P-1508

NETWORK TRAFFIC ANALYZER

A PROJECT REPORT

Submitted by

VANITHA.M. (71201104066)

VIDYAPRABHA.V. (71201104068)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006.

ANNA UNIVERSITY : CHENNAI 600 025

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"NETWORK TRAFFIC ANALYZER"** is the bonafide work of "**VANITHA.M (71201104066)** and **VIDYAPRABHA.V (71201104068)**" who carried out the project work under my supervision.

[illegible]
SIGNATURE

Dr . S . Thangasamy
HEAD OF THE DEPARTMENT

[illegible]
SIGNATURE

Ms.Amutha Venkatessh
SUPERVISOR
SENIOR LECTURER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641006.

Submitted for the Viva-voce Examination held on 21 · 4 · 2005

[illegible] 21/4
Internal Examiner

[illegible] 21/4
External Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We express our immense gratitude to **Prof.K.Arumugam** Correspondent , Kumaraguru College of Technology , Coimbatore for giving us an opportunity to study in their prestigious institution and to take up the project in partial fulfillment of the regulations for the B.E. program.

We are thankful to **Dr . K . K . Padmanabhan** , Principal , Kumaraguru College of Technology , Coimbatore , **Dr.S.Thangasamy**, HOD, Computer Science & Engineering Department for the facilities made available during the course.

We also convey our heartfelt thanks to **Ms . Devaki** , B.E., M.S., Project Coordinator, Kumaraguru College of Technology , Coimbatore , for providing us the support which really helped us make this project a success.

We are indebted to our project guide **Ms.Amutha Venkatessh** , M.E., Senior Lecturer, Kumaraguru College of Technology , Coimbatore , for her guidance and enthusiasm for doing the project.

We would also thank all staff members of the Computer Science & Engineering Department , for their constant encouragement and guidance throughout the work.

ABSTRACT

ABSTRACT

The Network Traffic Analyzer deals with monitoring the network and provides the network administrator, the details about the network traffic, packets transmission, participation of individual node in the network traffic,packet types transmitted,protocols used for transmission,etc.

It monitors the network data stream, which consists of all of the information transferred over a network at any given time. Prior to transmission, this information is divided by the networking software into smaller segments, called frames or packets. All the packets that are flowing in the network are captured. Then the analysis of each and every packet is done and the details are stored.

It allows administrators to look at the details of network packets, perform remote captures on a packet anywhere on the network, and gather network statistics about a group of personal computers. It enables network administrators to capture and analyze network traffic and detect problems or potential network bottlenecks.

CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

ACK	:	Acknowledgement
AWT	:	Abstract Window Toolkit
DNS	:	Domain Name Server
FIN	:	Final
FTP	:	File Transfer Protocol
HTTP	:	Hypertext Transfer Protocol
ICMP	:	Internet Control Message Protocol
IGMP	:	Internet Group Message Protocol
IP	:	Internet Protocol
LAN	:	Local Area Network
NTA	:	Network Traffic Analyzer
PSH	:	Push
RST	:	Reset
SQL	:	Structured Query Language
SYN	:	Synchronization
TCP	:	Transmission Control Protocol
UDP	:	User Datagram Protocol
WAN	:	Wide Area Network

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

1.1. PROJECT OVERVIEW

In order to understand Network Traffic Analyzer some of the basic concept of network must be understood which can be explained as follows. TCP/IP reference model has four layers.

Fig 1.1 TCP/IP Reference Model

Application Layer
Transport Layer
Internet Layer
Host-to-network Layer

Internet Layer:

The Internet layer controls the operation of the subnet. A key design issue is to determine how packets are routed from source to destination. Routes can be based on static tables that are "wired into" the network and rarely changed. They can also be determined at the start of each conversation for example, a terminal session.

Finally they can be dynamic, being determined a new for each packet, to reflect the current network load. The internet layer defines an official packet format and protocol called IP Internet Protocol). Finally the job of this layer is to deliver IP packets.

Transport Layer:

The basic function of the transport layer is to accept data from above, split it up into smaller units, pass these to the internet layer and ensure that the pieces all arrive correctly at other end. Further more all these must be efficiently done. The transport layer is true end-to-end layer, all the way from the source to the destination.

The protocols are between each machine and its immediate neighbors, and not between the ultimate source and destination machines, which may be separated by many routers. The first one, TCP is a reliable connection-oriented protocol that allows a byte stream originating on one machine to be delivered without error on any machine in the internet.

The second protocol in this layer, UDP is a connectionless protocol for applications that do not want TCP's sequencing or flow control and wish to provide their own.

Application Layer:

The application layer contains a variety of protocols that are commonly needed by users. One widely-used application protocol is HTTP, which is the basis for the World Wide Web. When a browser wants a web page, it sends the name of the page it wants to the server using HTTP. The server sends the page back. Other application protocols are used for file transfer, electronic mail, and network news. Some of the protocols used in this layer are SMTP, FTP, protocols for mapping host name (DNS) into their network addresses.

Host-to-Network Layer:

The TCP/IP reference does not really say about what happens here, except to point out that the host has to connect to the network using some protocol so it can send IP packets to it.

Introduction to TCP:

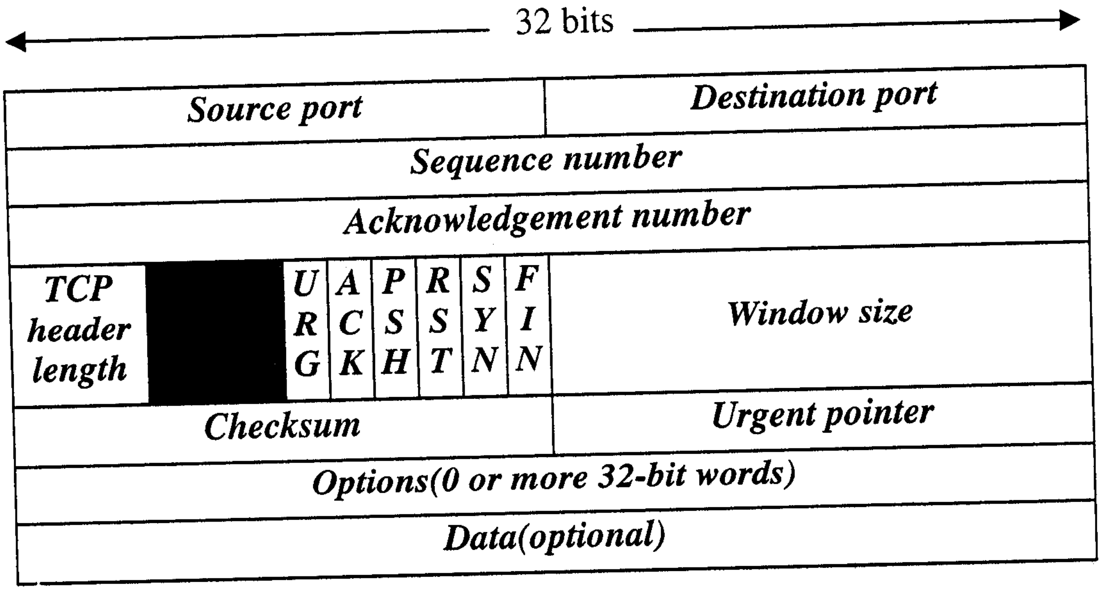


Fig 1.2 TCP Header Format

Every TCP segment begins with a fixed format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535-20-20=65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

The **Source port** and **Destination port** fields identify the local end points of the connection. The **Sequence number** and **Acknowledgement number** fields perform their usual functions. The **TCP Header length** tells how many 32-bit words are contained in the TCP header. This information is needed because the Options field is of variable length, so the header is, too. Technically, this field really indicates the start of the data within the segment, measured in 32-bit words.

Next comes a 6-bit field that is not used. Next field is **URG**. It is set o 1 if the Urgent pointer is in use. The **ACK** bit is set to 1 to indicate that the Acknowledge number is valid. The **PSH** bit indicates **Push**ed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received.

The **RST** bit is used to reset a connection that has become confused due to a host crash or some other reason. The **SYN** is used to establish connections. The **FIN** bit is used to release a connection.

The **Window size** field tells how many bytes may be sent starting at the byte acknowledged. A **Checksum** is also provided for extra reliability. The **Options** field provides a way to add extra facilities not covered by the regular header. The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept.Next comes the optional **Data** field.

Introduction to UDP:

32 bits

Source port	*Destination port*
UDP Length	*UDP Checksum*

Fig 1.3 UDP Header Format

The **Source port** is primarily needed when a reply must be sent back to the source, by copying the source port field from the incoming segment into the **Destination port** field of the outgoing segment. The **Source port** and **Destination port** fields identify the local end points of the connection.

The **UDP length field** includes the 8-byte header and the data. The **UDP checksum** is optional and stored as 0 if not computed.

NTA:

Core Modules:

NTA consists of the following modules

1. Packet Sniffing Module
2. Post processing Module
3. User Interface

Packet Sniffing Module:

Packet Sniffing is a system that counts, and copies all the frames it detects, to its capture buffer, which is a reserved storage area in memory. This process is referred to as capturing.

The first thing to understand is the general layout of a pcap (pcap is nothing but a packet capturing library) sniffer. The flow of code is as follows:

1. Selection of interface to sniff on.
2. Initialize pcap.
3. Setting filtering criteria if any.
4. Finally, the pcap enters its primary execution loop.

Post Processing Module:

Each frame contains the following information:

- The source address of the computer that sent the message, which is a unique hexadecimal number that identifies the computer on the network

- The destination address of the computer that received the frame
- Headers from each protocol used to send the frame
- The data or a portion of the information being sent

User Interface:

In this module using Java AWT controls a good interface is provided to the user. According to user options, they can view the contents of the file they select. Further the summary of the Network Traffic Analyzer is also provided.

1.1.1. EXISTING SYSTEM AND ITS LIMITATIONS:

Existing System:

Various monitoring tools are available, like Ethereal, TCP Dump. Most of these tools are stand alone monitors. They monitor only a single system or node in which they are implemented. These tools just monitors the incoming and outgoing packets of a system and the packet details are displayed. Some tools like TCP Dump generates and log files.

