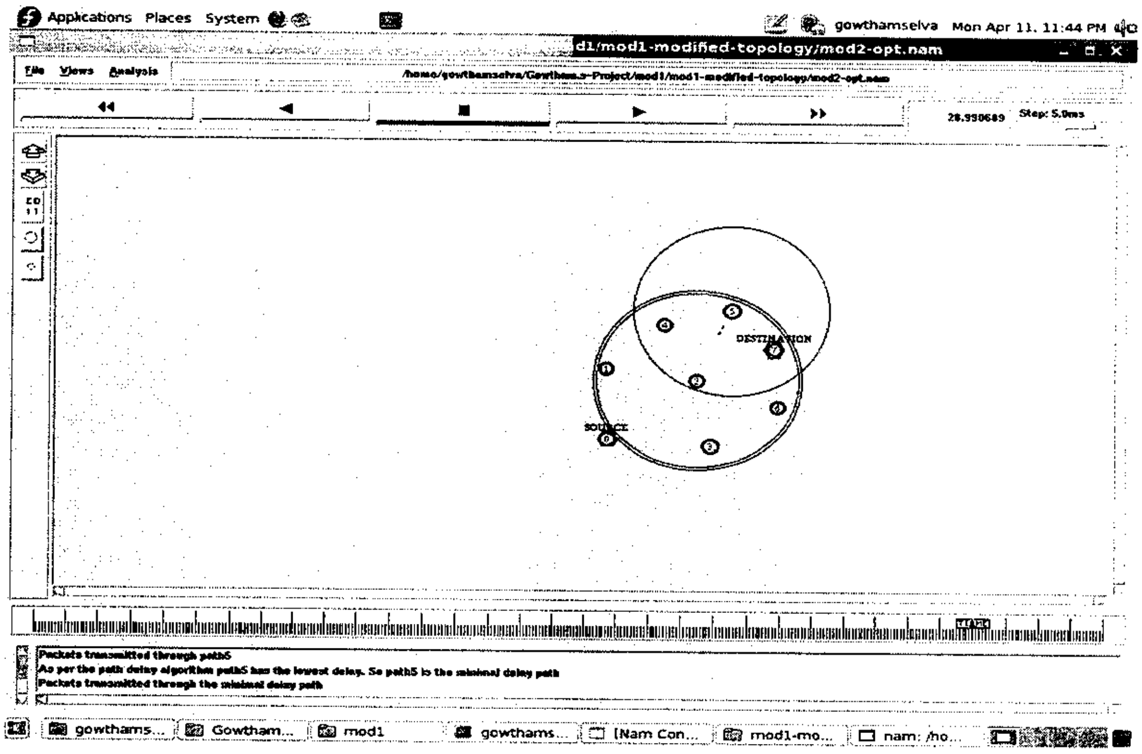


```
path1 delay: 0.0108105
path2 delay: 0.0112102
path3 delay: 0.000832621
path4 delay: 0.0108908
path5 delay: 0.00105254


As per the delay analysis path3[0-3-6-7] has lowest delay. So path3 is the minimal delay path
```

Applications Places System
gowthamselva Mon Apr 11, 11:37 PM
File Edit View Search Tools Documents Help
New Open Save Print
Paste Find Replace
path-delay.tr
Ln 1, Col 1
INS
[gowtha... [Gowth... [mod1] mod1-b... [gowtha... [Nam C... [nam: /... path-del...

Figure 3.3: Display of delay computation for each path

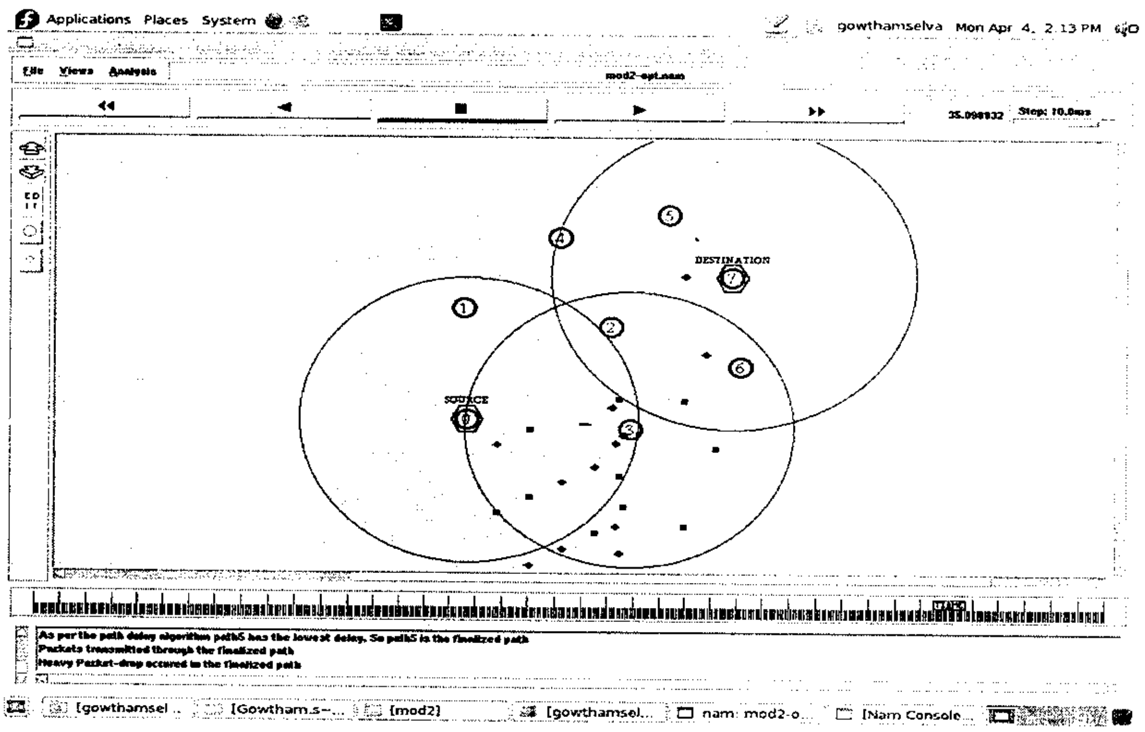


```
Packets transmitted through path5
As per the path delay algorithm path5 has the lowest delay. So path5 is the minimal delay path
Packets transmitted through the minimal delay path
```

Applications Places System
gowthamselva Mon Apr 11, 11:44 PM
d1/mod1-modified-topology/mod2-opt.nam
File Views Analysis
SOURCE
DESTINATION
Step: 5.0ms
gowthams... Gowtham... mod1 gowthams... [Nam Con... mod1-mo... nam: /ho...

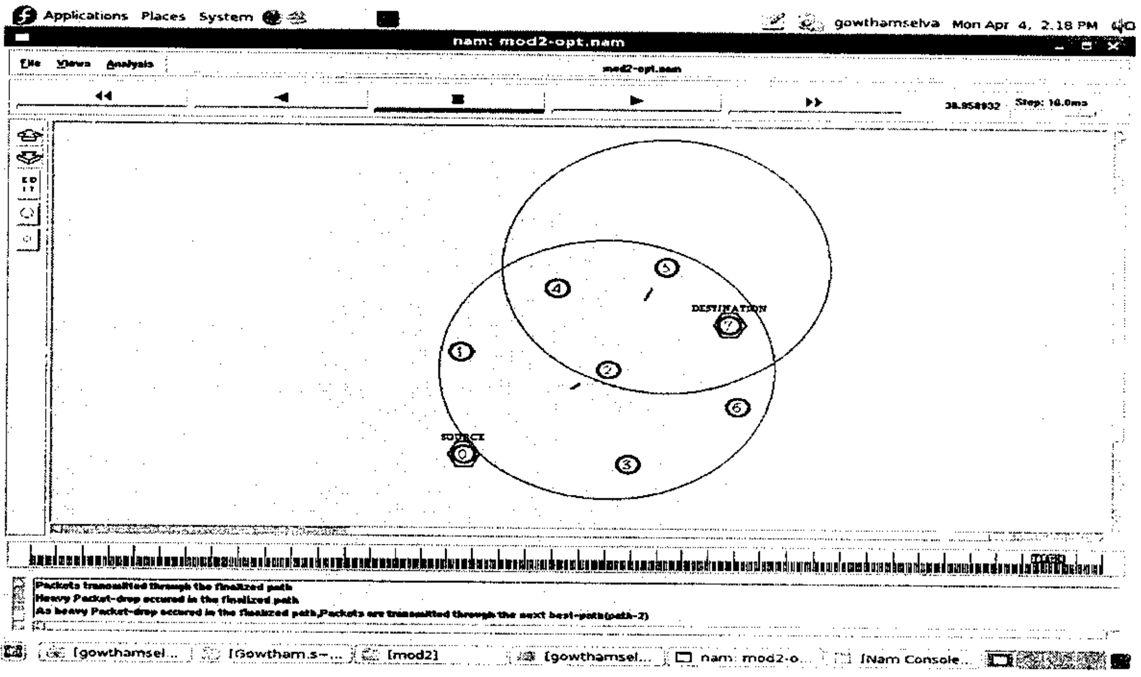
Figure 3.4: Path analysis for modified network

4.2.2 SIMULATION OF PACKET DROP SCENARIO



```
As per the path delay algorithm path5 has the lowest delay. So path5 is the finalized path
Packets transmitted through the finalized path
Heavy Packet-drop occured in the finalized path
```

