P-1163

ABE ROUTING ALGORITHM

Project Report

Submitted in partial fulfillment of the
Requirement for the award of the degree of the

Bachelor of Engineering in Information Technology of Bharathiar University, Coimbatore.

Submitted by

N.Matheswaran
0027S0088

R.Thirunavukkarasu
0027S0116

Under the guidance of

Mrs.J.Cynthia M.E.,
Lecturer

DEPARTMENT OF INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE – 641006.

MARCH 2004.

DEPARTMENT OF INFORMATION TECHNOLOGY
KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University, Coimbatore)

CERTIFICATE

This is to certify that the project entitled

ABE ROUTING ALGORITHM

Is done by

N.Matheswaran
0027S0088

R.Thirunavukkarasu
0027S0116

And submitted in partial fulfillment of the
Requirement for the award of the degree of the

Bachelor of Engineering in Information Technology of
Bharathiar University, Coimbatore.

[illegible]

Professor & Head of the department
(Dr.S.THANGASAMY)

[illegible] 24/3/4

Project Guide
(Mrs.J. CYNTHIA)

Certified that the candidates were examined by us in the project work
Viva voce examination held on 27-03-04.

[illegible] 27/3

Internal Examiner

[illegible] 27/3/04

External Examiner

DECLARATION

DECLARATION

We,

N.Matheswaran	**0027S0088**
R.Thirunavukkarasu	**0027S0116**

declare that the project entitled "ABE Routing Algorithm ", is done by us and to the best of our knowledge, a similar work has not been submitted earlier to the Bharathiar University or any other institution, for fulfillment of the requirement of the course study.

This project report is submitted on the partial fulfillment of the requirement for all awards of the degree of Bachelor of Engineering in Information Technology of Bharathiar University.

[illegible]

Place: Coimbatore. **[N.Matheswaran]**

[illegible]

Date : 24-03-04 **[R.Thirunavukkarasu]**

Project Guided by

[illegible] 24/3/4

Mrs. J. Cynthia M.E.,

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

The exhilaration achieved upon the successful completion of any task should be definitely shared with the people behind the venture. This project is an amalgam of study and experience of many people without whose help this project would not have taken shape.

At the onset, we take this opportunity to thank the management of our college for having provided us excellent facilities to work with. We express our deep gratitude to our Principal Dr.K.K.Padmanabhan for ushering us in the path of triumph.

We are always thankful to our beloved Professor and the Head of the Department, Dr.S.Thangasamy whose consistent support and enthusiastic involvement helped us a great deal.

We are greatly indebted to our beloved guide Mrs.J.Cynthia M.E., Lecturer, Department of Information Technology for his excellent guidance and timely support during the course of this project. As a token of our esteem and gratitude, we would like to honour her for her assistance towards this cause.

We also thank our project coordinator Mrs.S.Devaki M.S., and our beloved class advisor Ms.P.Sudha B.E., for their invaluable assistance.

We also feel elated in manifesting our deep sense of gratitude to all the staff and lab technicians in the Department of Information Technology.

We feel proud to pay our respectful thanks to our Parents for their enthusiasm and encouragement and also we thank our friends who have associated themselves to bring out this project successfully.

SYNOPSIS

SYNOPSIS

ABE (Alternative Best-Effort) is a novel service for IP networks. It provides a low bounded queuing delay service in the Internet. The service is best-effort, and requires no additional charging or usage control. Its goal is to help applications with stringent real time constraints, such as interactive audio. With ABE, it is not required to police how much traffic uses the low delay capability, the service being designed to operate equally well in all traffic scenarios. Applications choose between receiving a lower end-to-end delay and receiving more overall throughput. Every best effort packet is tagged as either green or blue. Green packets receive a low, bounded queuing delay. To ensure blue packets do not suffer as a result, green flows receive fewer throughputs during bouts of congestion.

To test the router, separate network traffic simulator software will be necessary (particularly for Green). While simulating network traffic by sending periodic or random packets, there must be a variety of packet types to simulate real network traffic. Packet editing features will be necessary to cook-up a packet before transmission. In this project, we develop another custom made software which can generate various types of packets to simulate multiprotocol network traffic for testing the performance of our ABE implemented router.

CONTENT

CONTENTS

INTRODUCTION

1. INTRODUCTION

Our project deals with routing the packets from one network to another network.

1.1 Existing System and Limitations

Router connects networks, thus working on the network layer of the OSI-model (Layer 3). It is protocol depended and the network address on one interface is different from that on another interface. A router consists of a computer with at least two network interface cards supporting the IP protocol. The router receives packets from each interface via a network interface and forwards the received packets to an appropriate output network interface. Received packets have all link layer protocol headers removed, and transmitted packets have a new link protocol header added prior to transmission.

A router introduces delay (latency) as it processes the packets it receives. The total delay observed is the sum of many components including:

- Time taken to process the frame by the data link protocol
- Time taken to select the correct output link (i.e. filtering and routing)
- Queuing delay at the output link (when the link is busy)
- Other activities which consume processor resources (computing routing tables, network management, generation of logging information)

The router queue of packets waiting to be sent also introduces a potential cause of packet loss. Since the router has a finite amount of buffer memory to hold the queue, a router which receives packets at too high a rate may experience a full queue. In this case, the router has no other option than to simply discard excess packets. If required, these may later be retransmitted by a transport protocol.

Limitations

All the routing algorithms which are currently used consider the multimedia and ordinary text packets as the same. Since the multimedia and ordinary packets are given the same priority in routing, both packets have same queuing delay. So in Internet applications like video chatting, video conferencing, etc.. the delay between the frames is high (i.e) the frames cannot be continuously viewed.

1.2 Proposed System and Advantages

This proposed system assumes that the router has only output port queuing. It is based on a new scheduling concept, DSD. One of the first schemes to implement ABE that might spring to mind is a first come first served (FCFS) scheduling discipline with a threshold drop policy to filter green packets. In such a scheme, blue packets would be accepted when the buffer is not full, while green packets would only be accepted if they can be served with a delay no greater than some maximum d. Most of the time, though, there would be little or no incentive to be green. What is desired is to provide green with the best service possible while still ensuring that green does not hurt blue. Any significant extra gain by blue packets is at the expense of green ones. The gain blue packets would enjoy under ABE should be kept to a minimum such that there is still an incentive to use green packets whenever appropriate. This can be formalized by the following optimization problem: minimize the number of green losses subject to the following constraints:

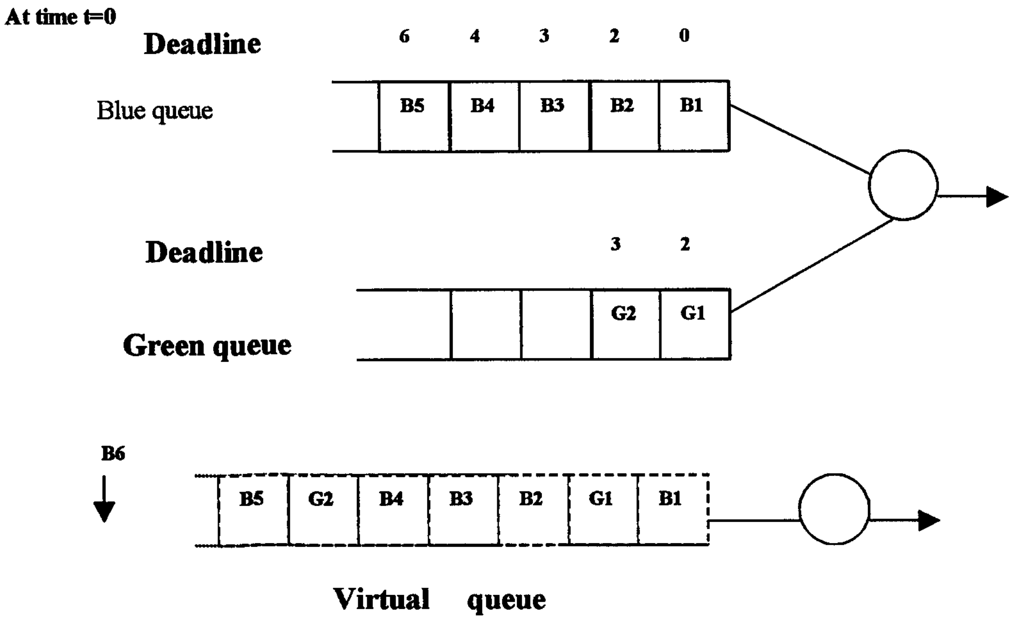
- Green packets receive a queuing delay no larger than d.
- The scheduling is work-conserving.
- No reordering: blue (respectively green) packets are served in the order of arrival.

