J-NETVIEW

A PROJECT REPORT

Submitted by

SANGEETHA. K **71201205052**

SHREENANDHINI. S **71201205056**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY : CHENNAI 600 025

APRIL 2005

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**J-NETVIEW**" is the bonafide work of "**SANGEETHA.K, SHREENANDHINI.S**" who carried out the project work under my supervision.

SIGNATURE

Dr. S. THANGASWAMY

HEAD OF THE DEPARTMENT

Computer Science and Engineering,
Kumaraguru College of Technology,
Coimbatore - 641006.

SIGNATURE

13/4/05

Mrs.N.SUGANTHI

SUPERVISOR

Senior Lecturer

Information Technology,
Kumaraguru College of Technology,
Coimbatore - 641006.

The candidates with University Register Nos. **71201205052, 71201205056** were examined by us in the project viva-voce examination held on ...19.:.04.:.05...

INTERNAL EXAMINER

EXTERNAL EXAMINER

19/04/05.

ACKNOWLEDGEMENT

We are extremely grateful to **Dr.K.K.Padmanabhan**, B.Sc. (Engg.)., M.Tech., Ph.D., Principal, Kumaraguru College of Technology for having given us a golden opportunity to embark on this project.

We are deeply obliged to **Dr.S.Thangasamy**, Ph.D, Head of the Department of Information Technology for his valuable guidance and useful suggestions during the course of this project.

We also extend our heartfelt thanks to our project coordinator **Mr.K.R.Baskaran,** M.S., Assistant Professor Department of Information Technology for providing the support which really helped us to make this project a success.

We are indebted to our project guide **Ms.N.Suganthi**, M.E., Senior Lecturer Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staff of our Department for providing us the technical support in the duration of our project.

We also thank all of our friends who helped us to complete this project successfully.

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS

API	: Application Programming Interface.
SRS	: Software Requirement Specification
JDK	: Java Development Kit
JCAP	: Java Packet Capture
WINPCAP	: Windows Packet Capture
AWT	: Abstract Window Toolkit
ACK	: Acknowledgement
HTTP	: Hypertext Transfer Protocol
IP	: Internet Protocol
SNMP	: Simple Network Transport Protocol
LAN	: Local Area Network
TCP	: Transmission Control Protocol
ARP	: Address Resolution Protocol
ICMP	: Internet Control Message Protocol
NIC	: Network Interface Card

2. SYSTEM REQUIREMENTS AND ANALYSIS

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

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

2.1 Product Definition

Our intention is to develop a network monitoring software which allows the administrator to have a complete view about the network and capture the details of the packets arrived.

2.2 Project Plan

In the analysis phase, we learned about the various technologies and identified the technologies most suitable for the project implementation. From this study, the most suitable languages were selected as Java. Differentiation of modules, user interface design and other crucial aspects were identified and designed as necessary. Other ambiguities which may exist are identified and take care of.

From the analysis phase, the design phase commences in which the various module's functionalities are identified. The complete system's flow of control and data are identified and depicted in the form of diagrams.

Next the implementation phase is taken care of in which the design is translated into code. Each module is coded separately and finally integrated to form the entire system. Care is taken to make the code easily understandable by future users.

In the testing phase, each module is tested thoroughly and finally the integrated modules are tested together to ensure the correct working of the entire system. Testing is also done to ensure that the product satisfies the specified requirements and set criteria.

3. SOFTWARE REQUIREMENTS SPECIFICATION

3.1 Introduction

3.1.1 Purpose:

The purpose of this project "J-NetView" is develop a network monitoring software which allows the administrator to have a complete view about the network and capture the details of the packets arrived.

3.1.2 Scope:

The scope of the system is to capture the packets and log the details.A graph is drawn for the number of packets captured. The ports are scanned to know about their status.SNMP trap messages are received and if any error occurs in the network during monitoring, the administrator is given an alert through E-mail.

3.1.3 Definition:

Customer:

A person or organization, internal or external to the producing organization, who takes financial responsibility for the system. In a large system this may not be the end user. The customer is the ultimate recipient of the developed product and its artifacts.

User:

A person who will use the system that is developed.

Analyst:

The Analyst details the specification of the system's functionality by describing the requirements aspect and other supporting software requirements

3.1.4 Abbreviation:

SRS	: Software Requirement Specification
JDK	: Java Development Kit
JCAP	: Java Packet Capture
WINPCAP	: Windows Packet Capture
ACK	: Acknowledgement
IP	: Internet Protocol
SNMP	: Simple Network Transport Protocol

3.2 General Description

3.2.1 Product Overview:

It is a pure java application of network monitoring software using SNMP implementation. The system has various functionalities like packet capture, logging the details of the packet in a database, graphical representation of the traffic and receives trap messages from SNMP capable devices and also can transfer the trap message to desired address via email.

3.2.2 User Characteristics:

The main user of this system is the administrator who is in charge of the network.He is expected to have a basic knowledge of networking and its operations.

3.2.3 General Constraints:

The clients are programmed to run in the windows platform but can be migrated to other platforms if the need arises. The server however is a multi-platform application.

3.2.4 General assumptions:

The network administrator should view the database at least once in a day to know about the traffic present in the network.

3.3 Specific Requirements

3.3.1 Inputs and Outputs:

The main input of the system is specifying the device address for packet capture. The other inputs are specifying the host address and the number of ports which are to be scanned.

The expected outputs of the system are displaying the details of the packets captured and the status of the ports that are scanned.

3.3.2 Functional requirements:

i. The system should allow the user to enter only valid port number and device address.

ii. The system should allow the network administrator to perform the desired functions without any problems.

3.3.3 System Requirements:

3.3.3.1 Hardware Requirements:

Processor	:	Pentium IV
RAM size	:	128 MB RAM
Hard disk capacity:		20GB

3.3.3.2 Software Requirements:

Operating System	:	Windows 2000/XP/98
Database Used	:	Microsoft Access
Language	:	JDK 1.5

3.3.4 Performance constraints:

The error alert through E mail to the administrator should be within 10 seconds.

3.3.5 Software Constraints:

The system requires Winpcap and jpcap API's installed .The administrator should have some web server in order to check for error messages. The server requires that jdk1.5 be installed and running in the server system.

4. SYSTEM STUDY

4.1 INTRODUCTION TO JAVA:

Java is an object-oriented programming language with a built-in application programming interface (API) that can handle graphics and user interfaces and that can be used to create applications or applets. Because of its rich set of API's, similar to Macintosh and Windows, and its platform independence, Java can also be thought of as a platform in itself. Java also has standard libraries for doing mathematics.

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer

The inventors are Java wanted to design a language, which could offer solution to some of the problems encountered in modern programming. They wanted the language to be not only reliable, portable and distributed.

Although the above appears to be a list of buzzwords, they aptly describe the full potential of the language. These features have made Java the first application language of the World Wide Web. Java will also become the primer language for general-purpose stand-alone applications.

4.1.1 Properties of java:

- Compiled and interpreted
- Platform-Independent and Portable
- Robust and Secure
- Multithreading and Interactive
- Dynamic and Extensible

4.1.2 Compiled and Interpreted:

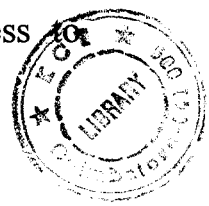
Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as bytecode instructions.

4.1.3 Platform Independent and Portable:

The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, anywhere anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet, which inter connects different kinds of systems worldwide. We can download a Java applet from a remote computer on to our local system via Internet an extension of the user's basic system providing practically unlimited number of accessible applets and applications.

4.1.4 Robust and Secure:

Security becomes an important issue for a language that is used for programming on Internet. Threat of virus and abuse of resources is everything. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.



4.1.5 Distributed:

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data programs. Java applications can open and access remote objects on Internet as easily as they can in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

4.1.6 Multithreaded:

Multithreaded means handling multiple tasks simultaneously. Java supports multithreading programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip time download an applet from a distance computer. This feature greatly improves the interactive performance of graphical applications.

4.1.7 Dynamic and Extensible:

Java is a dynamic language. Java is capable of dynamically linking in new class libraries methods and objects. Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program, depending on the response.

4.1.8 Advantages of JAVA

Java has gained enormous popularity since it first appeared. Its rapid ascension and wide acceptance can be traced to its design and programming features, particularly in its promise that we can write a program once, and run it anywhere. Java was chosen as the programming language for network computers (NC) and has been perceived as a universal front end for the enterprise database. As stated in Java language white paper by Sun Microsystems: "Java is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic."