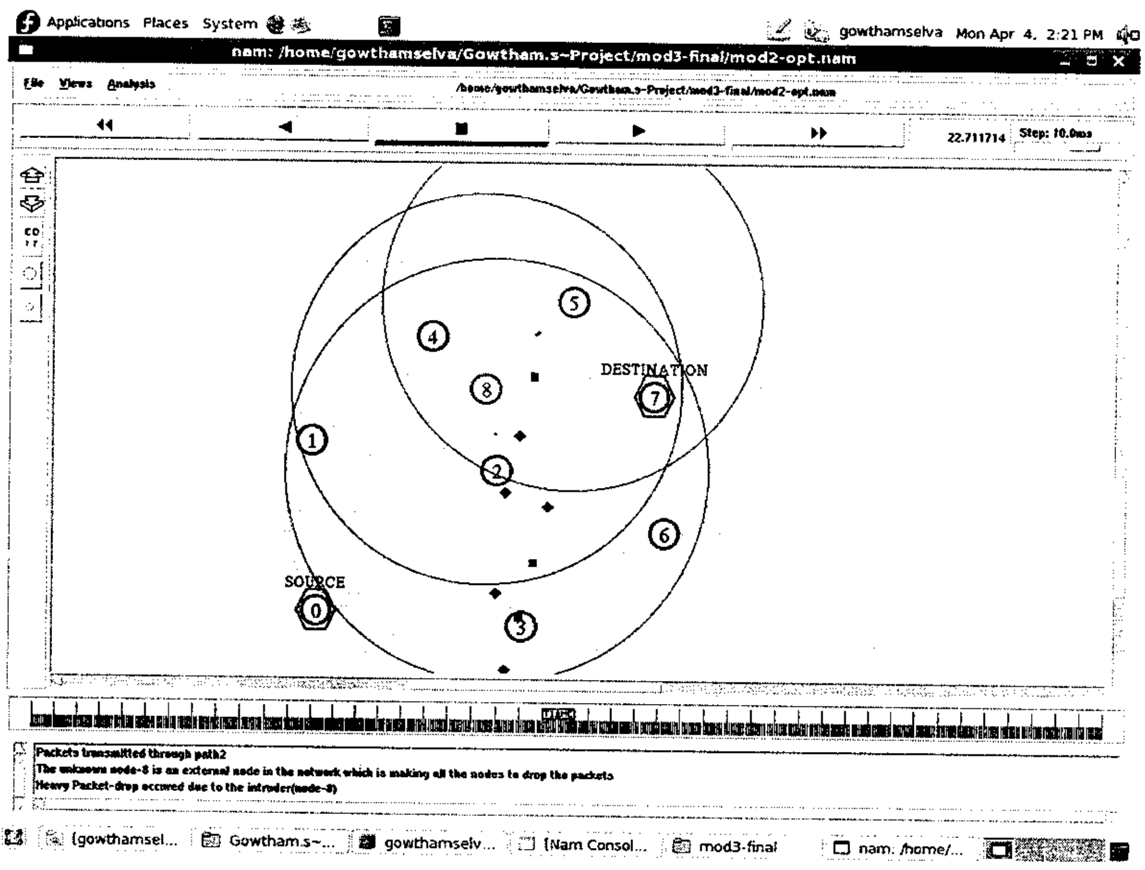
Figure 3.5: Depicting the packet drop in path5



```
Packets transmitted through the finalized path
Heavy Packet-drop occured in the finalized path
As heavy Packet-drop occured in the finalized path,Packets are transmitted through the next best-path(path-2)
```

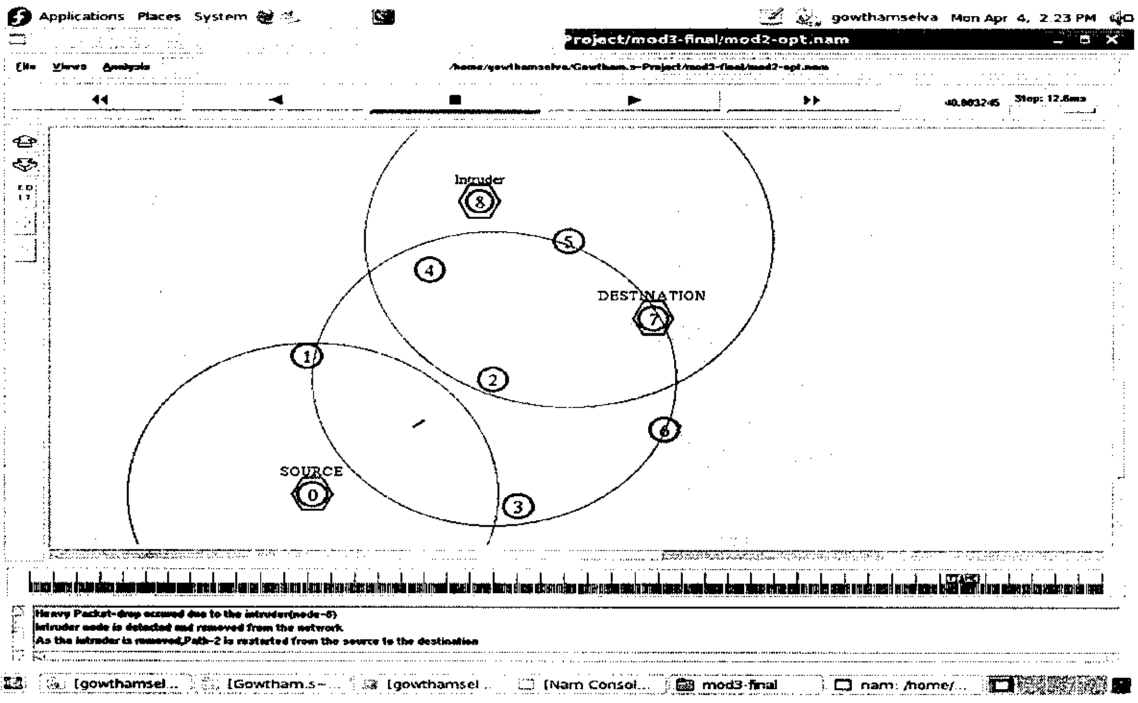
Figure 3.6: Depicting the restart transmission through path2

4.2.3 INTRUDER ATTACK SCENARIO


Applications Places System
gowthamselva Mon Apr 4, 2:21 PM
nam: /home/gowthamselva/Gowtham.s~Project/mod3-final/mod2-opt.nam
File Views Analysis
/home/gowthamselva/Gowtham.s~Project/mod3-final/mod2-opt.nam
22.711714
Step: 10.0ms
DESTINATION
SOURCE
0
1
2
3
4
5
6
7
8
(gowthamsel...
Gowtham.s~...
gowthamselv...
[Nam Consol...
mod3-final
nam: /home/...

```
Packets transmitted through path2
The unknown node-8 is an external node in the network which is making all the nodes to drop the packets
Heavy Packet-drop occured due to the intruder(node-8)
```

Figure 3.7: Depicting the intruder attack scenario



Heavy Packet-drop occured due to the intruder(node-8)
Intruder node is detected and removed from the network
As the intruder is removed,Path-2 is restarted from the source to the destination

