

LOAD BALANCING IN DYNAMIC MULTICAST NETWORKS



ISO 9001:2000

Thesis submitted in partial fulfillment of the requirements
for the award of the degree of

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE
AND ENGINEERING
OF BHARATHIAR UNIVERSITY**

By

Ms.LIYA JOHN	(Reg.No.9927K0135)
Ms. MAHALAKSHMI .R	(Reg.No.9927K0136)
Ms. NITHYA PRABHA .V	(Reg.No.9927K0146)
Ms.THILLAIKARASI .R	(Reg.No.9927K0169)

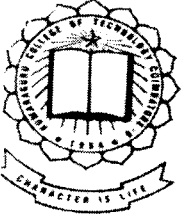
Under the guidance of

Mrs.J.CYNTHIA,M.E.Senior Lecturer,
Department of Computer Science and Engineering

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University)**

COIMBATORE-641 006

2002-2003



CERTIFICATE
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE, TAMILNADU-641 006



ISO 9001:2000

Department Of Computer Science and Engineering

Certified that this is a bonafide report of thesis work done by

Ms.LIYA JOHN	(Reg.No.9927K0135)
Ms. MAHALAKSHMI .R	(Reg.No.9927K0136)
Ms. NITHYA PRABHA .V	(Reg.No.9927K0146)
Ms.THILLAIKARASI .R	(Reg.No.9927K0169)

During the Year 2002-2003

Rajini
.....
j (Guide)

.....
(Head of the department)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE-641 006

Place: Coimbatore

Date: 13-03-03

Submitted for viva-voce examination held at
Kumaraguru College of Technology on 13-03-03

Rajini
.....
(Internal Examiner)

[Signature]
.....
(External Examiner)

Acknowledgement

ACKNOWLEDGEMENT:

This endeavor over a long period has been successful by the advice and support of many well-wishers. We avail this opportunity to express our gratitude and appreciation to all of them.

We would like to express our profound respect to our beloved Principal **Dr.K.K.Padmanabhan, B.Sc(Engg), M.Tech., Ph.D.**, for having provided the necessary facilities to complete this project.

We are greatly indebted to our beloved Head of the Department, **Dr.S.Thangasamy, B.E(Hons), Ph.D.**, Computer Science and Engineering, Kumaraguru college of Technology, for the inspiration and encouragement rendered by him.

We are greatly privileged to express our deep sense of gratitude to our course coordinator **Ms.S.Rajni, B.E , M.I.S.T.E , Senior Lecturer**, Department of Computer Science and Engineering, who has been a motivating force behind all our deeds.

We are grateful to express our gratitude and sincere thanks to our guide **Mrs.J.Cynthia M.E** and our class advisor **Mr.M.N.Gupta, B.E ,** Department of Computer Science and Engineering, who has been a constant source of encouragement for our project.

We like to express our special thanks to all staff members and **lab technicians** in the Department of Computer Science and Engineering, who helped us in the successful completion of the project. Finally we express our deep sense of gratitude to our parents, friends and all others who had been directly or indirectly involved in this project for their invaluable help.

Synopsis

SYNOPSIS:

Multicasting is the process of sending a packet from one sender to multiple receivers with a single send operation. Networks are capable of performing multiple multicast communications with different QOS requirements that compete for the network resources. The goal of multicast routing is to find a tree of links of interconnected routers that have hosts of the multicast group attached. The packets will be transmitted via this tree.

The factors involved in Multicast routing are the scalability of the network, the excess traffic received, traffic concentration and optimality of the forwarding .Our aim is to provide scalable, reliable multicast routing on an optimal routing path. The project also proposes a heuristics to avoid congestion by routing the traffic through alternate paths (if one is available).

The project deals with two main problems of multicast routing

1.Multicast Packing

2.Multicast Retransmission-Scoping

The amount of load on the link can be defined as the number of packets transmitted through it. The use of the same link on multiple groups is one source of increasing the network load. The proposed approach identifies alternate paths and also assures that the cost of the tree constructed is acceptable.

Retransmission on the network can lead to misuse of the valuable bandwidth allotted for the transmission .The complexity of retransmission problems affecting the network load is increased with a multicast network. The proposed approach uses multiple multicast channels.

Contents

CONTENTS:

1.INTRODUCTION	1
1.1 CURRENT STATUS OF THE PROBLEM	3
1.1.1 PROBLEM DEFINITION	5
1.2 RELEVANCE AND IMPORTANCE	7
2.LITERATURE SURVEY	11
3.SOFTWARE REQUIREMENTS	12
3.1 PROGRAMMING ENVIRONMENT	12
3.1.1 LINUX	13
3.1.2 ABOUT C	13
3.2 MODULE-I	14
3.2.1 FUNCTIONAL REQUIREMENTS	14
3.2.1.1 PHASE-I	15
3.2.1.2 PHASE-II	16
3.3 MODULE-II	16
3.4 EXCEPTION HANDLING	17
4.PROPOSED APPROACH TO THE PROBLEM	18
4.1 MODULE-I	18
4.2 MODULE-II	19
4.2.1 SENDER	20
4.2.2 RECEIVER	21
4.2.3ROUTER	21

5.DESIGN DETAILS	22
5.1MULTICAST PACKING	
5.1.1 PHASE-I	23
5.1.2 ALGORITHMS USED	
5.1.2.1 DIJKSTRA'S ALGORITHM	25
5.1.2.2 BRANCH AND BOUND ALGORITHM	26
5.1.2.3 CUTTING PLANE ALGORITHM	28
5.1.3 PHASE-II	28
5.2 MODULE-II	29
5.2.1ALGORITHM'S USED	30
5.3 DATA STRUCTURES USED	31
5.4 TOPOLOGY OF THE NETWORK	32
6.IMPLEMENTATION DETAILS	33
6.1MODULE-I	
6.1.1 PLATFORM'S USED	34
6.1.2 IMPLEMENTATION DETAILS	34
6.2MODULE-II	
6.2.1 PLATFORM'S USED	35
6.2.2 IMPLEMENTATION DETAILS	35
7.TESTING	
7.1 UNIT TESTING	38
7.2 INTEGRATION TESTING	39
	40

8.CONCLUSION AND FUTURE OUTLOOK	41
9.REFERENCES	43
10.APPENDIX	45
10.1 MODULE-I	
10.1.1 SAMPLE SOURCE CODE	46
10.1.2 SAMPLE OUTPUT	52
10.2 MODULE-II	
10.2.1 SAMPLE SOURCE CODE	54
10.2.2 SAMPLE OUTPUT	69

Introduction

1. INTRODUCTION:

The applications that demand the use of multicasting are bulk data transfer (the transfer of a software upgrade from the software developer to users needing upgrade), streaming continuous media the transfer of the audio, video and text of live lecture to a set of distributed lecture participants, shared data application (for e.g. a white board or teleconferencing application), data feeds, WWW (World Wide Web) cache updating and interactive gaming. Traffic management in a dynamic multicast network that demands access to multimedia network, low end-to-end delays, reliable and bulk data transport is of great consideration to increase the utility of the network resources.

Traffic management is the set of policies and mechanisms that allow a network to efficiently satisfy a diverse range of service requests. Traffic management subsumes many ideas traditionally classified under congestion control. A resource (link) is said to be congested, when it is overloaded and this results in performance degradation and loss of network efficiency. Congestion control policies either restrict access to the resource or scale back user demand dynamically so that the overload situation disappears.

Generally bottlenecks experienced by data traversing through the links of the Internet can be categorized according to their locations.

1. That occurs near the server.
2. That occurs near the client.
3. That occurs at the intermediate links and nodes.

A number of factors influence the congestion in the network link. The number of transmitted but yet to be acknowledged packets will affect the network traffic. Less number of packets for each acknowledgement will overload the network with the acknowledgements.

An approach to prevent congestion is flow control. Congestion arises when the nodes that are sending messages to a particular receiving node from the network users perspective. Flow control is a mechanism that prevents those messages from entering the network that cannot be delivered in the predefined time. Network flow control can also be used as a mechanism for distributing the traffic equally among network nodes. As such, flow control can reduce message delays and prevent one part of the network from becoming overloaded by another part of the network.

1.1 CURRENT STATUS OF THE PROBLEM:

1.1.1 PROBLEM DEFINITION:

Data in a multicast network is not transmitted to a single receiver. This requires multicast trees to be set up within the network. Routing metrics might include number of transmissions links, data rate, load on a link and number of links to reach the destinations and these affect the calculation done on the routing table and hence the multicast routing. Load on the transmission link can depend on the length of the corresponding queue and one can also evaluate the delay of data units within the intermediate system. The routing tables are updated based on the routing metrics and consequently adapted dynamically to reflect the current network load and group membership. Multicast routing should account for the dynamic changes in the group membership ,since changes can vary depending on the application.

In the multicast path, network load, loops and concentration points of traffic need to be avoided. Routing algorithm should therefore work incrementally and not monolithically. Thus the changes in the group membership should not necessitate a complete recalculation of the routing information for an entire network or for an entire group. Instead it should be possible to deal with these changes locally. Construction of the spanning trees avoids loops by creating

an overlay network that eliminates some of the links and intermediate systems in the network. The spanning tree ensures that all network works and end systems can be reached through it.

The transport layer is located above the network layer, whereas the network layer basically handles routing in multicast communication. The transport layer is responsible for tasks very similar to those in point to point communication. These tasks mainly include connection management, error detection, and error recovery and flow and congestion control. The aspect of reliability (a reliable service is one in which all the data is delivered to the receiver in the correct order without any errors and without any duplication) takes a special significance in the group communication.

SRM (Scalable Reliable Multicast) provides a semi reliable multi peer transport service. SRM is a receiver-oriented protocol in which the receivers themselves are responsible for error detection. Bit errors can be detected through checksum and lost data units through the use of sequence numbers.

SRM deals with the problem of lost data units remaining undetected until the receipt of the following data units by requiring all group members to send regular status reports, called sessions reports to the group. Reports are sent per multicast, other group members are able to compare this information which their own. Regular status reports sent by all group members are a problem for large group because they contribute to the network load and processing a large number of session reports may overload the member. To achieve a better scalable network, length of the intervals for sending status report are based on group size.