4.2 INTRODUCTION TO WINPCAP

WinPcap is a free, public system for direct network access under Windows.

Most networking applications access the network through widely used system primitives, like sockets. This approach allows to easily transfer data on a network, because the OS copes with low level details (protocol handling, flow reassembly, etc.) and provides an interface similar to the one used to read and write on a file.

Sometimes however the 'easy way' is not enough, since some applications need a low level view in order to directly handle the network traffic. Therefore, they need raw access to the network, without the intermediation of entities like protocol stacks.

The purpose of WinPcap is to give this kind of access to Win32 applications; it provides facilities to:

- capture raw packets, both the ones destined to the machine where it's running and the ones exchanged by other hosts (on shared media)
- filter the packets according to user-specified rules before dispatching them to the application
- transmit raw packets to the network
- gather statistical values on the network traffic

This set of capabilities is obtained by means of a device driver, which is installed inside the networking portion of the Win32 kernels, plus a couple of DLLs.

All these features are exported through a powerful programming interface, easily exploitable by the applications and portable on different OSes.

WinPcap can be used by different kind of tools for network analysis, troubleshooting, security and monitoring. In particular, classical tools that rely on WinPcap are:

- network and protocol analyzers
- network monitors
- traffic loggers
- traffic generators
- user-level bridges and routers

- network intrusion detection systems (NIDS)
- network scanners
- security tools

WinPcap receives and sends the packets *independently* from the protocols of the host, like TCP-IP. This means that it isn't able to block, filter or manipulate the traffic generated by other programs on the same machine: it simply sniffs the packets that transit on the wire. Therefore, it cannot be used by applications like traffic shapers, QoS schedulers and personal firewalls.

WinPcap is an architecture for packet capture and network analysis for the Win32 platforms. It includes a kernel-level packet filter, a low-level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll).

Why we use the term "architecture" rather than "library"? Because packet capture is a low level mechanism that requires a strict interaction with the network adapter and with the operating system, in particular with its networking implementation, so a simple library is not sufficient.

The following figure shows the various components of WinPcap:

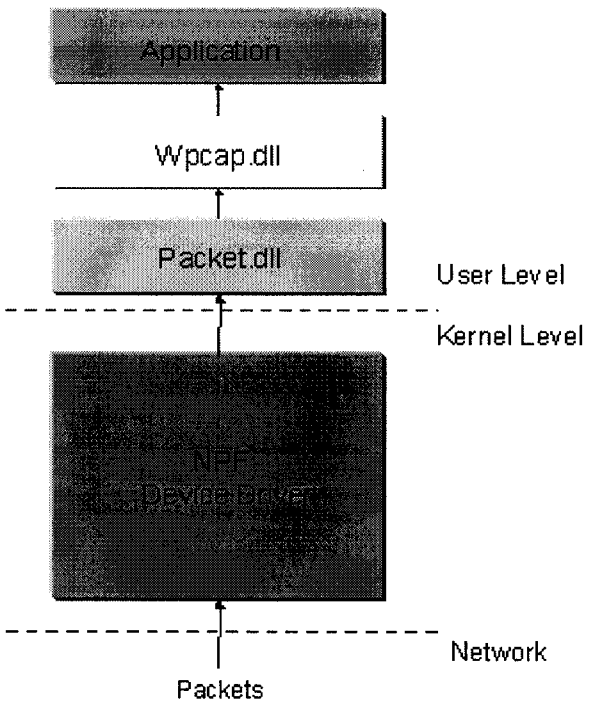


Figure 4.0 Main components of WinPcap.

First, a capture system needs to bypass the protocol stack in order to access the raw data transiting on the network. This requires a portion running inside the kernel of OS, interacting directly with the network interface drivers. This portion is very system dependent, and in our solution it is realized as a device driver, called Net group Packet Filter (NPF); we provide currently different versions of the driver for Windows 95, Windows 98, Windows ME, Windows NT 4, Windows 2000 and Windows XP. These drivers offer both basic features like packet capture and injection, as well as more advanced ones like a programmable filtering system and a monitoring engine. The first one can be used to

restrict a capture session to a subset of the network traffic (e.g. it is possible to capture only the ftp traffic generated by a particular host), the second one provides a powerful but simple to use mechanism to obtain statistics on the traffic (eg. it is possible to obtain the network load or the amount of data exchanged between two hosts).

Second, the capture system must export an interface that user-level applications will use to take advantage of the features provided by the kernel driver. WinPcap provides two different libraries: *packet.dll* and *wpcap.dll*.

The first one offers a low-level API that can be used to directly access the functions of the driver, with a programming interface independent from the Microsoft OS.

The second one exports a more powerful set of high level capture primitives that are compatible with libpcap, the well known UNIX capture library. These functions allow capturing packets in a way independent from the underlying network hardware and operating system.

4.2.1 NPF and NDIS:

Net group Packet Filter (NPF) is the kernel portion of WinPcap. NPF is the WinPcap component that does the hard work, processing the packets that transit on the network and exporting capture, injection and analysis capabilities to user-level.

NDIS (Network Driver Interface Specification) is a standard that defines the communication between a network adapter (or, better, the driver that manages it) and the protocol drivers (that implement for example TCP/IP). Main NDIS purpose is to act as a wrapper that allows protocol drivers to send and receive packets onto a network (LAN or WAN) without caring either the particular adapter or the particular Win32 operating system.

NDIS supports three types of network drivers:

1. **Network interface card or NIC drivers.** NIC drivers directly manage network interface cards, referred to as NICs. The NIC drivers interface directly to the hardware at their lower edge and at their upper edge present an interface to allow upper layers to send packets on the network, to handle interrupts, to reset the NIC, to halt the NIC and to query and set the operational characteristics of the driver. NIC drivers can be either miniports or legacy full NIC drivers.

    - Miniport drivers implement only the hardware-specific operations necessary to manage a NIC, including sending and receiving data on the NIC. Operations common to all lowest level NIC drivers, such as synchronization, is provided by NDIS. Miniports do not call operating system routines directly; their interface to the operatingsystem is NDIS.
    - A miniport does not keep track of bindings. It merely passes packets up to NDIS and NDIS makes sure that these packets are passed to the correct protocols.

- Full NIC drivers have been written to perform both hardware-specific operations and all the synchronization and queuing operations usually done by NDIS. Full NIC drivers, for instance, maintain their own binding information for indicating received data.

2. **Intermediate drivers**. Intermediate drivers interface between an upper-level driver such as a protocol driver and a miniport. To the upper-level driver, an intermediate driver looks like a miniport. To a miniport, the intermediate driver looks like a protocol driver. An intermediate protocol driver can layer on top of another intermediate driver although such layering could have a negative effect on system performance. A typical reason for developing an intermediate driver is to perform media translation between an existing legacy protocol driver and a miniport that manages a NIC for a new media type unknown to the protocol driver. For instance, an intermediate driver could translate from LAN protocol to ATM protocol. An intermediate driver cannot communicate with user-mode applications, but only with other NDIS drivers.
3. **Transport drivers or protocol drivers**. A protocol driver implements a network protocol stack such as IPX/SPX or TCP/IP, offering its services over one or more network interface cards. A protocol driver services application-layer clients at its upper edge and connects to one or more NIC driver(s) or intermediate NDIS driver(s) at its lower edge.

NPF is implemented as a protocol driver. This is not the best possible choice from the performance point of view, but allows reasonable independence from the MAC layer and as well as complete access to the raw traffic.

NPF is able to perform a number of different operations: capture, monitoring, dump to disk, packet injection.

4.2.2 Packet Capture

The most important operation of NPF is packet capture. During a capture, the driver sniffs the packets using a network interface and delivers them intact to the user-level applications.

