IPv6 Network Monitoring Tool

by

Yogita Kanesin

Dissertation submitted in partial fulfilment of
the requirements for the
Bachelor of Technology (Hons)
(Information Systems)

JUNE 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t
TK
5105.5
.Y54
2004

1. Computer networks -- Management
2. IT/IS -- Thesis

CERTIFICATION OF APPROVAL

IPv6 Network Monitoring Tool

By

Yogita Kanesin

A project dissertation submitted to the
Information Systems Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirements for the
BACHELOR OF TECHNOLOGY (Hons)
(INFORMATION SYSTEMS)

Approved by,

[illegible]

(Miss. Rozana Kasbon)

UNIVERSITI TEKNOLOGI PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan
June 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

(YOGITA KANESIN)

ABSTRACT

IPv6 is a new version of the internetworking protocol designed to address the scalability and service shortcomings of the current standard, IPv4.Unfortunately, IPv4 and IPv6 are not directly compatible, so programs and systems designed to one standard can not communicate with those designed to the other. Consequently, it is necessary to develop smooth transition mechanisms that enable applications to continue working while the network is being upgraded. In this paper the author presents the design and implementation of a network monitoring tool for the latest Internet Protocol; IPv6 which is designed for Microsoft Windows platform. The development of network has increased the need to monitor the nodes that is operating across the same network. The network monitoring tool aims to capture and analyze IP related packets (IPv6 packets) before executing report on the results found.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank those who have helped make this project a success. First and foremost, I would like to acknowledge God for it is through Him that I garner the strength to proceed during times of difficulty. I would also like to thank my parents for their unwavering support and encouragement. They have been and always will be there for me through thick and thin and for that I will be eternally grateful. I would also like to extend my greatest gratitude to my supervisor, Miss Rozana Kasbon, for her guidance and support throughout the duration of this project. Coupled with your patience and understanding, I have managed to accomplish this task with minimum difficulty. Finally, I take this opportunity to thank all those who have helped me in one way or another in completing this task. Although I am unable to list down all your names, your assistance is greatly appreciated. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

BPF	BSD Packet Filter
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ICMP	Internet Protocol Control Message
ICMPv6	Internet Protocol Control Message version 6
LAN	Local Area Network
MAC	Medium Access Layer
NDIS	Network Driver Interface Specification
NPF	Netgroup Packet Filtering
OS	Operating System
OSI	Open Systems Interconnect
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network

CHAPTER 1
INTRODUCTION

1.1 Background of Study

IETF, Internet Engineering Task Force, started looking for a successor to IPv4 in late 1990 when projections indicated that the address field of IPv4 would become a limiting resource. In 1993 IETF investigated several proposals a described the technical criteria for choosing the successor. In January 1995 was published and it describes a recommendation for the Internet Protocol Next Generation, IPng, as IPv6 was called back then. A specification for IPv6 was published in the end of 1995.

IPv6 (Internet Protocol version 6) is a new version of the internetworking protocol designed to address the scalability and service shortcomings of the current standard, IPv4 (Internet Protocol version 4). The current internetworking protocol, IPv4 eventually will be unable to adequately support additional nodes or the requirements of new applications. IPv4's 32-bit address was generous when it was first introduced, but the addresses are running out fast. This shortage has caused serious difficulties particularly in companies- that have gone ahead and allocated made-up addresses for their internal networks. This works of course until the company decides that it needs to be connected to the global Internet, at which point some unpleasant things can start happening. If those made-up IP addresses are unallocated, then the chances are that the Internet connection would not be problematic, until of course they are allocated. By expanding the address space we make the allocation of legal addresses easier, and remove the need for network managers to make up addresses to new host and workstations.

IPv6 is a new network protocol that features improved scalability and routing, security, ease-of-configuration, and higher performance compared to IPv4. IPv6 increases the IP addresses from 32 bits to 128 bits to support more levels of the addressing hierarchy, a much greater number of addressable nodes, and simpler auto-configuration.