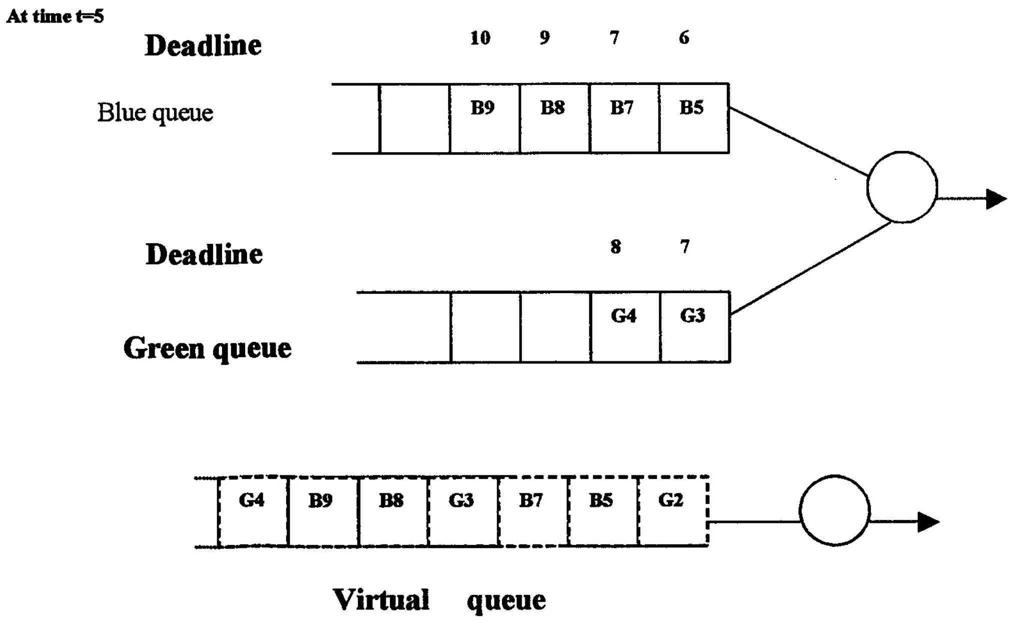
A solution to this problem is the DSD, a new scheduling algorithm based on the concept of ***duplicates***. Dead-lines are assigned to packets as they arrive, green and blue packets are queued separately, and the deadlines of the packets at the head of blue and green queues are used to determine which is to be served next. As previously discussed, throughput transparency as well as local transparency is required for ABE to ensure rate-adaptive green flows do not hurt blue. This is facilitated by the use of a parameter g, which is used in deciding which queue should be served in the event that the deadlines of the packets at the head of each queue can both be met if the other queue was served beforehand. The value of g used at any given time is determined by a control loop as described later. We can now describe DSD in detail.

1.2.1 Duplicate Scheduling with Deadlines (DSD)

Duplicates of all incoming packets are sent to a virtual queue with buffer size *Buff*. A duplicate is admitted if the virtual buffer is not full. Packets in the virtual queue are served according to FCFS at rate c, as they would be in flat best effort. The times at which duplicates will be served are used to assign blue packets deadlines at which they would have (approximately) been served in flat best effort. The original arriving packets are fed according to their color into a green and a blue queue. Blue packets are always served at the latest their deadline permits subject to work conservation. Green packets are served in the meantime if they have been in the queue for less than d s, and dropped otherwise. A blue packet is dropped if its duplicate was not accepted in the virtual queue. Otherwise, it is tagged with a deadline, given by the time at which its duplicate will be served in the virtual queue, and placed at the back of the blue queue.

A green packet is accepted if it passes what is called the *green acceptance test* and dropped otherwise. A green packet arriving at time t fails the test if the sum of the length of the green queue at time t (including this packet), and of the length of the first part of the blue queue that contains packets tagged with a deadline less than or equal to $t + d + pg$ new, where pg new is the transmission delay for the incoming green packet, is more than c $(d + pg$ new), and passes otherwise. The use of the test ensures the total buffer occupancy, namely the sum of the green and blue queue lengths, does not exceed *Buff*, which is discussed later. To facilitate understanding, we consider first the case where green packets do not undergo the green acceptance test and where $g = 1$. The maximal buffer size is *Buff* $= 7$ packets. The maximum green queue wait is $d = 3$ packets. B and G denote blue and green packets, respectively. In the first snap-shot, B 1 is served at time $t = 0$ in order to meet its deadline, then G 1, B 2, B 3, and B 4. G 2 has to be dropped from the green queue because it has to wait for more than *d* 3, whereas *B* 6 had to be dropped because the virtual queue length was *Buff* when it arrived.





Two snapshots as an example of DSD, at time t = 0 (top)

and t = 5 (bottom).

At time $t = 5$, we reach the situation of the second snapshot. Since no blue packet has reached its deadline yet, *G* 3 can be served, followed by *B* 5, *B* 7, *G* 4, *B* 8, and *B* 9. Consider again the example in Fig. 2, except green packets are now enqueued only if they pass the green acceptance test. This amounts here to accepting a green packet at time *t* if the number of green packets in the queue at time t, augmented by the number of blue packets in the queue with a deadline between [t, $t + 4$], is no more than 4. The only difference from Fig. 2 is that *G* 2 is no longer enqueued. Indeed, when it arrived, the green queue already contained packet *G* 1, and the blue queue contained packets *B* 1, *B* 2, and *B* 3. The total queue length at time *t* was 5 packets (including *G* 2), so *G* 2 fails the test. An accepted green packet is then assigned a deadline which is the sum of its arrival time plus its maximum waiting time d, and placed at the back of the green queue. At each service time, a decision is made as to which queue to serve.

The serving mechanism's primary function is to ensure that blue packets are always served no later than their deadlines. The best performance green could receive would be to then serve the green queue as much as possible, subject to this restriction. However, as previously discussed, in addition to local transparency, throughput transparency is needed to ensure that green adaptive applications do not benefit too much from lower delay. It can happen at service time that both blue and green packets at the head of their respective queues are able to wait, since letting the other packet go first would still allow it to be served within its deadline. For the purpose of supporting throughput transparency, when this situation arises the packet serving algorithm uses the current value of the ***green bias*** g, a value in the range [0,1], to determine the extent to which green is favored over blue. More precisely, when both blue and green packets can wait, *g* is the probability that the green packet is served first.

The value $g = 1$ corresponds to the case where green is always favored. Conversely, the value $g = 0$ corresponds to the systematic favoring of blue packets. In

Fig. 2, the packets served would have thus been, successively, $B1$, $B2$, $G1$, $B3$, $B4$, $B5$, $B7$, $G3$, $G4$, $B8$, and $B9$. A value of g less than 1 causes the delay for green traffic to be increased. This increase in delay for green TCP-friendly traffic reduces their throughput, thereby enabling blue traffic to increase its throughput. Increasing the delay of non-TCP-friendly traffic may not reduce their throughput, but blue flows are, in the worst case, equally as protected from this type of traffic as in a flat best effort service. The value of g choice is made according to a control loop, described later. All green packets who miss their deadline by waiting for more than d seconds (these packets are said to have become *stale*) are removed from the green queue. At service time, the possible events that arise and packets served by DSD are shown in Table 1. Pseudo-code for DSD is given below. Let *now* be the cur-rent time, p.