Drawbacks in the Existing System:

These tools lack many features and facilities that are essential for an effective network administration. These tools do not provide any effective way to control or analyze the network and the individual client nodes in the network.

The network administrator is not furnished with flexible features and controls. So the existing system does not play a vital role in the network management and administration.

The drawbacks observed in these existing systems are summarized as,

No client-server centralized monitoring:

Using this existing tool, all nodes in the network cannot be monitored with a centralized control.

No Storage:

Ethereal does not stores the details or the summary to a database or any separate log files, which will be very useful for the network administrator for information retrieval.

No effective & clear statistical reports are generated:

Reports about the network traffic and flow of packets are very essential to a network administrator for flawless network control and analysis. But these existing tools do not generate any statistical reports. The statistical report includes total number of IP packets transmitted (TCP, UDP, ICMP, and IGMP).

1.1.2. <u>PROPOSED SYSTEM</u>:

The problems and drawbacks faced in the existing system are solved with optimal solution strategies in the proposed system. The solution strategies are developed with careful procedural approach to give out effective and optimal solution. The modules that are designed and developed as the required solution strategies are as below,

Packet Monitoring:

Individual client monitors are implemented in each node of the network. The client monitors analyses and monitors all the incoming and outgoing packets and the details such as Source IP address & Port, Destination IP address & port, packet size, protocol type, etc are recorded.. The packets are monitored by libpcap functions.

Storage:

The server monitor stores the information gathered from the client monitors and stores it in the separate files. This summary is stored permanently for report generation and future reference.

Statistics Generation:

The monitored details are stored and the corresponding statistical reports are generated. The statistical report includes total number of TCP, UDP, ICMP packets generated.

CHAPTER 2
PROPOSED LINE OF ATTACK

2. PROPOSED LINE OF ATTACK

Network traffic is examined by real-time monitors. These monitors test network traffic for a specific set of number of packets, and when those conditions are satisfied the packets are categorized according to the protocol to which it belong and stored in separate files.

DETAILED BLOCK DIAGRAM:

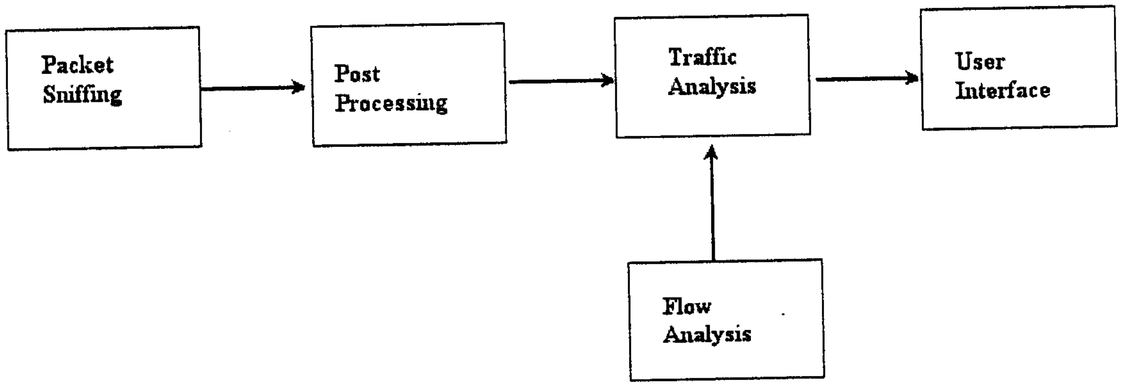


Fig 2.1 NTA Block Diagram

With Network traffic analyzer, the following tasks can be performed:

- Capture frames (also called packets) directly from the local network
- Capture frames from a remote computer
- Display and filter captured frames
- Edit and transmit captured frames onto the network to test network resources or to reproduce network problems
- Display statistics on frames captured locally or on a remote computer

In promiscuous mode a machine listens to all traffic on a network and not only the traffic, which is legitimately directed to it. As a result the network analyzer when in promiscuous mode has access to all the data and control packets flowing over a network. So the promiscuous mode is set first.

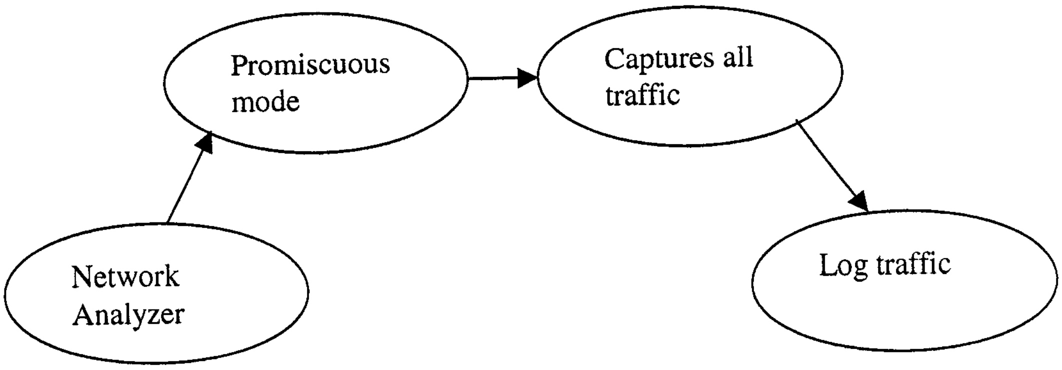


Fig 2.2 NTA Flow Diagram

After setting the promiscuous mode the following are the important functional tasks that the network analyzer has to carryout:

- Creation of a socket in order to provide the network analyzer with a listening point or ear.
- Creation of Headers for TCP and IP information.
- Capture of packets.
- Packets are categorized according to the type.

Appropriate testing procedures are to be followed to ensure that the constructed network analyzer functions flawlessly in the working environment of a LAN. Unit and integration testing are also to be adhered to ensure smooth working of the final entity.

CHAPTER 3
PROPOSED METHODOLOGY

3. PROPOSED METHODOLOGY

PROJECT PLAN

- The **life cycle model** for our project is a simple **SDLC** consisting of the System Engineering., Analysis, Design, Coding and testing.

The various phases that are in the development of the product are as follows:

1. System study
2. Requirements
3. Design
4. Implementation
5. Testing
6. Documentation

- **Team structure** is **democratic** in which every member is to contribute equally to the project development.

CHAPTER 4
PROGRAMMING ENVIRONMENT

4. PROGRAMMING ENVIRONMENT

4.1. HARDWARE REQUIREMENTS

Minimum Requirements:

- Processor: Celeron
- RAM: 128 MB
- Hard disk drive: 20GB

4.2. SOFTWARE REQUIREMENTS

- Platform: Red Hat Linux 8.0
- Language: Java , C
- Desktop: GNOME

4.3. NETWORK REQUIREMENTS:

A minimum configured Ethernet LAN.

CHAPTER 5
IMPLEMENTATION DETAILS

5. IMPLEMENTATION DETAILS

Network traffic analyzer monitors the network data stream, which consists of all of the information transferred over a network at any given time. Prior to transmission, this information is divided by the networking software into smaller segments, called frames or packets. Modules are explained as follows:

Core Modules:

NTA consists of the following modules

1. Packet Sniffing Module
2. Post processing Module
3. User Interface

Packet Sniffing Module:

Packet Sniffing is a system that counts, and copies all the frames it detects, to its capture buffer, which is a reserved storage area in memory. This process is referred to as capturing.

The first thing to understand is the general layout of a pcap sniffer. The flow of code is as follows:

1. The interface want to be sniffed on is first determined. In Linux this may be something like eth0, in BSD it may be xl1, etc. this device can be either define in a string, or can be use pcap to provide us with the name of an interface that will do the job.

2. Initialize pcap. This is where we actually tell pcap what device we are sniffing on. We can, if we want to, sniff on multiple devices. How do we differentiate between them? Using file handles. Just like opening a file for reading or writing, we must name our sniffing "session" so we can tell it apart from other such sessions.
3. In the event that we only want to sniff specific traffic (e.g.: only TCP/IP packets, only packets going to port 23, etc) we must create a rule set, "compile" it, and apply it. This is a three phase process, all of which is closely related. The rule set is kept in a string, and is converted into a format that pcap can read (hence compiling it.) The compilation is actually just done by calling a function within our program; it does not involve the use of an external application. Then we tell pcap to apply it to whichever session we wish for it to filter.
4. Finally, we tell pcap to enter its primary execution loop. In this state, pcap waits until it has received however many packets we want it to. Every time it gets a new packet in, it calls another function that we have already defined. The function that it calls can do anything we want; it can dissect the packet and print it to the user, it can save it in a file, or it can do nothing at all.

Post processing Module:

Each frame contains the following information:

- The source address of the computer that sent the message, which is a unique hexadecimal number that identifies the computer on the network
- The destination address of the computer that received the frame

- Headers from each protocol used to send the frame
- The data or a portion of the information being sent

Consider the captured frame structure name is u_char.

How does this work? Consider the layout of the packet u_char in memory. Basically, all that has happened when pcap stuffed these structures into a u_char is that all of the data contained within them was put in a string, and that string was sent to our callback. The convenient thing is that, regardless of the values set to these structures, their sizes always remain the same. On some workstation, for instance, a sniff_ethernet structure has a size of 14 bytes. A sniff_ip structure is 20 bytes, and likewise a sniff_tcp structure is 20 bytes. The u_char pointer is really just a variable containing an address in memory. That's what a pointer is; it points to a location in memory. For the sake of simplicity, we'll say that the address this pointer is set to the value X. Well, if our three structures are just sitting in line, the first of them (sniff_ethernet) being located in memory at the address X, then we can easily find the address of the other structures.

Variable	**Location (in bytes)**
sniff_ethernet	X
sniff_ip	X + 14
sniff_tcp	X + 14 + 20
Payload	X + 14 + 20 + 20

Table 1.1 Address Calculation

The sniff_ethernet structure, being the first in line, is simply at location X. sniff_ip, who follows directly after sniff_ethernet, is at the location X, plus however much space sniff_ethernet consumes (14 in this example). sniff_tcp is after both sniff_ip and sniff_ethernet, so it is location at X plus the sizes of sniff_ethernet and sniff_ip (14 and 20 byes, respectively). Lastly, the payload (which isn't really a structure, just a character string) is located after all of them.