The capture process relies on two main components:

- A packet filter that decides if an incoming packet has to be accepted and copied to the listening application. Most applications using NPF reject far more packets than those accepted, therefore a versatile and efficient packet filter is critical for good over-all performance. A packet filter is a function with boolean output that is applied to a packet. If the value of the function is true the capture driver copies the packet to the application; if it is false the packet is discarded. NPF packet filter is a bit more complex, because it determines not only if the packet should be kept, but also the amount of bytes to keep. The filtering system adopted by NPF derives from the **BSD Packet Filter** (BPF), a virtual processor able to execute filtering programs expressed in a pseudo-assembler and created at user level. The application takes a user-defined filter (e.g. "pick up all UDP packets") and, using

wpcap.dll, compiles them into a BPF program (e.g. "if the packet is IP and the *protocol type* field is equal to 17, then return true"). Then, the application uses the *BIOCSETF* IOCTL to inject the filter in the kernel. At this point, the program is executed for every incoming packet, and only the conformant packets are accepted. Unlike traditional solutions, NPF does not *interpret* the filters, but it *executes* them. For performance reasons, before using the filter NPF feeds it to a JIT compiler that translates it into a native 80x 86 functions. When a packet is captured, NPF calls this native function instead of invoking the filter interpreter, and this makes the process very fast. The concept behind this optimization is very similar to the one of Java jitters.

- A circular buffer to store the packets and avoid loss. A packet is stored in the buffer with a header that maintains information like the timestamp and the size of the packet. Moreover, alignment padding is inserted between the packets in order to speed-up the access to their data by the applications. Groups of packets can be copied with a single operation from the NPF buffer to the applications. This improves performances because it minimizes the number of reads. If the buffer is full when a new packet arrives, the packet is discarded and hence it's lost. Both kernel and user buffer can be changed at runtime for maximum versatility: packet.dll and wpcap.dll provide functions for this purpose.

The size of the user buffer is very important because it determines the *maximum* amount of data that can be copied from kernel space to user space within a single system call. On the other hand, it can be noticed that also the *minimum* amount of data that can be copied in a single call is

extremely important. In presence of a large value for this variable, the kernel waits for the arrival of several packets before copying the data to the user. This guarantees a low number of system calls, i.e. low processor usage, which is a good setting for applications like sniffers. On the other side, a small value means that the kernel will copy the packets as soon as the application is ready to receive them. This is excellent for real time applications (like, for example, ARP redirectors or bridges) that need the better responsiveness from the kernel. From this point of view, NPF has a configurable behavior that allows users to choose between best efficiency or best responsiveness (or any intermediate situation).

The wpcap library includes a couple of system calls that can be used both to set the timeout after which a read expires and the minimum amount of data that can be transferred to the application. By default, the read timeout is 1 second, and the minimum amount of data copied between the kernel and the application is 16K.

4.2.3 Packet injection

NPF allows writing raw packets to the network. To send data, a user-level application performs a WriteFile () system call on the NPF device file. The data is sent to the network as is, without encapsulating it in any protocol, therefore the application will have to build the various headers for each packet. The application usually does not need to generate the FCS because it is calculated by the network adapter hardware and it is attached automatically at the end of a packet before sending it to the network.

In normal situations, the sending rate of the packets to the network is not very high because of the need of a system call for each packet. For this reason, the possibility to send a single packet more than once with a single write system call has been added. The user-level application can set, with an IOCTL call (code pBIOCSWRITEREP), the number of times a single packet will be repeated: for example, if this value is set to 1000, every raw packet written by the application on the driver's device file will be sent 1000 times. This feature can be used to generate high speed traffic for testing purposes: the overload of context switches is no longer present, so performance is remarkably better.

4.2.4 Network monitoring

WinPcap offers a kernel-level programmable monitoring module, able to calculate simple statistics on the network traffic. The idea behind this module is shown in Figure 2: the statistics can be gathered without the need to copy the packets to the application that simply receives and displays the results obtained from the monitoring engine. This allows avoiding great part of the capture overhead in terms of memory and CPU clocks.

The monitoring engine is made of a *classifier* followed by a *counter*. The packets are classified using the filtering engine of NPF, which provides a configurable way to select a subset of the traffic. The data that pass the filter go to the counter, that keeps some variables like the number of packets and the amount of bytes accepted by the filter and updates them with the data of the incoming packets. These variables are passed to the user-level application at regular intervals whose period can

be configured by the user. No buffers are allocated at kernel and user level.

4.2.5 Dump to disk

The dump to disk capability can be used to save the network data to disk directly from kernel mode.

In traditional systems, the path covered by the packets that are saved to disk is the one followed by the black arrows in Figure 3: every packet is copied several times, and normally 4 buffers are allocated: the one of the capture driver, the one in the application that keeps the captured data, the one of the stdio functions (or similar) that are used by the application to write on file, and finally the one of the file system.

When the kernel-level traffic logging feature of NPF is enabled, the capture driver addresses the file system directly, hence the path covered by the packets is the one of the red dotted arrow: only two buffers and a single copy are necessary, the number of system call is drastically reduced, therefore the performance is considerably better.

Current implementation dumps to disk in the widely used libpcap format. It gives also the possibility to filter the traffic before the dump process in order to select the packet that will go to the disk.

4.2.6 Packet Driver API:

`Packet.dll` is a dynamic link library that offers a set of low level functions to:

- install, start and stop the NPF device driver
- sniff the network traffic
- send packets to the network
- obtain the list of the available network adapters
- retrieve various information about an adapter, like the description and the list of addresses and net masks
- query and set various low-level parameters of an adapter

There are two versions of packet.dll: the first one works in Windows 95/98/ME, the second in Windows NT/2000/XP.

Packet.dll was born to provide a layer granting access the low level functionalities of WinPcap in a system independent way. This library copes with the system dependent details (like managing the devices, interacting with the OS to handle the adapters, looking for the information in the registry and so on), and export an API that is the same on the different Win32 OSes. In this way, applications or libraries that use it can run without being recompiled under any Win32 operating system.

However, currently this portability does not apply to the whole packet.dll API: some of the most recent features, like for example kernel-mode dump, and are present only in the WinNTx version of WinPcap, therefore packet.dll for Win9x does not export them. On the other side, the NTx version is a superset of the 9x one, in other words all the function present in the Win9x version are present also in WinNTx.

The other important role of this library is the handling of the NPF driver. Packet.dll transparently installs and starts the driver when an application attempts to access an adapter. This avoids the manual installation of the driver through the control panel.

4.2.7 Remote capture

WinPcap 3.0 comes with Remote Capture capabilities. This is an highly experimental feature that allows to interact to a remote machine and capture packets that are being transmitted on the remote network.

This requires a **remote daemon** (called `rpcapd`) which performs the capture and sends data back and a **local client** that sends the appropriate commands and receives the captured data.

WinPcap 3.0 extends the standard WinPcap code in such a way that all WinPcap-based tools can exploit remote capture capabilities. For instance, the capability to interact with a remote daemon is added to the client software without any explicit modification to it. Vice versa, the remote daemon must be explicitly installed (and configured) on the remote machine.

4.2.8 Remote Capture Running Modes

The Remote Capture Protocol (RPCAP) can work in two modes:

Passive Mode (default): the client (e.g. a network sniffer) connects to the remote daemon, it sends them the appropriate commands, and it starts the capture.

Active Mode: the remote daemon tries to establish a connection toward the client (e.g. the network sniffer); then, the client sends the appropriate commands to the daemon and it starts the capture. This name is due to the fact that the daemon becomes *active* instead of *waiting* for new connections.

The Active Mode is useful in case the remote daemon is behind a firewall and it cannot receive connections from the external world. In this case, the daemon can be configured to establish the connection to a given host, which will have been configured in order to *wait* for that connection. After establishing the connection, the protocol continues its job in almost the same way in both Active and Passive Mode.

Analyzer has a set of commands (in the **Capture** menu) that allows you to accept a remote connection and then start the capture on the remote device. Currently, Analyzer is the only tool that is able to work in active mode, since it requires some modifications to the application code.

4.2.9 Configuring the Remote Daemon (rpcapd)

The Remote Daemon is a standard Win32 executable running either in console mode or as a service. The executable can be found in the `WinPcap` folder and it has the following syntax:

rpcapd [-b <address>] [-p <port>] [-6] [-l <host_list>] [-a <host, port>]
[-n] [-v] [-d] [-s <file>] [-f <file>]

The daemon can be compiled and it is actually working on Linux as well.

Here there is a brief description of the allowed commands:

Switch	Description
-b <address>	It sets the address the daemon has to bind to (either numeric or literal). Default: it binds to all local IPv4 and IPv6 addresses.
-p <port>	It sets the port the daemon has to bind to. Default: it binds to port 2002.
-4	It binds only to IPv4 addresses. Default: both IPv4 and IPv6 waiting sockets are used.
-l <host_list_file>	It specifies a file that keeps the list of the hosts which are allowed to connect to this daemon (if more than one, the file keeps them one per line). We suggest to use literal names (instead of numeric ones) in order to avoid problems with different address families (IPv4 and IPv6).
-n	It permits NULL authentication (usually used with '-l', that guarantees that only the allowed hosts can connect to the daemon). Default: the username/password authentication mechanism is required.
-a <host, port>	It forces the daemon to run in active mode and to connect to 'host' on port 'port'. This does not exclude that the daemon is still able to accept passive connections.
-v	It forces the daemon to run in active mode only (default: the daemon always accepts active connections, even if the '-a' switch is specified).