SOFTWARE REQUIREMENT ANALYSIS

2. SOFTWARE REQUIREMENT ANALYSIS

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. System study is conducted with the following objectives.

- Identify the needs.
- Evaluate the system concept for feasibility.
- Perform economic and technical analysis.
- Allocate function to hardware, software, people and other system elements.
- Create a system definition that forms the foundation for all subsequent engineering works.

2.1 Product Definition

In the Internet, while video conferencing and video chatting the frame rate is so low that the images move frame by frame. Since it moves frame by frame the picture is not so clear. In this routing algorithm we identify the multimedia packets and we give more precedence for it but we must see to that ordinary text packets don't suffer. We use a new algorithm called Duplicate scheduling with deadline. This routing algorithm is most simple and flexible one. The ABE-unaware sources receive the same service as they would if the network were flat best effort.

2.2 Project Plan

This project entitled "ABE Routing Algorithm" is to develop a software package to implement the routing algorithm which would be used in the internet for faster transmission of multimedia packets. We develop an another custom made software 'Packet Generator' to generate different kinds of packets which is helpful in testing the algorithm.

Three important steps in implementation of ABE Algorithm is

- Packet Generation
- Packet Analysis
- Implementation of ABE algorithm

SOFTWARE REQUIREMENT SPECIFICATION

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 Purpose

The primary purpose of the Software Requirement Specification (SRS) is to document the previously agreed functionality, attributes and the performance of the "ABE Routing Algorithm". This specification is the primary document upon which all of the subsequent design, source code and test plan will be based.

3.2 Scope

The scope of the document is to describe the requirements definition effort. The SRS is limited to description of the routing algorithms for the stringent real time applications like video conferencing etc.

3.3 Product Overview and Summary

Our product provides a reliable, robust and efficient means of routing packets from one network to another network. Our product basically deals with reduction of queuing delay for the multimedia packets. The product is developed using the concept of Duplicates Scheduling with Deadlines where the multimedia packets are given higher precedence. The Packet Generator first generates different kinds of packets to create traffic in the network. Then the packets are analyzed to check whether it is multimedia packets or ordinary text packets. Then ABE Algorithm is used to route the packets with the high precedence to the multimedia packets but the ordinary text packets are not affect much.

3.4 Development and Operating Environment

The development environment gives the minimum hardware and software requirements.

Hardware Specifications

<u>Server</u>

System Processor	:	Pentium II
Main Memory	:	64 Mb
Hard Disk	:	10 Gb
Monitor	:	15" Color Monitor
Floppy Drive	:	1.44 FDD
Ethernet card	:	Intex PCI (rtl8029)

<u>Clients</u>

System Processor	:	Pentium II
Main Memory	:	32/16 Mb
Hard Disk	:	1.2/4.2 Gb
Monitor	:	15" Color Monitor
Floppy Drive	:	1.44 FDD
Ethernet card	:	RL2000 PCI (rtl8029)

Software Specifications

Software (server)

Operating System	:	RedHat Linux 7.2
Programming Language	:	C++
Main Libraries	:	bpf USI++ libpcap (LIBPCAP 0.6.2), ncurses
Protocols	:	All the Available Protocols in the network

Software(clients for testing)

Operating System	:	RedHat Linux 7.2
Utilities	:	telnet, ping, ftp, http
Protocols	:	All the Available Protocols in the network

3.5 Functional Specifications

The descriptions of the modules are as follows:

- **Packet Generator**

The traffic is monitored by the sniff_pack function of derived object pcap of Pcap class. The timeout function is used to generate packets with respect to the traffic connection; and also it is used to do update the screen outputs and process the key inputs. To send packets, the objects ip, tcp and udp, which were derived from the classes IP, TCP, UDP were used. The time delay was generated by randnum() function or sleep() function depending on the choice we made during the input.

The keyboard interaction is indirectly implemented by using basic termios function calls through the functions kbhit() and readkey(). In UNIX, there will not be kbhit() function to interrupt the program during execution. This keyboard handling functions were developed from a c source code from internet.

The ncurses library functions were used to generate a pleasant window based text screen output in UNIX terminal by using ETI screen mode.

- **Packet Analysis & Implementation of the algorithm**

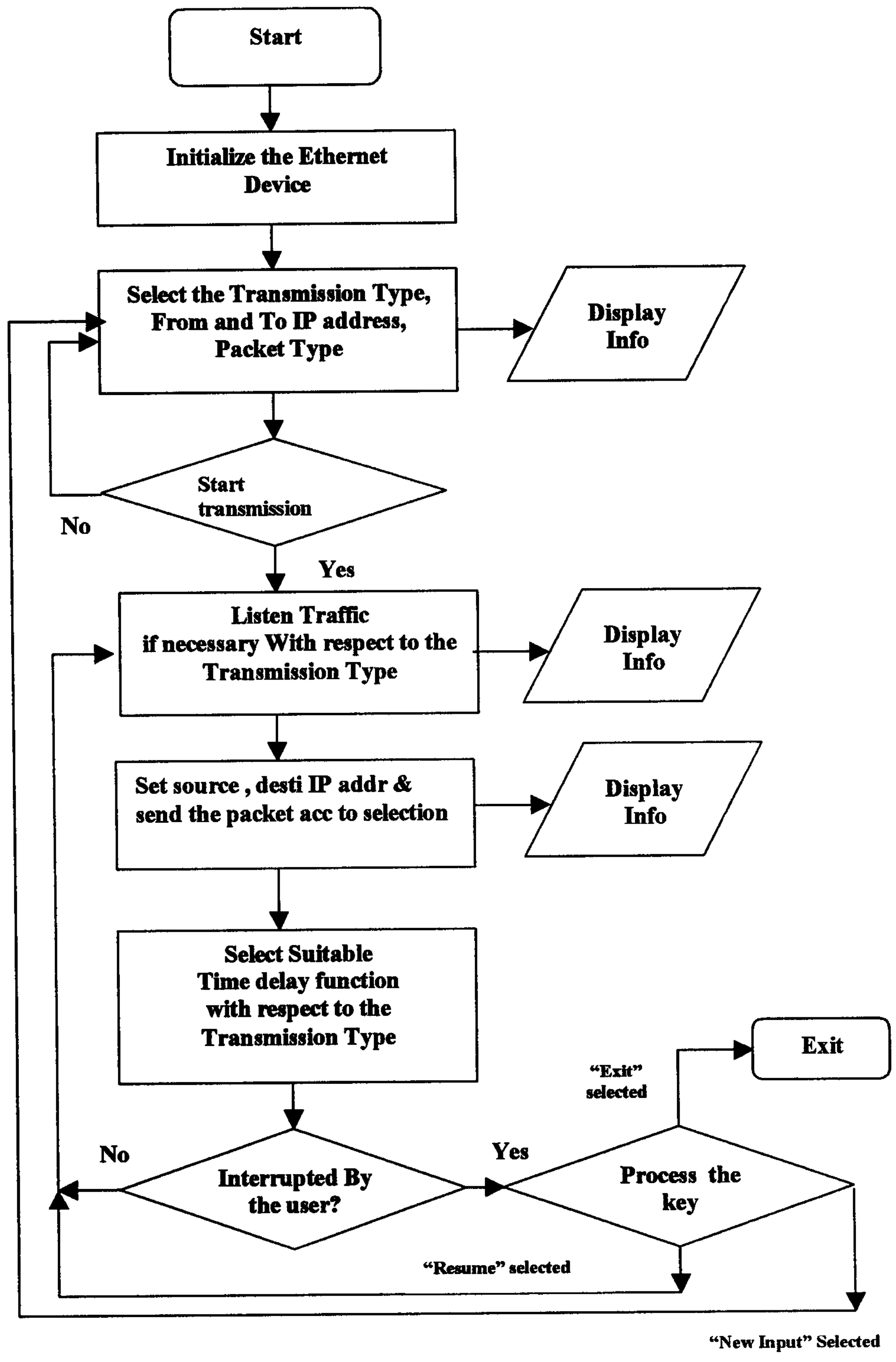
The Routing requests and ICMP messages were monitored by the sniff_pack function of derived object pcap of Pcap class. The received packets were enqueued and serviced according to the ABE algorithm. The time_out function is used to service the packets which already enqueued in the queues; and also it is used to do update the screen outputs and process the key inputs.