Figure 3.8: Depicting the restart transmission after the attack

4.2.4 RESOURCE SHARING SCENARIO

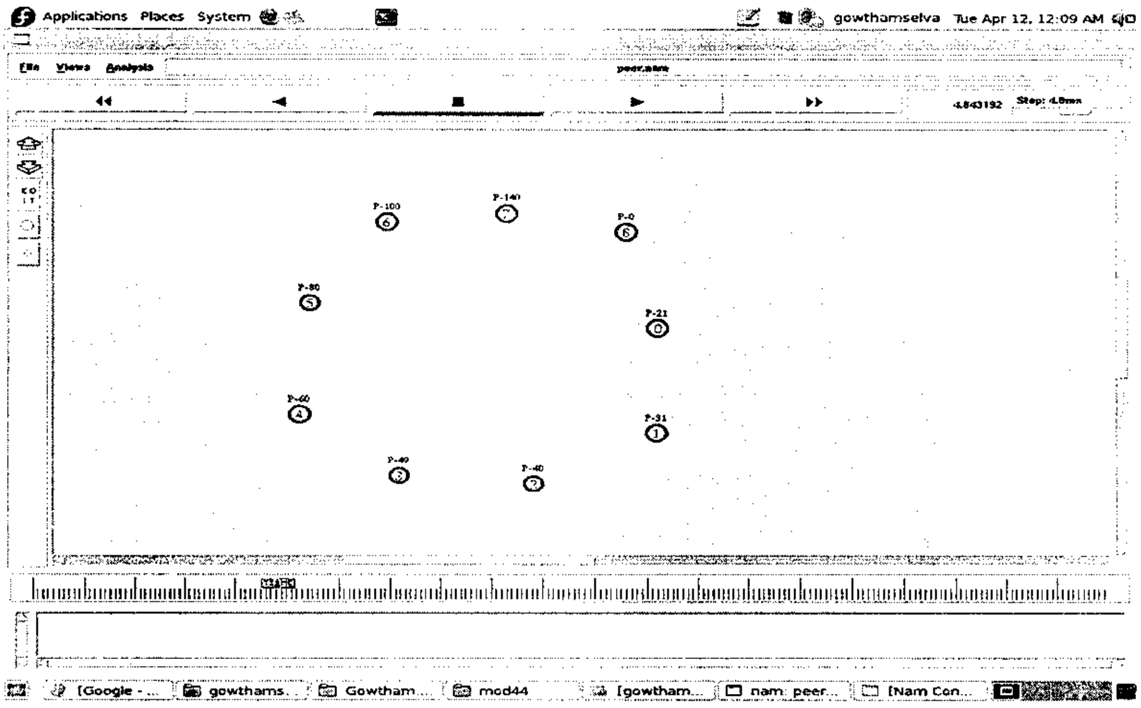
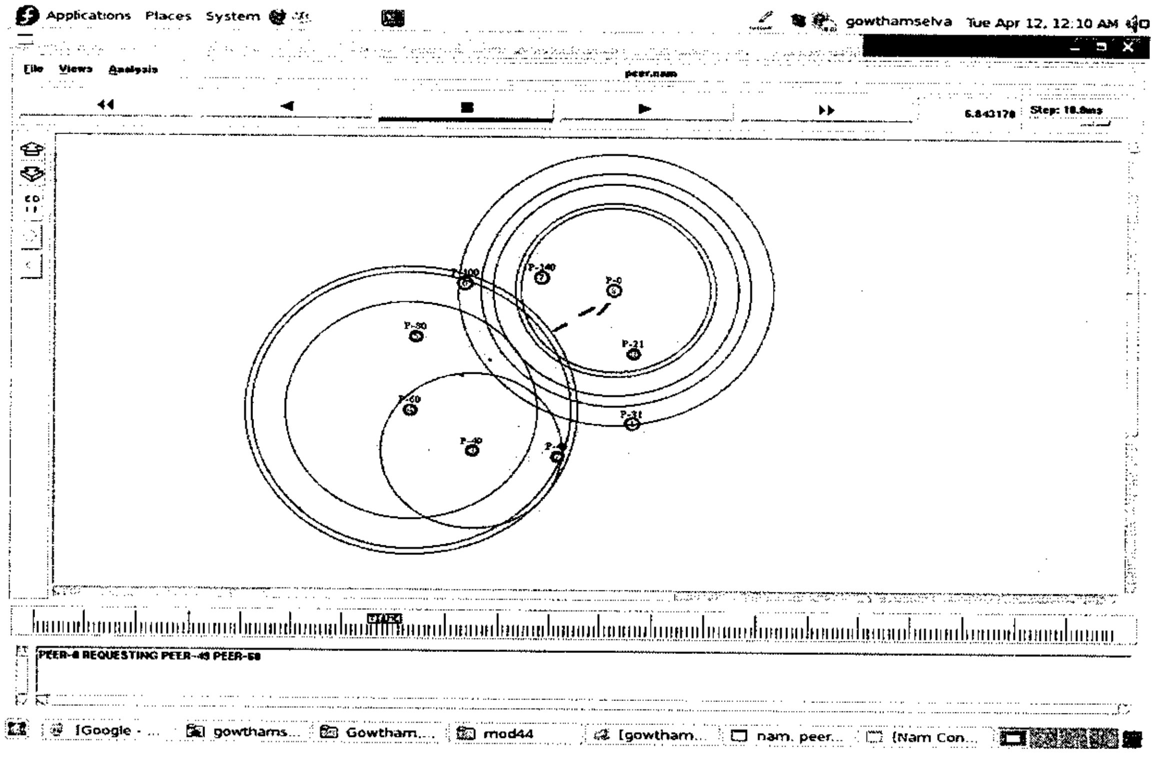
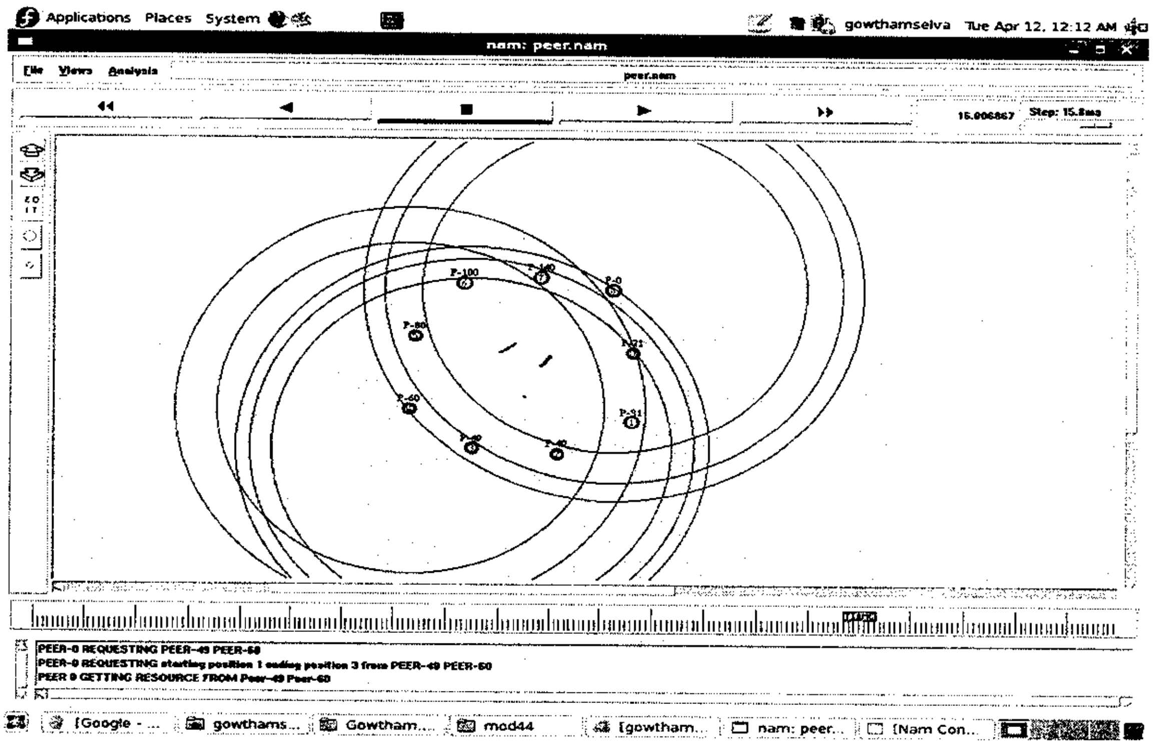


Figure 3.9: Depicting the ringed fashion network for resource sharing



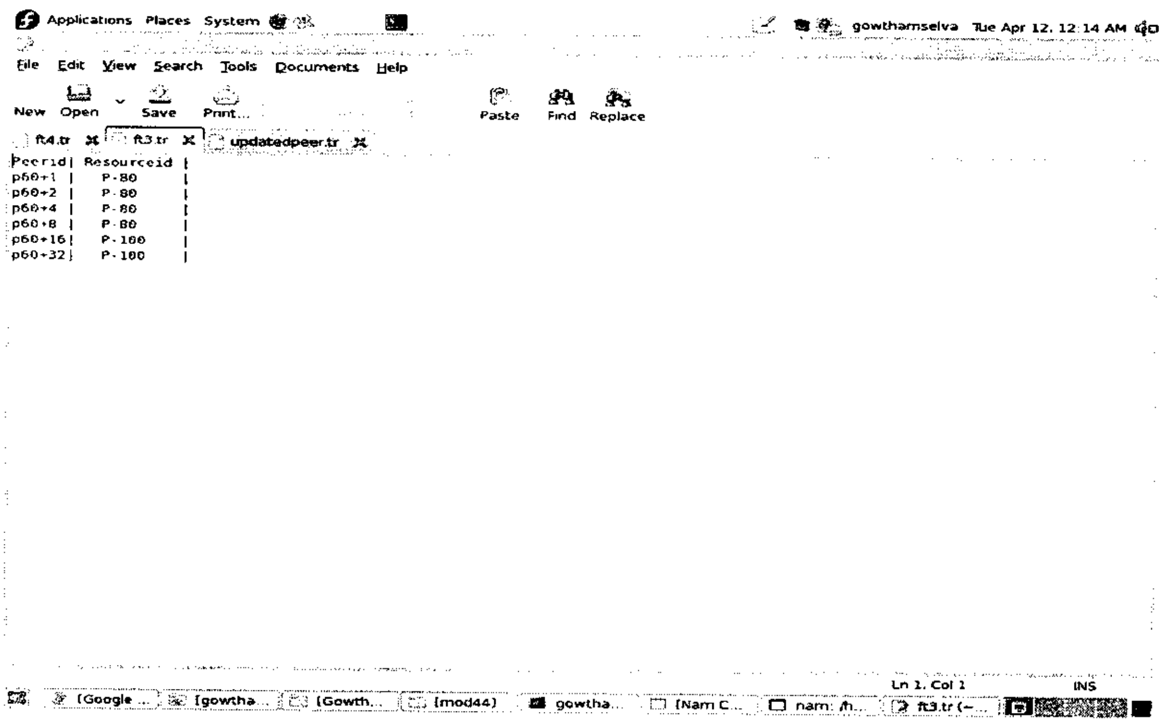
```
PEER-0 REQUESTING PEER-49 PEER-60
```

Figure 4.0: Depicting the requisition of nodes



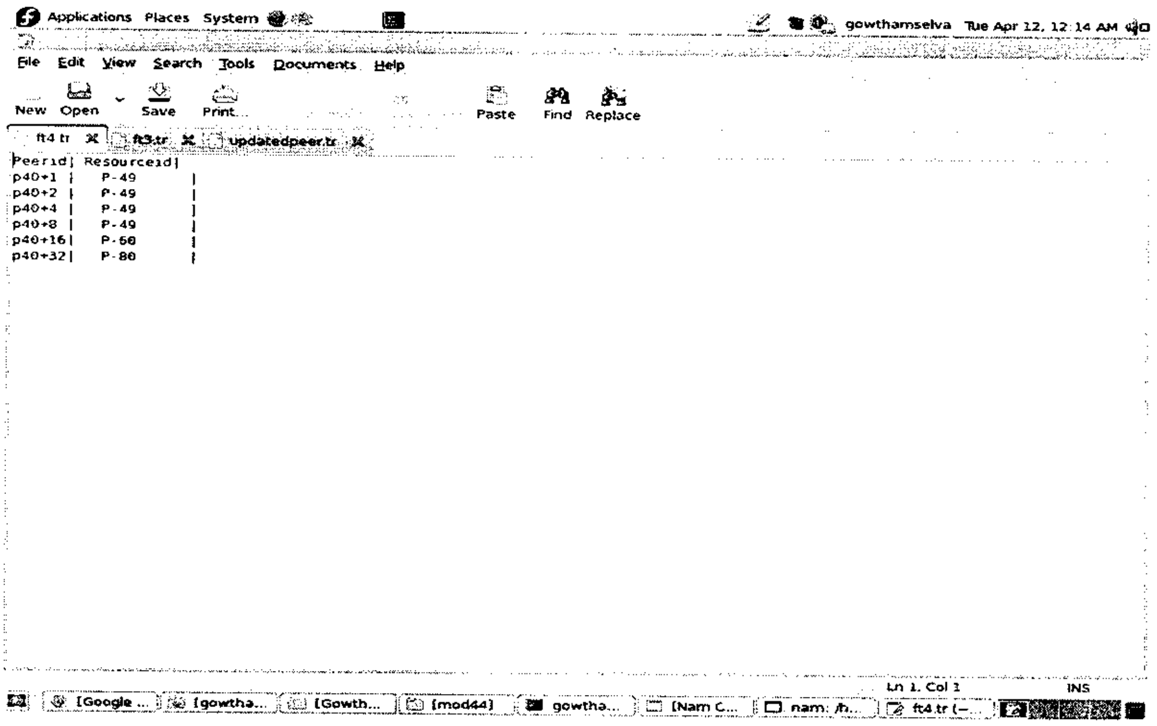
```
PEER-0 REQUESTING PEER-49 PEER-60
PEER-0 REQUESTING starting position 1 ending position 3 from PEER-49 PEER-60
PEER 0 GETTING RESOURCE FROM Peer-49 Peer-60
```

Figure 4.1: Depicting the share with its resource id and position


Applications Places System
gowthamselva Tue Apr 12, 12:14 AM
File Edit View Search Tools Documents Help
New Open Save Print... Paste Find Replace
ft4.tr ft3.tr updatedpeer.tr
Ln 1, Col 1 INS
[Google ... [gowtha... [Gowth... [mod44] gowtha... [Nam C... nam: /h... ft3.tr (~...