If on the basis of the sequence numbers, a group member determines that one or more data units have been lost, the host sends a negative acknowledgment to the group. It thereby requests the retransmission of the data unit involved. With SRM, all members of a multicast group are potentially involved in retransmission in order to ensure a reliable delivery of data. Negative acknowledgements however, always incorporate the risk of an acknowledgement implosion, if multiple receivers have not received data units. Performance of the SRM may

degrade heavily, if there is a crying baby –a receiver that loses packets frequently. Performance degrades because the repair requests are multicast to the group and many members may retransmit the repair to the entire group.

1.2 RELEVANCE AND IMPORTANCE:

1. OPTIMAL CONSTRUCTION OF MANY TO MANY MULTICAST TREES:

We propose an integer programming formulation for the optimal tree and discuss a solution procedure that combines implicit enumeration with cutting plane technique. We can solve the problem to optimality without running into computational problems for network with several hundred nodes. The branch and cut algorithm can be explained as follows. Given the integer-programming problem (with many constraints), the idea is to recursively partition the solution space and to solve the relaxation of the formulations without generating all of the constraints at once. Branching on variables to zero or one, partitions the solution space. In many to many multicasting, any node can be source node and they share multicast routing graph. Construction of an efficient multicast graph is necessary for the better use of the network resources.

2. EFFECTIVE CONGESTION CONTROL- BETTER TRAFFIC MANAGEMENT:

The project employs a two-way approach to control congestion

- 1) It attempts to construct an optimal tree and the heuristic aims at removing the most congested link, provided one exists. Such a proposal is a preventive mechanism to congestion control. This enhances the multicast routing graph and hence the protocol performance is also enhanced.

2) It attacks the retransmission-scoping problem by the allocation of multiple multicast channels. At the receiver side, the unwanted processing is done (when retransmission packets arrive) by the receivers, which have not initiated any negative acknowledgment. Since the retransmission occurs on a separate channel those receivers that are error free don't suffer any delay in receiving packets.

3.RECEIVER BASED APPROACH:

The receiver is necessary for identifying the error in the packet transmission and initiating the negative acknowledgment .It does not require only the sender to retransmit the packet, any secondary member that has received the packets in the right order can retransmit them. This approach hence avoids the overheads that are involved with the central source. This approach also avoids the bottleneck, that occurs only the sender transmits, detects errors and also performs retransmission.

4.BENEFITS OF SCALABLE RELIABLE MULTICAST:

- 1) Scalable reliable multicast using multiple multicast channels is effective because it overcomes the RETRANSMISSION SCOPING PROBLEM.
- 2) It provides reduced receiver processing overhead and reduced network bandwidth communication because of the use of multiple multicast channels.

Literature Survey

2. LITERATURE SURVEY:

Reliable multicast has been an active research area in the last few years. Several architectures and protocols have been proposed.

One of the most popular existing reliable multicast protocols are scalable reliable multicast .SRM is a NAK (Negative Acknowledgement)-based protocol that has been implemented for a shared whiteboard application. In its basic form, SRM suffers from the problem of unwanted redundant packets being sent to, and processed at, receivers. Local recovery enhancements in SRM are likely to scale down this problem but not solve it. Local recovery helps to isolate the domains of loss and thereby reduce global retransmissions. For a multicast transmission with thousands of local neighborhoods, unless receivers are arranged in a hierarchy with a small bounded degree and re transmissions if only one channel is used for both transmissions and retransmissions. Log-based reliable multicast (LBRM), are reliable multicast transport protocol (RMTP), are two hierarchical approaches in which designated receivers at a certain level supply repairs to lower-level designated receivers or loggers. The problem of placing these designated receivers and determining their processing and storage requirements is still being studied.

There has been an increasing interest in using processing and storage inside the network for enhancing reliable multicast performance. Control-on-demand and active reliable multicast (ARM) propose to maintain retransmission state for selective forwarding of retransmissions and for duplicate NAK suppression. Here we propose to use the control-on-demand architecture for implementing multiple multicast channels. ARM, control-on-demand propose the use of buffering "current -data" at strategic locations inside the network for providing local retransmissions. Even though it has been shown that buffering data inside the network for purpose of retransmissions results in higher performance, the issues of where to place buffers and when and how to invoke

“repair service “remain open questions. Here the focus is made on error recovery only form the sender.

Reliable multicast could benefit form the use of forward error correction (FEC). This is because FEC techniques allow recovery of multiple lost packets with the help of single FEC packet. FEC based loss recovery; using end-to-end means only, will not perform well in the presence of heterogeneous loss. Even if only a few receivers experience very high loss, large number of FEC packet will be generated at the sender and will be sent everywhere thereby wasting network bandwidth and causing unnecessary packet processing all the other receivers. FEC technique could be combined with the mechanism of using additional router support for implementing multiple channels to enhance our work.

In an approach based on IP time-to-live (TTL) has been proposed for scoping retransmissions. There are two problems with TTL-based scoping. First, TTL based scoping limits the packets within a radius and is not suitable for tree structure as in the case of multicast. Second, it is hard to approximate a good TTL value.

It is proposed to use multiple multicast groups for flow and congestion control, but not for error recovery. The possibility of using separate multicast groups for defining “local groups” for local recovery has been suggested. Holbrook proposes the use of separate retransmission channel, for recovery as future work.

The idea of destination set splitting for improving the throughput of some specific positive acknowledgment-based point-to-multipoint protocols is also proposed. They suggested that the receivers could be divided into groups based on their capabilities and that the sender would carry out as many simultaneous independent conversations as the number of groups. The two proposed system mentioned above differ in three significant ways. First we do not group the receivers based on there capabilities. Rather we group packet such that retransmissions of packet belonging to each group is done on separate multicast channel. Second is the protocol we have considered is the generic NAK-based protocol instead of specific ACK-based protocols. Third, in addition to point-to-

multi-point scenario, we have also considered the multipoint-to-multipoint scenario.

Among the existing analytical work on reliable multicast, the work of Towsley, Kurose and Pingali provides a simple analytical framework for studying the performance of reliable multicast protocols. They have used this framework for a quantitative demonstration of the superiority of receiver-initiated NAK-based approaches over sender-initiated ACK-based approaches. This work has subsequently been used in several performance analyses. This forms the basis of our analyses.



Software Requirements

3. SOFTWARE REQUIREMENTS:

3.1 PROGRAMMING ENVIRONMENT:

3.1.1 LINUX:

Linux is an operating system for several types of computer platforms but primarily for INTEL based pc. The system has been designed and built by hundreds of programmers scattered around the world. The goal has been to create a UNIX clone free of any commercially copyright software, which the entire world can use.

Linus Torvalds developed Linux in the early 1990's along with other programmers around the world.

Linux is basically a UNIX clone which means that with Linux we get many of the advantages of UNIX. Linux multitasking is fully pre-empted meaning that we can run multiple programs at the same time and each program seems to process continuously.

Linux allows starting a file transfer, printing a document, copying a file using a CD-ROM and playing a game all that the same time.

WHY USE LINUX?

LINUX is selected because it is the only operating system today that is freely available to provide multitasking and multiprogramming capabilities for multiple users on IBM PC –compatible hardware platforms. You aren't logged into upgrading every few years and paying outrageous to update all your applications. Many applications for Linux are freely available on the Internet. Thus we have access to the source code to modify and expand the OS to our needs something we can't do with commercial OS such as WINDOWS NT and WINDOWS 95.

FEATURES OF LINUX:

Here are some reasons why LINUX could be the best operating system.

A Linux distribution has thousands of dollars worth of software for no cost. Linux is a complete operating system that is

- 1) **STABLE**-Crash of an application is much less likely to bring down the operating system under Linux
- 2) **RELIABLE**-Linux servers are offered to run up for hundreds of day compared with the regular reboots required with a windows system.
- 3) **Extremely powerful.**

Linux comes with a complete development environment, including C, C++, FORTRAN compilers, toolkits such as QT and scripting languages such as PERL, AWK. Excellent networking facilities: allow you to share modems etc., The ideal environment to run servers such as a web server (e.g. Apache) or FTP server. Wide variety of commercial software is available if your needs aren't satisfied by the free software. An operating system that is easily upgradeable.

2.1.2 ABOUT C:

The language selected for this project is "C" efficient for network-oriented project. This language is efficient, powerful and compact.

C has emerged as the language of choice for most applications due to speed. Portability and compactness code is highly portable; programs running on computer can be used with another computer with different operating systems with slight or no modification.

Program written in C are efficient and fast which is due to its variety of data types and powerful operators. Several standard functions are available in C, which can be used for developing programs.

3.2 MODULE-I:

In module-I the project actually deals with the establishment of the routing graph with minimized congestion.

3.2.1) FUNCTIONAL REQUIREMENTS:

3.2.1.2) PHASE-I:

INPUT:

The set of nodes and the links connecting them on the network under consideration.

OUTPUT:

Multicast routing graph for each of the groups [each group for a different application and each identified by a common multicast Address.

FUNCTIONS PERFORMED:

Solve for the optimization problem, which results in a solution that connects all the multicast nodes. Each routing graph connects the source node or the sender with all receivers, based on the weight of the link. The solution assures that it connects the nodes with the minimum load.

3.2.1.2) PHASE-II:

INPUT:

Set of multicast trees each for a group.

OUTPUT:

Refined multicast tree.

FUNCTIONS PERFORMED:

Each tree constructed is embedded on the physical network. The links that are common to two or more links are the ones with the greater risk of congestion. The link with the maximum congestion is identified. The congestion value is checked if it is well below the allowable load on the link, the path is left unaltered. Else the

solution searches for an alternate path and if one is found, the tree is updated else left as such. The user of the network or the protocol is expected to take care of the latter case and make adjustments accordingly by manually turning of the link or by sending packets at reduced rate or by preventing pack transmission until congestion is over.