The keyboard interaction is indirectly implemented by using basic termios function calls through the functions kbhit() and readkey(). In UNIX, there will not be kbhit() function to interrupt the program during execution. This keyboard handling functions were developed from a c source code from internet.

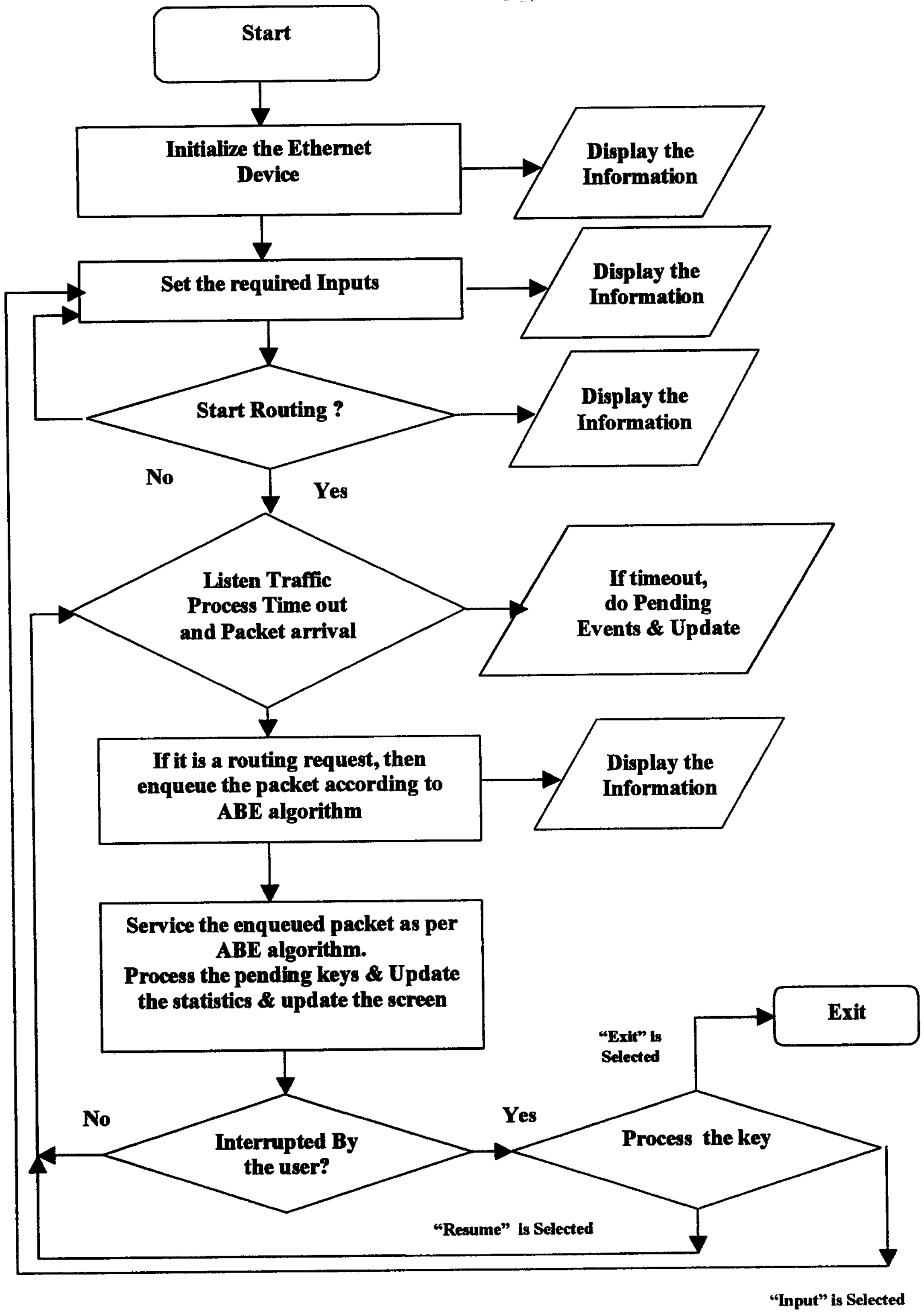
The ncurses library functions were used to generate a pleasant window based text screen output in UNIX terminal by using ETI screen mode.

3.6. Functional Flowchart

Flow chart for traffic simulator


Start
Initialize the Ethernet Device
Select the Transmission Type, From and To IP address, Packet Type
Display Info
Start transmission
No
Yes
Listen Traffic if necessary With respect to the Transmission Type
Display Info
Set source , desti IP addr & send the packet acc to selection
Display Info
Select Suitable Time delay function with respect to the Transmission Type
Exit
"Exit" selected
No
Yes
Interrupted By the user?
Process the key
"Resume" selected
"New Input" Selected

Flowchart for ABE Router


Start
Initialize the Ethernet Device
Display the Information
Set the required Inputs
Display the Information
Start Routing ?
Display the Information
No
Yes
Listen Traffic Process Time out and Packet arrival
If timeout, do Pending Events & Update
If it is a routing request, then enqueue the packet according to ABE algorithm
Display the Information
Service the enqueued packet as per ABE algorithm. Process the pending keys & Update the statistics & update the screen
Exit
"Exit" is Selected
No
Yes
Interrupted By the user?
Process the key
"Resume" is Selected
"Input" is Selected

3.7 Exception Handling

In this router simulation model, we concentrated only on ABE implementation. So, even it will not satisfy all the requirements of an IPV4 Router as per the rfc 1812. In the ABE Traffic simulating program, we handled very basic information while sending a packet. But there are few more things in the header and option fields of the typical packet.

3.8 Product Optimization

The product is about to be implemented on Linux 7.2 (c++). If it can be implemented with the concept of multitasking and inter-process communication of Linux then the code would be said to optimized.

SYSTEM DESIGN

4. SYSTEM DESIGN

The system design is the high level strategy for solving the problem and building a solution. System design includes decisions about the organization of the system into subsystems, allocation of subsystems to hardware and software components, and major conceptual and policy decisions that form the framework for the detailed design.

In the ABE Router software, the Main Module code is kept under the name "ABEroute.cc". Similarly, the Listening Module, Packet Enqueing and serving Module and the Output Module of the project are kept in the corresponding files namely, "listen.cc", "enquesrv.cc" and "output.cc".

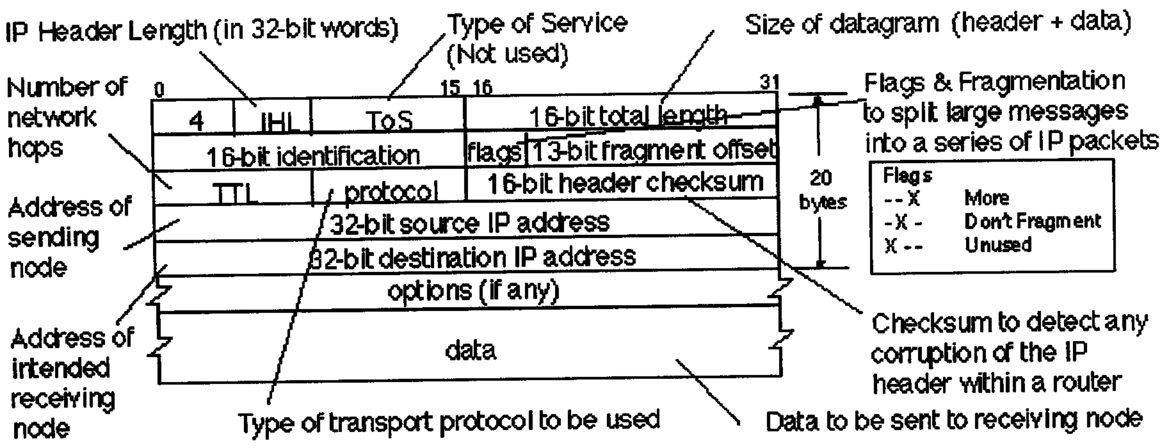
In the ABE Traffic Simulator software, the Main Module code is kept under the name "TrafficSim.cc". Similarly, the Input Module, Output Module, Packet Sending Module of the project is kept in the corresponding files namely, "input.cc", "output.cc" and "transmit.cc".