Finally in this module the captured packet is analyzed and we determine what kind of packet it is, i.e. we analyze each packet and separate the fields for determining which protocol it belongs to. There are different protocols in network transmission namely TCP, UDP, ICMP, IGMP.... .Each protocol has a different structure.

By analyzing the structure we determine which protocols it belongs to and we store the separated packets in a separate file.

User Interface:

In this module using Java AWT controls we provide a good interface to the user. According to user options, we enable them to view the contents of the file they select. Further we provide a summary of the Network Traffic Analyzer.

Pcap Functions:

Some of the important pcap functions are explained as follows.

pcap_lookupdev() returns a pointer to a network device suitable for use with **pcap_open_live**() and **pcap_lookupnet**(). If there is an error, **NULL** is returned and ***errbuf*** is filled in with an appropriate error message.

pcap_t ***pcap_open_live**(char *device, int snaplen, int promisc, int to_ms, char *ebuf)

The first argument is the device which was specified in the previous section. snaplen is an integer which defines the maximum number of bytes to be captured by pcap. The promisc, when set to true, brings the interface into promiscuous mode (however, even if it is set to false, it is possible under specific cases for the interface to be in promiscuous mode, anyway). to_ms is the read time out in milliseconds (a value of 0 sniffs until an error occurs; -1 sniffs indefinitely). Lastly, ebuf is a string where the error messages are stored (same as the errbuf). The function returns the session handler.

```
#include <pcap.h>]

   pcap_t *handle;

      handle   =   pcap_open_live(somedev,   BUFSIZ,   1,   0,   errbuf);
```

A packet contains many attributes, it is not really a string, but actually a collection of structures (for instance, a TCP/IP packet would have an Ethernet header, an IP header, a TCP header, and lastly, the packet's payload). This u_char is the serialized version of these structures. To make any use of it, typecasting must be done.

First, the actual structures must be defined before typecasting is done. The following is the structure definitions that describe a TCP/IP packet over Ethernet. The libraries vary slightly from platform to platform, making it complicated to implement them quickly. Here are the structures:

Ethernet header

```
struct sniff_ethernet {
   u_char ether_dhost[ETHER_ADDR_LEN]; /* Destination host address */
   u_char ether_shost[ETHER_ADDR_LEN]; /* Source host address */
   u_short ether_type; /* IP? ARP? RARP? etc */
};
```

IP header

```
struct sniff_ip {
  #if BYTE_ORDER == LITTLE_ENDIAN
  u_int ip_hl:4, /* header length */
  ip_v:4; /* version */
  #if BYTE_ORDER == BIG_ENDIAN
  u_int ip_v:4, /* version */
  ip_hl:4; /* header length */
  #endif
  #endif /* not _IP_VHL */
  u_char ip_tos; /* type of service */
  u_short ip_len; /* total length */
  u_short ip_id; /* identification */
  u_short ip_off; /* fragment offset field */
  #define IP_RF 0x8000 /* reserved fragment flag */
  #define IP_DF 0x4000 /* dont fragment flag */
  #define IP_MF 0x2000 /* more fragments flag */
  #define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
  u_char ip_ttl; /* time to live */
  u_char ip_p; /* protocol */
  u_short ip_sum; /* checksum */
  struct in_addr ip_src,ip_dst; /* source and dest address */
 };
```

```
if(str==("cancel"))
{ System.exit(0); }

if(str==("ok") && flag<=2)
{
passwordstr=passwordtext.getText();
userstr=usertext.getText();
if(userstr.equals("admin"))
{
if(passwordstr.equals("admin")) {
Buttondemos bd= new Buttondemos();
d.setVisible(false);
bd.setSize(500,600);
bd.setResizable(false);
bd.setBounds(0,0,800,600);
bd.setVisible(true);
bd.setTitle("INTRODUCTION");

}
else
{
JOptionPane.showMessageDialog(this,"Invalid
Password","Error",JOptionPane.ERROR_MESSAGE);
flag++;
}//pss
}
else
{
JOptionPane.showMessageDialog(this,"Invalid
Username","Warning",JOptionPane.WARNING_MESSAGE);
flag++;
}//user
}
else{System.exit(0);}//ok
}//action

public static void main(String arg[])
{
d=new design1();
d.setSize(800,250);
d.setResizable(false);
d.setBounds(150,200,400,250);
d.setVisible(true);
d.setTitle("LOGIN");
}//main

}//design1cls


/*Intrduction Frame*/


class Buttondemos extends JFrame  implements ActionListener {
```

APPENDICES

APPENDIX 1

APPENDICES

APPENDIX 1 SAMPLE CODE

```
import newpack.*;
import intro.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.io.*;
import java.util.*;
import javax.swing.border.*;
import javax.swing.text.*;

/* Initial frame for login*/

public class design1 extends JFrame implements ActionListener
{

JLabel username,password;
JTextField usertext;
JPanel panelLabel;
JPasswordField passwordtext;
JButton okbutton,cancelbutton;
String userstr,passwordstr;
Container con;
```

```
int flag;
static design1 d;



public design1()
{

Color cl=new Color(0,204,255);
con=getContentPane();
con.setLayout(null);
flag=0;
username=new JLabel("Administrator Name:");
username.setForeground(Color.black);
username.setBounds(70,30,250,20);

usertext=new JTextField(15);
usertext.setBounds(200,30,100,20);

password=new JLabel("Pasword: ");
password.setForeground(Color.black);
password.setBounds(70,70,250,20);

passwordtext=new JPasswordField(15);
passwordtext.setBounds(200,70,100,20);

okbutton=new JButton("ok");
okbutton.setBounds(80,120,100,20);
okbutton.addActionListener(this);

cancelbutton = new JButton("cancel");
cancelbutton .setBounds(200,120,100,20);
cancelbutton.addActionListener(this);

panelLabel=new JPanel();
panelLabel.setBackground(cl);
panelLabel.setBounds(0,0,800,800);

con.add(username);
con.add(usertext);
con.add(password);
con.add(passwordtext);
con.add(okbutton);
con.add(cancelbutton);
con.add(panelLabel);

addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});

}//design1

public void actionPerformed(ActionEvent e)
{
String str=e.getActionCommand();
```

TCP header

```
struct sniff_tcp {
  u_short th_sport; /* source port */
  u_short th_dport; /* destination port */
  tcp_seq th_seq; /* sequence number */
  tcp_seq th_ack; /* acknowledgement number */
  #if BYTE_ORDER == LITTLE_ENDIAN
  u_int th_x2:4, /* (unused) */
  th_off:4; /* data offset */
  #endif
  #if BYTE_ORDER == BIG_ENDIAN
  u_int th_off:4, /* data offset */
  th_x2:4; /* (unused) */
  #endif
  u_char th_flags;
  #define TH_FIN 0x01
  #define TH_SYN 0x02
  #define TH_RST 0x04
  #define TH_PUSH 0x08
  #define TH_ACK 0x10
  #define TH_URG 0x20
  #define TH_ECE 0x40
  #define TH_CWR 0x80
  #define TH_FLAGS
 (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
u_short th_win; /* window */
u_short th_sum; /* checksum */ };
```

pcap_open_dead() is used for creating a **pcap_t** structure to use when calling the other functions in libpcap. It is typically used when just using libpcap for compiling BPF code.

pcap_lookupnet() is used to determine the network number and mask associated with the network device **device**. Both *netp* and *maskp* are *bpf_u_int32* pointers. A return of -1 indicates an error in which case *errbuf* is filled in with an appropriate error message.

int **pcap_compile**(pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask)

The first argument is the session handle (pcap_t *handle in the previous example). Following that is a reference to the place where the compiled version of our filter is stored. Then comes the expression itself, in regular string format. Next is an integer that decides if the expression should be "optimized" or not (0 is false, 1 is true.) Finally, the net mask of the network the filter must be specified. The function returns -1 on failure; all other values imply success.

After the expression has been compiled, it is time to apply it. Enter pcap_setfilter(). Following the format of explaining pcap, the pcap_setfilter() prototype is shown:
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
This is very straight forward. The first argument is session handler, the

second is a reference to the compiled version of the expression (presumably the same variable as the second argument to pcap_compile()).

pcap_setfilter() is used to specify a filter program. *fp* is a pointer to a *bpf_program* struct, usually the result of a call to **pcap_compile()**. **-1** is returned on failure, in which case **pcap_geterr()** may be used to display the error text; **0** is returned on success.

The prototype for pcap_next() fairly simple: u_char ***pcap_next**(pcap_t *p, struct pcap_pkthdr *h) The first argument is our session handler. The second argument is a pointer to a structure that holds general information about the packet, specifically the time in which it was sniffed, the length of this packet, and the length of his specific portion (incase it is fragmented, for example.) pcap_next() returns a u_char pointer to the packet that is described by this structure.

Consider the following pcap method:

- **int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)**

This is used to provide the core functionality of our engine. When **pcap_loop(..)** is called it will grab cnt packets (it will loop infinitely when cnt is -1) and pass them to the callback function which is of type **pcap_handler.**

CHAPTER 6
CONCLUSION

6. CONCLUSION

Network Traffic Analyzer enables the administrator to monitor and detect problems with traffic on a specified LAN. The information that NTA provides comes from the traffic itself, which is divided into frames. These frames contain information such as the address of the computer that sent the frame, the address of the computer to which the frame was sent, and the protocols that exist within the frame.

NTA has been tested on a LAN and the data captured has been logged. The logged data has provided useful insights into the network activity during the capture interval. Also the built in automation feature has proved to be very useful to Network Administrators who have automated the execution of the packet capture utility according to their convenience. Further the promiscuous mode detection utility identify if any other sniffing activity is going on a local host or subnet.

This can be modified to suit the needs of the administrator for monitoring. And hope that this project serves the purpose of simplicity and reliability.

CHAPTER 7
FUTURE ENHANCEMENTS

7. FUTURE ENHANCEMENTS

Network Traffic Analyzer requires an administrator of a LAN to watch the tool. So we need some proactivity for monitoring the network without the human presence. Our future enhancements is that to use a server (another process) running on managed nodes. With this we can launch actions on managed nodes automatically, so human presence is not required.