3.3 MODULE-II:

3.3.1 FUNCTIONAL REQUIREMENTS:

In a reliable multicast scenario using a single multicast channel all packet transmissions and retransmissions are done. Each receiver therefore receives the entire retransmitted packet irrespective of whether it receives packets correctly or not. The above imposes unnecessary receiver processing overhead and wastes network bandwidth on links. Hence the fundamental problem in reliable multicast is how to scope the retransmissions so as to shield receivers and the links leading to them from loss recovery due to other receivers.

Here we implement an approach that allows one to overcome this retransmission-scoping problem in a multicast scenario. The approach consists of using multiple channels for transmission of data reliably. A lost packet is retransmitted through a separate channel. This approach will reduce the receiver processing due to reception of unwanted packets as well as the network bandwidth consumption.

3.4 EXCEPTION HANDLING

Exception handling, including the actions to be taken and the messages to be displayed in response to undesired situations or events is briefly outlined here.

1. The user interface is well defined. Validation of the user input is performed and corresponding message specifying the corrective actions to be taken are displayed on the screen.

For e.g.: the user's input regarding the node number is verified to check, if it specifies the number indicates the node under test, else an error message is displayed on the screen and the user is directed to give a valid input.

2. If during the search of the path, the nodes in the intermediate path cannot be linked, then the user is informed regarding this using a message

*Proposed Approach To The
Problem*

4 PROPOSED APPROACH TO THE PROBLEM:

4.1 MODULE-I:

The project aims at providing an efficient method for the optimal sharing of the network resources among several multicast groups that co-exist in the network. A shared tree is considered as the backbone of the group multicast session.

Considering each multicast session in isolation an independently may cause congestion on some links and reduce network utilization. Thus we define the multicast packing problem in which the network tries to accommodate simultaneously all the multicast groups while trying to avoid bottleneck on some links for higher throughput. Our project aims to reduce the sharing of the link while ensuring that the size of the multicast tree will never exceed the αOPT^k where opt is the size of the optimum multicast group k in isolation. Optimum multicast tree for each group (in isolation) is computed by using cutting plane inequalities and branch & bound algorithm. Let us denote the physical network by $G=(V, E)$ where V is the set of all nodes in the network and E is the set of all edges. With each link we associate a weight $w_e > 0$ based on the distance .a multicast group $M \subseteq V$, and the objective of the design problem is to find a sub-graph that spans M and has the minimum total cost. The sub-graph is again a tree and cost is measured as the sum of weight of the edges in the solution.

The optimality criterion is the minimization of the network congestion, which is defined as the total load of the most congested edge in the network. The load of the edge is the total traffic demand summed over the multicast groups using that edge.

PROBLEM FORMULATION:

In the formulation we define the traffic load for a multicast group k is t^k

The objective function is

Minimize λ

Where $\lambda = \max_e \{Z_e\}$

$$Z_e = \sum_{k \in K} t^k$$

$\lambda \rightarrow$ Denotes the maximum congestion on the link e

$Z_e \rightarrow$ denotes the total traffic demand summed on the link e due to the multicast group k .

The objective function takes the congestion on the link as the main parameter and this objective is under the major consideration to remove congestion on the links.

Subject to the constraints

$$x^k \in ST^k \text{ for all } k \in K$$

$$ST^k = \{x^k \in \{0,1\}^{|E|} : x^k \text{ induces a Steiner tree spanning } M\}$$

We use integer-programming formulations, which uses Steiner-cut inequalities to construct such trees. Given the optimum multicast tree for each step the preprocessing step computes the congestion value on each link. Starting from the most congested link and the tree that uses this link is identified .the approach searches for an alternate path if this link is disconnected temporarily. If one is found it updates the new routing path as per the new one.

4.2 MODULE-II:

The project aims to implement the concept of Scalable reliable multicast using multiple multicast channels by the following ways.

4.2.1 SENDER:

The sender is responsible for sending data from a specific file by framing it into packets. The sender is identified by the IP address and uses a specific port

for the transmission of data. All the receivers and routers have information about the sender in a header file, which has defined in it the IP address of the sender

Sender keeps track of the number of packets sent and displays the sequence number of packets after sending each packet. Sender is also responsible for the retransmission of data packet incase of the receiver not receiving packets due to loss or timeout. Sender consists of two processes, which control transmission and retransmission of packets through a designated channel.

The sender may refuse to transmit data packets incase the socket creation is out of bound. Usually port numbers are 16-bit unsigned binary numbers, each one in the range 1 to 65535(0 is reserved). Certain applications have default port numbers assigned to them. For example port number 24 has been assigned to the File Transfer Protocol .we have used port numbers above 1024 for transmission and retransmission of data packets.

The retransmission process listens to the default port number for requests from the receiver reporting data loss and requesting retransmits. The sender usually initiates the transaction by sending the data packets to the router.

4.2.2 RECEIVERS:

The receivers are usually connected to a router, which forwards the packet to them. Receivers usually timeout after a few seconds and then request sender for a retransmit of the particular data packet through a separate channel which is designated by a different port number.

The receiver usually performs the following:

- 1) It sends a NACK to the sender on receiving a packet, which is not in sequence.
- 2) It sends a NACK to the sender requesting retransmission of data packet in case of a time out.

The receiver usually sends the NACK and waits for a while and after timeout sends a NACK. This process is usually tried certain number of times. The receiver is usually bound to a port number so that it can receive packets from the router.

4.2.3 ROUTERS:

A computer network usually consists of machines interconnected by communication channels. We call these machines hosts and routers. Hosts are computers that run applications such as web browsers; the application programs that are running in hosts are the users of the network. Routers are machines whose job is to relay or forward information from one communication channel to another.

Routers are important simply because it is not practical to connect every host directly to every other host. Instead, a few hosts connect to a router, which connects to other routers and so on to form the network. This arrangement lets each machine get by with a relatively small number of communication channels; Programs that exchange information over the network, however do not interact directly with routers and are generally unaware of their existence.

Three routers are used for the implementation. These include a source router and two routers connected to this source router. The sender is connected to the source router. The receivers are connected to the routers. The router usually operates by reading a file to obtain the IP address of the receivers to which it must forward the packets.

The routers are also responsible for forwarding the NACK from the receivers to the sender if such a situation arises due to packet loss or timeout.

Design Details

5. DESIGN DETAILS:

According to Webster the process of the design involves “conceiving and planning out in the mind” & “making a drawing, pattern or sketch of “ and here is a brief sketch of our solution to the problem we have discussed.

As briefed out so far, the project has two major modules

1.Multicast Packing

2.Multicast Retransmission Scoping

Modular system that we intend to develop has well defined, manageable units with well-defined interfaces among the units. The functions that are used in the subsystem have a single and a well-defined purpose.

5.1 MULTICAST PACKING:

In module-1 our goal is to find an efficient, less congested routing graph. The module is further subdivided into two phases.

5.1.1 PHASE-1:

Here the input to the processing is the set V which is of all nodes of the network and set of edges E that connects two nodes of V .The input also gives a list of set of multicast nodes M for each network.

The following are the steps involved in processing

- 1.Construct a complete graph (M, E') such that $E'=\{i, j: i \neq j \text{ and } i, j \in M\}$ and $w_e \in E' = d_{i j}$ (Distance between i, j computed using dijkstra's algorithm)
- 2.Find the minimum spanning tree of (M, E') using branch and bound algorithm.
- 3.For every pair of nodes in the spanning tree, identify a path on the actual network.
- 4.If that node is not present in final group of multicast nodes add on to the final vertex list V_m and update edge list.

SET REPRESENTATION:

$V \rightarrow$ SET OF VERTICES = $\{M1, M2, M3, \dots\}$

Where each M is the set of vertices for the multicast group

$E \rightarrow$ SET OF EDGES = $\{E1, E2, E3, \dots\}$

Where each E is the set of edges for the multicast group M

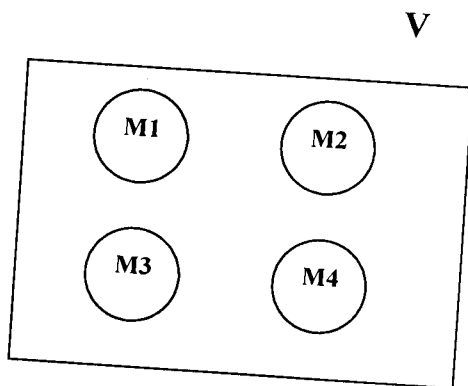


Fig (i)

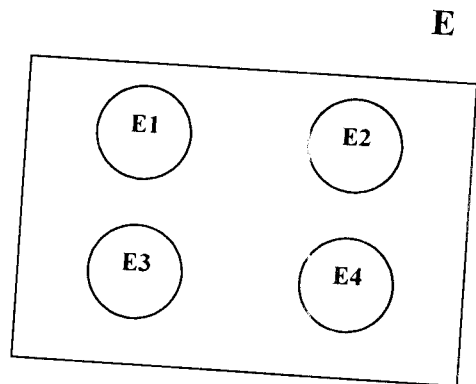


Fig (ii)

For each (M_i, E_i) draw a complete graph

For e.g. for $M1 = \{1,2,3,4,5\}$, the complete graph

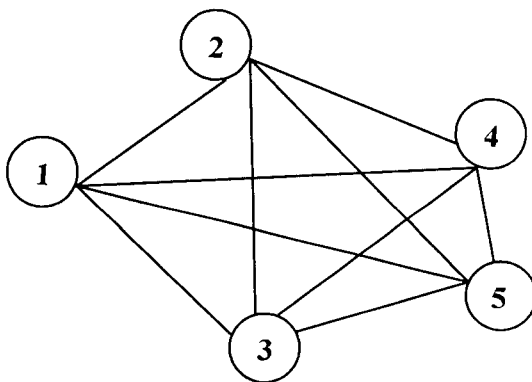


Fig (iii)

5.1.2. ALGORITHM'S USED:

5.1.2.1. DIJKSTRA'S ALGORITHM:

This algorithm is used to estimate the weight of the link connecting two nodes. This mainly takes into account the number of hops required to reach the destination and also the initial weight that is assigned to each link can take into care the link capacity between the two nodes. The algorithm can be basically explained as follows.

We define two sets of nodes P and T (P stands for permanent and T stands for temporary). Set P is the set of nodes to which the shortest has been found and set T is the set of nodes to which we are considering shortest paths. We start by initializing P to the current node, and T to null. The algorithm repeats the following steps.