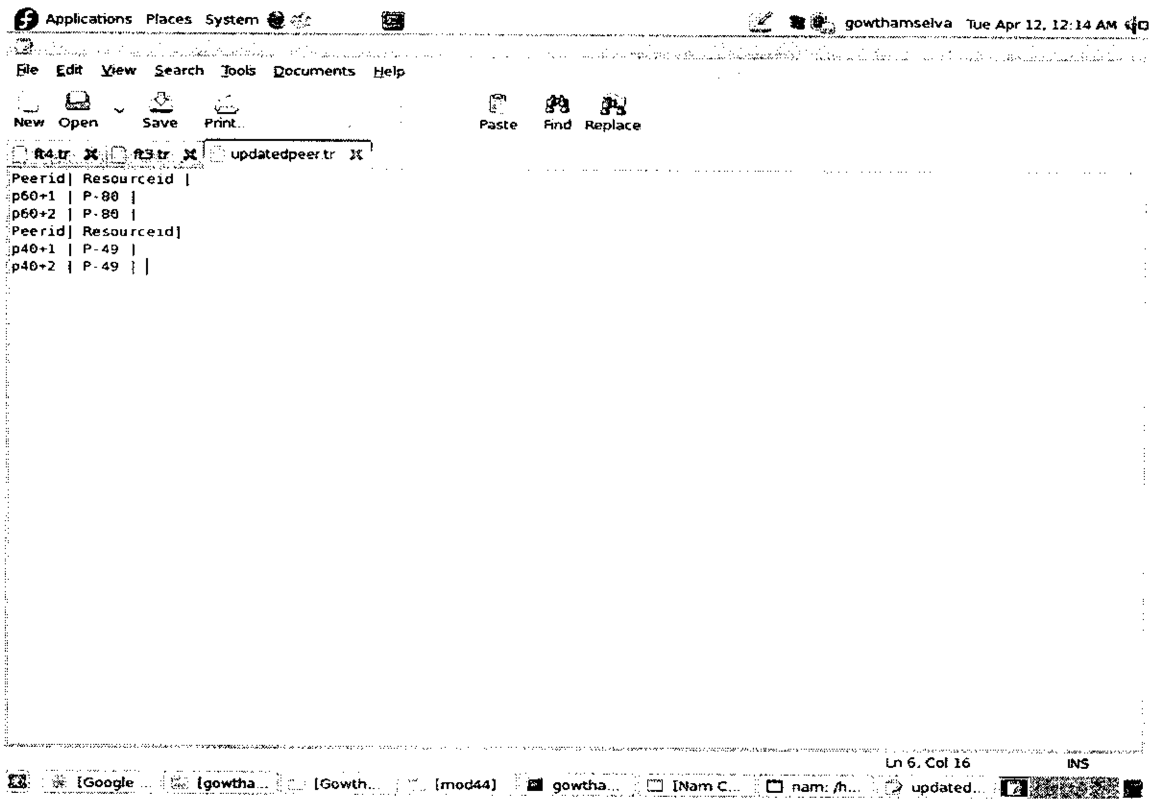
```
Peerid| Resourceid |
p60+1  |    P-80     |
p60+2  |    P-80     |
p60+4  |    P-80     |
p60+8  |    P-80     |
p60+16|    P-100    |
p60+32|    P-100    |
```

Figure 4.2: display of resources in node 3


Applications Places System
gowthamselva Tue Apr 12, 12:14 AM
File Edit View Search Tools Documents Help
New Open Save Print... Paste Find Replace
ft4.tr ft3.tr updatedpeer.tr
Ln 1, Col 1 INS
[Google ... [gowtha... [Gowth... [mod44] gowtha... [Nam C... nam: /h... ft4.tr (~...

```
Peerid| Resourceid|
p40+1  |    P-49      |
p40+2  |    P-49      |
p40+4  |    P-49      |
p40+8  |    P-49      |
p40+16|    P-60      |
p40+32|    P-80      |
```

Figure 4.3: display of resources in node 4


Applications Places System
gowthamselva Tue Apr 12, 12:14 AM
File Edit View Search Tools Documents Help
New Open Save Print..
Paste Find Replace
ft4.tr
ft3.tr
updatedpeer.tr
Ln 6, Col 16
INS

```
Peerid| Resourceid |
p60+1 | P-80 |
p60+2 | P-80 |
Peerid| Resourceid|
p40+1 | P-49 |
p40+2 | P-49 | |
```

Figure 4.4: Shared resource from node 3 and node 4(based on the starting and ending position)

4.3 PERFORMANCE EVALUATION

The performance evaluation of the analysis is done by means of three performance metrics like drop, throughput and bit error rate.

Now these three different metrics is analyzed by depicting the simulated scenario graphically. The brief description of the metrics is given below.

- ✓ **Throughput**

  Throughput is the measure of the average rate of successful message delivery.

✓ **Drop**

Drop is the ratio between the number of packets received to the total number of packets sent.

✓ **Bit error rate**

The bit error rate (BER) is the percentage of bits that have errors relative to the total number of bits received in a transmission.

The screen shots of the graphs are given below.