The implementation of IPv6 over the network has brought some major changes in the network environment. These features have made IPv6 more robust and convenient. These features include;

- **Stateless and stateful Configuration –** IPv6 supports both stateful and stateless address configurations. IPv6 will work with or without a DHCP server. With stateless address configuration, hosts on a link automatically configure themselves with IPv6 addresses for the link (called link-local addresses) and with the addresses derived from prefixes advertised by local routers. Even in the absence of a router, hosts on the same link can automatically configure themselves with link-local addresses and communicate without manual configuration.

- **Packet Authentication is mandatory** – Security will be improved as the IP stack now natively supports extensions for authentication, data-integrity and confidentiality (encryption).

- **Streamlined IPv6 headers –** IPv6 header has a new format that is designed to keep header overhead to a minimum. This format is achieved by moving both nonessential fields and option fields to extension headers that are placed after the IPv6 header. The streamlined IPv6 header provides more efficient processing at intermediate router.

- **Built-in Security** – Support for IPSec is an IPv6 protocol suite requirement. This requirement provides a standards-based solution for network security and promotes interoperability between different IPv6 implementations.

- **Real-Time Performance-** IPv6 offers a packet prioritization feature that provides the real-time and near real-time applications an improved response time. Consequently, IPv6 will become the protocol of choice for those applications.

- **Better Support for QoS (Quality of Standard)** – New fields in the IPv6 header define how traffic is handled and identified. Traffic identification using a flow label field in the IPv6 header allows routers to identify and provide special handling for packets belonging to a flow, which is a series of packets between a source and destination. Because traffic is identified in the IPv6 header, support for QoS can be achieved even when the packet payload is encrypted through IPSec.

1.2 Problem Statement

1.2.1 Problem Identification

Most of the network monitoring tools designed is available for IPv4 only. As IPv6 will be deployed in the near future, current applications used such as the network monitoring tool for IPv4 is not directly compatible for IPv6. Thus, a network monitoring tool embedded with functions to capture IPv6 packets in the network.

1.2.2 Significant of the Project

In the long term, the main goal of IPv6 is to replace IPv4 due to the deficiency in this protocol designed in the 1970. During this transition time, IPv6 services must be deployed in the Internet for organisations to slowly but steadily move their network to IPv6 protocol. Thus, this application will be usable to the network administrators to capture and analyse all the IPv6 packets in a network as well for monitoring purposes.

1.3 Objectives and Scope of Study

1.3.1 Objectives

The objectives that are to be achieved by the end of this project:

1. To develop an agent that captures IPv6 packets in the network.
2. To develop an agent that analyses and categorizes the captured packets.

1.3.2 Scope of Study

The scope of study for this project will cover on the research on IPv6 packet capturing, socket programming, analyzing and categorizing the captured packets on the network. Study will also be carried out on how to collaborate all the resources to develop a functioning network monitoring tool for IPv6. This project is relevant to be carried out as IPv6 will be deployed in a few years time, we should be ready to accept and use it. Thus in a way of preparation to deploy IPv6 in the network field, it is better to be ready with appropriate application such as the network monitoring tool. This monitoring tool is similar to other monitoring tools but is expanded to read IPv6 packets on the network. Since IPv6 uses long address representation and stateless address configuration method, this tool is essential to monitor the number of nodes (computers) on a network and the source and destination of the sent packets.