1. For the node P just added to P, add each of its neighbors n to T such that

a) If n is not in T, add annotating it with the cost to reach it through P and P'.

b) If n is already in T and the path to n through P has a lower cost, then remove the earlier instance of n and add the new instance annotated with the cost to reach it through P and P'.

2. pick the node n that has the smallest cost in T and if not already in P add it to P use its annotation to determine the router P to use to reach n. If T is empty we are done.

5.1.1.2) BRANCH AND BOUND ALGORITHM:

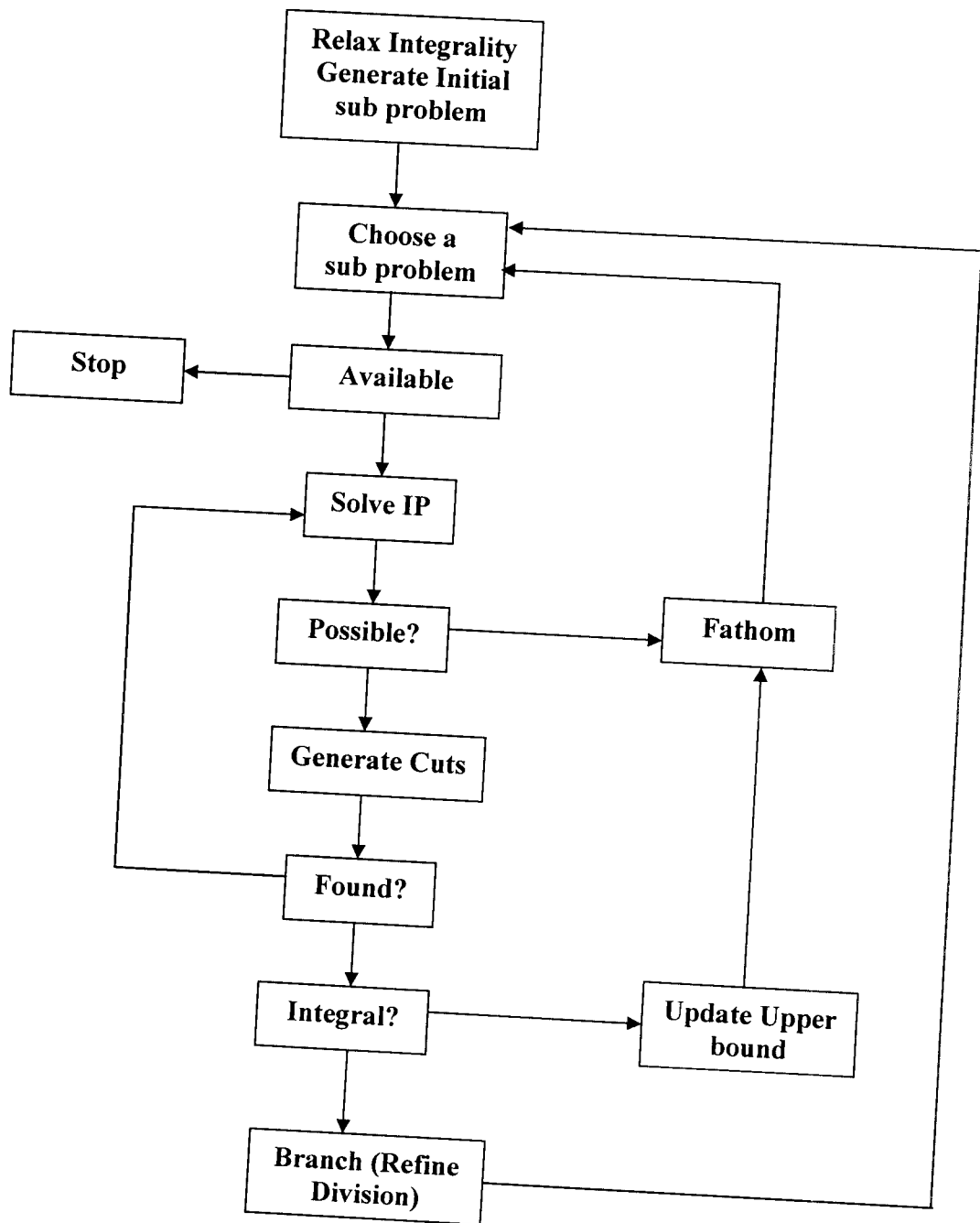


Fig (iv)

BRANCH AND BOUND ALGORITHM:

In practice, using a technique of branch and bound solves most integer programming problems. Branch and bound methods find the optimal solutions to an IP by efficiently enumerating the points in a sub problems feasible region and the fig(iv) describes an outline of the algorithm.

The procedure starts by solving the LP relaxation of the IP. Our next step is to partition the feasible region for the LP relaxation in an attempt to find out more about the location of the IP's optimal solutions. We create sub problems by branching on the variables. A tree like structured as shown in fig(v) is created.

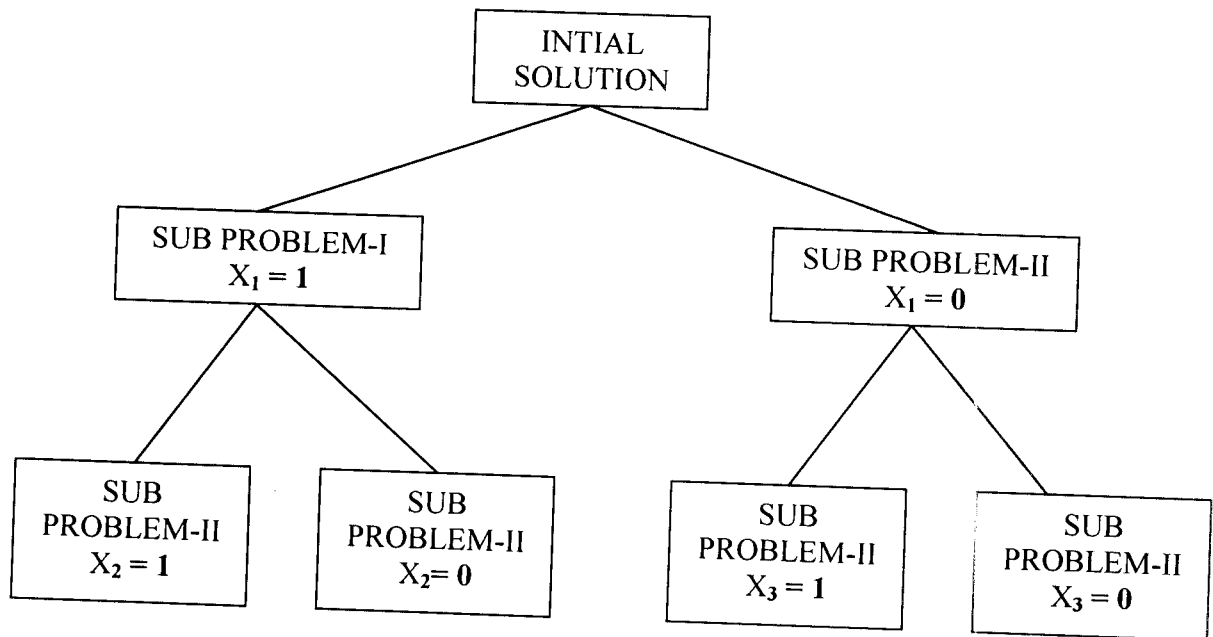


Fig (v)

The constraints associated with any node of the tree are the constraints for the LP relaxation on plus the constraints associated with the arcs leading from the sub problem 1 to the node. When further branching on a sub problem does not yield any useful information, we say that sub problem is **fathomed**.

5.1.1.3 CUTTING PLANE ALGORITHM:

To apply cutting plane algorithm, we begin by choosing any constraint in the LP relaxation's optimal solution in which a basic variable is functional. We can choose any constraint arbitrarily. The cutting plane algorithm then proceeds by adding the new constraint to already existing formulation. The newly generated constraint is called a cut. The cut will satisfy any feasible point for the IP and the current optimal solution to the LP relaxation will not satisfy the cut and proceed until we get an optimal solution.

5.1.3. PHASE-II:

Here our input is set of individual multicast trees T_1, \dots, T_n for each of the nodes and a bound on the tree is $\alpha \text{ OPT}^K$ where α can be any arbitrarily value depending upon the application. In this we obtain revised multicast trees T' .

The processing steps can be outlined as follows

1. Sort all edges of a tree by the value of congestion.
2. Choose an edge e' with maximum congestion.
3. Locate the trees with the edge e' .
4. Update the tree by avoiding the link e' by calling the procedure rebuild.
5. Update the new Z_e values.

5.1.3.1. PROCEDURE REBUILD:

1. The congestion of the link is compared with maximum link capacity of the node and two situations as below may result.

(i) Congestion of the link is less than MAXCONGESTION, the link has got no congestion and the link is efficiently used. The procedure rebuild returns control with the message "efficient routing graph".

(ii) Congestion of the link is above MAXCONGESTION. In such a case; the procedure searches for an alternate path. If one is found the tree is checked for the size it is less than αOPT^k is updated else it cannot be updated. The user is informed about the congested link and he is to take the alternative steps.

5.2 MODULE-II:

Some applications require reliable delivery of data. Examples of such applications include file distribution and collaborative work. The IP multicast service model does not guarantee delivery and there is no estimate of group size - any packet can be sent to any subset. In scalable reliable multicast we provide an approach for providing reliable, scalable multicast communication, involving the use of multiple multicast channels for reducing the receiver processing costs and reducing the network bandwidth in a multicast session. This is based on a new paradigm that retransmission of packets are done on a separate channels which receivers dynamically join and leave.

5.2.1 ALGORITHMS USED:

The sender, router and the receivers work as per the respective algorithms. Retransmission requests usually arise whenever the receiver timeout or receives a packet which is not in sequence.

5.2.1.1. SENDER ALGORITHM:

It consists of two processes these include the transmission process and the retransmission process

The flow of the algorithm for the transmission process is as follows:

- 1) Data is read from the file and framed into packets.
- 2) Packets are sent to the source router using a sendto() call.