4.3.1 Performance analysis of packet drop scenario

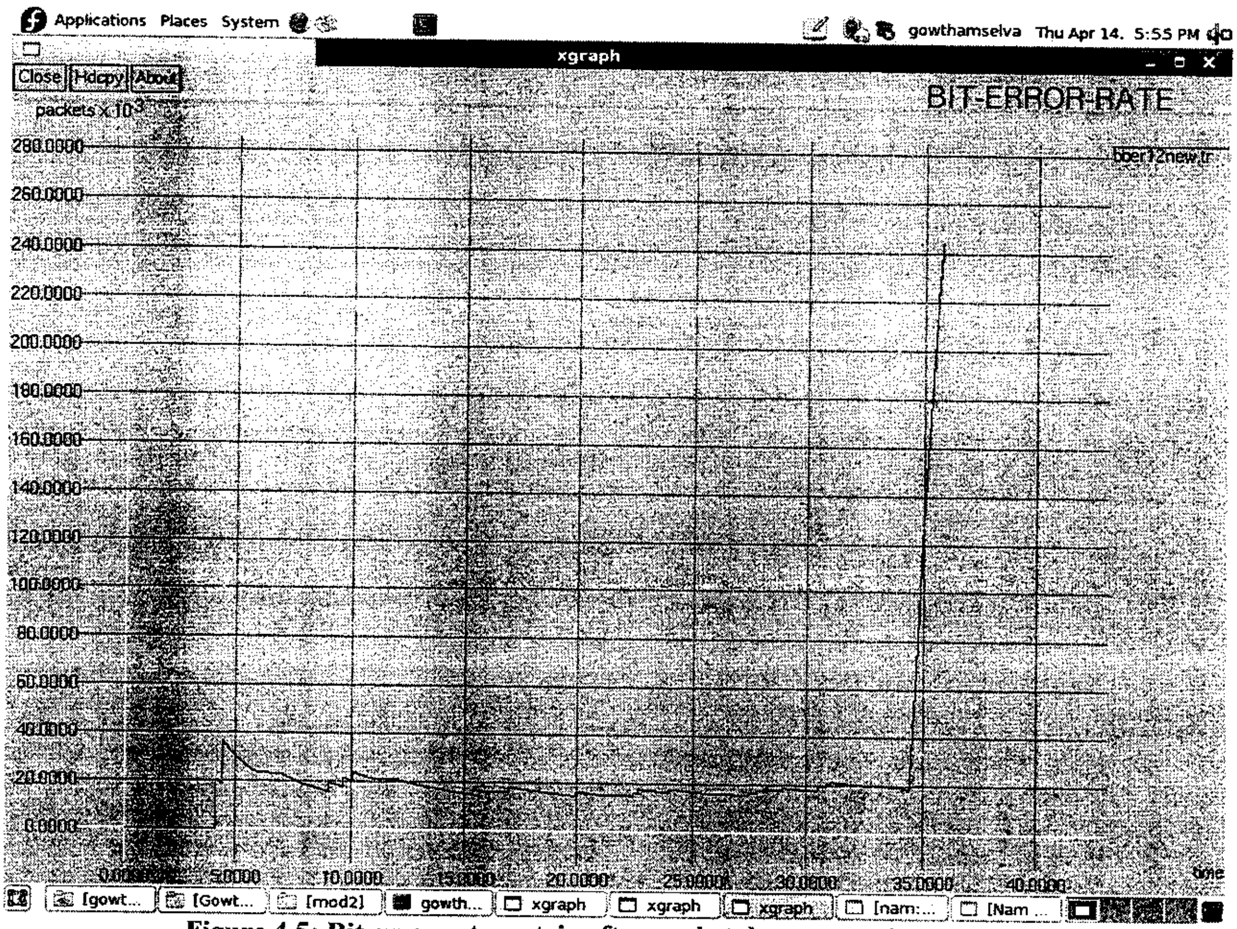


Figure 4.5: Bit error rate metric after packet drop scenario

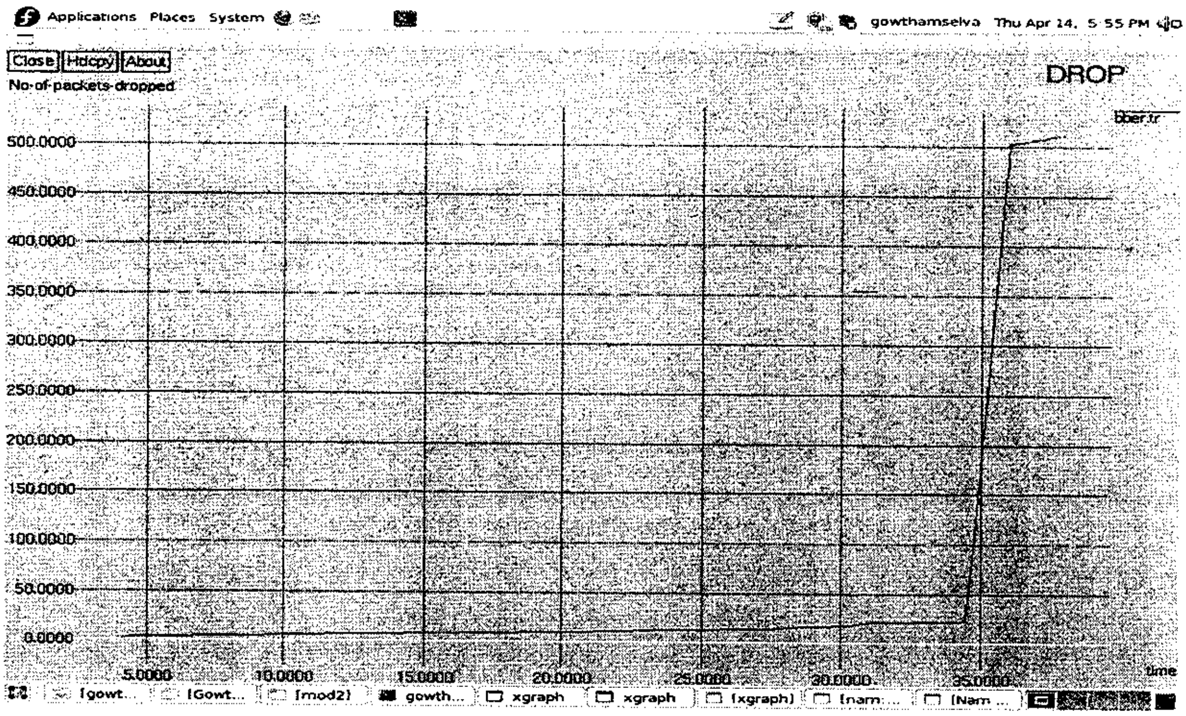


Figure 4.6: Drop metric after packet drop scenario

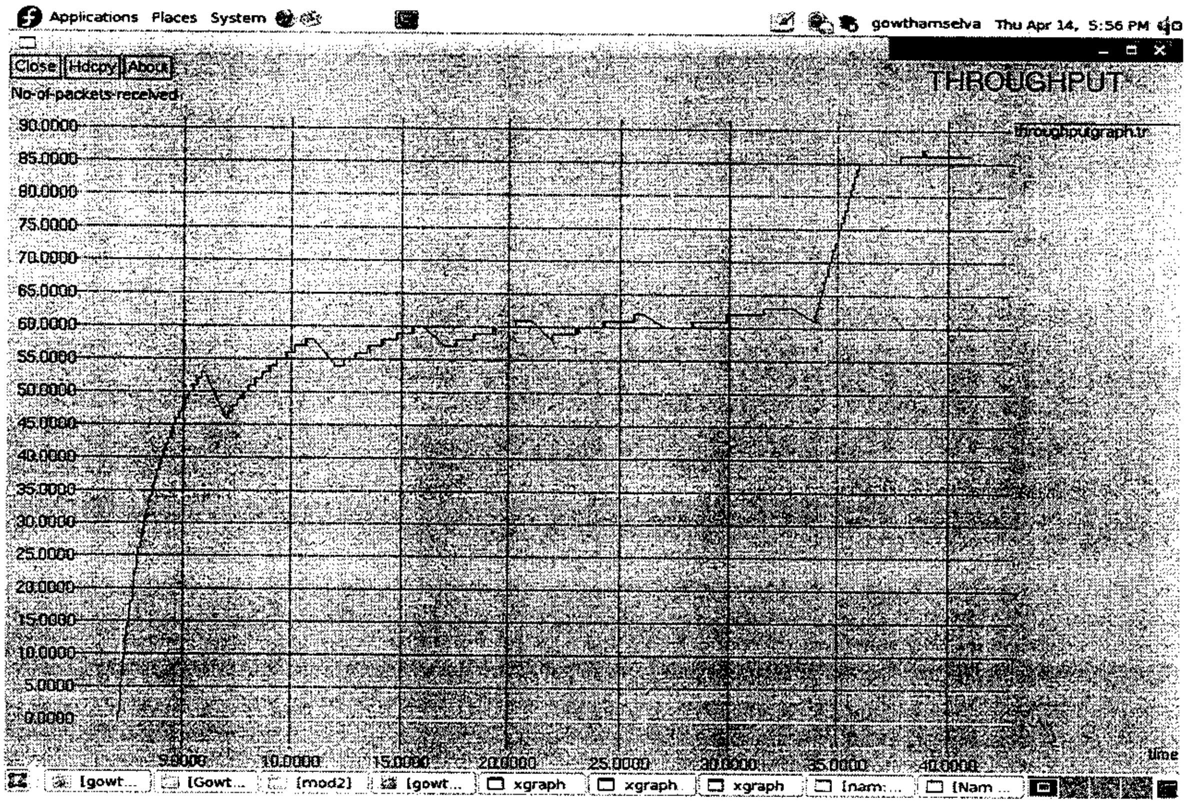


Figure 4.7: Throughput metric after packet drop scenario

4.3.2 Performance analysis of intruder attack scenario

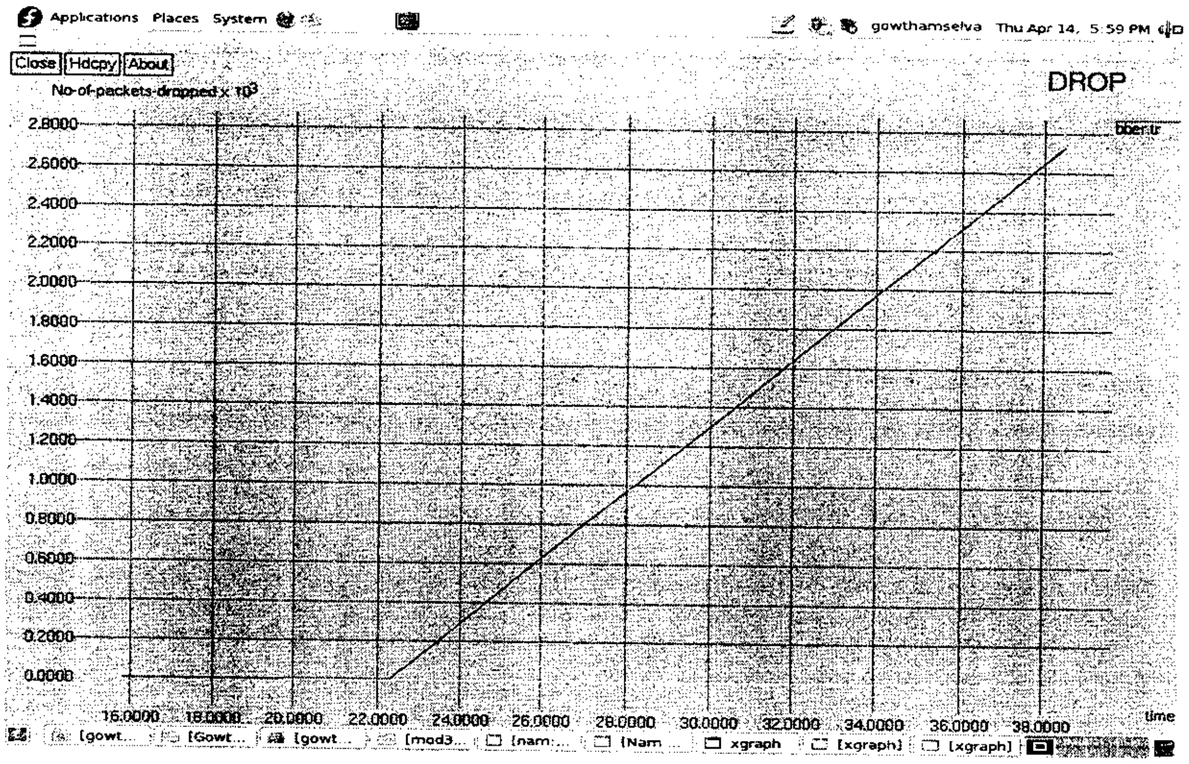


Figure 4.8: Drop metric after the attack scenario

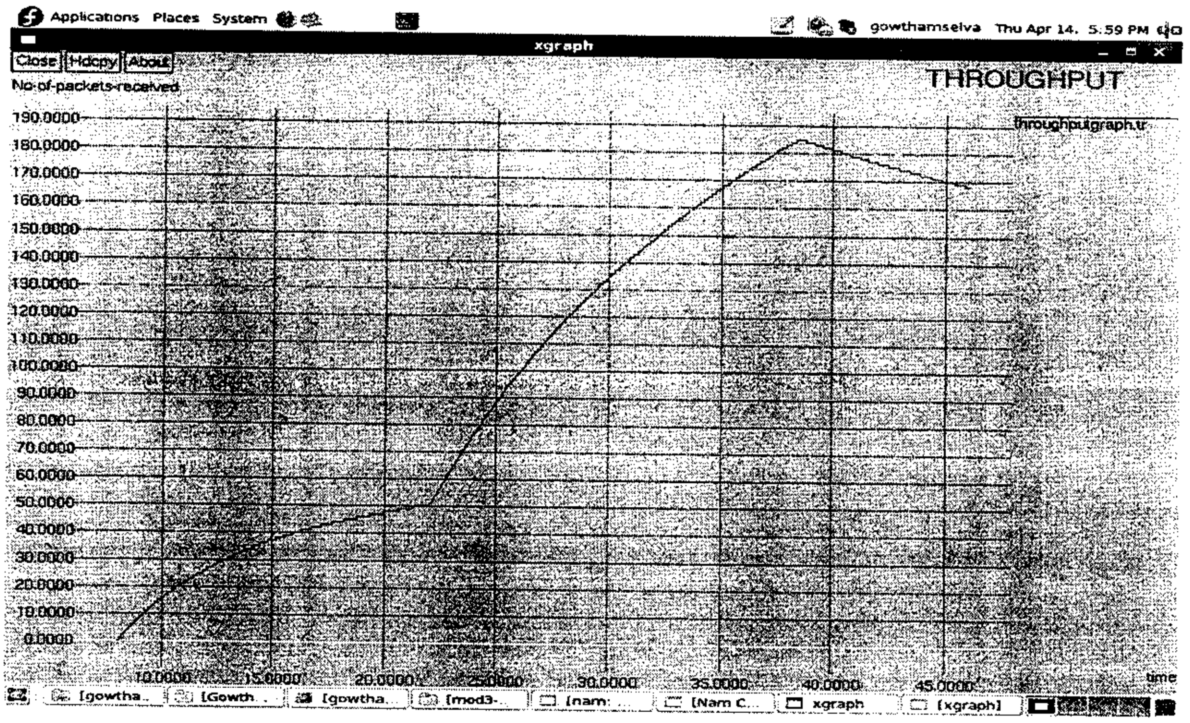


Figure 4.9: Throughput metric after the attack scenario

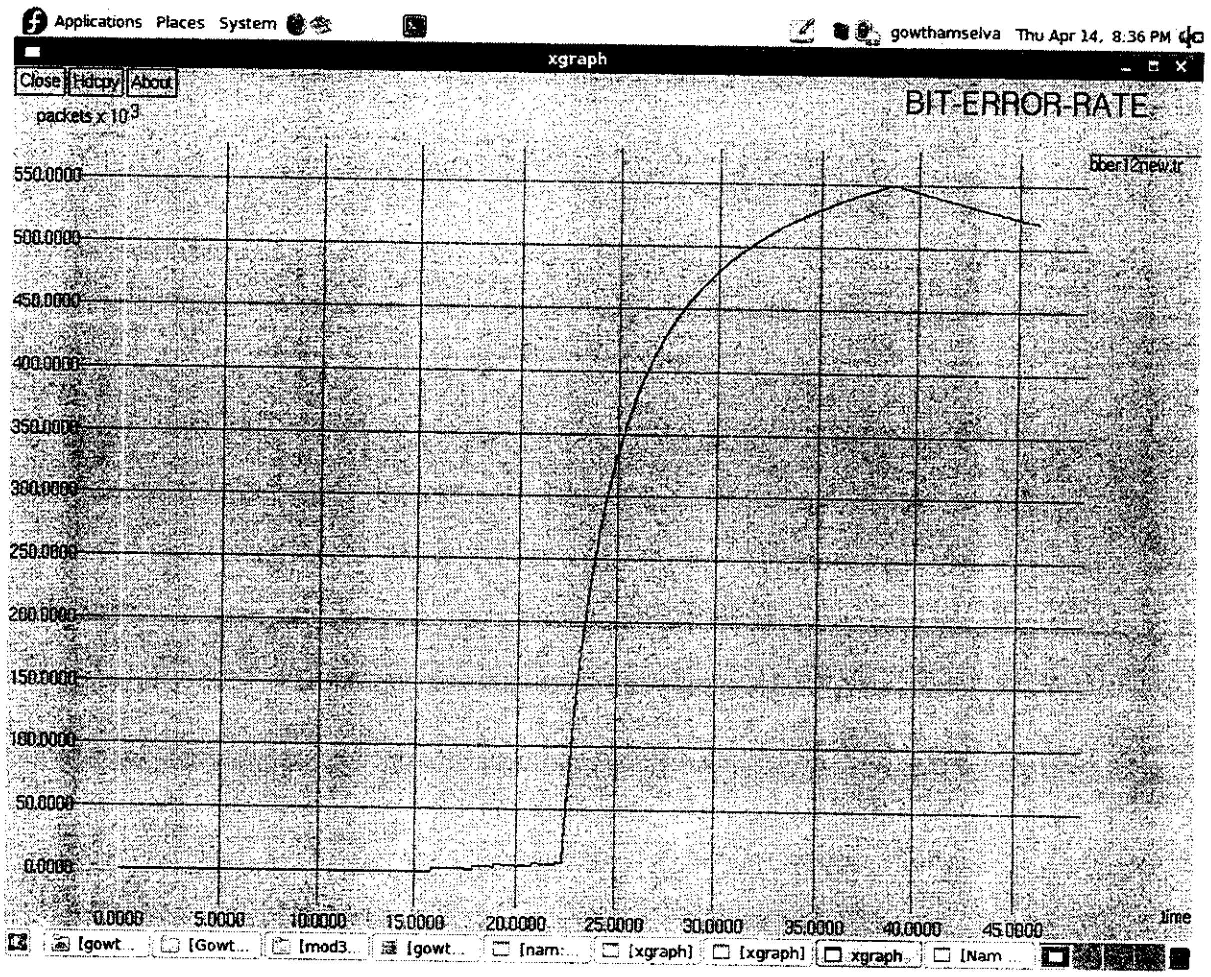


Figure 5.0: Bit error rate metric after the attack scenario

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENTS

In this project, we have analyzed the minimal delay paths in unstructured peer to peer networks by our proposed delay computation algorithm. And we have verified the performance of the network by imposing the constraints like network traffic and intruder attack, and depicting some graphical representations in terms of some performance metrics like drop, throughput and bit error rate. And we have successfully demonstrated the experiment of dynamic resource sharing between the peers using split up and add methodology. Thus we have given a complete analysis report of the above concepts proposed and the output is verified.

In future this can be extended with the calculation of distances between each and every node from source to destination, and both the calculations can be compared to yield the nearest optimal path in the network. By doing so, the reliability of the network can be increased.

And dynamic resource sharing experiment can be further enhanced by designing the environment in such a way that it can fetch data from any number of nodes maintaining large amount of data. And not from the pre defined number of nodes.

CHAPTER 6

APPENDICES

6.1 SAMPLE CODE

6.1.1 Path Analysis for existing network

Path analysis.tcl

```
#
======================================================================
=
# Define options
#
======================================================================
=
set val(chan)             Channel/WirelessChannel          ;#Channel Type
set val(prop)             Propagation/TwoRayGround         ;# radio-Popagation model
set val(netif)            Phy/WirelessPhy                  ;# network interface type
set val(mac)              Mac/802_11                       ;# MAC type
set val(ifq)              CMUPriQueue                      ;# 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)               8                                ;# number of peernodes
set val(rp)               DSR                              ;# routing protocol
set val(x)                1000                             ;# X axis distance
set val(y)                1000                             ;# Y axis distance
set opt(energymodel)      EnergyModel                      ;# Initial Energy
set opt(radiomodel)       RadioModel                       ;# Transmission Model
set opt(initialenergy)    100                              ;# Initial energy in Joules
```

```
#
=====================================================================
=
# Main Program
#
=====================================================================
=

#
# Initialize Global Variables
#
set ns_          [new Simulator]
set namTracefile [open opt.nam w]
$ns_ namtrace-all-wireless $namTracefile $val(x) $val(y)
set tracefd     [open opt.tr w]
$ns_ trace-all $tracefd
#$ns_ use-newtrace

# set up topography object
set topo       [new Topography]

$topo load_flatgrid 5000 5000

#
# Create God
#
create-god $val(nn)


# configure node

        $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 ON \
                              -rxPower 2.0 \
                              -txPower 2.5 \
                              -movementTrace ON \


for {set i 0} {$i < $val(nn) } {incr i} {
                    set node_($i) [$ns_ node]
$node_($i) random-motion 0             ;# disable random motion
$node_($i) color black
$node_($i) set X_ 0.0
$node_($i) set Y_ 0.0
$node_($i) set Z_ 0.0

$ns_ initial_node_pos $node_($i) 30

        }

$ns_ at 0.0 "$node_(0) setdest 413.41 375.466 1000"
$ns_ at 0.0 "$node_(1) setdest 549.088 552.949 1000"
$ns_ at 0.0 "$node_(2) setdest 583.002 425.902 1000"
$ns_ at 0.0 "$node_(3) setdest 652.767 303.471 1000"

$ns_ at 0.0 "$node_(4) setdest 743.801 614.644 1000"
$ns_ at 0.0 "$node_(5) setdest 784.231 479.62 1000"
$ns_ at 0.0 "$node_(6) setdest 853.385 357.772 1000"