The present NTA is used to monitor limited number of protocols like TCP/IP, UDP, ICMP and IGMP. In future it can be modified to monitor Application layer protocols like FTP, HTTP, SMTP…

In our project we use separate log files for storing the information. This log files can be modified to separate database using tools like MYSQL.

```
JPanel paneluplabel,panelbutton,paneldownlabel,panelpic;
JButton ok;

TextArea txt;
JTextArea abs;
Container con;
JLabel nta,abst,pt;
Font font1,font2;
ImageIcon sqr;


 public Buttondemos()
 {

 con=getContentPane();
 con.setLayout(null);
 con.setBackground(Color.black);
 font1=new Font("Arial",Font.BOLD+Font.ITALIC,50);
 font2=new Font("Arial",Font.BOLD+Font.ITALIC,20);

 sqr=new ImageIcon("imgintro.gif");
 paneluplabel= new JPanel();
 paneluplabel.setBackground(Color.black);
 paneluplabel.setBounds(0,0,800,100);

 panelpic= new JPanel();
 panelpic.setBounds(5,100,300,410);
panelpic.setBackground(Color.black);
pt=new JLabel(sqr);
panelpic.add(pt);

nta=new JLabel("Network Traffic Analyser");
nta.setForeground(new Color(255,206,106));
nta.setFont(font1);
paneluplabel.add(nta);



paneldownlabel= new JPanel();
paneldownlabel.setBackground(Color.black);
paneldownlabel.setBounds(0,510,800,100);



abs ab=new abs();
JTextArea abs=new JTextArea(ab.note);
abs.setEditable(false);
abs.setBounds(300,100,500,350);
abs.setBackground(Color.black);
abs.setForeground(new Color(0,183,183));
abs.setFont(font2);

panelbutton= new JPanel();
panelbutton.setBackground(Color.pink);
panelbutton.setBounds(300,350,500,50);
```

```
ok=new JButton("OK");
ok.setBounds(450,460,200,30);
ok.setBackground(new Color(255,206,106));
panelbutton.add(ok);

ok.addActionListener(this);

con.add(abs);
con.add(ok);

con.add("North",paneluplabel);
con.add("South",paneldownlabel);
con.add("West",panelpic);
con.add("East",panelbutton);

addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});

}//butdemo

public void actionPerformed(ActionEvent ae)
{

String str=ae.getActionCommand();
if(str.equals("OK"))
{
```

```
this.setVisible(false);
Color cl=new Color(102,153,204);
design3 d3=new design3();
d3.setSize(500,600);
d3.setTitle("PACKET CAPTURING");
d3.setResizable(false);
d3.setBounds(0,0,800,600);
d3.setBackground(cl);
d3.setVisible(true);
}
}//act

/* Frame Executing Packet capturing in background*/

public class design3 extends JFrame implements ActionListener
{

JPanel panelpiclabel,paneldownbutton;
JButton bstart,bstop,bprocess;
Container con;
JLabel pict,pictt;
ImageIcon act_img,act_imgg;
Font font1;
JTextArea txt;
int flag1,flag2;

Color cl=new Color(102,153,204);
```

```
public design3()
{

con=getContentPane();
con.setLayout(null);
con.setBackground(cl);
act_img=new ImageIcon("packet3.gif");
pict=new JLabel(act_img);
act_imgg=new ImageIcon("packet.gif");
pictt=new JLabel(act_imgg);

font1=new Font("Arial",Font.BOLD+Font.ITALIC,20);

panelpiclabel= new JPanel();
panelpiclabel.setBounds(0,0,800,400);
panelpiclabel.setBackground(cl);
panelpiclabel.add(pict);
pict.setVisible(false);
pict.setBounds(150,0,550,400);
panelpiclabel.add(pictt);
pictt.setVisible(true);
pictt.setBounds(150,0,550,400);

paneldownbutton= new JPanel();
paneldownbutton.setBackground(cl);
paneldownbutton.setBounds(0,400,800,200);
```

```
bstart=new JButton("Start");
bstart.setBounds(10,450,100,30);
bstart.addActionListener(this);
bstart.setBackground(new Color(221,164,193));

bstop=new JButton("Stop");
bstop.setBounds(310,450,100,30);
bstop.addActionListener(this);
bstop.setBackground(new Color(221,164,193));
bstop.setEnabled(false);

bprocess=new JButton("Processdetails");
bprocess.setBounds(510,450,100,30);
bprocess.addActionListener(this);
bprocess.setBackground(new Color(221,164,193));


paneldownbutton.add(bstart);
paneldownbutton.add(bstop);
paneldownbutton.add(bprocess);

con.add("North",panelpiclabel);
con.add("South",paneldownbutton);

addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});
```

```
}//design3

public void actionPerformed(ActionEvent ae)
{
String str=ae.getActionCommand();

if(str.equals("Start"))
{
pict.setVisible(true); pictt.setVisible(false); bstop.setEnabled(true);
backdeamon b=new backdeamon();
}
if(str.equals("Stop"))
{
pict.setVisible(false);
this.setVisible(false);
Color cl=new Color(102,153,204);
designpic d4=new designpic();
d4.setSize(500,600);
d4.setTitle("POST PROCESSING");
d4.setResizable(false);
d4.setBounds(0,0,800,600);
d4.setBackground(cl);
d4.setVisible(true);
prostop ps=new prostop();
}// To stop the process running in the background
```

```
if(str.equals("Processdetails"))
{
details det =new details();
det.setSize(500,600);
det.setTitle("PACKET CAPTURING DETAILS");
det.setResizable(false);
det.setBounds(400,150,400,450);
det.setVisible(true);} }//act
}//design3cls
/*Running the process in the background*/
class backdeamon{ Runtime r=Runtime.getRuntime();process p=null;
try{ p=r.exec("/bin/t/./Tes");}catch(Exception e){}
}
/* Frame explaining packet capturing details*/
class details extends JFrame implements ActionListener{
JButton ok;
JTextArea txt;
Color cl=new Color(255,159,255);
Container con;
Font font3;
String str;
public details()
{
font3=new Font("Arial",Font.BOLD,15);
con=getContentPane();
con.setLayout(null);
```

```
ok=new JButton("OK");
ok.setBounds(150,385,100,30);
ok.addActionListener(this);
//ok.setBackground(cl);

pktdetails pktd= new pktdetails();
txt=new JTextArea(pktd.pkts);
txt.setBounds(0,0,400,385);
txt.setBackground(cl);
txt.setEditable(false);
txt.setFont(font3);
con.add(txt);
con.add(ok);
}//details
public void actionPerformed(ActionEvent ae)
{
str=ae.getActionCommand();
if(str.equals("OK"))
{
this.setVisible(false);
} }//act }//destailscls
/* Frame executing Post Processing module*/

class designpic extends JFrame implements ActionListener
{
JPanel panelpiclabel;
Container con;
```

```
JButton ok;
JLabel pict;
ImageIcon act_img;
Font font1,font3;
JTextArea txt;

Color cl=new Color(102,153,204);
public designpic()
{
font3=new Font("Arial",Font.BOLD,15);
con=getContentPane();
con.setLayout(null);
con.setBackground(cl);
act_img=new ImageIcon("postpacket.gif");

pict=new JLabel(act_img);
pict.setVisible(true);
pict.setBounds(0,0,550,400);

postdetails pd= new postdetails();
txt=new JTextArea(pd.posts);
txt.setBounds(400,0,400,500);
txt.setBackground(cl);
txt.setEditable(false);
txt.setFont(font3);
ok=new JButton("OK");
ok.setBounds(525,500,100,30);
ok.addActionListener(this);
ok.setBackground(new Color(221,164,193));
con.add(ok);


font1=new Font("Arial",Font.BOLD,20);
panelpiclabel= new JPanel();
panelpiclabel.setBounds(0,0,400,600);
panelpiclabel.setBackground(cl);
panelpiclabel.add(pict);
//panelpiclabel.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.
RAISED));

con.add(panelpiclabel);
con.add(txt);

addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});

}//designpic

public void actionPerformed(ActionEvent ae)
{
String str=ae.getActionCommand();
if(str.equals("OK"))
{
```

```
this.setVisible(false);

output op=new output();
op.setSize(800,250);
op.setTitle("PACKET INFORMATION");
op.setResizable(false);
op.setBounds(0,0,800,600);
op.setVisible(true);
}

}//act
/*Frame displaying the captued packet*/



class output extends JFrame implements ActionListener {
JPanel paneluplabel,panelbutton,paneldownlabel,paneleastlabel;
Container con;
TextArea txt;
JButton bnext;
JLabel nta;
Font font1,font2,font3;

public output()
{

con=getContentPane();
con.setLayout(null);
```

```
font3=new Font("Arial",Font.BOLD,15);
font1=new Font("Arial",Font.BOLD+Font.ITALIC,50);
font2=new Font("OCR A Extended",Font.BOLD+Font.ITALIC,20);
paneluplabel= new JPanel();
paneluplabel.setBackground(Color.black);
paneluplabel.setBounds(0,0,800,100);

nta=new JLabel("Network Traffic Analyser");
nta.setForeground(new Color(255,206,106));
nta.setFont(font1);
paneluplabel.add(nta);



paneldownlabel= new JPanel();
paneldownlabel.setBackground(Color.black);
paneldownlabel.setBounds(0,500,800,100);

paneleastlabel= new JPanel();
paneleastlabel.setBackground(Color.pink);
paneleastlabel.setBounds(650,100,150,400);
paneleastlabel.setLayout(new GridLayout(1,1));

txt=new TextArea();
txt.setBounds(150,100,500,400);
txt.setBackground(Color.white);
txt.setForeground(Color.black);
txt.setFont(font3);
txt.setEditable(false);

panelbutton= new JPanel();
panelbutton.setBackground(Color.pink);
panelbutton.setBounds(0,100,150,400);
panelbutton.setLayout(new GridLayout(5,1));
panelbutton.add( new MyButton("IP",txt ));
panelbutton.add( new MyButton("TCP",txt));
panelbutton.add( new MyButton("UDP",txt));
panelbutton.add( new MyButton("ICMP",txt));
panelbutton.add( new MyButton("IGMP",txt));

bnext=new JButton("NEXT>>>");
bnext.setBounds(670,500,100,50);
bnext.addActionListener(this);
bnext.setFont(font2);
bnext.setBackground(Color.pink);
paneleastlabel.add(bnext);
con.add(txt);
con.add("North",paneluplabel);
con.add("South",paneldownlabel);
con.add("West",panelbutton);
con.add("East",paneleastlabel);
addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});
```

```
}//op
class MyButton extends Button {
TextArea textArea;
public MyButton(String text,TextArea newTextArea) {
   super(text);
   super.setFont(font2);
   super.setBackground(Color.pink);
   textArea = newTextArea;
   textArea.setBackground(Color.white);
   textArea.setEditable(false);
 }//my button

 public boolean action(Event event,Object arg) {

 if("IP".equals(event.arg)||"TCP".equals(event.arg)||"UDP".equals(event.arg)||
 "ICMP".equals(event.arg)||"IGMP".equals(event.arg)){
  String inFile = event.arg+".txt";
  inFile=inFile.toLowerCase();
  readFile(inFile);
  return true;
  }//if
 return false;
  }//act

  public void readFile(String file) {
   DataInputStream inStream;
   try{
      inStream = new DataInputStream(new FileInputStream(file));
  }
catch (IOException ex){
      System.out.println("Error opening file");
       return;
  }
try{
      String newText="";
      String line;
      while((line=inStream.readLine())!=null)
      newText=newText+line+"\n";
     textArea.setText(newText);
      inStream.close();
   }
 catch (IOException ex){
       System.out.println("Error reading file");
   }
 }//read
}//myclass
public void actionPerformed(ActionEvent ae)
{
String str=ae.getActionCommand();
if(str.equals("NEXT>>>"))
{
finalop fp=new finalop();
fp.setSize(800,250);
fp.setResizable(false);
fp.setBounds(0,0,800,600);
fp.setVisible(true);
}
}//act

/*Frame displaying final statistics*/

class finalop extends JFrame implements ActionListener {
int s;
JPanel paneluplabel,panelbutton;
Container con;
JTextArea ttxt,utxt,itxt,txt1,txt5;
Button b1,b2,b3;
JButton bexit;
JLabel nta;
Font font1,font2;
Randdom ran=new Randdom();
String str;
public finalop()
{
Color cl=new Color(0,204,255);
con=getContentPane();
con.setLayout(null);
con.setBackground(Color.black);
font1=new Font("Arial",Font.BOLD+Font.ITALIC,50);
font2=new Font("OCR A Extended",Font.BOLD+Font.ITALIC,17);
paneluplabel= new JPanel();
```

```
paneluplabel.setBackground(Color.black);
paneluplabel.setBounds(0,0,800,60);
nta=new JLabel("Traffic Statistics");
nta.setForeground(new Color(255,206,106));
nta.setFont(font1);
paneluplabel.add(nta);
txt1=new JTextArea("PACKETS RECEIVED :");
str=" "+ran.total;
txt1.append(str);
txt1.setBounds(300,60,500,60);
txt1.setFont(font2);
txt1.setBackground(Color.black);
txt1.setForeground(cl);
panelbutton= new JPanel();
panelbutton.setBackground(new Color(221,164,193));
panelbutton.setBounds(150,150,500,30);
panelbutton.setLayout(new GridLayout(1,3));
b1=new Button("TCP");
b2=new Button("UDP");
b3=new Button("ICMP");
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
panelbutton.add(b1);
panelbutton.add(b2);
panelbutton.add(b3);
ttxt=new JTextArea();
```