The class IP of USI++ library contains all the necessary functions to send and receive an IP packet. Similarly, the class TCP and UDP contains all the necessary functions to send and receive TCP and UDP packets. In this project, Pcap class is handled in an efficient manner to monitor the traffic and respond with respect to it.

Descriptions of the Packets & Packet headers

Structure IP packet header

The IP packet header consists of 20 bytes of data. An option exists within the header which allows further optional bytes to be added, but this is not normally used.
The full header is shown below:



IP Header Length (in 32-bit words)
Type of Service (Not used)
Size of datagram (header + data)
Number of network hops
0
15 16
31
4
IHL
ToS
16-bit total length
16-bit identification
flags
13-bit fragment offset
TTL
protocol
16-bit header checksum
32-bit source IP address
32-bit destination IP address
options (if any)
data
20 bytes
Flags & Fragmentation to split large messages into a series of IP packets
Flags
--X More
-X- Don't Fragment
X-- Unused
Address of sending node
Address of intended receiving node
Checksum to detect any corruption of the IP header within a router
Type of transport protocol to be used
Data to be sent to receiving node

Figure 1: The IP Header

The header fields are discussed below:

- **Version** (always set to the value 4, which is the current version of IP)
- **IP Header Length** (number of 32 -bit words forming the header, usually five)
- **Type of Service**, now known as **Differentiated Services Code Point (DSCP)** (usually set to 0, but may indicate particular Quality of Service needs from the network, the DSCP defines one of a set of class of service)
- **Size of Datagram** (in bytes, this is the combined length of the header and the data)
- **Identification** (16-bit number which together with the source address uniquely identifies this packet - used during reassembly of fragmented datagrams)
- **Flags** (a sequence of three flags (one of the 4 bits is unused) used to control whether routers are allowed to fragment a packet (i.e. the Don't Fragment, DF, flag), and to indicate the parts of a packet to the receiver)
- **Fragmentation Offset** (a byte count from the start of the original sent packet, set by any router which performs IP router fragmentation)
- **Time To Live** (Number of hops /links which the packet may be routed over, decremented by most routers - used to prevent accidental routing loops)
- **Protocol** (Service Access Point (SAP) which indicates the type of transport packet being carried (e.g. 1 = ICMP; 2= IGMP; 6 = TCP; 17= UDP).
- **Header Checksum** (A 2's complement checksum inserted by the sender and updated whenever the packet header is modified by a router - Used to detect

processing errors introduced into the packet inside a router or bridge where the packet is not protected by a link layer cyclic redundancy check. Packets with an invalid checksum are discarded by all nodes in an IP network)

- **Source Address** (the IP address of the original sender of the packet)
- **Destination Address** (the IP address of the final destination of the packet)
- **Options** (not normally used, but when used the IP header length will be > 5 32-bit words to indicate the size of the options field)

'C' Structure of IP packet header

```
struct ip
{
     u_int8_t  ip_vhl;          /* header length, version */
     u_int8_t  ip_tos;          /* type of service */
     u_int16_t ip_len;          /* total length */
     u_int16_t ip_id;           /* identification */
     u_int16_t ip_off;          /* fragment offset field */
     u_int8_t  ip_ttl;          /* time to live */
     u_int8_t  ip_p;            /* protocol */
     u_int16_t ip_sum;          /* checksum */
     struct    in_addr ip_src,ip_dst;   /*src &dest address*/
};
```

The User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is a transport layer protocol defined by the US Department of Defence (DoD) for use with the IP network layer protocol. It provides a best-effort datagram service to an End System.

The service provided by UDP is an unreliable service which provides no guarantees for delivery and no protection from duplication (if this arises due to software errors within an Intermediate System (IS)). The simplicity of UDP reduces the overhead from using the protocol and the services may be adequate in many cases.

A computer may send UDP packets without first establishing a connection to the recipient. The computer completes the appropriate fields in the UDP header (PCI) and forwards the data together with the header for transmission by the IP network layer.

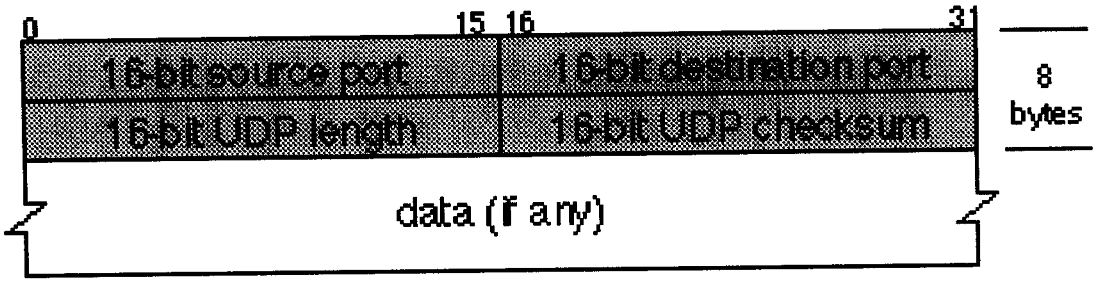


Figure 2: The UDP protocol header

The UDP header consists of four fields each of 2 bytes in length:

- **Source Port** (UDP packets from a client use this as a service access point (SAP) to indicate which session on the local client originated the packet. UDP packets from a server carry the server SAP in this field)
- **Destination Port** (UDP packets from a client use this as a service access point (SAP) to indicate which service is required from the remote server. UDP packets from a server carry the client SAP in this field)
- **UDP length** (The number of bytes of data)
- **UDP Checksum** (A checksum to verify that the end to end data has not been corrupted by routers or bridges in the network or by the processing in an end system. If this check is not required, the value of 0x0000 is placed in this field, in which case the data is not checked by the receiver.)
  The UDP header and data are not processed by Intermediate Systems (IS) in the network, and are delivered to the final destination in the same form as originally transmitted.

Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is a connection-oriented reliable protocol. It provides a reliable transport service between pairs of processes executing on End Systems (ES) using the network layer service provided by the IP protocol.

TCP providing reliable data transfer to FTP over an IP network using Ethernet
TCP is stream oriented, that is, TCP users exchange streams of data. The data are placed in buffers and transmitted by TCP in transport Protocol Data Units (sometimes known as "segments"). TCP is much more complex than UDP (which provides the Best Effort service). TCP implements a number of protocol timers to ensure reliable and synchronized communication between the two End Systems.

The TCP transport service is used by such applications as telnet, World Wide Web (WWW), ftp, electronic mail. The transport header contains a Service Access Point which indicates the protocol which is being used (e.g. 23 = Telnet; 25 = Mail; 69 = TFTP; 80 = WWW (http)).

'C' Structure of TCP packet header

```
struct tcphdr
{
     u_int16_t th_sport;              /* source port */
     u_int16_t th_dport;              /* destination port */
     tcp_seq   th_seq;                /* sequence number */
     tcp_seq   th_ack;                /* ack number */
     u_int8_t  th_offx2;              /* data offset, rsvd */
     u_int8_t  th_flags;              /* flags */
     u_int16_t th_win;                /* window */
     u_int16_t th_sum;                /* checksum */
     u_int16_t th_urp;                /* urgent pointer */
};
```

The functions handled from USI++ library

The listening of network traffic has been implemented by using the Pcap Class. The receiving objects were inherited from the parent class RX and the Transmitting objects were inherited from TX class.