CHAPTER 2
LITERATURE REVIEW AND THEORY

"Years ago, nay Sayers claimed that the IP address system would soon run out of addresses and that we would be stuck in an Internet Protocol version four (IPv4) world, teetering towards disaster. The only way out was to migrate to the next-generation address paradigm, IPv6. But few have made that migration, particularly in the U.S., and the sky has yet to fall, thanks to tricks such as Network Address Translation (NAT), which helps organizations conserve IP addresses. But now more and more IPv6-compatible products are hitting the market, sparking more interest in the technology." (Jim Rendon, News Writer, 29 Dec 2003)

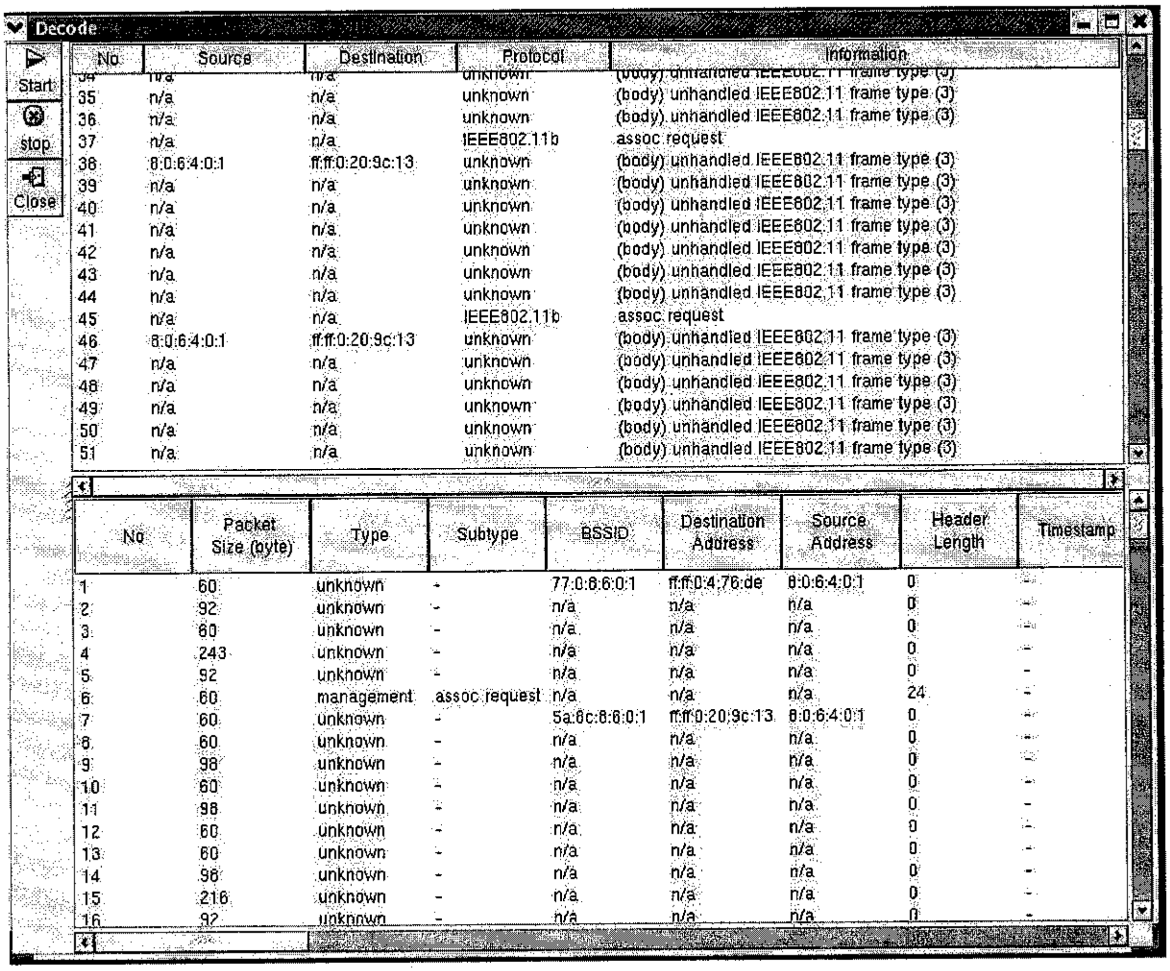
2.1 Linux Based Monitoring System

This monitoring tool was built in Linux platform to enable real time monitoring to be done. With the use of 3rd party device drivers and **libpcap**, the system was able to be built without much complication. With the resources and raw packets provided, further analysis on the packets can be executed. The usage of **libpcap** is mainly to enable packet capturing to be done across the network.