The Retransmission process does the following:

- 1) Listens to NACK in the retransmission channel.
- 2) Retransmits packets from an array based on the NACK.

5.2.1.2. RECEIVER ALGORITHM:

The receivers are usually connected to any 1 of the routers. The flow of the algorithm for the receiver is as follows:

- 1) The receiver usually binds to a particular port using the bind () call.
- 2) It listens to the port for any incoming packet using a recvfrom () call.
- 3) If there are no incoming packets then a NACK is sent to the sender after a timeout.
- 4) Step 3 is usually tried out for a certain number of times after which it is given up.

5.2.1.3. ROUTER ALGORITHM:

The routers execute a socket program that which receives a data packet and forwards it on its entire outgoing links.

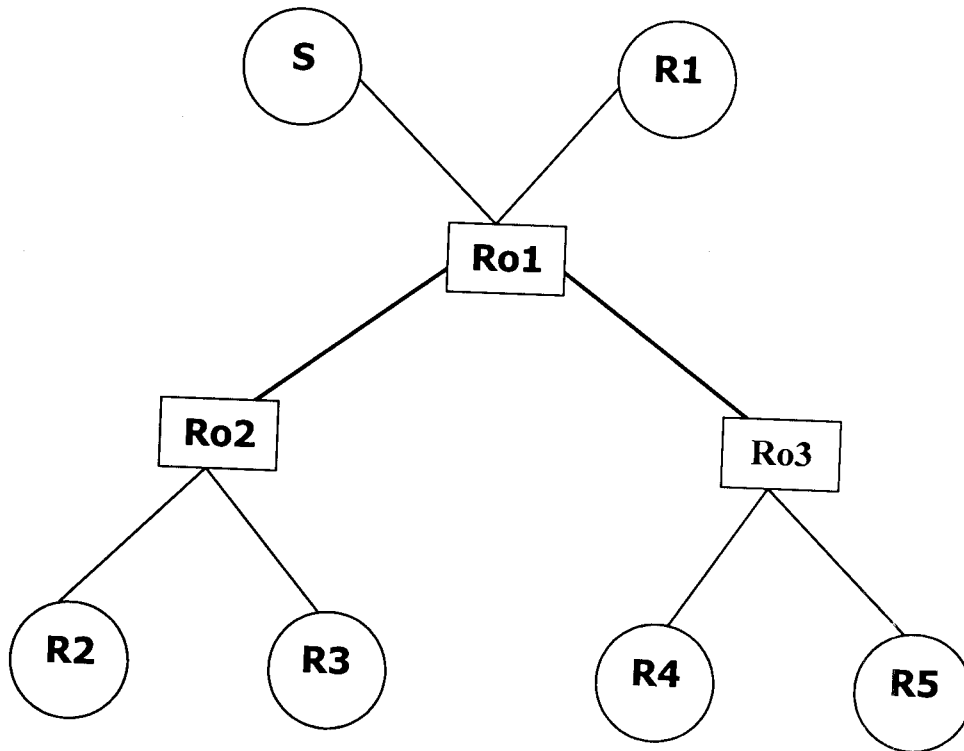
The flow of the router algorithm is as follows:

- 1) Get the IP address of all its nodes.
- 2) Open a connection to each of its node through the transmission channel.
- 3) Open a connection to each of its node through the retransmission channel.
- 4) Send the packets to the nodes through the transmission channel and make sure there is no duplication.
- 5) If packet is a NACK it is forwarded to the sender for retransmission of the packet through the retransmission channel avoiding duplication.

5.3. DATA STRUCTURES USED:

All basic structures such as int, float and character variables are used. The algorithm uses linked list to maintain the list of nodes under construction. The algorithm uses an adjacency matrix to represent the network with value of the array $[i, j]=1$, if the link exists else array $[i, j] =0$, if no link exists. The other basic structure in use is the arrays

5.4 TOPOLOGY OF THE NETWORK:



S → Sender
R → Receiver
Ro → Router

The topology used for testing included a source and another system. The multicast tree is formed by the inter connection of **three routers Ro1, Ro2, Ro3** and the designated receivers are **R1, R2, R3, R4**.

Implementation Details

6. IMPLEMENTATION DETAILS:

This part of the project is concerned with the translation of the design specifications into source code.

6.1 MODULE-I:

In module-1, we have developed the simulator to test the efficiency of the approach specified. The simulator was designed to test all aspects of the approach specified.

6.1.1 PLATFORMS USED:

Operating system: Windows

Programming language: C

6.1.2 IMPLEMENTATION DETAILS:

6.1.2.1 PHASE-I:

In PHASE-I, dijkstra's algorithm is implemented using a link list and iterative procedures are used to perform the operations until the weight is calculated for each of the edges.

6.1.2.2 PHASE-II:

Initially we define the Steiner part ion $P = \{ S_1, S_2, \dots, S_n \}$ where each S_i is a singleton set containing a single node starting with this we generate the multicasts and proceed by branching on the solution using integrity constraints $1 \geq X_e \geq 0$

6.2 MODULE-II:

6.2.1 PLATFORMS USED:

Operating system: LINUX

Programming language: C

6.2.2 IMPLEMENTATION DETAILS:

Implementation of scalable reliable multicast is done in the sender, router and the receiver by programming using sockets as follows:

6.2.2.1 SENDER:

The sender performs the following functions:

1) Opening a socket:

The sender issues the socket call to ask the operating system to create a socket. The operating system enters the local IP address and the port number in the local socket field of the created socket

2) The sender repeats the following steps as long as it has packets to send :

- **Sending:** The sender reads data from a file and stores them as packets in a packet array. After receiving the socket descriptor from the operating system, the sender issues the send to calls to send the packets to the router.
- **Receiving:** The sender issues the Recvfrom to receive NACK from the receivers.

3) Closing: When the sender finishes sending the packets to the router it issues a close call.

6.2.2.2 ROUTER:

The router performs the following functions:

1) Reading IP address from the file:

The router opens a file, which contains the IP address of its node and reads the IP address to which it must forward the packets.

2) Opening the sockets:

The router issues the socket call to ask the operating system to create a socket for the normal transmission of the packet. The operating system enters the local IP address and the port number in the local socket field of the created socket. Another socket is created for the retransmission channel.

3) Bind the socket:

The router issues the bind call to prevent other programs entering in the same port.

4) Receiving packets:

The router issues a Recvfrom to receive packets from the sender

5) Send packets to nodes:

The router issue a send to () call to forward the packets to all it's associated nodes and to the other two routers incase it is the source router.

6) Send NACK to the receiver:

The sender issues a recvfrom call to listen to the retransmission channel for NACK to be forwarded to the sender and also for packets, which need to be retransmitted to the receiver.

6.2.2.3. RECEIVER:

1) Opening a socket:

The receiver issues the socket call to ask the operating system to create a socket. The operating system enters the local IP address and the port number in the local socket field of the created socket.

2) Bind the socket:

The receiver issues the bind call to prevent other programs in entering in the same port.

3) Recvfrom ():

The receiver issues a recvfrom () to receive packets from the router. It also starts a timer and when the timer goes out the receiver usually asks for a retransmit by sending a NACK to the sender.

Testing

7. TESTING:

Verification is performed at each of the phases to ensure that only the error free products are passed on to subsequent products. Validation is performed at the end to check if the product conforms to the user requirements or not. The following are the testing performed in the project.

7.1 UNIT TESTING:

Unit testing comprises the set of tests performed by an individual programmer prior to integration of the unit into a larger system.

Each of the algorithms described so far are considered as single program unit. Functions are implemented to perform the operations of the algorithm.

Functional tests are conducted to check for the reliability, correctness of the code and data input to the function are verified and the transactions are also validated.

As said adjacency matrix is used to represent the networks. Thus for adjacency matrix is just a 2×2 array and since the graph is a complete one the representation is considered to be more effective. The network congestion is defined as the total load of the most congested edge in the network. The load of an edge is the total traffic demand summed over the multicast groups using edge.

Stress tests are also conducted. The algorithms are found to be more reliable and they are also found to be more efficient.

Debugging the execution path for a wide range of the input value performs structure tests

7.2 INTEGRATION TESTING:

The various program units are integrated to provide the required functionality. Interfaces here are defined by passing the output of one function as a parameter to the subsequent function. Global variables are also accessed and they provide another form of interaction between functions.

Bottom up integration is followed, i.e. first exhaustive unit testing, followed by the sub system testing, followed by the testing of the entire system. The fig (vi) gives an overview of the testing involved in module-I

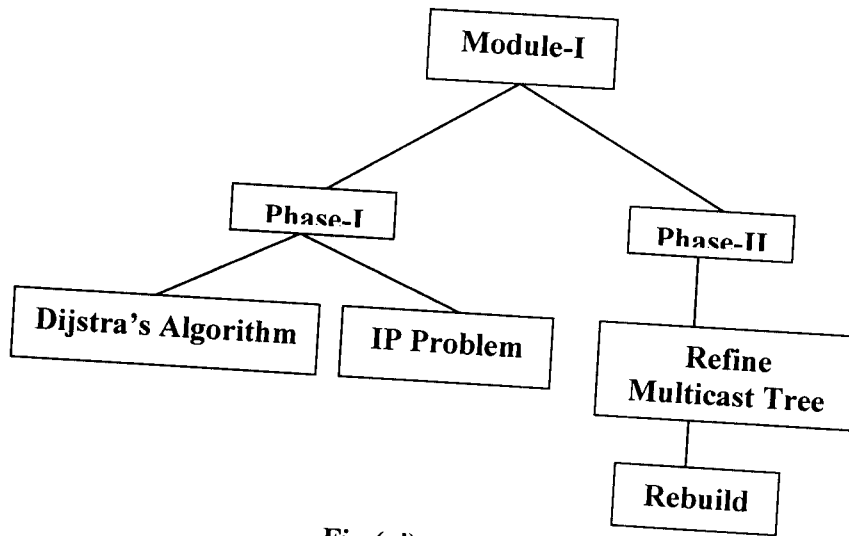


Fig (vi)

The simulator has been listed until the maximum number of nodes. Due to memory requirements of the current system, a maximum of 15 nodes were taken and the maximum size of each multicast group is 5 nodes. The system is thoroughly tested for its maximum volume of the input it takes in and it is found to be more reliable.