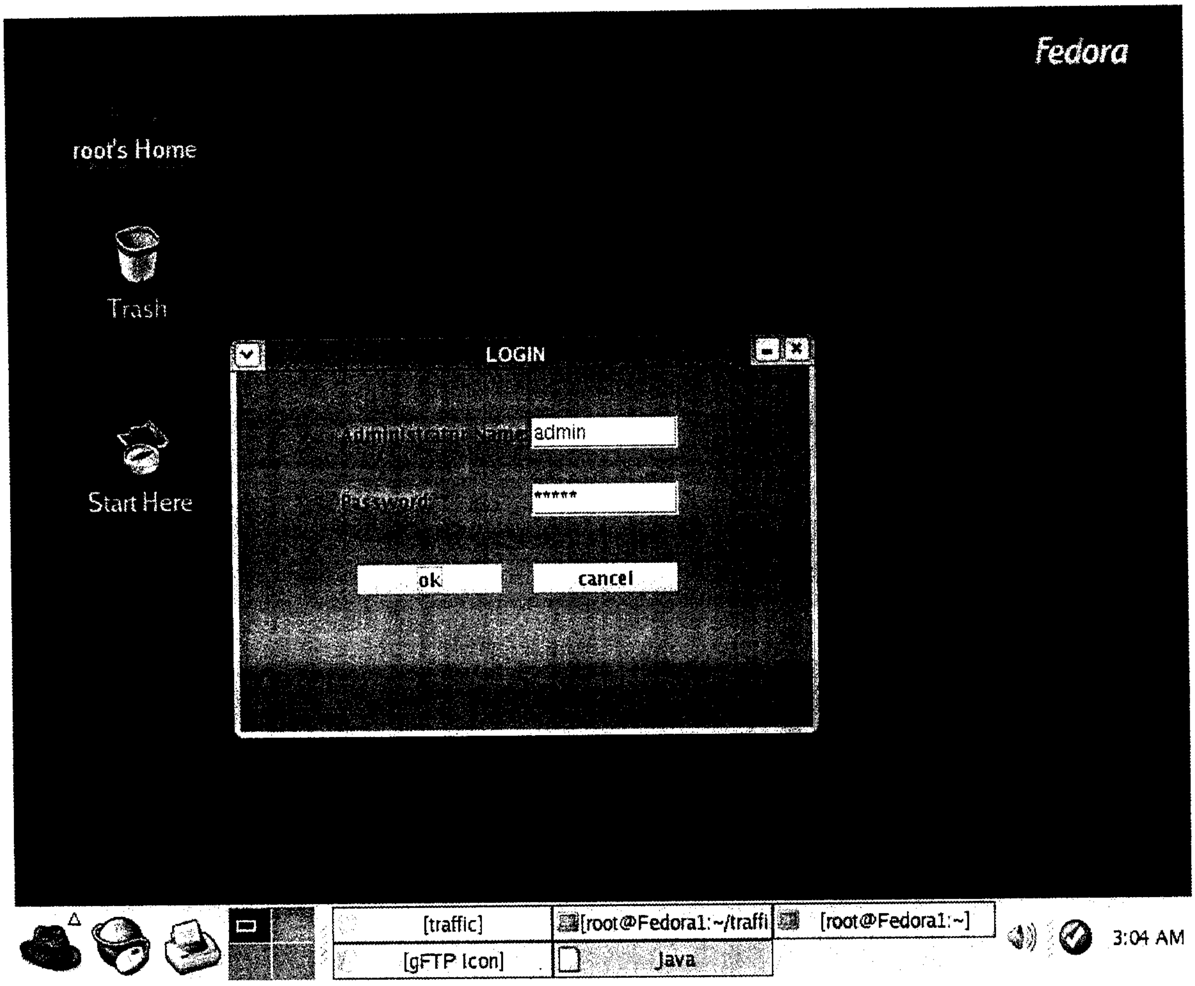
```
ttxt.setBounds(150,210,175,50);
ttxt.setBackground(Color.black);
ttxt.setForeground(Color.white);
ttxt.setFont(font2);
ttxt.setEditable(false);
con.add(ttxt);
utxt=new JTextArea();
utxt.setBounds(150,270,175,50);
utxt.setBackground(Color.black);
utxt.setForeground(Color.white);
utxt.setFont(font2);
utxt.setEditable(false);
con.add(utxt);
itxt=new JTextArea();
itxt.setBounds(150,330,175,50);
itxt.setBackground(Color.black);
itxt.setForeground(Color.white);
itxt.setFont(font2);
itxt.setEditable(false);
con.add(itxt);

dropdetails d=new dropdetails();
txt5=new JTextArea(d.drops);
txt5.append("0");
txt5.setFont(font2);
txt5.setBounds(100,400,500,20);
txt5.setBackground(Color.black);
```

```
txt5.setForeground(cl);
bexit=new JButton("EXIT");
bexit.setBounds(670,500,100,50);
bexit.addActionListener(this);
bexit.setFont(font2);
bexit.setBackground(new Color(221,164,193));

con.add("North",paneluplabel);
con.add(txt1);
con.add("West",panelbutton);
con.add(txt5);
con.add(bexit);
addWindowListener(new WindowAdapter(){public void
windowClosing(WindowEvent e){
        System.exit(0);}});

}//finalop

public void actionPerformed(ActionEvent ae)
{
String str=ae.getActionCommand();
if(str.equals("EXIT"))
{
System.exit(0);
}
if(str.equals("TCP"))
{
```

```
str="TCP TOTAL : "+ran.val[2]+"     ";
ttxt.append(str);
}
if(str.equals("UDP"))
{
str="UDP TOTAL : "+ran.val[1]+"     ";
utxt.append(str);
 }
 if(str.equals("ICMP"))
 {
 str="ICMP TOTAL : "+"2"+"     ";
 itxt.append(str);
  }
 }//act
 }//finalopclass
 }//opclass
 }//designpiccls
 }//design1cls
```