$ns_ at 0.0 "$node_(7) setdest 947.997 538.504 1000"

$ns_ at 1.2 "$node_(0) add-mark c4 green4 hexagon"
$ns_ at 1.2 "$node_(7) add-mark c4 green4 hexagon"
$ns_ at 1.2 "$node_(0) label SOURCE"
$ns_ at 1.2 "$node_(7) label DESTINATION"
$ns_ at 1.2 "$node_(0) color red"
$ns_ at 1.2 "$node_(7) color red"

# set transmission
# Agent Creation for Sink

for { set i 0 } { $i < 8 } { incr i } {
set sink($i) [new Agent/LossMonitor]
$ns_ attach-agent $node_($i) $sink($i)
}

#Creating the Application/Traffic for Data Transmission

proc attach-CBR-traffic { node sink size interval } {

  #Get an instance of the simulator
  set ns_ [Simulator instance]
  #Create a CBR sink14 agent and attach it to the node
  set cbr [new Agent/CBR]
  $ns_ attach-agent $node $cbr
  $cbr set packetSize_ $size
  $cbr set interval_ $interval

  #Attach CBR source to sink;
  $ns_ connect $cbr $sink
  return $cbr
 }

set cbr(0) [attach-CBR-traffic $node_(0) $sink(1) 1024 .075]
set cbr(1) [attach-CBR-traffic $node_(1) $sink(4) 1024 .075]
set cbr(2) [attach-CBR-traffic $node_(4) $sink(7) 1024 .075]

for { set i 0 } { $i < 3 } { incr i } {
$ns_ at 2.0 "$cbr($i) start"
$ns_ at 6.0 "$cbr($i) stop"
}

$ns_ at 2.0 "$ns_ trace-annotate \"Packets transmitted through path1 \" "

$ns_ at 2.0 "$node_(1) color red"
$ns_ at 2.0 "$node_(4) color red"

$ns_ at 6.0 "$node_(1) color black"
```

```
$ns_ at 6.0 "$node_(4) color black"


set cbr(3) [attach-CBR-traffic $node_(0) $sink(2) 1024 .075]
set cbr(4) [attach-CBR-traffic $node_(2) $sink(5) 1024 .075]
set cbr(5) [attach-CBR-traffic $node_(5) $sink(7) 1024 .075]

for { set i 3 } { $i < 6 } { incr i } {
$ns_ at 7.0 "$cbr($i) start"
$ns_ at 11.0 "$cbr($i) stop"
}

$ns_ at 7.0 "$node_(2) color red"
$ns_ at 7.0 "$node_(5) color red"

$ns_ at 11.0 "$node_(2) color black"
$ns_ at 11.0 "$node_(5) color black"

$ns_ at 7.0 "$ns_ trace-annotate \"Packets transmitted through path2 \" "


set cbr(6) [attach-CBR-traffic $node_(0) $sink(3) 1024 .075]
set cbr(7) [attach-CBR-traffic $node_(3) $sink(6) 1024 .075]
set cbr(8) [attach-CBR-traffic $node_(6) $sink(7) 1024 .075]

for { set i 6 } { $i < 9 } { incr i } {
$ns_ at 12.0 "$cbr($i) start"
$ns_ at 16.0 "$cbr($i) stop"
}

$ns_ at 12.0 "$node_(3) color red"
$ns_ at 12.0 "$node_(6) color red"

$ns_ at 16.0 "$node_(3) color black"
$ns_ at 16.0 "$node_(6) color black"

$ns_ at 12.0 "$ns_ trace-annotate \"Packets transmitted through path3 \" "


set cbr(9) [attach-CBR-traffic $node_(0) $sink(1) 1024 .075]
set cbr(10) [attach-CBR-traffic $node_(1) $sink(5) 1024 .075]
set cbr(11) [attach-CBR-traffic $node_(5) $sink(7) 1024 .075]

for { set i 9 } { $i < 12 } { incr i } {
$ns_ at 17.0 "$cbr($i) start"
$ns_ at 21.0 "$cbr($i) stop"
}

$ns_ at 17.0 "$ns_ trace-annotate \"Packets transmitted through path4 \" "

$ns_ at 17.0 "$node_(1) color red"
$ns_ at 17.0 "$node_(5) color red"

$ns_ at 21.0 "$node_(1) color black"
$ns_ at 21.0 "$node_(5) color black"


set cbr(12) [attach-CBR-traffic $node_(0) $sink(3) 1024 .075]
set cbr(13) [attach-CBR-traffic $node_(3) $sink(5) 1024 .075]
set cbr(14) [attach-CBR-traffic $node_(5) $sink(7) 1024 .075]

for { set i 12 } { $i < 15 } { incr i } {
$ns_ at 22.0 "$cbr($i) start"
$ns_ at 26.0 "$cbr($i) stop"
}

$ns_ at 22.0 "$ns_ trace-annotate \"Packets transmitted through path5 \" "

$ns_ at 22.0 "$node_(3) color red"
$ns_ at 22.0 "$node_(5) color red"

$ns_ at 26.0 "$node_(3) color black"
$ns_ at 26.0 "$node_(5) color black"


$ns_ at 26.5 "$ns_ trace-annotate \"As per the path delay algorithm path3 has the lowest delay.
So path3 is the minimal delay path \" "

for { set i 6 } { $i < 9 } { incr i } {
$ns_ at 27.0 "$cbr($i) start"
```

```
 $ns_ at 33.0 "$cbr($i) stop"
 }

 $ns_ at 27.0 "$node_(3) color red"
 $ns_ at 27.0 "$node_(6) color red"

 $ns_ at 33.0 "$node_(3) color black"
 $ns_ at 33.0 "$node_(6) color black"

 $ns_ at 27.0 "$ns_ trace-annotate \"Packets transmitted through the minimal delay path \" "


#***************************************************************************
**
#***************************************************************************
**
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 50.0 "$node_($i) reset";
}

$ns_ at 55.0 "stop"
$ns_ at 55.01 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd n1
   $ns_ flush-trace
     close $tracefd
         exec awk -f delay.awk opt.tr
   exec awk -f cal-del.awk sent-detail receive-detail sent1-detail receive1-detail sent2-detail
receive2-detail sent3-detail receive3-detail sent4-detail receive4-detail &


         puts "running nam..."
         exec nam opt.nam &
}
puts "Starting Simulation..."
$ns_ run
```