2.2 Enabling Ethernet card into promiscuous mode (in Linux platform)

The basic design and architecture of Wifi (in this research project, researcher worked with wireless network cards) network cards varies from one vendor to another. The behavior each react with the operating system is another issue, for example most cards do not support "**promiscuous**" mode. The tool becomes dependent to device drivers that come from each vendor to provide with the communication tools. Most of these drivers also have limited access and functions inhibiting us from developing monitoring tools with it. Developing or creating your own device driver is not an easy task as a substantial amount of expertise in device driver development is needed. At the end, the tool is dependent on 3rd party device drivers which are scarce. This issue is more apparent in Microsoft Windows operating systems as there are very few open source resources. Fortunately, Linux operating systems provide more open sources and has 3rd party device drivers to support multiple Wifi devices. These drivers allow us to retrieve raw packets from the wireless network interfaces as if they are Ethernet cards with **promiscuous** capabilities.

2.3 Generating report in Linux



No	Source	Destination	Protocol	Information
35	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
36	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
37	n/a	n/a	IEEE802.11b	assoc request
38	8:0:6:4:0:1	ff:ff:0:20:9c:13	unknown	(body) unhandled IEEE802.11 frame type (3)
39	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
40	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
41	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
42	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
43	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
44	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
45	n/a	n/a	IEEE802.11b	assoc request
46	8:0:6:4:0:1	ff:ff:0:20:9c:13	unknown	(body) unhandled IEEE802.11 frame type (3)
47	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
48	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
49	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
50	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)
51	n/a	n/a	unknown	(body) unhandled IEEE802.11 frame type (3)