APPENDIX 2

APPENDIX 2 RESULTS

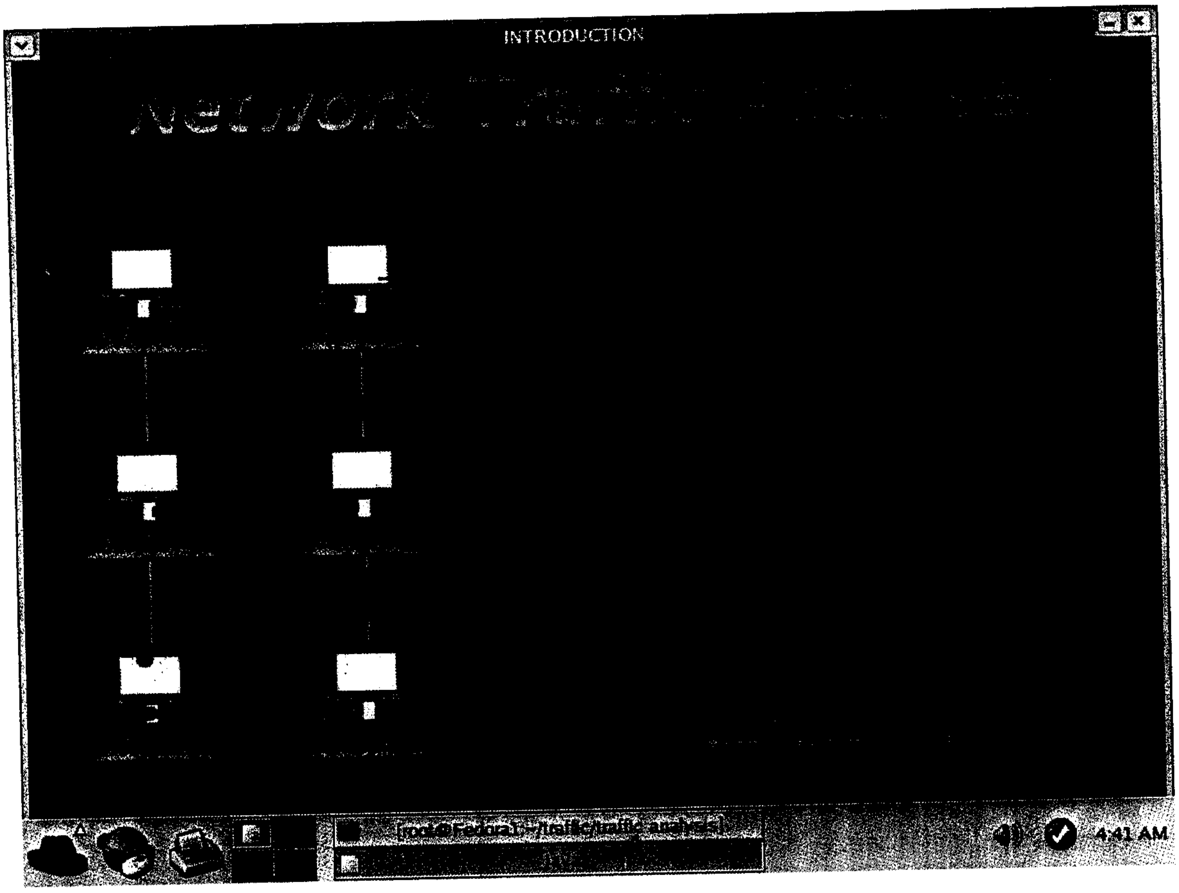


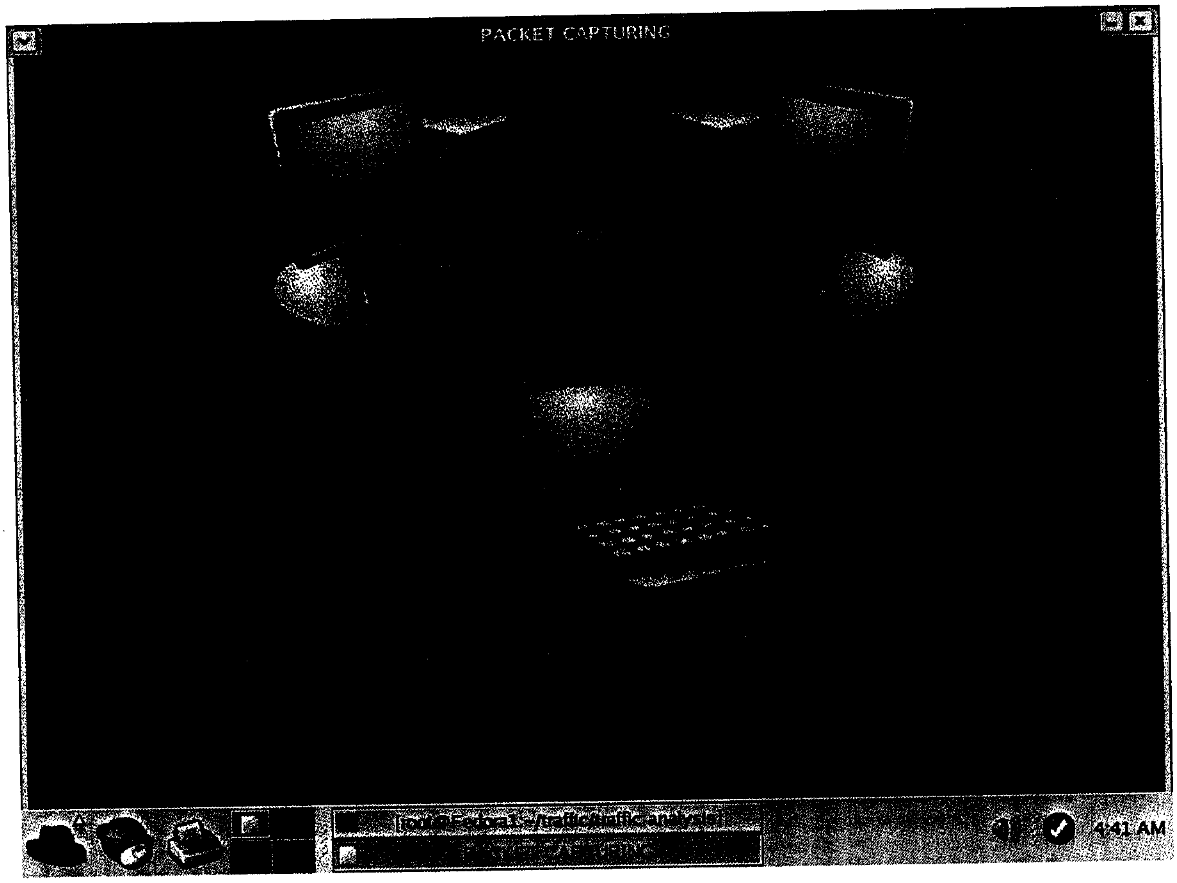
LOGIN

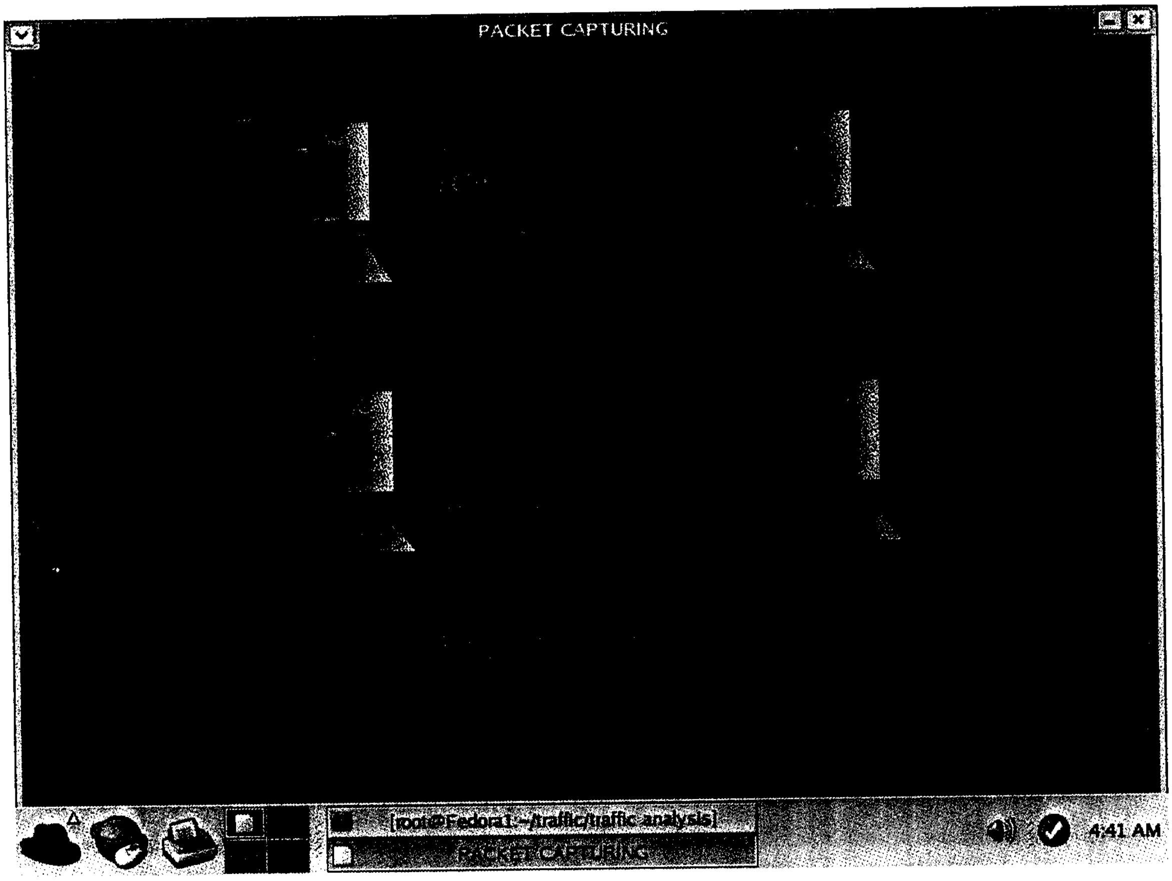
Administrator Name: admin

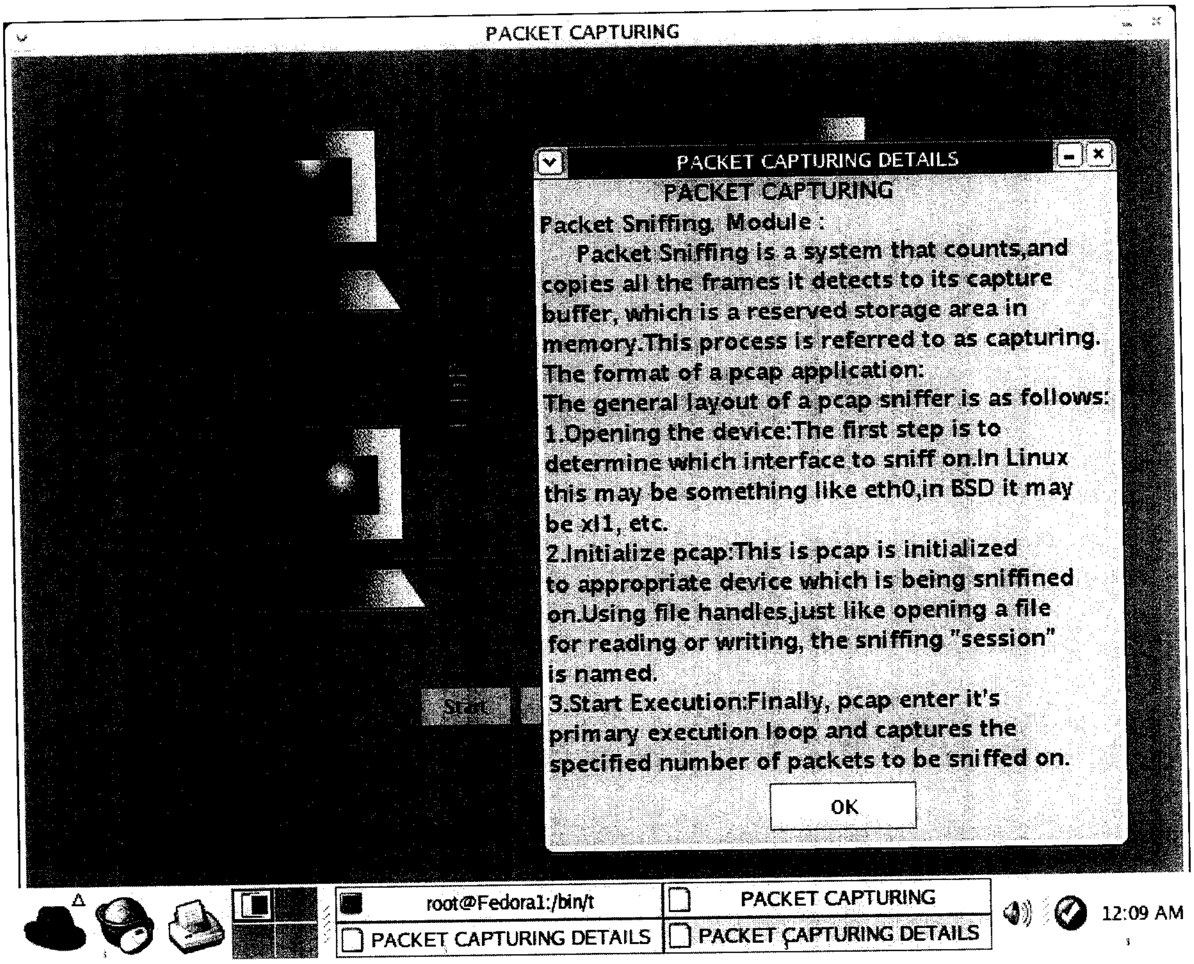
Password: *****

ok cancel


INTRODUCTION
Network
[root@Fedora1:~/traffic/traffic_analysis]
4:41 AM


PACKET CAPTURING
[root@Fedora1:~/traffic/traffic analysis]
PACKET CAPTURING
4:41 AM


PACKET CAPTURING
[root@Fedora1:~/traffic/traffic analysis]
PACKET CAPTURING
4:41 AM



PACKET CAPTURING DETAILS