*Conclusion And Future
Outlook*

8. CONCLUSIONS AND FUTURE OUTLOOK:

Our proposed approach will result in an efficient and a congestion-less network.

The future scope of the project is to implement the concepts so far discussed on a multicast networks on the Internet backbone. The approach takes into care the congestion and also the maximum group size. It is expected to more efficient and has got a good performance due to the use of branch and bound algorithm. The algorithm rebuilds works only incrementally and not monolithically and hence other parts of the network are not affected.

SRM is a framework for reliable multicast that is still in progress, with areas such as local recovery, congestion control, and ADU naming still under investigation. This project has mainly explained the working of SRM in the application layer. Further studies on how to implement SRM in the network layer by configuring IP address and configuring each LINUX box as a m-router might be undertaken. Reliable multicast schemes often cannot scale to large receiver sets due to the problems of state explosion and message implosion.

Current work in SRM includes the Erasure Correcting Scalable Reliable Multicast, ECSRM. ECSRM is based on the SRM framework proposed by Floyd et. al., which utilizes NACK suppression to reduce message implosion. ECSRM makes a number of modifications to SRM to addressed enhanced scalability and rate control. Most notably, instead of re-sending lost packets, erasure-correcting encoded packets are sent in response to NACK messages.

References

REFERENCES:

- 1.M.Grotchel A. Martin and R.Weismantel
“Packing Steiner: a cutting plane algorithm and computation”
Mathematical Programming
- 2.Andrew S.Tanenbaum
“Computer Networks”
Prentice –Hall of India Private Limited, 2001
- 3.Ralph Witt man and Martina Zitterbart
“ Multicast Communication- Protocols and Applications”
Morgan Kaufman Publishers, 2001
- 4.Vijay Ahuja
“Design and analysis of computer communication networks”
McGraw-Hill International editions.
5. www.aciri.org/floyd/srm.html
6. research.microsoft.com/barc/mbone/ecsrn.htm
7. www.ieee.org

Appendix

10.APPENDIX:

10.1 MODULE-I:

10.1.1 SAMPLE SOURCE CODE:

10.1.1.1 Code for dijkstra's algorithm

```
while(src<=no_of_nodes+1)
{
    for(i=0;i<no_of_nodes;i++)
    {
        dist[i]=a[src-1][i];
    }
    for(i=0;i<no_of_nodes;i++)
    {
        if(dist[i])
        {
            dir_edge[src-1][i]=1;
            dir_weight[src-1][i]=1;
            path[src-1][i]=src-1;
        }
        else
        {
            if((src-1)==i)
            {
                dir_edge[src-1][i]=0;
            }
        }
    }
}
```



```

        dir_weight[src-1][i]=0;
        path[src-1][i]=-1;
    }
}
}
for(i=0;i<no_of_nodes;i++)
{
    if(dist[i])
    {
        for(j=0;j<no_of_nodes;j++)
        {
            if(a[i][j])
            {
                if(dir_edge[src-1][j]==0)
                {
                    dir_edge[src-1][j]=1;
                    dir_weight[src-1][j]=dir_weight[src-1][i]+1;
                    dist[j]=1;
                    if(dir_edge[src-1][j]!=0)
                        path[src-1][j]=i;
                }
            }
        }
    }
}
src++;
}
}

```

10.1.1.2 Code for spanning tree generation:

```

first=(node *)malloc(sizeof(node));

```

```

data=1;
create(first);
noVertex=no_of_multi_nodes;
span_vertex[0]=multicast_member[0];
noEdges=edge_count(edge);
noSEdges=edge_count(span_tree);
while((noEdges!=0)&&(noSEdges!=(no_of_multi_nodes-1)))
{
    for(n=0;n<=k;n++)
    {
        l=span_vertex[n];
        tnode=first;
        do
        {
            m=tnode->data;
            if(edge[l][m])
            {
                if(weight[l][m]<less_weight)
                {
                    less_weight=weight[l][m];
                    less_l=l;
                    less_m=m;
                }
            }
            tnode=tnode->next;
        }while(tnode!=NULL);
    }
    k++;
    span_vertex[k]=less_m;
    first=link_del(first,less_m);
    edge[less_l][less_m]=0;
}

```

```

        span_tree[less_l][less_m]=1;
        less_weight=35;
        noEdges=edge_count(edge);
        noSEdges=edge_count(span_tree);
    }
    noSEdges=edge_count(span_tree);
    cleardevice();
    if(noSEdges==(noVertex-1))
        sprintf(output,"Correct Spanning\n\t\tTree");
    else
        sprintf(output,"No Spanning Tree Can Be found");

```

10.1.1.3 Code for congestion computation and checking link removal:

```

for(i=0;i<no_of_nodes;i++)
{
    for(j=0;j<no_of_nodes;j++)
    {
        if(span_tree[i][j]==1)
        {
            final_edge[i][j]=1;
            final_weight[i][j]+=1;
            temp1=weight[i][j]*traffic[mn];
            congestion[mn][i][j]+=temp1;
        }
    }
}
fflush(stdin);
}
If(max_val==0)
{
    sprintf(final_str,"There is no Alternate Path.The Multicast Path is");

```

```

}
else
{
    if(max_val<MAXCONGESTION)
    {
        printf(final_str,"There Is No Congestion In The Network");
    }
    else
    {
        printf(final_str,"The Updated Final Multicast Path Is");
        decision=1;
    }
}
if(decision==1)
{
    final_edge[maxx][maxy]=0;
    for(i=0;i<count;i++)
        final_edge[i][i+1]=1;
}

```

10.1.1.4 Code for identifying maximum congestion:

```

int max()
{
    int i,j,k,temp;
    static int maximum=0;
    for(i=0;i<no_of_groups;i++)
    {
        for(j=0;j<no_of_nodes;j++)
        {
            for(k=0;k<no_of_nodes;k++)

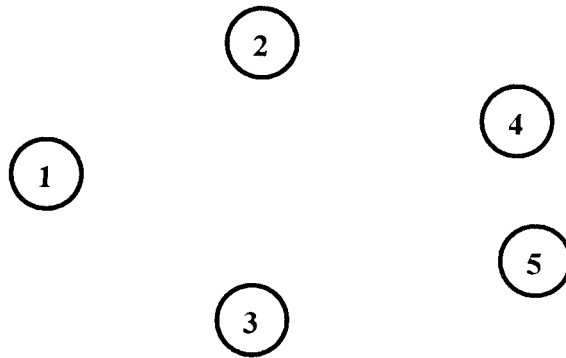
```

```

        {
            if(final_edge[j][k]==1)
            {
                if(maximum<congestion[i][j][k])
                {
                    maximum=congestion[i][j][k];
                    maxgroup=i;
                    maxx=j;
                    maxy=k;}
            }
        }
    }
}
test_edge[maxx][maxy]=0;
temp=maxx;
path[0]=temp;
count=0;
for(j=0;j<no_of_nodes;j++)
{
    k=j;
    while(test_edge[temp][k]==1)
    {
        ++count;
        path[count]=k;
        temp=k;
        k++;
    }
    if(k==maxy)
        return maximum;
}
maximum=0;
return maximum;
}

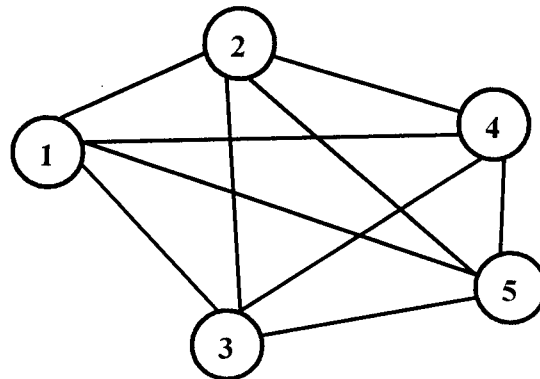
```

10.1.2 SAMPLE OUTPUT:



Input Nodes

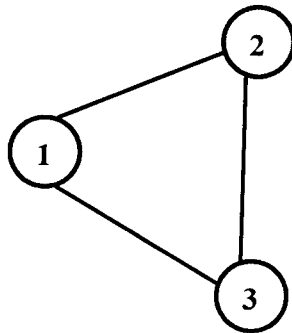
Space To Continue... Esc To Abort



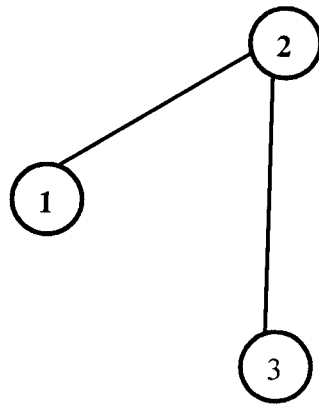
Press any key To Continue....