-d	Forces the daemon to run in background, i.e. as a daemon (UNIX only) or as a service (Win32 only). **Warning** (Win32): this switch is provided automatically when WinPcap installs this daemon into the Win32 services (control panel - administrative tools - services).
-s <file>	It saves the current configuration to file.
-f <file>	It loads the current configuration from file; all the switches specified from the command line are ignored and the file settings are used instead.
-h	It prints an help screen.

4.3 INTRODUCTION TO JPCAP

If you want to capture network packets in your Java program, you'll need a little help because no parts of the core Java API give access to low-level network data. However, Jpcap is a Java API that provides you with this access on Windows or UNIX systems.

Jpcap isn't a pure Java solution; it depends on the use of native libraries. On either Windows or UNIX, you must have the required third-party library, WinPcap or libpcap, respectively.

Jpcap uses an event model to allow you to process packets. To get started, you must first create a class that implements the interface jpcap.JpcapHandler.

In order to capture packets, you need to tell Jpcap which network device you want to listen with. The API provides the jpcap.Jpcap.getDeviceList () method for this purpose.

After choosing a device, you open it for listening by using the method Jpcap.openDevice (). The openDevice() method requires four arguments: the device name to be opened, the maximum number of bytes to read from the device at one time, a Boolean value specifying whether to put the device into promiscuous mode, and a timeout value that will be used if you later call the processPacket() method.

The openDevice () method returns a reference to a Jpcap object that will be used for capturing. Now that you have the Jpcap instance, you can start listening by calling either processPacket () or loopPacket (). Both of the methods take two arguments: The maximum number of packets to capture can be -1 to indicate no limit and an instance of a class that implements JpcapHandler.

To execute the class, you must make sure that the virtual machine can find the Jpcap native library.

5. DOCUMENT DESIGN

5.1 SOFTWARE DESIGN

5.1.1 INTRODUCTION:

A Software design is a model of a real world system that has many participating entities and relationships. This design is used in a number of different ways. It acts as a basis for detailed implementation. It serves as a communication medium between the designers of the subsystems; it provides information to system maintainers about the original intentions of the system designers.

5.1.2 UML DIAGRAMS

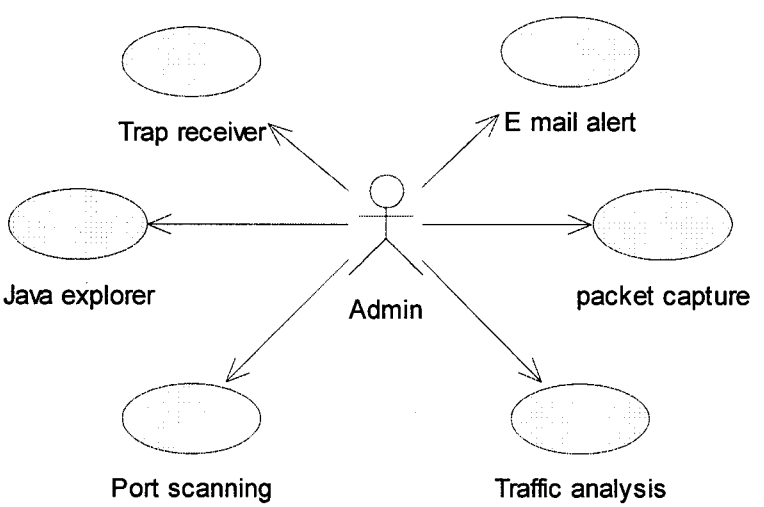


Figure 5.0 Use-Case Diagram

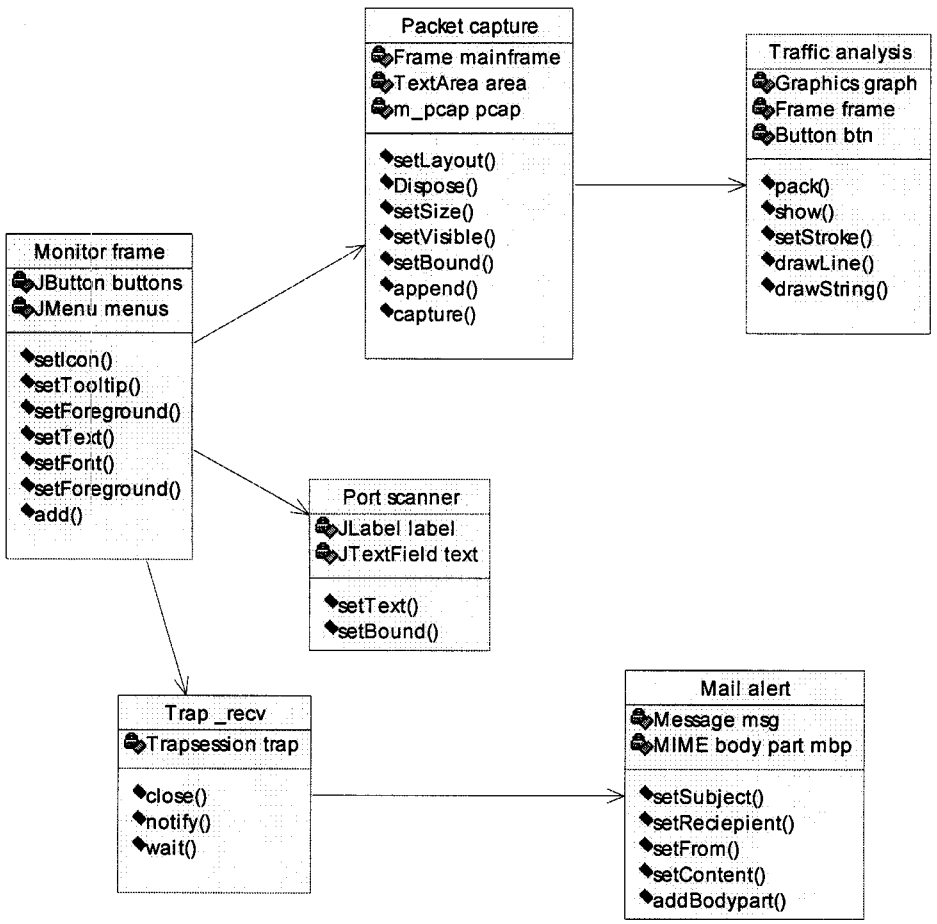


Figure 5.1Class Diagram

5.2 DATABASE DESIGN:

The database consists of the Packet details table.

5.2.1 Packet Details:

Field Name	Field Type	Field Length	Field Description
Packet_type	Text	20	Type of the arrived packet
Src_address	Text	20	Source address of the packet
Dest_address	Text	50	Destination address of the packet
Time	Time	-	Arrival time of the packet
Src_port	Number	4	Destination port of the packet
Dest_port	Number	4	Source port of the packet

5.3 INPUT/OUTPUT DESIGN FORMATS:

5.3.1 Packet Capture

Input: Device address

Output: Packets arrived at the specified device address are displayed

5.3.2 Port Scanning

Input: Port number and the host address

Output: The status of the ports scanned is displayed.

6. SOFTWARE IMPLEMENTATION MODEL

6.1 Module Description:

There are 6 modules in this system. They are:

- Packet Capture
- Traffic Analysis
- Port Scanning
- Explorer
- Trap receiver
- E-mail alert

6.1.1 Packet Capture:

The packets that are passing through the network are captured and the details of the packets like type, source address, destination address, source port, destination port are extracted and logged for the administrator to view the load.

6.1.2 Traffic Analysis:

The details of the captured packets are logged in a database.The load in the network is viewed by means of a graph with the time in X-axis and the no.of packets in Y-axis.The traffic statistics is also viewed in the form of a table with the total no.of packets arrived at a particular time.

6.1.3 Port Scanning:

The host IP address and the range of the ports to be scanned is given as the input to this module.It scans all the ports in the specified range and displays the status of the scanned port.