PACKET CAPTURING

Packet Sniffing Module :

Packet Sniffing is a system that counts,and copies all the frames it detects to its capture buffer, which is a reserved storage area in memory.This process is referred to as capturing.

The format of a pcap application:

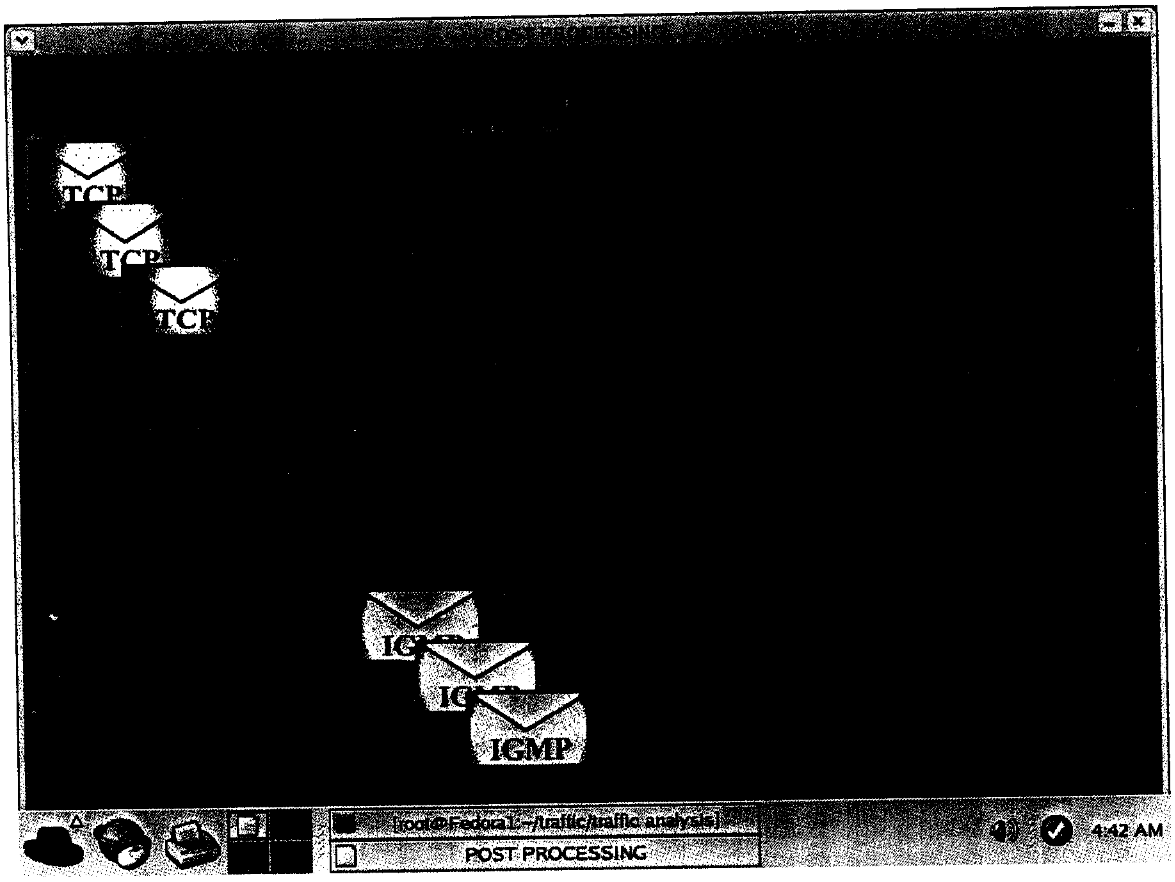
The general layout of a pcap sniffer is as follows:

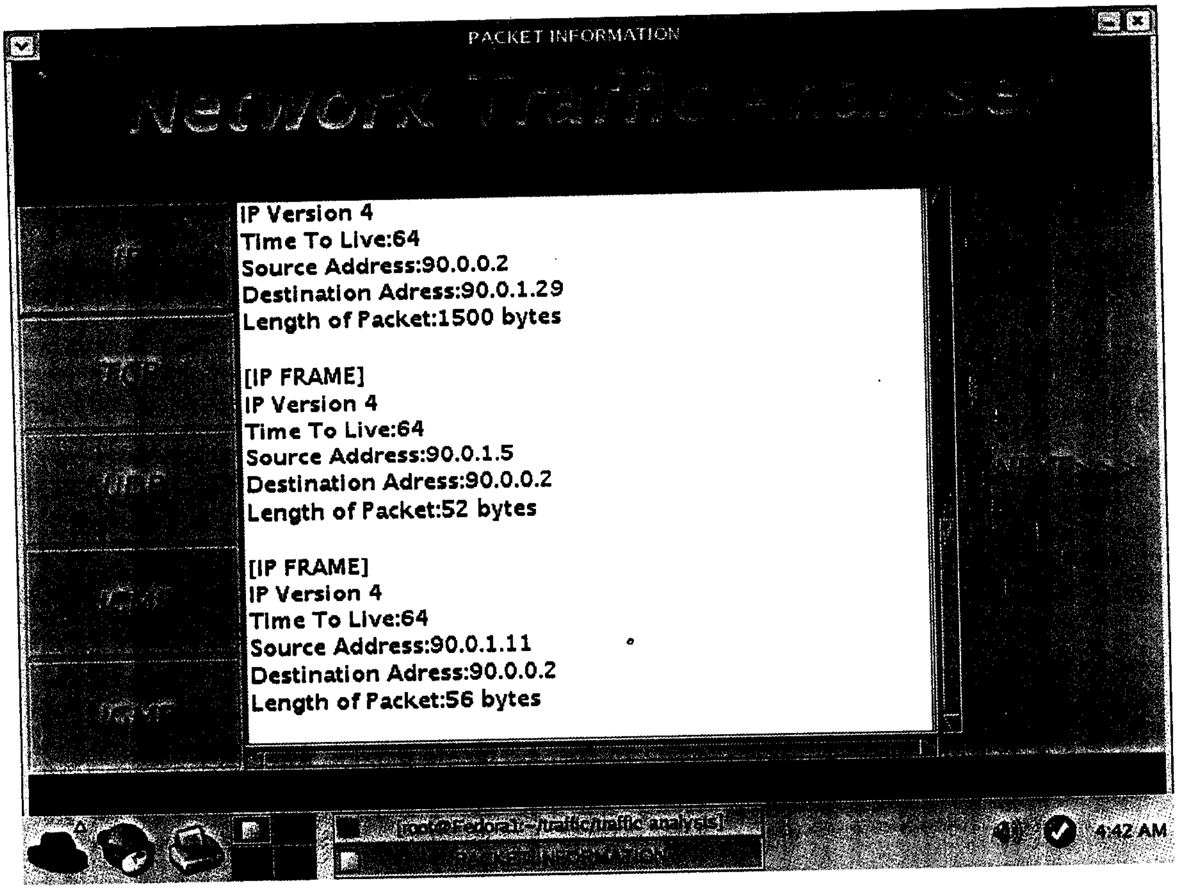
1.Opening the device:The first step is to determine which interface to sniff on.In Linux this may be something like eth0,in BSD it may be xl1, etc.