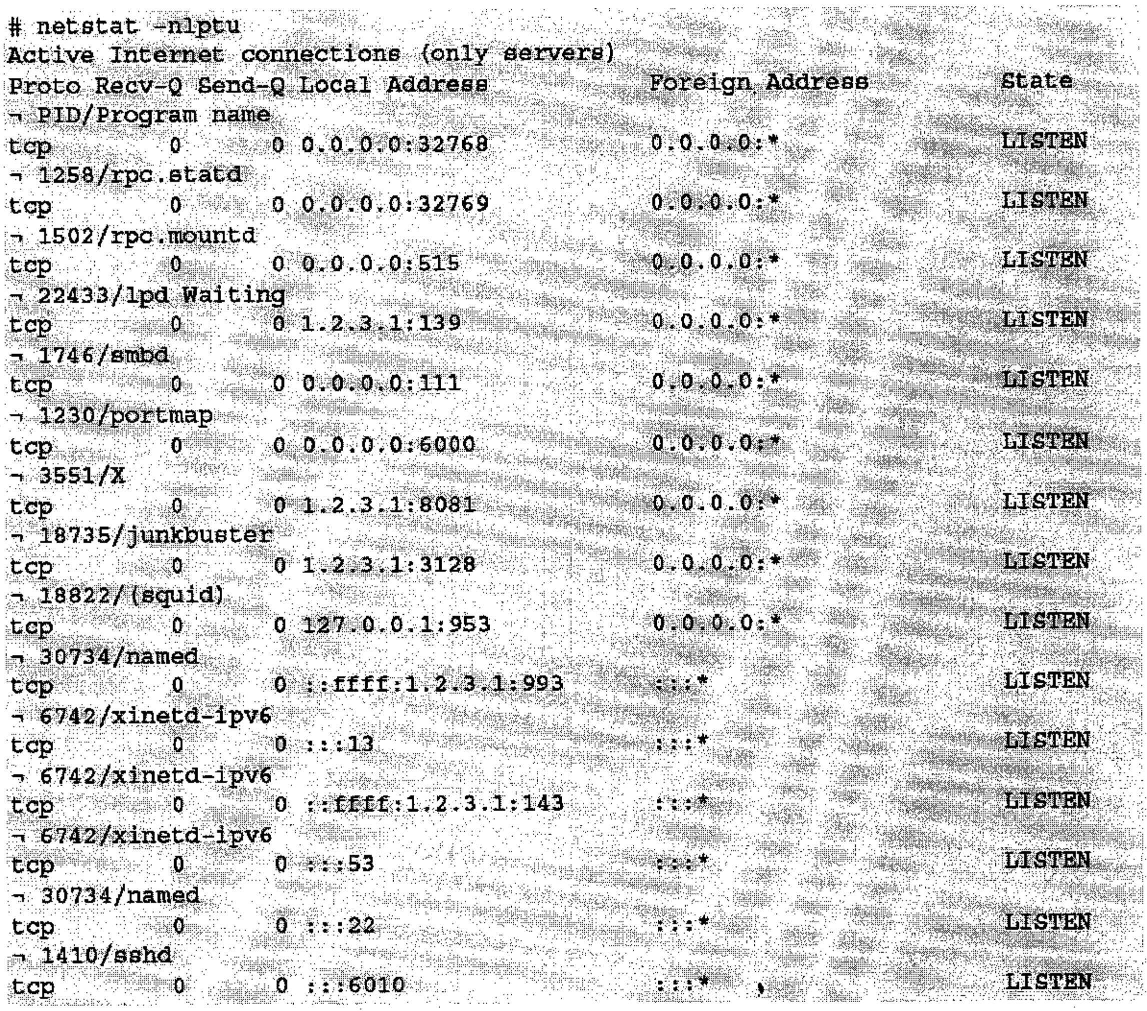
No	Packet Size (byte)	Type	Subtype	BSSID	Destination Address	Source Address	Header Length	Timestamp
1	60	unknown	-	77:0:8:6:0:1	ff:ff:0:4:76:de	8:0:6:4:0:1	0	-
2	92	unknown	-	n/a	n/a	n/a	0	-
3	60	unknown	-	n/a	n/a	n/a	0	-
4	243	unknown	-	n/a	n/a	n/a	0	-
5	92	unknown	-	n/a	n/a	n/a	0	-
6	60	management	assoc request	n/a	n/a	n/a	24	-
7	60	unknown	-	5a:8c:8:6:0:1	ff:ff:0:20:9c:13	8:0:6:4:0:1	0	-
8	60	unknown	-	n/a	n/a	n/a	0	-
9	98	unknown	-	n/a	n/a	n/a	0	-
10	60	unknown	-	n/a	n/a	n/a	0	-
11	98	unknown	-	n/a	n/a	n/a	0	-
12	60	unknown	-	n/a	n/a	n/a	0	-
13	60	unknown	-	n/a	n/a	n/a	0	-
14	98	unknown	-	n/a	n/a	n/a	0	-
15	216	unknown	-	n/a	n/a	n/a	0	-
16	92	unknown	-	n/a	n/a	n/a	0	-

Figure 1: Sample of report generated via the wireless monitoring tool on Linux platform

The report generated shows the source address, destination address and the type of packets passed through the network. The report generated is on real time basis, and generated as an on going process for the given time extend (for example 30-40 seconds).