Multicast Group-I:

Tree with the given edges and vertices:

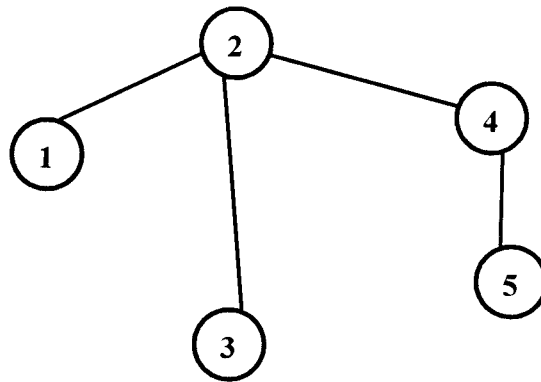


Multicast Tree:



The spanning tree is generated for each of the groups and finally embedded into the network. The final network may look like

Final Network With Embedded Multicast Tree:



10.2 MODULE-II:

10.2.1 SAMPLE SOURCE CODE

HEADER FILE FOR RECEIVER

```
#ifndef _REC_H
#define _REC_H
#define TIMEOUT_SECS 5 /* seconds between retransmits */
#define MAXTRIES 10
    void myHandler(int ignored);
    void prepareNACK(PACKET *, int);
#endif
```

HEADER FILE FOR SENDER

```
#ifndef _SENDER_H
#define _SENDER_H
#define PACKMAX 1030 /* The maximum size of the packet to be sent */
#define NORMAL 0
#define MAX_PACKETS 20
#include "common.h"
#endif
```

HEADER FILE FOR ROUTER

```
#ifndef _ROUTER_H
#define _ROUTER_H
#define MAX_NEIGHBOURS 10
#define MAX_LEN 20
#define NUM_SOCKETS 2
#define NUM_PORTS 2
```

```

#include "common.h"
int isNack(char *, char **, int);
#endif

```

COMMON.H FOR THE SENDER:

```

#ifndef _COMMON_H
#define _COMMON_H
#define TX_PORT 2000          /* The normal transmission channel */
#define RX_PORT 3000        /* The retransmission channel */
#define PACKSIZE 1008
#define SOURCE_ROUTER "90.0.0.94" /*The IP address of the source
router */
#define ROUTER_ONE "90.0.0.96"    /* The IP address of the other two routers */
#define ROUTER_TWO "90.0.0.97"
#define MY_IP "90.0.0.82" /* The IP address of this machine */
#define NACK 0
#define DATA 1
typedef struct packet
{
    int seqno;                /* Sequence number */
    int type;                 /* Packet type */
    char data[PACKSIZE];     /* The actual data payload */
} PACKET;
#endif

```

CODE FOR SENDER PROGRAM:

```

#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdlib.h>

```

```

#include<string.h>
#include<unistd.h>
#include "sender.h"          /* Constants for the sender */

main()
{
    int sock;
    char packbuffer[PACKMAX];
    char packetArray[MAX_PACKETS][PACKMAX];
    PACKET pkt;
    struct sockaddr_in destaddr;
    struct sockaddr_in fromaddr;
    FILE *fp;
    int i;
    int notover;
    int numPackets;
    printf("*****\n");
    printf("PROGRAM FOR THE SENDER \n");
    printf("*****\n");
    /* Read from file and put into data array */
    fp = fopen("input", "r");
    i=0;
    notover=1;
    while (notover == 1)
    {
        if (fread(packetArray[i],1,DATASIZE,fp) < DATASIZE)
        {
            printf("End of file reached\n");
            notover = 0;
        }
        else

```

```

        ++i;
    }
    numPackets = i;

    /* creation */
    if((sock=socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP))<0)
        printf("socket() failed ");

    /* server address structure */
    memset(&destaddr,0,sizeof(destaddr));
    destaddr.sin_family = AF_INET;
    destaddr.sin_addr.s_addr = inet_addr(SOURCE_ROUTER);
    destaddr.sin_port = htons(TX_PORT);

    /* send packet */
    /* TBD: while not end of data */
    /* Keep reading from array and sending it to the socket after some delay */
    for(i=0; i < numPackets; ++i)
    {
        pkt.type = DATA;
        pkt.seqno = i;
        memcpy(pkt.data,packetArray[i],DATASIZE);
        if(sendto(sock, (char *) &pkt,PACKSIZE,0,(struct sockaddr *)
&destaddr,sizeof(destaddr)) != PACKSIZE )
            printf("Error: In sending data\n");
        printf("Sent one packet\n");
        sleep(10);
    }
    close(sock);
}

```

CODE FOR ROUTER PROGRAM:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
/* Includes for select call */
#include <sys/select.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include "router.h"

/* Check if the IP address should be given the retransmission */
int isNACK(char * ipAddress, char ** nackArray, int max)
{
    int i;
    int retval=0;
    for(i=0; i < max && retval==0; ++i)
        if (strcmp(ipAddress,nackArray[i]) == 0)
            retval=1;
    return retval;
}
main()
{
    FILE * fp;
```

```

fd_set sockSet;
char ipAddress[MAX_NEIGHBOURS][MAX_LEN];
char buffer[PACKSIZE];
PACKET * pptr;
int neighbours;
int i=0;
int tx,rx;
struct sockaddr_in myAddress,rAddress;
int rLen,recvLength;
int maxDescriptor;
char nackArray[MAX_NEIGHBOURS][MAX_LEN];
int numNacks;
char * ip_address;
fp = fopen("ipaddress","r");
while (fscanf(fp, "%s", ipAddress[i]) != EOF) ++i;
neighbours = i;
printf("*****\n");
printf("PROGRAM FOR THE ROUTER \n");
printf("*****\n");

/*Testing if the reading is correct */
for(i=0; i < neighbours; ++i)
printf("The IP Address of node is %s \n", ipAddress[i]);
numNacks = 0;
if ( (tx = socket(PF_INET, SOCK_DGRAM,IPPROTO_UDP)) < 0)
printf("Error: Unable to create socket for normal channel\n");
if ( (rx = socket(PF_INET, SOCK_DGRAM,IPPROTO_UDP)) < 0)
printf("Error: Unable to create socket for normal channel\n");

/* Bind to port TX_PORT on socket tx */
memset(&myAddress,0,sizeof(myAddress));

```

```

myAddress.sin_family = AF_INET;
myAddress.sin_addr.s_addr = htonl(INADDR_ANY);
myAddress.sin_port = htons(TX_PORT);

if (bind(tx, (struct sockaddr *) & myAddress, sizeof(myAddress)) < 0)
    printf("Error: Unable to bind to port %d\n", TX_PORT);

/* Bind to port RX_PORT on socket rx */
memset(&myAddress,0,sizeof(myAddress));
myAddress.sin_family = AF_INET;
myAddress.sin_addr.s_addr = htonl(INADDR_ANY);
myAddress.sin_port = htons(RX_PORT);
if (bind(rx, (struct sockaddr *) & myAddress, sizeof(myAddress)) < 0)
    printf("Error: Unable to bind to port %d\n", TX_PORT);
if (tx > rx)
    maxDescriptor = tx;
else
    maxDescriptor = rx;
while (1)
{
    FD_ZERO(&sockSet);
    FD_SET(tx, &sockSet);
    FD_SET(rx, &sockSet);
    if (select(maxDescriptor+1, &sockSet, NULL, NULL,0) == 0)
        printf("Message:No activity on either ports\n");
    else
    {
        if (FD_ISSET(tx, &sockSet) )
        {
            /* TBD: Do something */;
            rLen = sizeof(rAddress);

```

```

if ((recvLength = recvfrom(tx,buffer,PACKSIZE,0,(struct sockaddr *) &
rAddress, &rLen)) < 0 )
printf("Error: In receiving data \n");

/* Send data to all normal nodes associated with this router */
for(i=0; i < neighbours; ++i)
{
memset(&rAddress,0,sizeof(rAddress));
rAddress.sin_family = AF_INET;
rAddress.sin_addr.s_addr = inet_addr(ipAddress[i]);
rAddress.sin_port = htons(TX_PORT);
if (sendto(tx,buffer,recvLength,0,(struct sockaddr *) &rAddress, rLen) !=
recvLength)
printf("Error: In sending data to address %s\n", ipAddress[i]);
}

if (strcmp(MY_IP, SOURCE_ROUTER) == 0)
{
memset(&rAddress,0,sizeof(rAddress));
rAddress.sin_family = AF_INET;
rAddress.sin_addr.s_addr = inet_addr(ROUTER_ONE);
rAddress.sin_port = htons(TX_PORT);
if (sendto(tx,buffer,recvLength,0,(struct sockaddr *) &rAddress, rLen) !=
recvLength)
printf("Error: In sending data to address %s\n", ROUTER_ONE);
memset(&rAddress,0,sizeof(rAddress));
rAddress.sin_family = AF_INET;
rAddress.sin_addr.s_addr = inet_addr(ROUTER_TWO);
rAddress.sin_port = htons(TX_PORT);
if (sendto(tx,buffer,recvLength,0,(struct sockaddr *) &rAddress, rLen) !=
recvLength)

```



```

        printf("Error: In sending data to address %s\n", ROUTER_TWO);
    }
}

```

```

if (FD_ISSET(rx, &sockSet))
/* TBD: Do something */
{
    rLen = sizeof(rAddress);
    if ((recvLength = recvfrom(rx,buffer,PACKSIZE,0,(struct sockaddr *) &
rAddress, &rLen)) < 0 )
        printf("Error: In receiving data \n");

    /* Figure out the IP address of the source of this packet */
    strcpy(ip_address,inet_ntoa(rAddress.sin_addr));
    pptr = ( PACKET *)buffer;
    if (pptr->type == NACK)
    {
        /* Add an entry to the nackArray */
        ++ numNacks;
        strcpy(nackArray[numNacks],ip_address);
        if (strcmp(MY_IP,SOURCE_ROUTER) != 0)
        {
            /* Forward it to the source router */
            memset(&rAddress,0,sizeof(rAddress));
            rAddress.sin_family = AF_INET;
            rAddress.sin_addr.s_addr = inet_addr(SOURCE_ROUTER);
            rAddress.sin_port = htons(RX_PORT);
            if (sendto(rx,buffer,recvLength,0,(struct sockaddr *) &rAddress, rLen) !=
recvLength)
                printf("Error: In sending NACK to address %s\n", SOURCE_ROUTER);
        }
    }
}

```

```

    }
else
    {
        /* Forward the NACK to the source machine */
        memset(&rAddress,0,sizeof(rAddress));
        rAddress.sin_family = AF_INET;
        rAddress.sin_addr.s_addr = inet_addr(SENDER);
        rAddress.sin_port = htons(RX_PORT);
        if (sendto(rx,buffer,recvLength,0,(struct sockaddr *) &rAddress, rLen)
            != recvLength)
            printf("Error: In sending NACK to address %s\n", SENDER);
    }
}
else
    {
        /* Send it to all those who missed it */
        /* And remove it from the NACK array */
        for(i=0; i< numNacks; ++i)
        {
            memset(&rAddress,0, sizeof(rAddress));