2.Initialize pcap:This is pcap is initialized to appropriate device which is being sniffined on.Using file handles,just like opening a file for reading or writing, the sniffing "session" is named.

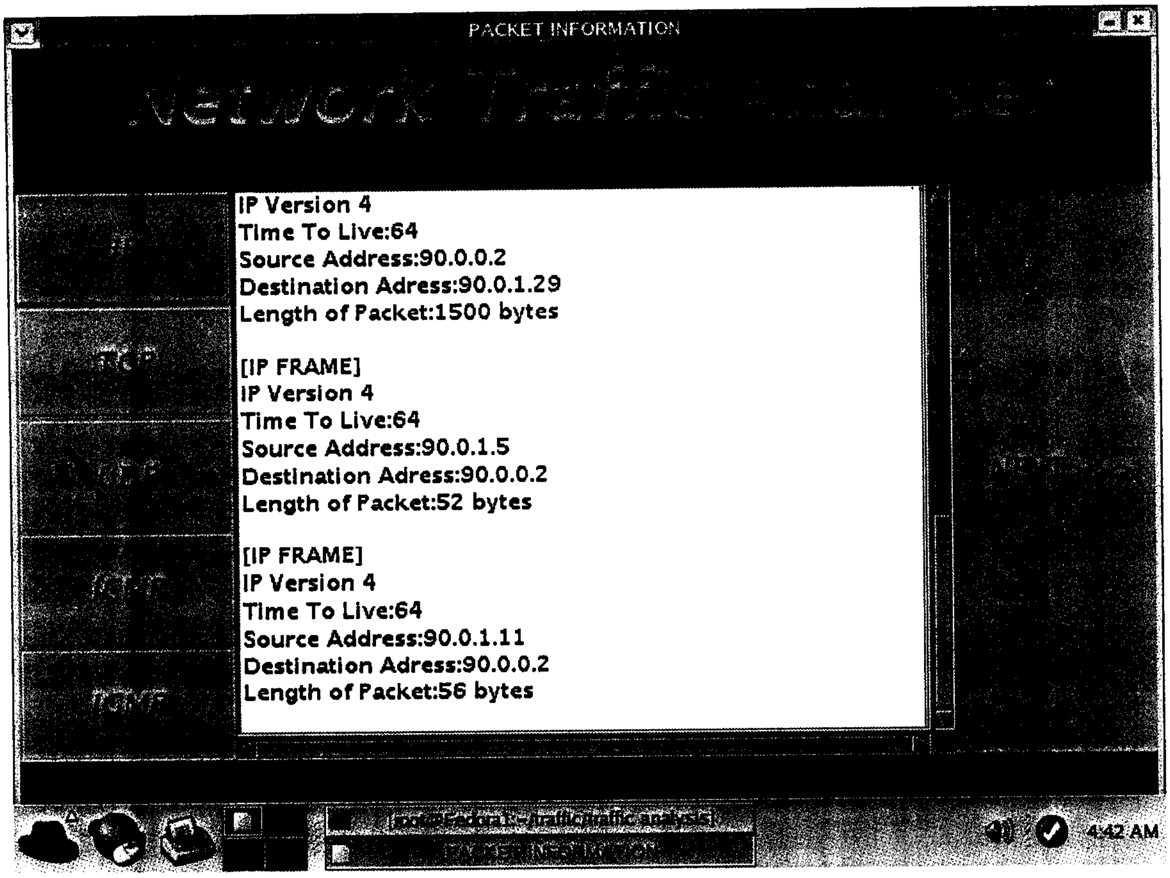
3.Start Execution:Finally, pcap enter it's primary execution loop and captures the specified number of packets to be sniffed on.

OK

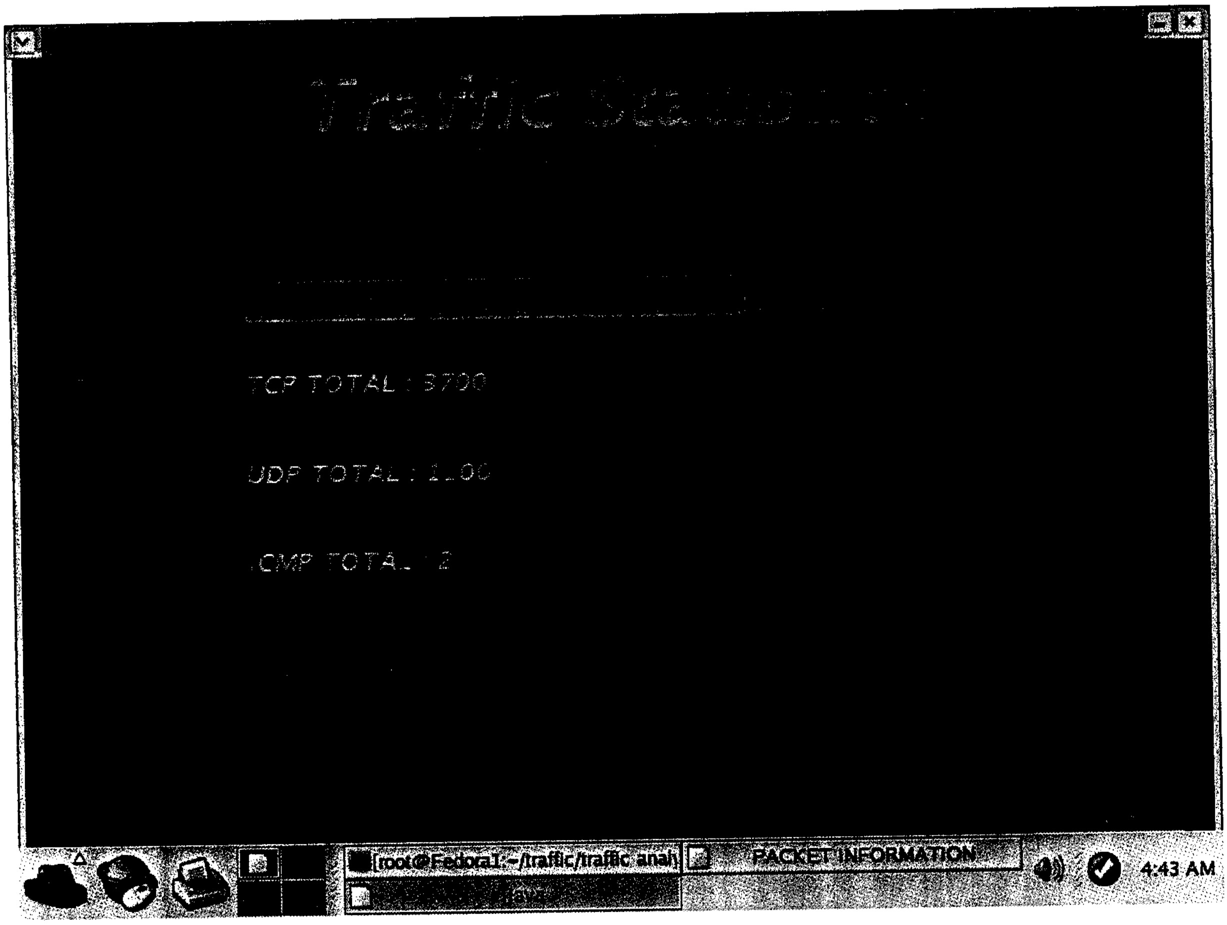

POST PROCESSING
TCP
TCP
TCP
IGMP
IGMP
IGMP
[root@Fedora1:~/traffic/traffic analysis]
POST PROCESSING
4:42 AM



```
IP Version 4
Time To Live:64
Source Address:90.0.0.2
Destination Adress:90.0.1.29
Length of Packet:1500 bytes

[IP FRAME]
IP Version 4
Time To Live:64
Source Address:90.0.1.5
Destination Adress:90.0.0.2
Length of Packet:52 bytes

[IP FRAME]
IP Version 4
Time To Live:64
Source Address:90.0.1.11
Destination Adress:90.0.0.2
Length of Packet:56 bytes
```



```
IP Version 4
Time To Live:64
Source Address:90.0.0.2
Destination Adress:90.0.1.29
Length of Packet:1500 bytes

[IP FRAME]
IP Version 4
Time To Live:64
Source Address:90.0.1.5
Destination Adress:90.0.0.2
Length of Packet:52 bytes

[IP FRAME]
IP Version 4
Time To Live:64
Source Address:90.0.1.11
Destination Adress:90.0.0.2
Length of Packet:56 bytes
```



Traffic Statistics

TCP TOTAL : 3700

UDP TOTAL : 1100

ICMP TOTAL : 2

REFERENCES

REFERENCES

1. Andrew S. Tannenbaum, "Computer Networks", Prentice Hall, Inc.
2. Behrouz A. Forouzan, (1999) "TCP/IP Protocol suite", Tata Mc Graw Hill.
3. Douglas Comer, "TCP/IP and Intranetworking", Volume 2, Second Edition.
4. Gittleman Art, "Internet programming with JAVA 2 platform".
5. Herbert Schldit, "JAVA 2 Complete Reference".
6. James F. Kurose and Keith W. Ross, (2001) "Computer networking a top down approach", Addison Wesley Longman.
7. Joseph L.Weber, "Using JAVA 2 Platform Specified Edition".
8. Richard Stone, Neil Mathew and Alan Cocx, "Beginning Linux Programming".
9. www.anonymizer.com
10. www.cert.org
11. www.ecst.csuchico.edu/~beej/guide/net/
12. www.ietf.org
13. www.java.sun.org
14. www.tcpdump.org