The following functions are handled from USI++ library. The descriptions of this functions were taken from the HTML document pages of USI++. So for further details, it can be referred from the manual page itself. Here, only the functions from the classes which are related or used in this project are described.

usipp::IP Class

int usipp::IP::sendpack (void * *payload*, size_t *paylen*) `[virtual]`
Used to Send an IP Packet.

int usipp::IP::set_dst (const char *)
Used to set destination IP address. Not needed if the destination given in the constructor is OK.

int usipp::IP::set_hlen (u_int8_t)
Set header-len in number of 32 bit words. 5 (5*4 = 20) in normal case. Contructor does this for you, so you should not use this.

int usipp::IP::set_id (u_int16_t)
Used to set the ID-field of the IP packet.

int usipp::IP::set_proto (u_int8_t)
Used to set protocol number. If you use TCP {} or such, you don't need to do it yourself.

int usipp::IP::set_src (const char * *ip_or_name*)
Used to set source-adress IP address. Not needed if the destination given in the constructor is OK.

usipp::Pcap Class

usipp::Pcap::Pcap (char *)
This constructor should be used to initialize raw-datalink-objects, means not IP/TCP/ICMP etc. We need this b/c unlike in derived classes, datalink::init_device() cannot set a filter!

int usipp::Pcap::get_datalink ( )
Return the actual datalink of the object.

u_int16_t usipp::Pcap::get_etype ( )
Get protocol-type of ethernet-frame maybe moves to ethernet-class in future?

char * usipp::Pcap::get_hwdst (char *, size_t)
Fill buffer with dst-hardware-adress of actual packet, use 'data link' to determine what HW the device is. Now only ethernet s supported, but it's extensional.

char * usipp::Pcap::get_hwsrc (char *, size_t)
Fill buffer with src-hardware-adress of actual packet, use 'data link' to determine what HW the device is. Now only ethernet s supported, but it's extensional.

into usipp::Pcap::init_device (char * *dev*, int *promisc*, size_t *snaplen*) `[virtual]`
Initialize a device ("eth0" for example) for packet- capturing. It MUST be called before sniffpack() is launched. Set 'promisc' to 1 if you want the device running in promiscuous mode. Fetch at most 'snaplen' bytes per call.
Reimplemented from usipp::RX.

int usipp::Pcap::sniffpack (void *, size_t) `[virtual]`

sniff a packet

Reimplemented from usipp::RX.

usipp::TCP Class

int usipp::TCP::sendpack (char * ***pay_string*** **)** `[virtual]`

Used to send a string.

Reimplemented from usipp::IP.

int usipp::TCP::sendpack (void * ***payload*****, size_t** ***paylen*** **)** `[virtual]`

Used to send a packet.

Reimplemented from usipp::IP.

int usipp::TCP::set_dstport (u_int16_t)

Set destination-port

int usipp::TCP::set_srcport (u_int16_t)

Used to set source-port

int usipp::TCP::set_tcpopt (char ***kind*****, unsigned char** ***len*****, union tcp_options** ***t*** **)**

Used to set a TCP-option of particular kind

int usipp::TCP::set_tcpsum (u_int16_t)

Used to set TCP-checksum. Doing these will prevent sendpack() from doing this for you. It's not recommended that you do so, because the sum will almost be weak.

usipp::ICMP Class

u_int8_t usipp::ICMP::get_code ( )

Get ICMP-code.

u_int16_t usipp::ICMP::get_icmpId ( )

Get the id field from actual ICMP-packet.

u_int16_t usipp::ICMP::get_seq ( )

Get the sequence-number of actual ICMP-packet

u_int8_t usipp::ICMP::get_type ( )

Get the type-field from the actual ICMP-packet.

int usipp::ICMP::init_device (char * *dev*, int *promisc*, size_t *snaplen*) `[virtual]`

Initialize a device ("eth0" for example) for packet- capturing. It MUST be called before sniffpack() is launched. Set 'promisc' to 1 if you want the device running in promiscous mode. Fetch at most 'snaplen' bytes per call. Reimplemented from usipp::IP.

ICMP & usipp::ICMP::operator= (const ICMP &)

Assign-operator

int usipp::ICMP::sendpack (char * *pay_string*) `[virtual]`

send a ICMP-packet with string 'payload' as payload. Reimplemented from usipp::IP.

int usipp::ICMP::sendpack (void * *payload*, size_t *paylen*) `[virtual]`

send an ICMP-packet containing 'payload' which is 'paylen' bytes long Reimplemented from usipp::IP.

int usipp::ICMP::set_code (u_int8_t)

Set ICMP-code.

int usipp::ICMP::set_icmpId (u_int16_t)

Set id field in the actual ICMP-packet

int usipp::ICMP::set_seq (u_int16_t)

Set the sequence number of the actual ICMP-packet.

int usipp::ICMP::set_type (u_int8_t)

Set the type-field in the actual ICMP-packet.

int usipp::ICMP::sniffpack (void * *buf*, size_t *len*) `[virtual]`

handle packets, that are NOT actually for the local address! Reimplemented from usipp::IP.

The functions handled from ncurses library

Screen Initialization

To initialize the routines, the routine initscr or newterm must be called before any of the other routines that deal with windows and screens are used. The routine endwin must be called before exiting. To get character-at-a-time input without echoing (most interactive, screen oriented programs want this), the following sequence should be used: initscr(); cbreak(); noecho();

Windows and the newwin Function

The ncurses library permits manipulation of data structures, called windows, which can be thought of as two-dimensional arrays of characters representing all or part of a CRT screen. A default window called stdscr, which is the size of the terminal screen, is supplied. Others may be created with newwin. Note that curses does not handle overlapping windows, that's done by the panel(3X) library. This means that one can either use stdscr or divide the screen into tiled windows and not using stdscr at all. Mixing the two will result in unpredictable, and undesired, effects.

Windows are referred to by variables declared as WINDOW *. These data structures are manipulated with routines described here and elsewhere in the ncurses manual pages.

The move and addch functions

The most basic routines are move and addch. More general versions of these routines are included with names beginning with w, allowing the user to specify a window. The routines not beginning with w affect stdscr.) After using routines to manipulate a window, refresh is called, telling curses to make the user's CRT screen look like stdscr.

The characters in a window are actually of type chtype, (character and attribute data) so that other information about the character may also be stored with each character. Special windows called pads may also be manipulated. These are windows which are not constrained to the size of the screen and whose contents need not be completely displayed. See curs_pad(3X) for more information.

The Video Attributes, Colors & Lines

In addition to drawing characters on the screen, video attributes and colors may be supported, causing the characters to show up in such modes as underlined, in reverse video, or in color on terminals that support such display enhancements.

Line drawing characters may be specified to be output. On input, curses is also able to translate arrow and function keys that transmit escape sequences into single values. The video attributes, line drawing characters, and input values use names, defined in <curses.h>, such as A_REVERSE, ACS_HLINE, and KEY_LEFT. If the environment variables LINES and COLUMNS are set, or if the program is executing in a window environment, line and column information in the environment will override information read by terminfo. This would effect a program running in an AT&T 630 layer, for example, where the size of a screen is changeable (see ENVIRONMENT).

If the environment variable TERMINFO is defined, any program using curses checks for a local terminal definition before checking in the standard place.

The integer variables LINES and COLS are defined in <curses.h> and will be filled in by initscr with the size of the screen. The constants TRUE and FALSE have the values 1 and 0, respectively. The curses routines also define the WINDOW * variable curscr which is used for certain low-level operations like clearing and redrawing a screen containing garbage. The curscr can be used in only a few routines.

Routine and Argument Names

Many curses routines have two or more versions. The routines prefixed with w require a window argument. The routines prefixed with p require a pad argument. Those without a prefix generally use stdscr.

The routines prefixed with mv require a y and x coordinate to move to before performing the appropriate action. The mv routines imply a call to move before the call to the other routine. The coordinate y always refers to the row (of the window), and x always refers to the column. The upper left-hand corner is always (0,0), not (1,1).