6.1.4 Explorer:

The explorer is a file browser that gives a tree view of all the drives and the files and folders present in the drives.

6.1.5 Trap Receiver:

It can receive trap messages from SNMP capable devices and also can transfer the trap message to desired address via email.

6.1.6 E-Mail Alert:

When an error occurs while monitoring the network like the destination unreachable or timeout, an error message can be sent via email to the desired address destination, thus intimating the occurrence of error in the network.

7. SYSTEM TESTING

7.1 PRODUCT TESTING

Testing is done to detect the errors in the software. This implies not only to the coding phase but to uncover errors introduced in all the previous phases.

The following are the types of tests that were performed.

7.1.1 Unit testing:

Each and every module is tested separately to check if its intended functionality is met. Some unit testing performed are,

- Checking the proper working of packet capturing module.
- Checking for proper updation in the database of the captured packets.
- Ensuring that the SNMP trap messages are properly received and negative trap messages are sent to the network administrator.
- Verify the graph representing traffic load against the number of packets captured in a particular time.

7.1.2 Integration testing:

It is the testing performed to detect errors on interconnection between modules. Here, the module packet capture is integrated with data logging and graphical representation and tested to ensure synchronization.

7.1.3 System testing:

The system is tested against the system requirements to see if all the requirements are met and if the system performs as per the specified requirements. The system is tested as a whole to check for its functionality.

7.1.4 Validation testing:

This test is done to check for the validity of the entered input.The input here is the device address, host name and range of ports. Invalid characters and symbols are recognized and properly handled.

8. FUTURE ENHANCEMENTS

As this project J-NetView is completely done in java, it provides greater flexibility and reusability. Any kind of change can be made to the project without any major change in the underlying functionalities. The classes are defined clearly with necessary access parameters so that they can be modified easily and any additional class can be added incase of any additional functionality.

In the future, this application can be extended to monitor many interrelated networks. Instead of static network monitoring, modules that detect and monitor the adaptive nature of the network can be added to the existing system.

Another possible enhancement would be to present the charts and graphs with three dimensional diagrams. This would give more clarity and would become more comprehensive.

Currently the network traffic is being analyzed with the packet flow in the network. The low level parameters such as Host congestions, Server load, etc can be taken into account and a complete network monitoring suite can be developed with an easy user interface.

9. CONCLUSION

The project "J-NetView"is completed and tested. This system is developed in such a way that it is user friendly and hence the network administrator is very much benefited.

This is suitable for a typical Local Area Network that needs continuous monitoring and performance evaluation.

The project J-Net View offers the following advantages:

- Allows the administrator to constantly monitor the dataflow in the network.
- Network traffic is presented in graphs and charts so that its performance can be understood easily.
- Quite flexible in nature allowing addition of newer functions more easily.
- User friendly interface which requires no training.

```
setIconImage(getIconImage());
jButton1.setForeground(new java.awt.Color(0, 51, 255));
jButton1.setIcon(new javax.swing.ImageIcon("z:\\final\\network.gif"));
jButton1.setToolTipText("JNetView");
jButton1.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt){
jButton1ActionPerformed(evt);}});
getContentPane().add(jButton1);
jButton1.setBounds(40, 30, 370, 330);
jMenuBar1.setName("network");
jMenu1.setForeground(new java.awt.Color(51, 0, 51));
jMenu1.setText("Network");
jMenu1.setFont(new java.awt.Font("Comic Sans MS", 1, 16));
jMenu1.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
jMenu1ActionPerformed(evt);}});
jMenu4.setBorder(new javax.swing.border.EtchedBorder());
jMenu4.setForeground(new java.awt.Color(0, 51, 51));
jMenu4.setText("Monitor");
jMenu4.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem3.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem3.setText("Port Scanner");
jMenuItem3.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem3ActionPerformed(evt);}});
jMenu4.add(jMenuItem3);
jMenuItem4.setForeground(new java.awt.Color(0, 51, 51));
```

```
jMenuItem4.setText("SNMP");
jMenuItem4.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem4ActionPerformed(evt);}});
jMenu4.add(jMenuItem4);
jMenuItem7.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem7.setText("PacketCapture");
jMenuItem7.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem7ActionPerformed(evt);}});
jMenu4.add(jMenuItem7);
jMenu1.add(jMenu4);
jMenuItem2.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem2.setText("Explore");
jMenuItem2.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem2.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem2ActionPerformed(evt);}});
jMenu1.add(jMenuItem2);
jMenuItem1.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem1.setText("Exit");
jMenuItem1.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem1.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem1ActionPerformed(evt);}});
jMenu1.add(jMenuItem1);
jMenuBar1.add(jMenu1);
```

```
jMenu1.getAccessibleContext().setAccessibleParent(jMenuItem9);
jMenu2.setForeground(new java.awt.Color(51, 0, 51));
jMenu2.setText("Reports");
jMenu2.setFont(new java.awt.Font("Comic Sans MS", 1, 16));
jMenuItem5.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem5.setText("Graph");
jMenuItem5.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem5.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem5ActionPerformed(evt);}});
jMenu2.add(jMenuItem5);
jMenuItem6.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem6.setText("Traffic Statistics");
jMenuItem6.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem6.addActionListener(new java.awt.event.ActionListener(){
public void actionPerformed(java.awt.event.ActionEvent evt) {
jMenuItem6ActionPerformed(evt);}});
jMenu2.add(jMenuItem6);
jMenuBar1.add(jMenu2);
jMenu2.getAccessibleContext().setAccessibleParent(jMenuItem6);
jMenu3.setForeground(new java.awt.Color(51, 0, 51));
jMenu3.setText("Settings");
jMenu3.setFont(new java.awt.Font("Comic Sans MS", 1, 16));
jMenuItem9.setForeground(new java.awt.Color(0, 51, 51));
jMenuItem9.setText("Email-options");
jMenuItem9.setFont(new java.awt.Font("Comic Sans MS", 1, 15));
jMenuItem9.addActionListener(new java.awt.event.ActionListener() {
```

```
public void actionPerformed(java.awt.event.ActionEvent evt){
jMenuItem9ActionPerformed(evt);}});
jMenu3.add(jMenuItem9);
jMenuBar1.add(jMenu3);
setJMenuBar(jMenuBar1);
java.awt.DimensionscreenSize=
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
setBounds((screenSize.width-500)/2,(screenSize.height-500)/2,500,
500);}
private void jMenu1ActionPerformed(java.awt.event.ActionEvent evt) {}
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent
evt){
System.exit(1);}
private void jMenuItem7ActionPerformed(java.awt.event.ActionEvent
evt){
try{
pack=new packetcapscr();
pack.show();}
catch(Exception e){}}
private void jMenuItem6ActionPerformed(java.awt.event.ActionEvent
evt){
detail=new detailed();
detail.show();}
private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent
evt){
try{
agent=new snmpagent("192.168.100.42",1,"11");
```

```
}catch(Exception e){}}
private void jMenuItem9ActionPerformed(java.awt.event.ActionEvent
evt)
{
send=new Sendingmsg();
send.Sendingmsg();}
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent
evt)
{
FilenameFilter filter = new FilenameFilter(){
public boolean accept(File dir, String name) {
return (name.charAt(0) != '.');}};
filenode=new FileNode("my computer",filter);
browser=new FileBrowser(filenode);
browser.getFileBrowser("my computer");
JFrame frame=new JFrame("file browser");
JScrollPane scrollpane=new JScrollPane(browser.getFileBrowser("my
computer"));
frame.getContentPane().add(scrollpane,BorderLayout.CENTER);
frame.setSize(300, 600);
frame.setVisible(true); }
private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent
evt)
{
try {
ta=new TrafficAnalysis();
ta.start();
```

```
ta.show();
setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON
_CLOSE);
}catch(Exception e){}}
private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent
evt)
{
try {
portscan=new portscanner2();
portscan.show();
setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON
_CLOSE);
} catch(Exception e) {}}
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{}
public static void main(String args[]){
java.awt.EventQueue.invokeLater(new Runnable(){
public void run() {
new monitorframe().setVisible(true);}});}
```