delay.awk

```
BEGIN{
}
{
if(FILENAME=="opt.tr"){
if( ($2>2.0) && ($2<=6.0)){
if($1=="s"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "sent-detail"
}
}
}
if( ($2>2.0) && ($2<=6.0)){
if($1=="r"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "receive-detail"
}
}


if( ($2>7.0) && ($2<=11.0)){
if($1=="s"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "sent1-detail"
}
}
if( ($2>7.0) && ($2<=11.0)){
if($1=="r"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "receive1-detail"
}
}


if( ($2>12.0) && ($2<=16.0)){
if($1=="s"){
```

```
a=$2
nid=$3;
print "node "nid" "a " " $6 > "sent2-detail"
}
}
if( ($2>12.0) && ($2<=16.0)){
if($1=="r"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "receive2-detail"
}
}


if( ($2>17.0) && ($2<=21.0)){
if($1=="s"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "sent3-detail"
}
}
if( ($2>17.0) && ($2<=21.0)){
if($1=="r"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "receive3-detail"
}
}


if( ($2>22.0) && ($2<=26.0)){
if($1=="s"){
a=$2
nid=$3;
print "node "nid" "a " " $6 > "sent4-detail"
}
}
if( ($2>22.0) && ($2<=26.0)){
if($1=="r"){
a=$2
```

```
nid=$3;
print "node "nid" "a " " $6 > "receive4-detail"
}
}


}

END{

}
```

Cal delay.awk

```
BEGIN{

}
{
if(FILENAME=="sent-detail"){
if($4==0){
a=$3;
}
}
if(FILENAME=="receive-detail"){
if($4==172){
b=$3;
}
}

if(FILENAME=="sent1-detail"){
if($4==0){
d=$3;
}
}
if(FILENAME=="receive1-detail"){
if($4==336){
e=$3;
}
}
if(FILENAME=="sent2-detail"){
if($4==0){
g=$3;
}
}
if(FILENAME=="receive2-detail"){
if($4==501){
h=$3;
}
}

if(FILENAME=="sent3-detail"){
if($4==0){
j=$3;
}
}
if(FILENAME=="receive3-detail"){
if($4==663){
k=$3;
}
}

if(FILENAME=="sent4-detail"){
if($4==0){
m=$3;
}
}
if(FILENAME=="receive4-detail"){
if($4==825){
n=$3;
}
}


c=a-b;
f=d-e;
i=g-h;
l=j-k;
o=m-n;
}
```

```
END{
print "path1 delay: " c > "path-delay.tr"
print "path2 delay: " f > "path-delay.tr"
print "path3 delay: " i > "path-delay.tr"
print "path4 delay: " l > "path-delay.tr"
print "path5 delay: " o > "path-delay.tr"
print "                    " > "path-delay.tr"
print "                    " > "path-delay.tr"
print "As per the delay analysis path3[0-3-6-7] has lowest delay. So path3 is the minimal delay
path " > "path-delay.tr"
}
```

6.1.2 Calculation of bit error rate

Ber.awk:-

```
BEGIN {
drop=" ";
time=0.0;
m=0;

}
{
 drop=$1;
 time=$2;
 if(drop=="D"){
 m++;
  printf("%f %d \n",time,m) > "bber.tr";
  }
}
END {

   }
```

Bercorrect.awk

```
BEGIN {
drop=" ";
time=0.0;
m=0;
s=0;
}
```

```
{
 drop=$1;
 time=$2;
 if(drop=="D"){
 m++;
}
  if (( $1 == "s" && $7 == "cbr" && $4 == "AGT" )&& (time != 0.000000))
 {
   s++;
 }
 printf("%f %f \n",time,m/(s+10)) > "bber12new.tr";
 }

END {
}
```