            /* Check if the IP address should be given the retransmission */
            rAddress.sin_family = AF_INET;
            rAddress.sin_addr.s_addr = inet_addr(nackArray[i]);
            rAddress.sin_port = htons(RX_PORT);
            if (sendto(rx,buffer,recvLength,0,(struct sockaddr *) &rAddress,
                rLen) != recvLength)
                printf("Error: In sending retransmission to address %s\n",
                    nackArray[i]);
        }
        numNacks=0;
    }
}

```

```

    }
}
}
}}

```

CODE FOR RECEIVER PROGRAM:

```

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include "common.h"
int tries=0;                /* Global variable shared between procedures */
#include "rec.h"
void myHandler(int ignored)
{
    tries += 1;
}
void prepareNACK(PACKET * p, int seqno)
{
    p->type = NACK;
    p->seqno = seqno;
}
int main()
{
    int sock;
    int sno;
    int recvMsgSize,fromaddrlen;
    int packetlen;

```

```

unsigned int tosize;
unsigned int fromsize;
struct sigaction timerAction;
struct sockaddr_in fromaddr;
struct sockaddr_in toaddr;
PACKET * pptr;          /* NACKs to be sent */
PACKET * rptr;          /* Data received */
char packbuffer[PACKSIZE];
/* Setup signal handler */
timerAction.sa_handler = myHandler;
if (sigfillset(&timerAction.sa_mask) < 0)
    printf("Unable to set signals\n");
timerAction.sa_flags = 0;
if (sigaction(SIGALRM, &timerAction, 0) < 0)
    printf("Error: Unable to initialize handler\n");
pptr = (PACKET *) malloc(sizeof(PACKET));
memset(pptr,0,sizeof(PACKET));
if((sock=socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP))<0) /* socket creation */
    printf("socket() failed\n");

/* server address structure */
memset(&toaddr,0,sizeof(toaddr));
toaddr.sin_family = AF_INET;
toaddr.sin_addr.s_addr = INADDR_ANY;
toaddr.sin_port = htons(TX_PORT);
printf("*****\n");
printf("PROGRAM FOR THE RECEIVER \n");
printf("*****\n");
if ( bind(sock, (struct sockaddr *) & toaddr, sizeof(toaddr)) < 0)
    printf("Error: Unable to bind. Port maybe in use\n");
while(1)

```

```

{
    fromsize = sizeof(fromaddr);
    alarm(TIMEOUT_SECS);

    while ( (recvMsgSize = recvfrom(sock,packbuffer,PACKSIZE,0,(struct sockaddr *)
    &fromaddr, &fromsize) ) < 0)
        if (errno == EINTR) /* THE ALARM WENT OFF */
            {
                printf("Timeout\n");
                if (tries < MAXTRIES)
                    {
                        printf("Timed out,%d more tries..\n",MAXTRIES-tries);
                        prepareNACK(pptr,sno+1);
                        if (sendto(sock,(char*)pptr,PACKSIZE,0,(struct sockaddr *)
                        &toaddr,sizeof(toaddr)) != PACKSIZE)
                            printf("send() does not function\n");
                        alarm(TIMEOUT_SECS);
                    }
                else
                    printf("no response\n");
            }
            else
                printf("recvfrom() failed\n");
        }

    /* if recvfrom() works cancel the time */
    alarm(0);
    rptr = (PACKET *)packbuffer;

    printf("packet received is \n");

```

```

    tries=0;

/* Check if we got a packet which was wrong */
    if (rptr->seqno == sno+1)
        {
            sno = sno+1;
            printf("In sequence packet received\n");
        }
    else
        {
            prepareNACK(pptr,sno+1);
            memset(&toaddr,0,sizeof(toaddr));
            toaddr.sin_family = AF_INET;
            toaddr.sin_addr.s_addr = inet_addr(SOURCE_ROUTER);
            toaddr.sin_port = htons(RX_PORT);
            tosize = sizeof(toaddr);
            if (sendto(sock, (char *) pptr, PACKSIZE, 0, (struct sockaddr *) &toaddr,tosize)
                != tosize)
                printf("Error: In sendto\n");
        }
    }
close(sock);
}

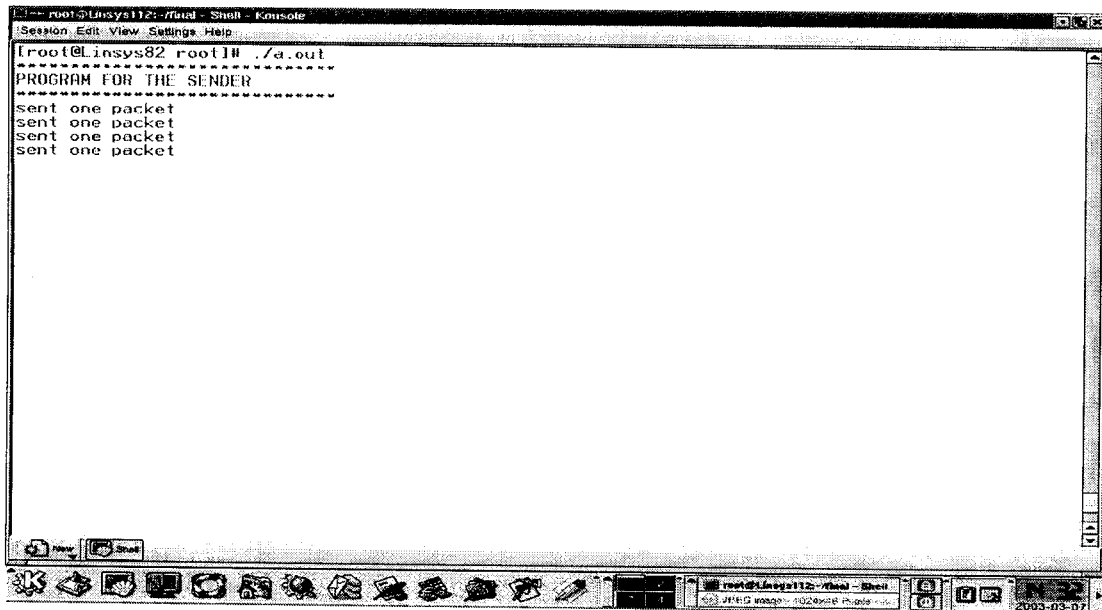
```

10.2.2 SAMPLE OUTPUT:

OUTPUT FOR THE SENDER PROGRAM:

10.2.2 SAMPLE OUTPUT:

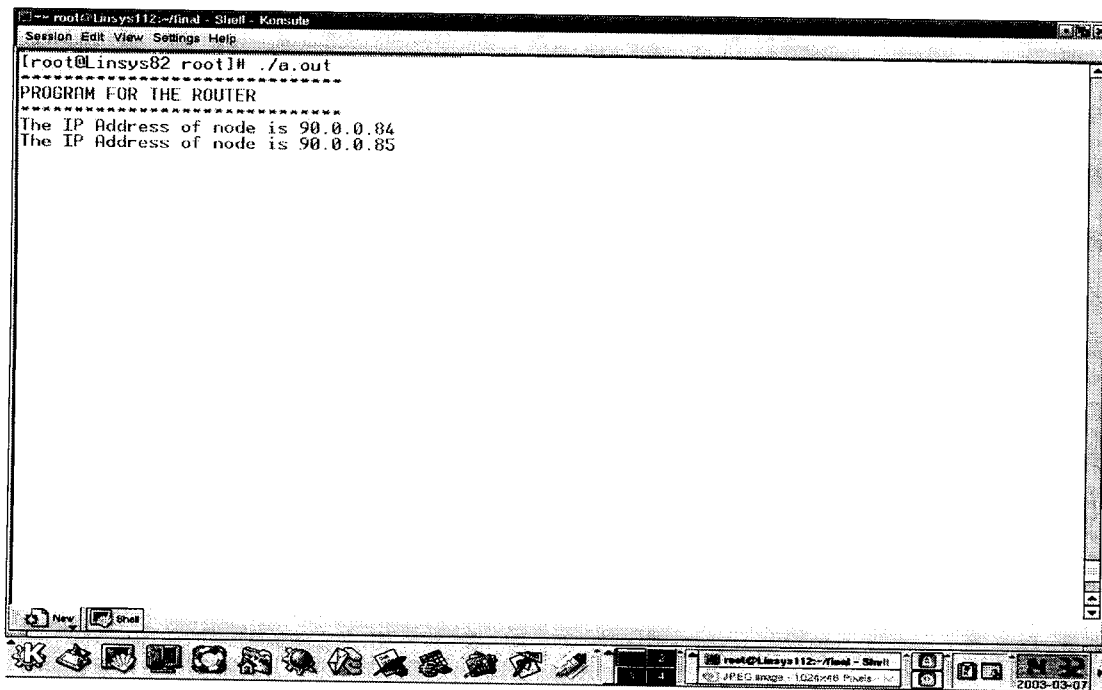
OUTPUT FOR THE SENDER PROGRAM:



A terminal window titled "root@Linsys112:~/fird - Shell - Konsole" showing the execution of a program. The prompt is "[root@Linsys82 root]# ./a.out". The output consists of a header "PROGRAM FOR THE SENDER" enclosed in dashed lines, followed by four lines of "sent one packet". The terminal has a standard Linux desktop environment with a taskbar at the bottom.

```
root@Linsys112:~/fird - Shell - Konsole
Session Edit View Settings Help
[root@Linsys82 root]# ./a.out
-----
PROGRAM FOR THE SENDER
-----
sent one packet
sent one packet
sent one packet
sent one packet
```

OUTPUT FOR THE ROUTER PROGRAM:



A terminal window titled "root@Linsys112:~/fird - Shell - Konsole" showing the execution of a program. The prompt is "[root@Linsys82 root]# ./a.out". The output consists of a header "PROGRAM FOR THE ROUTER" enclosed in dashed lines, followed by two lines of IP addresses: "The IP Address of node is 90.0.0.84" and "The IP Address of node is 90.0.0.85". The terminal has a standard Linux desktop environment with a taskbar at the bottom.

```
root@Linsys112:~/fird - Shell - Konsole
Session Edit View Settings Help
[root@Linsys82 root]# ./a.out
-----
PROGRAM FOR THE ROUTER
-----
The IP Address of node is 90.0.0.84
The IP Address of node is 90.0.0.85
```