PACKETCAPTURE. JAVA

```
public class packetcap1 extends java.awt.Panel{
public packetcap1() throws Exception
{
Pnl = new java.awt.Panel();
add(Pnl);
Pnl.setLayout(null);
```

```
f=new Frame();
f.setLayout(null);
f.setSize(500, 300);
f.setVisible(true);
f.addWindowListener(new java.awt.event.WindowAdapter(){
public void windowClosing(java.awt.event.WindowEvent evt) {
f.dispose(); }});P
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url="jdbc:odbc:packet";
Connection con =DriverManager.getConnection(url,"", "");
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);}
catch(ClassNotFoundException e){
System.out.println("exceptionindriver");}
      catch(SQLException e){
System.out.println("exception in sql");}
m_pcap = new PacketCapture();
m_device = m_pcap.findDevice();
m_pcap.open("\\Device\\NPF_{E9E7B594-DF2C-4AA8-9EAB-
B07A9CFB80DA}", true);
m_pcap.setFilter(FILTER, true);
m_pcap.addPacketListener(new PacketHandler());
m_pcap.capture(PACKET_COUNT);}
public static void main(String[] args) {
try {
packetcap1 pack = new packetcap1();}
```

```
catch(SQLException e){
System.out.println("sql error");}
TextArea1.append(sa+"\t"+da+"\t"          +hr+":"+mi+":"+se+"          "
+key+"\t"+dp+" "+sp+"\n");}
else if(packet instanceof UDPPacket) {
m_counter++;
UDPPacket udp=(UDPPacket)packet;
int dp=udp.getDestinationPort();
int sp=udp.getSourcePort();
String sa=udp.getSourceAddress();
String da=udp.getDestinationAddress();
String type = packet.getClass().getName();
String key="";
Date d=new Date();
int hr=d.getHours();
int mi=d.getMinutes();
int se=d.getSeconds();
String timeval=hr+":"+mi+":"+se;
StringTokenizer st=new StringTokenizer(type,".");
while(st.hasMoreTokens()){
key=st.nextToken();}
try{
query="INSERT                    INTO                    PACKET
VALUES("+"'"+sa+"'"+","+"'"+da+"'"+","+"'"+timeval+"'"+","+"'"+key+
"'"+","+dp+","+sp+")";
stmt.executeUpdate(query);}
catch(SQLException e){
```

```
System.out.println("sql error");}
TextArea1.append(sa+"\t"+da+"\t"          +hr+":"+mi+":"+se+"          "
+key+"\t"+dp+" "+sp+"\n");}
else if(packet instanceof ARPPacket) {
m_counter++;
ARPPacket arp=(ARPPacket)packet;
String sa=arp.getSourceProtoAddress();
String da=arp.getDestinationProtoAddress();
String type = packet.getClass().getName();
String key="";
Date d=new Date();
int hr=d.getHours();
int mi=d.getMinutes();
int se=d.getSeconds();
StringTokenizer st=new StringTokenizer(type,".");
while(st.hasMoreTokens()){
key=st.nextToken();}
TextArea1.append(sa+"\t"+da+"\t"          +hr+":"+mi+":"+se+"          "
+key+"\t"+type+"\n");}
else if(packet instanceof ICMPPacket) {
m_counter++;
ICMPPacket icmp=(ICMPPacket)packet;
String sa=icmp.getSourceAddress();
String da=icmp.getDestinationAddress();
String type = packet.getClass().getName();
String key="";
Date d=new Date();
```

```
int hr=d.getHours();
int mi=d.getMinutes();
int se=d.getSeconds();
StringTokenizer st=new StringTokenizer(type,".");
while(st.hasMoreTokens()){
key=st.nextToken();}
TextArea1.append(sa+"\t"+da+"\t"          +hr+":"+mi+":"+se+"          "
+key+"\t"+type);}
else if(packet instanceof IGMPPacket) {
m_counter++;
IGMPPacket igmp=(IGMPPacket)packet;
String sa=igmp.getSourceAddress();
String da=igmp.getDestinationAddress();
String type = packet.getClass().getName();
String key="";
Date d=new Date();
int hr=d.getHours();
int mi=d.getMinutes();
int se=d.getSeconds();
StringTokenizer st=new StringTokenizer(type,".");
while(st.hasMoreTokens()){
key=st.nextToken();}
TextArea1.append(sa+"\t"+da+"\t"          +hr+":"+mi+":"+se+"          "
+key+"\t"+type);}}
catch(Exception e) {
e.printStackTrace();}}}
public static class Reminder{
```

```
Timer timer;
public Reminder(int seconds) {
timer = new Timer(true);
timer.schedule(new RemindTask(), seconds*1000);}
class RemindTask extends TimerTask{
public void run(){
try{
System.out.println("No.of packets:"+m_counter);}
catch(Exception e){
e.printStackTrace();}}}}}
```

TRAFFICANALYSIS. JAVA

```
public class TrafficAnalysis extends Applet {
public TrafficAnalysis(){
int index1=0,minutes1=0,seconds1=0;
String timesub;
setBackground(Color.white);
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String url="jdbc:odbc:packet";
Connection con =DriverManager.getConnection(url,"", "");
stmt =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
query="select * from packet";
rs=stmt.executeQuery(query);
rs.first();
```

```
test=rs.getString(1);
while(index!=20) {
source=rs.getString(1);
destination=rs.getString(2);
sport=rs.getString(5);
dport=rs.getString(6);
timeval=rs.getString(3);
timesub=timeval.substring(3,5);
minutes=Integer.parseInt(timeval.substring(3, 5))*60;
if(index==0){
minutes1=minutes;}
if(minutes1!=minutes){
seconds1=60-seconds; }
seconds=Integer.parseInt(timeval.substring(6,8));
if(minutes1!=minutes){
minutes1=minutes;
totminutes=(minutes+seconds)/60;
if(index==0){
initialtime=seconds;
pcktcount=0;}
if((seconds-initialtime)<10){
pcktcount++;}
if(initialtime!=seconds){
String val=null;
Object valobj;
initialtime=seconds;
val=val.valueOf(pcktcount);
```

```
g.setColor(Color.red);
for(int j=0;j<=5;j++) {
g.setColor(Color.blue);
g.drawLine(35,163-j*20,30,163-j*20);
g.setColor(Color.red);
g.drawString(java.lang.Integer.toString(10*j),10,165-j*20); }
for(int j=0;j<=6;j++) {
g.setColor(Color.blue);
g.drawLine(35+j*50,164,35+j*50,169);
g.setColor(Color.red);
g.drawString(java.lang.Integer.toString(10*j),35+j*50-6,185); }
g.setColor(Color.black);
f = new Font("Helvetica",Font.ROMAN_BASELINE,15); g.setFont(f);
g.setColor(Color.black);
g.drawLine(37,115,335,115);
for(int j=1;j<6;j++){
g.drawLine(35+j*50,65,35+j*50,165);
int x1=0,x2=0,y1=0,y2=0,timervalue;
String timestring=null;
for(int i=0;i<5;i++){
g.setColor(Color.black);
if(i==0){
x1=35;
y1=163;}
packet=packets[i];
time=times[i];
x2=x1+time;
```