The routines prefixed with mvw take both a window argument and x and y coordinates. The window argument is always specified before the coordinates.

In each case, win is the window affected, and pad is the pad affected; win and pad are always pointers to type WIN DOW.

SYSTEM TESTING

5. SYSTEM TESTING

The project is divided into two modules. These two modules are developed separately and verified whether they function properly. After the completion of each module sample images are used and tested. The two modules are

1. Traffic generator software
2. Traffic analyzer & ABE Implementation

5.1 Testing the Traffic Generator

We tested the traffic generator using the ABE router software itself. We get the inputs from the user such as source IP, destination IP, type of packet. In the traffic analyzer part we get the packets from the ethernet card and display the information's about each packet (i.e.) source IP, destination IP, type of packet., source MAC, destination MAC addresses. And we check whether the packet which we have generated is traveling in the other side.

5.2 Testing the ABE Router Software

First we tested the ABE router software whether it works well for the multimedia packets (i.e.) we generated various packets like multimedia, ordinary packets in the network. In our ABE router software, higher precedence is given to the multimedia packets and we also see to that the ordinary packets doesn't suffer too much because of this. We checked it by having the two queues separately for multimedia packets and ordinary packets, here the multimedia queue was fastly served. Thus our routing software is tested. Also our software is tested for various kinds of traffic conditions by generating different kinds of packets in the network using our packet generator software.

FUTURE ENHANCEMENTS

6. FUTURE ENHANCEMENTS

This project is based on USI++ library and is developed to run in a UNIX server console only. This code can be ported to run in Microsoft Windows machines by using suitable library. In the ABE Router program, the overall speed is reduced to a very low level for the visual simulation and understanding of the ABE concept. But, if it will be a working router implementation, then the design of the program should be modified for maximum performance in terms of speed.

In this router simulation model, we concentrated only on ABE implementation. So, even it will not satisfy all the requirements of an IPV4 Router as per the rfc 1812. But we tried to implement some of the requirements such as ICMP. So in future versions we can satisfy some other requirements as per the rfc 1812 such as IGMP, ARP etc.,

In the ABE Traffic simulating program, we handled very basic information while sending a packet. But there are few more things in the header and option fields of the typical packet. If we develop the input interface and sending mechanism to use such fields to send a much customized packet in a specific manner.

The multi-processing and inter-process communicating capabilities of UNIX can be explored and can be implemented to enhance the performance of listening mechanism and packet enqueueing and packet servicing modules.

CONCLUSION

7. CONCLUSION

The ABE Router Program performed as we expected and it proves the efficiency of the ABE service. Developing the project under Linux was a challenging task but the resources of information and support were very rich in that platform

First, the program was tested in the network while only one or two nodes were active. After that, it was tested while more than ten nodes were active. In both the cases the performance was as expected. When more machines were active, it was found that the messages displayed in message window scrolled very fast. So it was not possible to perceive the screen output of the message window. But message window was useful while the error messages were redirected during initializing ether device.

A typical router program usually run in background as daemon process. But this program was designed to run in a console terminal which supports the enhanced terminal interface capabilities

UNIX is very powerful with its multi tasking capabilities. So, the multi tasking features can be used in future versions of ABE Router to improve the overall performance of the program. This can be implemented by running the packet enqueing module end, processing module, packet serving module and screen handling module of ABE Router as three separate processes to improve the performance. The extracted information from the captured packets can be buffered for future reference in an efficient manner to reduce the processing time while deciphering information from newly captured packet. After the development, the code can be optimized for better performance by making frequently using functions as inline and replacing necessary simple data types with pointer type data.

As for as the Traffic Simulator program is concerned, the software performed well and simulated virtual traffic as we expected. The network traffic was simulated in such a way that the packets were originated from lot of different IP clients. In the USI ++ library, duplicating ether MAC address is not properly documented, so we could not use that feature in our program. If it is possible to alter the MAC addresses of the out going packets by some means, then conceptually Traffic Simulator would be a fulfilled one.

REFERENCES

8. REFERENCES

1. Zimmermann, "IEEE Spectrum 2000, Volume 26, Number 4".
2. Douglous E Comer, "Internetworking with TCP/IP", Volume-3, Prentice Hall, 1995
3. Meeta Gandhi, Tilak Shetty, Rajiv Shah, "The 'C' Odyssy UNIX - The Open Boundless C ",(First Edition), BPB Publications,1992.
4. Zeyd M. Ben-Halim, Eric S. Raymond, Thomas E. Dickey, Linux Man Pages of ncurses, based on pcurses by Pavel Curtis.
5. Sebastian Krahmer, Document pages of USI++

Websites Visited

www.itprc.com/tcpipfaq/default.htm

www.tcpdump.org

www.itprc.com

www.ieee.org

ANNEXURE

9. ANNEXURE

9.1 Sample Code

```
void push_back_packet(int color);
void serve_packet(int color);
void serve(int color);
int enqueue_packet();
int green_acceptance();
int find_deadline_len();
void updatequeue();
extern void get_ipinfo(char []);
extern void disp_packet_info();

void push_back_packet(int cl)
{

   packet *c = new packet;
   memcpy(&newpacket.packetbuf,buf,1000);
   memcpy(c,&newpacket,sizeof(packet));
   virtual_queue.push_back(c);
   switch(color)
   {
   case 0:  blue_queue.push_back(c);
                 break;
   case 1: green_queue.push_back(c);
                 break;
   }
 }

void serve(int cl)
{
   switch(color)
{
case 0:  if(blue_queue.size())
             {
             get_ipinfo(blue_queue.front()->packetbuf);
             blue_queue.pop_front();
             virtual_queue.pop_front();
             srvprintmsg("\nRouting a Blue Packet");


             }
             break;
 case 1: if(green_queue.size())
             {
             get_ipinfo(green_queue.front()->packetbuf);
             green_queue.pop_front();
             virtual_queue.pop_front();
             srvprintmsg("\nRouting a Green Packet");
             }
             break;

  }
find_route(dip);
disp_packet_info();
sprintf(message,"\nGW/Met/Dev:%s",routestr);
srvprintmsg(message);
updatequeue();
}


//Packet Enqueuing Algorithm


int enqueue_packet()
```