6.1.3 To enable resource sharing between the peers

Enhancement.tcl

```
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)             CMUPriQueue                   ;# interface queue type
set val(ll)              LL                            ;# link layer type
set val(ant)             Antenna/OmniAntenna           ;# antenna model
set val(ifqlen)          250                           ;# max packet in ifq
set val(nn)              10                            ;# number of mobilenodes
set val(rp)              DSR                           ;# routing protocol
set val(x)               1000                          ;# X axis distance
set val(y)               1000                          ;# Y axis distance
set opt(energymodel)     EnergyModel                   ;# Initial Energy
set opt(radiomodel)      RadioModel                    ;# Transmission Model
set opt(initialenergy)   100                           ;# Initial energy in Joules

# Creating Simulator Object
set ns [new Simulator]
```

```
# Creating NAM File
set namTracefile [open peer.nam w]
$ns namtrace-all-wireless $namTracefile $val(x) $val(y)

# Creating Multiple Trace
set traceFile [open peer.tr w]
$ns trace-all $traceFile
$ns use-newtrace

# Creating Topology
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

# Parameters

Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set bandwidth_ 2e6
Phy/WirelessPhy set Pt_ 0.2818    ;# for 250.0
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0
Mac/802_11 set PLCPDataRate_ 11.0e6;#11Mbps
Agent/UDP set rate_ 8.0e6     ;# date rate 8Mbs

puts " Number of Nodes = 9 "

# Configuring the Nodes in the Topology.
$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) \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace ON \
                -idlePower 0.2 \
                -rxPower 0.5 \
                -txPower 1.0 \
        -sleepPower 0.1 \
        -transitionTime 0.003 \
        -channelType $val(chan) \
                -maxTransRange 75

# Creating Nodes
set node(0) [$ns node]
set node(1) [$ns node]
set node(2) [$ns node]
set node(3) [$ns node]
set node(4) [$ns node]
set node(5) [$ns node]
set node(6) [$ns node]
set node(7) [$ns node]
set node(8) [$ns node]
set node(9) [$ns node]

# Parameters to the Node
for { set i 0 } { $i<10 } { incr i } {
$node($i) random-motion 0
$node($i) set X_ 0.0
$node($i) set Y_ 0.0
$node($i) set Z_ 0.0
$node($i) color green
$node($i) shape circle
$ns initial_node_pos $node($i) 10
}

set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
set sink5 [new Agent/LossMonitor]
set sink6 [new Agent/LossMonitor]
set sink7 [new Agent/LossMonitor]

$ns attach-agent $node(0) $sink0
$ns attach-agent $node(1) $sink1
$ns attach-agent $node(2) $sink2
$ns attach-agent $node(3) $sink3
$ns attach-agent $node(4) $sink4
$ns attach-agent $node(5) $sink5
$ns attach-agent $node(6) $sink6
$ns attach-agent $node(7) $sink7

proc attach-CBR-traffic { node sink size interval } {
#Get an instance of the simulator
set ns [Simulator instance]
#Create a CBR sink14 agent and attach it to the node
set cbr [new Agent/CBR]
$ns attach-agent $node $cbr
$cbr set packetSize_ $size
$cbr set interval_ $interval
#Attach CBR source to sink;
$ns connect $cbr $sink
return $cbr
}

$ns at 2.2 "$node(0) setdest 301.657 556.09 3000"
$ns at 3.2 "$node(0) color brown"
$ns at 2.2 "$node(0) label P-21"
$ns at 2.2 "$node(1) setdest 301.297 486.477 3000"
$ns at 3.2 "$node(1) color brown"
$ns at 2.2 "$node(1) label P-31"
$ns at 2.2 "$node(2) setdest 234.077 449.626 3000"
$ns at 3.2 "$node(2) color brown"
$ns at 2.2 "$node(2) label P-40"
$ns at 2.2 "$node(3) setdest 161.594 454.833 3000"
$ns at 3.2 "$node(3) color brown"
$ns at 2.2 "$node(3) label P-49"
$ns at 2.2 "$node(4) setdest 110.737 500.965 3000"
$ns at 3.2 "$node(4) color brown"
$ns at 2.2 "$node(4) label P-60"
```

```
$ns at 2.2 "$node(5) setdest 115.026 570.812 3000"
$ns at 3.2 "$node(5) color brown"
$ns at 2.2 "$node(5) label P-80"
$ns at 2.2 "$node(6) setdest 155.975 622.754 3000"
$ns at 3.2 "$node(6) color brown"
$ns at 2.2 "$node(6) label P-100"
$ns at 2.2 "$node(7) setdest 220.897 627.626 3000"
$ns at 3.2 "$node(7) color brown"
$ns at 2.2 "$node(7) label P-140"
$ns at 2.2 "$node(8) setdest 281.925 611.897 3000"
$ns at 3.2 "$node(8) color brown"
$ns at 2.2 "$node(8) label P-0"

puts "Enter Response peer1 name (0 to 8)"
gets stdin nn1
puts "Enter Response peer2 name (0 to 8)"
gets stdin nn2

puts "Enter Requesting peer(0 to 8)"
gets stdin nn3

if { $nn1 == 0 } {
set stpr 21
}

if { $nn1 == 1 } {
set stpr 31
}
if { $nn1 == 2 } {
set stpr 40
}
if { $nn1 == 3 } {
set stpr 49
}
if { $nn1 == 4 } {
set stpr 60
}
if { $nn1 == 5 } {
set stpr 80
}
```

```
if { $nn1 == 6 } {
set stpr 100
}
if { $nn1 == 7 } {
set stpr 140
}
if { $nn1 == 8 } {
set stpr 0
}



if { $nn2 == 0 } {
set edpr 21
}

if { $nn2 == 1 } {
set edpr 31
}
if { $nn2 == 2 } {
set edpr 40
}
if { $nn2 == 3 } {
set edpr 49
}
if { $nn2 == 4 } {
set edpr 60
}
if { $nn2 == 5 } {
set edpr 80
}
if { $nn2 == 6 } {
set edpr 100
}
if { $nn2 == 7 } {
set edpr 140
}
if { $nn2 == 8 } {
set edpr 0
}
```

```
if { $nn3 == 0 } {
set destpr 21
}

if { $nn3 == 1 } {
set destpr 31
}
if { $nn3 == 2 } {
set destpr 40
}
if { $nn3 == 3 } {
set destpr 49
}
if { $nn3 == 4 } {
set destpr 60
}
if { $nn3 == 5 } {
set destpr 80
}
if { $nn3 == 6 } {
set destpr 100
}
if { $nn3 == 7 } {
set destpr 140
}
if { $nn3 == 8 } {
set destpr 0
}

puts "Enter starting position"
gets stdin sp
puts "Enter ending position"
gets stdin ep

if { $nn1 == 0 } {
set ff1 ft0.tr
}
if { $nn1 == 1 } {
set ff1 ft1.tr
```

```
}
if { $nn1 == 2 } {
set ff1 ft2.tr
}
if { $nn1 == 3 } {
set ff1 ft3.tr
}
if { $nn1 == 4 } {
set ff1 ft4.tr
}
if { $nn1 == 5 } {
set ff1 ft5.tr
}
if { $nn1 == 6 } {
set ff1 ft6.tr
}
if { $nn1 == 7 } {
set ff1 ft7.tr
}
if { $nn1 == 8 } {
set ff1 ft8.tr
}

if { $nn2 == 0 } {
set ff2 ft0.tr
}
if { $nn2 == 1 } {
set ff2 ft1.tr
}
if { $nn2 == 2 } {
set ff2 ft2.tr
}
if { $nn2 == 3 } {
set ff2 ft3.tr
}
if { $nn2 == 4 } {
set ff2 ft4.tr
}
if { $nn2 == 5 } {
set ff2 ft5.tr
```

```
}
if { $nn2 == 6 } {
set ff2 ft6.tr
}
if { $nn2 == 7 } {
set ff2 ft7.tr
}
if { $nn2 == 8 } {
set ff2 ft8.tr
}
set stcp2 [new Agent/TCP]
$ns attach-agent $node($nn3) $stcp2
set sftp2 [new Application/FTP]
$sftp2 attach-agent $stcp2
set ssink2 [new Agent/TCPSink]
$ns attach-agent $node($nn1) $ssink2
$ns connect $stcp2 $ssink2

set stcp3 [new Agent/TCP]
$ns attach-agent $node($nn3) $stcp3
set sftp3 [new Application/FTP]
$sftp3 attach-agent $stcp3
set ssink3 [new Agent/TCPSink]
$ns attach-agent $node($nn2) $ssink3
$ns connect $stcp3 $ssink3


set stcp4 [new Agent/TCP]
$ns attach-agent $node($nn1) $stcp4
set sftp4 [new Application/FTP]
$sftp4 attach-agent $stcp4
set ssink4 [new Agent/TCPSink]
$ns attach-agent $node($nn3) $ssink4
$ns connect $stcp4 $ssink4

set stcp5 [new Agent/TCP]
$ns attach-agent $node($nn2) $stcp5
set sftp5 [new Application/FTP]
$sftp5 attach-agent $stcp5
set ssink5 [new Agent/TCPSink]
```

```
$ns attach-agent $node($nn3) $ssink5
$ns connect $stcp5 $ssink5
$ns at 5.5 "$sftp2 start"
$ns at 10.5 "$sftp2 stop"
$ns at 5.5 "$sftp3 start"
$ns at 10.0 "$sftp3 stop"

$ns at 10.5 "$sftp4 start"
$ns at 20.5 "$sftp4 stop"
$ns at 10.5 "$sftp5 start"
$ns at 20.0 "$sftp5 stop"

$ns at 5.5 "$ns trace-annotate \" PEER-$destpr REQUESTING PEER-$stpr PEER-$edpr\""
$ns at 7.5  "$ns trace-annotate \"PEER-$destpr REQUESTING starting position $sp ending
position $ep from PEER-$stpr PEER-$edpr\""
$ns at 10.5  "$ns trace-annotate \" PEER $destpr GETTING RESOURCE FROM Peer-$stpr
Peer-$edpr\""

set  ptr1 [open xx w]
puts $ptr1 "$sp $ep"
close $ptr1

proc finish1 {} {
global ff1 ff2
exec awk -f check.awk xx $ff1 $ff2
exec xdg-open $ff1
exec xdg-open $ff2
exec xdg-open updatedpeer.tr
exit 0
}
# Finish Proc
proc finish {} {
global ns namTracefile
$ns flush-trace
close $namTracefile
exec nam -r 1m peer.nam &
}
# Calling Finish Procedure
$ns at 40.0 "finish"
```

```
$ns at 41.0 "finish1"
# Executing the Program
$ns run
```

Check.awk

```
BEGIN {
f=0
ff=0
}
{
if(FILENAME == "xx") {
st=$1
ed=$2
}
if(FILENAME == "ft1.tr" || FILENAME == "ft2.tr" || FILENAME == "ft3.tr" || FILENAME ==
"ft4.tr" || FILENAME == "ft5.tr" || FILENAME == "ft6.tr" || FILENAME == "ft7.tr" ||
FILENAME == "ft8.tr") {
if(f==0) {
NR=1
tf=FILENAME
f=1
}
if(tf!=FILENAME) {
if(ff==0) {
NR=1
ff=1
}
}
if((NR>=st)&&(NR<=ed)) {
print $1" "$2" "$3" "$4" "$5 > "updatedpeer.tr"
}
}
}
END {
}
#awk '{str = $1 ; getline < "gg1.tr" ; print str " " $1 > "ppp.tr"}' gg.tr
```

CHAPTER 7

REFERENCES

1. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," Proc.ACM SIGCOMM '01, pp. 161-172, Aug. 2001.

2. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H.Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc.CM SIGCOMM '01, pp. 149-160, Aug. 2001.

3. A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," Lecture Notes in Computer Science, pp. 161-172, Springer, Nov. 2001.

4. B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," IEEE J. Selected Areas in Comm., vol. 22,no. 1, pp. 41-53, Jan. 2004.

5. S. Sen and J. Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks," IEEE/ACM Trans. Networking, vol. 12, no. 2, pp. 219-232, Apr. 2004.

6. Gnutella, http://rfc-gnutella.sourceforge.net/, 2010

7. Y. Liu, L. Xiao, X. Liu, L.M. Ni, and X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems," IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 2, pp. 163-174, Feb.2005.

8. D.R. Karger and M. Ruhl, "Finding Nearest Neighbors in Growth-Restricted Metrics," Proc. 34th ACM Ann. Symp. Theory Computing (STOC '02), pp. 741-750, May 2002.

9. B. Bolloba´s, Random Graphs, second ed. Cambridge Univ. Press,2001.

10. K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03), pp. 314-329, Oct. 2003.

11. S. Jin and H. Jiang, "Novel Approaches to Efficient Flooding Search in Peer-to-Peer Networks," Computer Networks, vol. 51, no. 10, pp. 2818-2832, July 2007.

12. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," Proc. Ninth Int'l Workshop Modeling, Analysis, and Simulation of Computer and telecomm. Systems (MASCOTS '01), pp. 346-353, Aug. 2001

13. Introduction to network simulator, Teerawat Issariyakul, Ekram Hossain, Springer 2009.