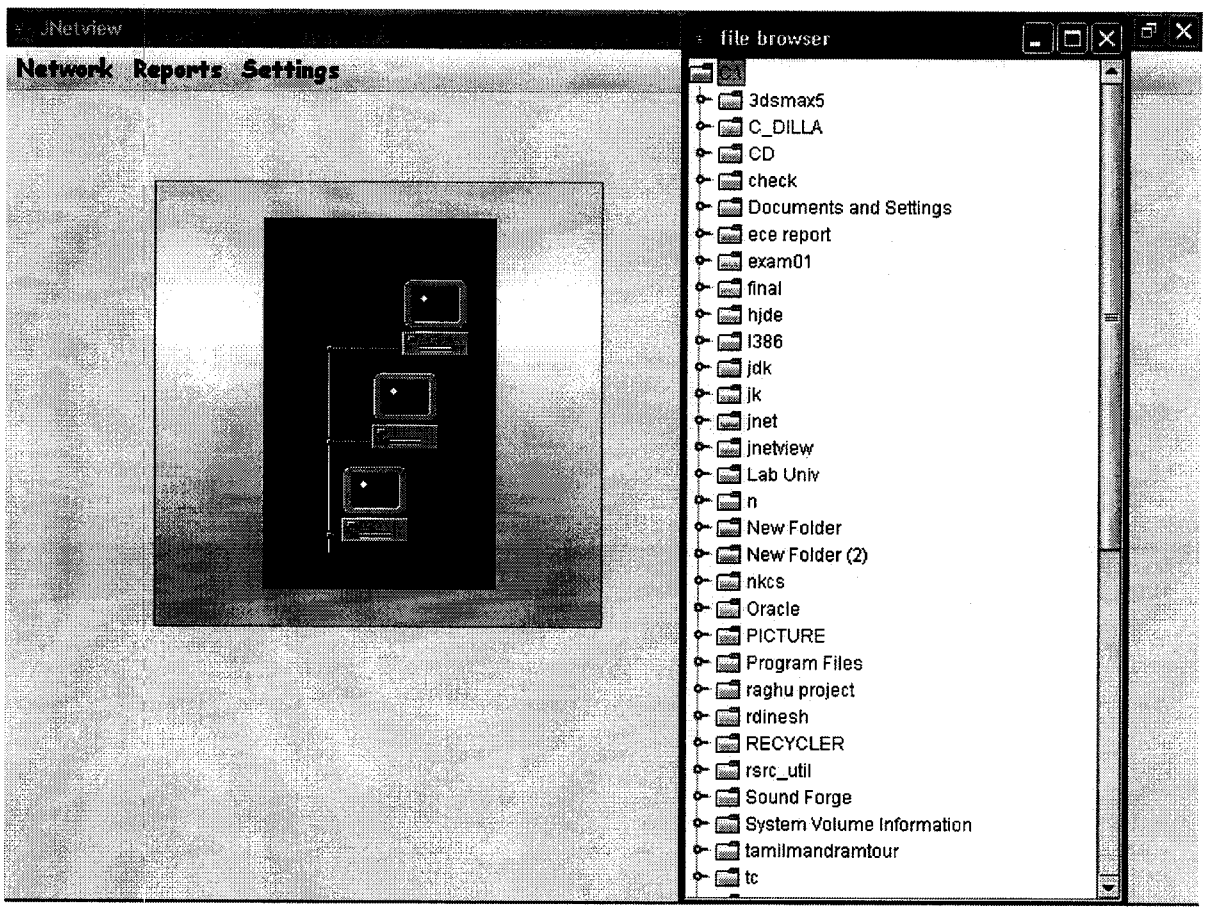
```
peer.setTimeout(PARAM_TIMEOUT);
SnmpParameters parameters = peer.getParameters();
parameters.setReadCommunity(PARAM_COMMUNITY);
m_session = new SnmpSession(peer);
m_session.setDefaultHandler(this);}
catch (Exception e) {}}
m_session.close();}
public void sendV1Trap(String message) throws Exception{
try{
SnmpPeer peer = m_session.getPeer();
SnmpParameters parameters = peer.getParameters();
parameters.setVersion(SnmpSMI.SNMPV1);
SnmpPduTrap trapPdu = new SnmpPduTrap();
trapPdu.setAgentAddress(new
SnmpIPAddress(InetAddress.getLocalHost().getAddress()));
trapPdu.setGeneric(PARAM_GENERIC_TYPE);
trapPdu.setSpecific(PARAM_SPECIFIC_TYPE);
trapPdu.addVarBind(newSnmpVarBind(new SnmpObjectId(m_trapOid),
new SnmpOctetString(message.getBytes())));
m_session.send(trapPdu);}
catch (Exception e){}}
public void sendV2Trap(String message) {
try {
SnmpPeer peer = m_session.getPeer();
SnmpParameters parameters = peer.getParameters();
parameters.setVersion(SnmpSMI.SNMPV2);
```

```
SnmpPduRequest trapPdu = new
SnmpPduRequest(SnmpPduPacket.V2TRAP);
trapPdu.addVarBind(new SnmpVarBind(new
SnmpObjectId(m_trapOid),
new SnmpOctetString(message.getBytes())));
m_session.send(trapPdu);}
catch (Exception e) {}}
public void snmpInternalError(SnmpSession session, int err, SnmpSyntax
pdu) {
System.out.println("SNMP internal error, code: " + err);}
public void snmpTimeoutError(SnmpSession session, SnmpSyntax pdu)
{
System.out.println("SNMP timeout");}
public void snmpReceivedPdu(SnmpSession session, int cmd,
SnmpPduPacket pdu) {}}
```