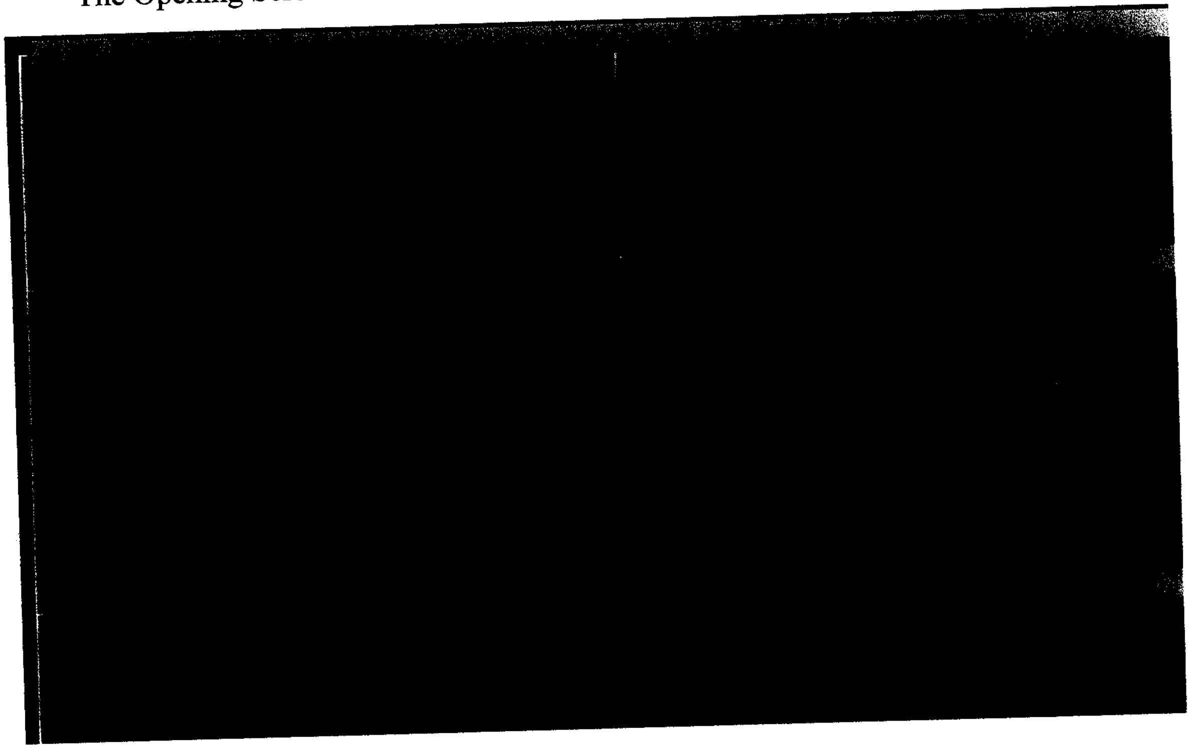
```
{
//packet p arrives at the output port
//dup = p
//Add dup to the virtual queue
struct timeval *tval;

gettimeofday(tval,(struct timezone *) NULL);
now=tval->tv_sec;

if (virtual_queue.size()==VQ_MAX_SIZE) //from virtual queue
{
   sprintf(message,"\nQueue Limit. Packet Dropped");
   printmsg(message);
   send_icmp(ICMP_SOURCE_QUENCH,0,dip,sip);
   return(0);
}

switch(color)
{
case 0:
        vd = virtual_queue.size() ; //queuing delay received by dup in virtual queue
        newpacket.deadline = now + vd;
        push_back_packet(0);
        sprintf(message,"\nBlue Packet Enqueued");
        printmsg(message);

        break;
case 1:
        if(!green_acceptance())//p fails "green acceptance test"
        {
                sprintf(message,"\nGreen Packet fails acceptance test");
                send_icmp(ICMP_SOURCE_QUENCH,0,dip,sip);
                printmsg(message);
                return(0);
        }
        else
           {
           newpacket.deadline=now + d;
           push_back_packet(1);
           sprintf(message,"\nGreen Packet Enqueued");
           printmsg(message);
           }
         break;
}
updatequeue();
return(1);
}

// green acceptance test
int green_acceptance()
{
int pgnew=1;              //new transmission delay for p
int lg=green_queue.size()+1;
int lb=find_deadline_len();//length of packets in blue queue with deadlines < now + d + pg
new
if(lg + lb > rate_c * (d + pgnew) )
   return(0);
else
   return (1);
}

int find_deadline_len()
```

```
{
int i;

for (i = 0; i < blue_queue.size(); i++)
 if ( blue_queue[i]->deadline < (now + d + pgnew) ) continue;
return(i);
}

//Packet Serving Algorithm
//     drop stale green packets,
//     those packets from green queue who cannot be served
//     within their deadline

 void serve_packet()
 {
 if(!green_queue.size()) // no green to serve
    {
    if(blue_queue.size()) //blue waits to be served
          serve(0);
    }
  else
    if(!blue_queue.size())  // no blue to serve
          serve(1);
    else // both queues contain packets
    {
 //        serve(1);
 //        return;
           pg   = 1;//now-green_queue.front()->deadline; //transmissionDelay;
           deadg= green_queue.front()->deadline;

           pb   = 1;//now - blue_queue.front()->deadline;  //transmissionDelay;
```

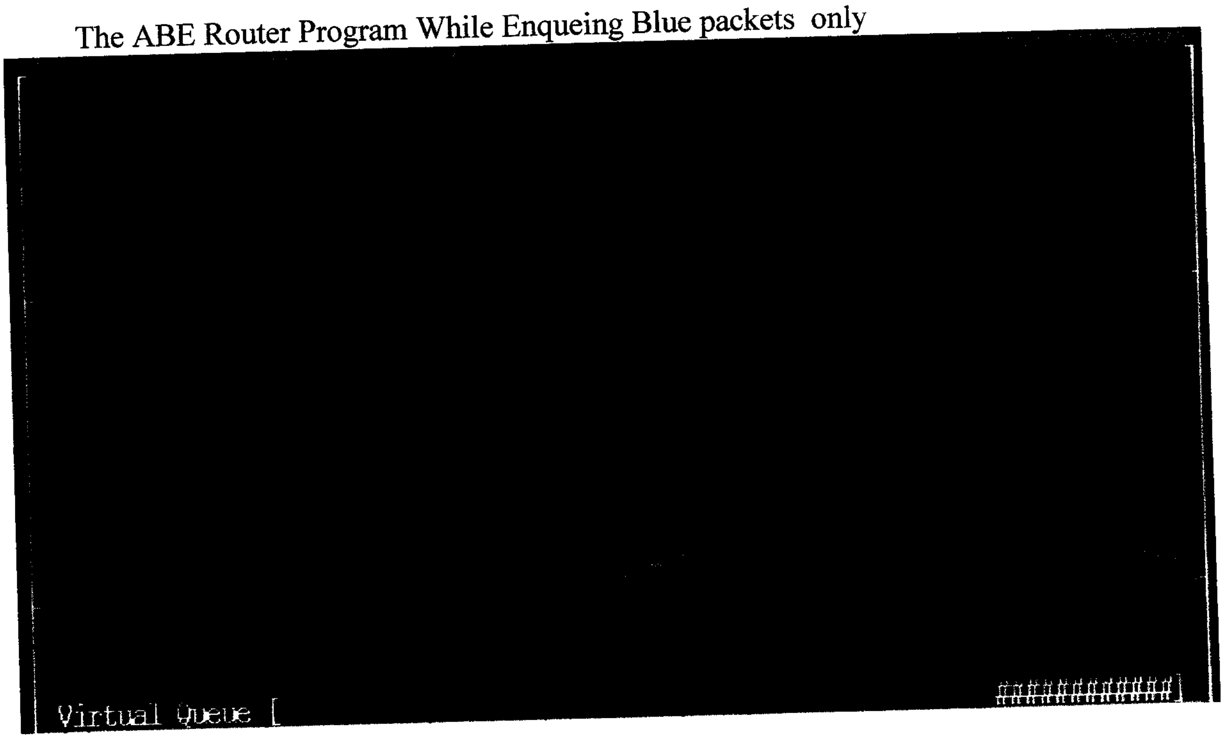
```
        deadb= blue_queue.front()->deadline;
        if (now > deadb - pg)
          serve(0); // because it cannot wait
        else
        {
          if (now > deadg - pb)
               serve(1); // because it cannot wait
          else
            if (probability_g)
                 serve(1);
                 else
                 serve(0);
        }
        }
}


void updatequeue()
{
if(blue_queue.size())
{
        sprintf(message,"Blue Queue   [%60.*s]",blue_queue.size(),queuechar);
        smessage(message,21,2,54);
}
else
{
        sprintf(message,"Blue Queue   [%60s]"," ");
        smessage(message,21,2,54);
}
if(green_queue.size())
{
        sprintf(message,"Green Queue  [%60.*s]",green_queue.size(),queuechar);
        smessage(message,22,2,52);
}
else
{
        sprintf(message,"Green Queue  [%60s]"," ");
        smessage(message,22,2,52);
}

if(virtual_queue.size())
{
        sprintf(message,"Virtual Queue [%60.*s]",virtual_queue.size(),queuechar);
        smessage(message,23,2,53);
}
else
{
        sprintf(message,"Virtual Queue [%60s]"," ");
        smessage(message,23,2,53);
}
        refresh();
}
```

9.2 Output Screen

The Opening Screen of the Router Program Interface while initializing Routing



The ABE Router Program While Enqueing Blue packets only



```
Virtual Queue [                                              ############]
```