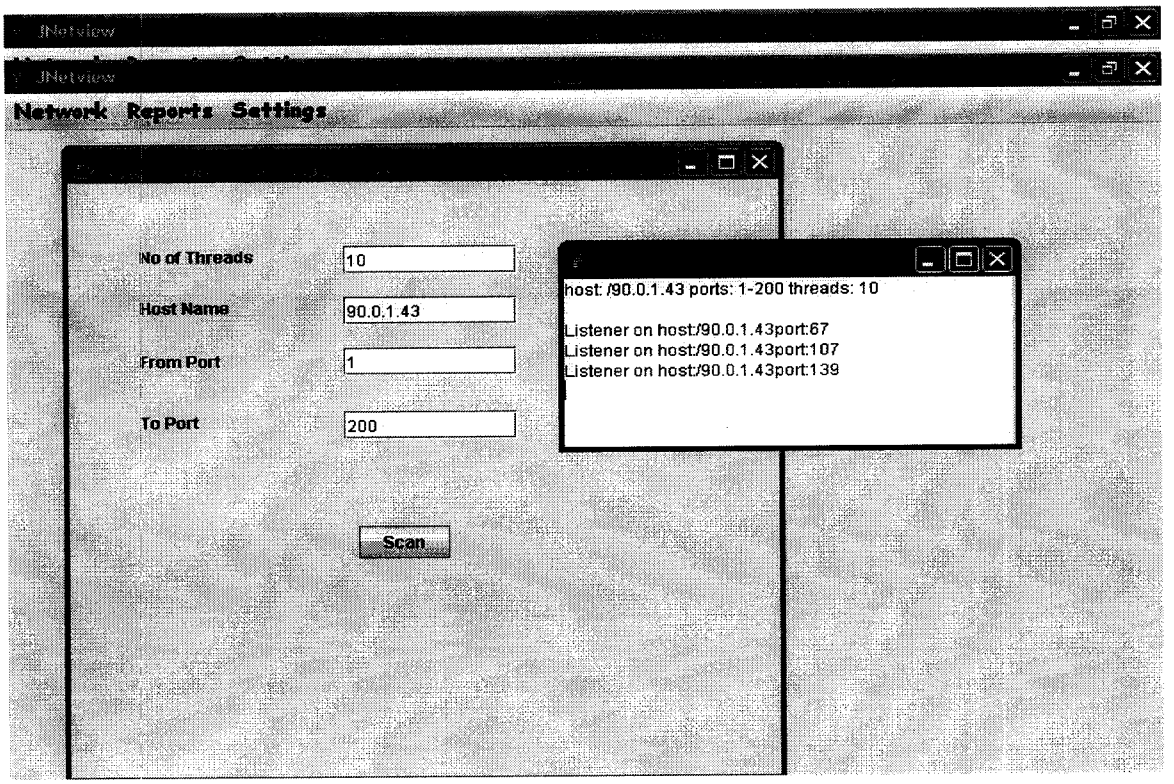
APPENDIX B

SCREENSHOTS

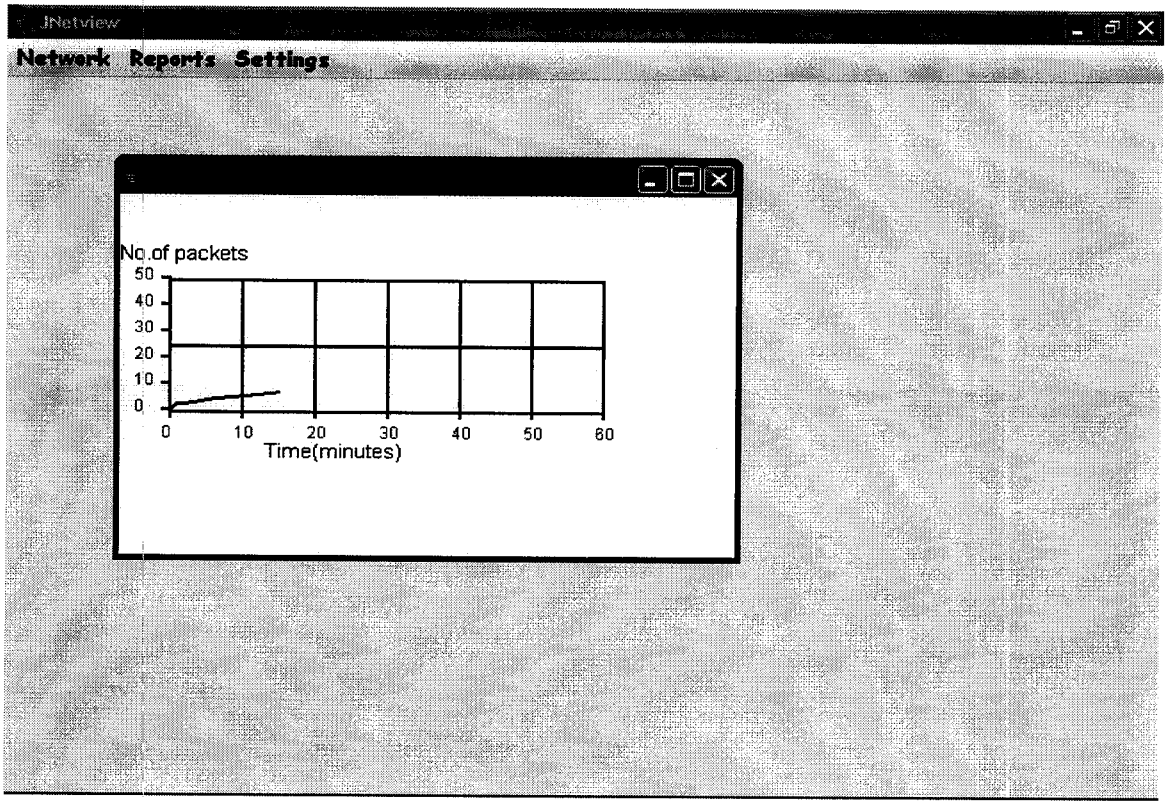
FILE BROWSER:



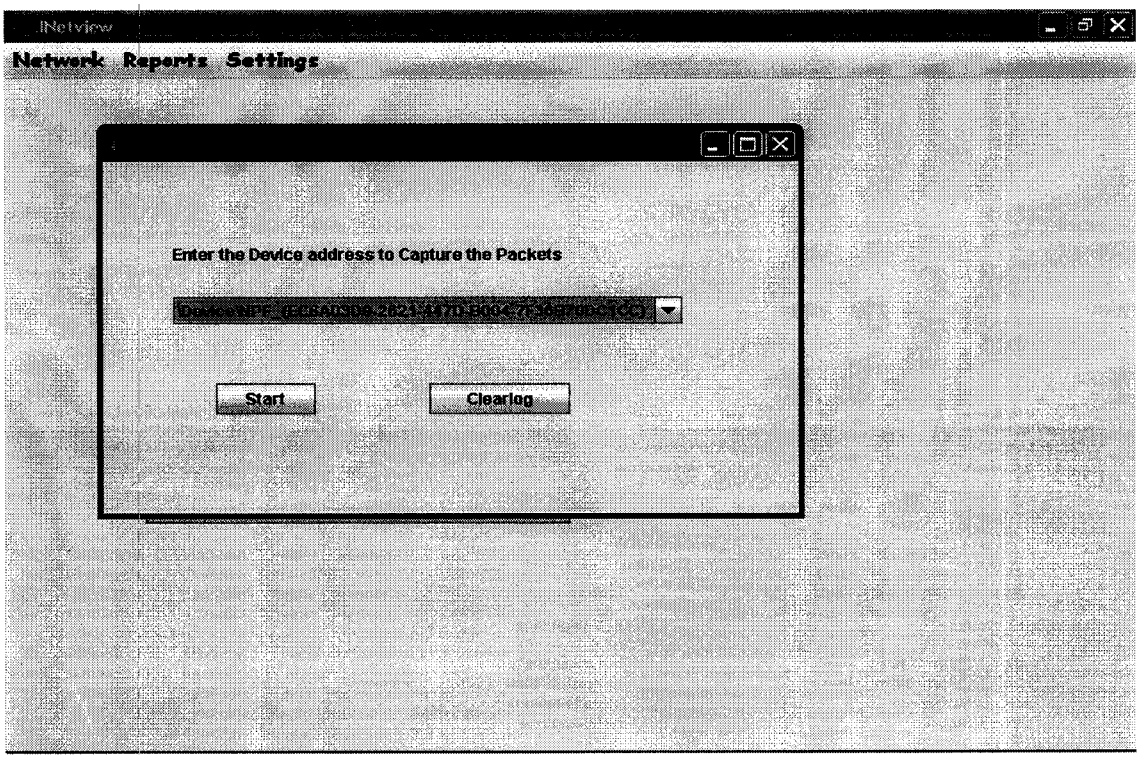
PORT SCANNING:



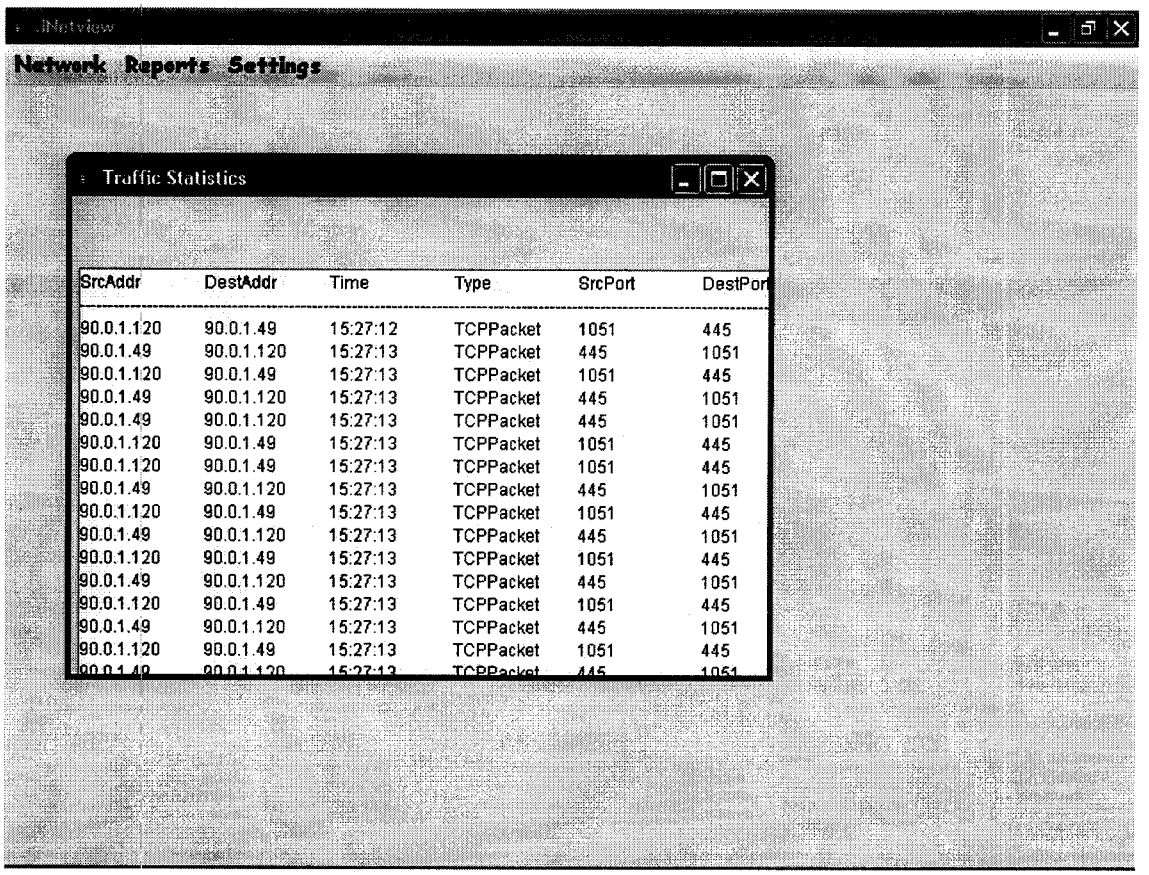
GRAPH:


JNetview
Network Reports Settings
No.of packets
50
40
30
20
10
0
0
10
20
30
40
50
60
Time(minutes)

PACKET CAPTURE:



TRAFFIC STATISTICS:



10. REFERENCES

1. Herbert Schldit, "JAVA 2 Complete Reference", 2nd Edition, McGraw-Hill Publishers.

2. Joseph L.Weber, "Using JAVA 2 Platform Specified Edition",Fourth Edition,McGraw-Hill Publishers.

3. Website: http://java.sun.com/j2se/1.5/docs/index.html

4. Website:http://java/swing.html