2.4 Socket Binding in Linux

Unlike the usual WinPcap used in normal windows environment, Linux uses netstat for socket binding purposes in order to sniff the packets in network. Example of retrieved packets from the network under Linux is as Figure 2 below;



```
# netstat -nlptu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
¬ PID/Program name
tcp        0      0 0.0.0.0:32768           0.0.0.0:*               LISTEN
¬ 1258/rpc.statd
tcp        0      0 0.0.0.0:32769           0.0.0.0:*               LISTEN
¬ 1502/rpc.mountd
tcp        0      0 0.0.0.0:515             0.0.0.0:*               LISTEN
¬ 22433/lpd Waiting
tcp        0      0 1.2.3.1:139             0.0.0.0:*               LISTEN
¬ 1746/smbd
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
¬ 1230/portmap
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
¬ 3551/X
tcp        0      0 1.2.3.1:8081            0.0.0.0:*               LISTEN
¬ 18735/junkbuster
tcp        0      0 1.2.3.1:3128            0.0.0.0:*               LISTEN
¬ 18822/(squid)
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
¬ 30734/named
tcp        0      0 ::ffff:1.2.3.1:993      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::13                   :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 ::ffff:1.2.3.1:143      :::*                    LISTEN
¬ 6742/xinetd-ipv6
tcp        0      0 :::53                   :::*                    LISTEN
¬ 30734/named
tcp        0      0 :::22                   :::*                    LISTEN
¬ 1410/sshd
tcp        0      0 :::6010                 :::*                    LISTEN
```

Figure 2: Server socket binding under Linux platform

The packet captured shows the type of header (TCP or UDP) and exhibits the status of the node in the network. A monitoring tool in Linux based platform uses the same concept as Winpcap in sniffing the packets across the network, and decoding them according to the IP header. The difference though is due to the usage of OS platform. In a windows based system, packet capturing is best applied using Winpcap.

2.5 Packet Sniffer (Sniff'em application)

A packet sniffer is a wiretap device that plugs into computer network to capture the information on the packets sent over the network. Most popular way of connecting computers is through Ethernet. Ethernet protocol works by sending packet information to all the hosts on the same segment. The packet header contains the address of the destination and source machine. Packets are sniffed by allowing the network card to be in a promiscuous mode. A machine that is accepting all packets, no matter what the packet header says, is said to be in promiscuous mode. Sniff'em application uses the promiscuous mode in the NDIS driver to enable the card to listen to data traffic. NDIS is a Windows device driver interface that enables a single network interface card (NIC) to support multiple network protocols. For example, with NDIS, a single NIC can support TCP/IP, IPX, and more protocols. Sniff'em application supports both high and low level protocols. High level protocol includes IP6 Header Compression, IPv6 Control Message Protocol and low level protocol includes CMP and IGMP. The requirement to implement the Sniff'em technology includes a network card which is set into promiscuous mode.

2.6 Packet Capturing Technologies

"The ability to capture and reconstruct a network conversation is a crucial component to any security administrator needing to investigate potential security incidents. A well executed packet capture or "wire tap" can be instrumental in determining what activity a hacker is up to on your network, what trade secrets are being transmitted illegally or what the latest malware is up to on your network." – Stevie Hendrie, steviehendrie.com

Stevie Hendrie.com has reviewed on two sufficient packet capturing tool in the market, TCPDump and Ethereal.

TCPDump is perhaps the most widely used packet capturing software in the Unix environment. Originally developed by the Network Research Group at Lawrence

Berkeley National Laboratory, it is now available under the Open Source BSD License. TCPDump utilizes the libpcap packet capture library which makes it a robust system that is independent as a packet capturing tool. In addition to the available Unix ports, it has also been ported to Windows as WinDump. Like its Unix counterpart, it utilizes an independent packet capture library called WinPCAP. Though, the application is a completely command line driven application without user interface being provided. This however causes discomfort among the users who are not familiar with command line formatting. With over 100 possible command line arguments and expressions, TCPDump can be extremely overwhelming to people who are either not very familiar with TCP/IP concepts or comfortable with command line interfaces.

The strength of TCPDump includes;.

1. Robust command line options allow for extremely granular filtering of packet captures. The command line allows user to easily capture as little or as much information as you want. Additionally, the tool can display only the headers of the communication or optionally write the entire payload to a file.
2. Supports most network protocols. Specifically ethernet, fddi, token ring, ip, ipv6, arp, rarp, decnet, tcp and udp.
3. Excellent filtering capabilities. It allows user to filter based at incredible granularity on source, destination, protocol, interface, host, network, and size.

Ethereal on the other hand is still officially a beta product. It is well known as "The world's most popular network protocol analyzer" with support for multiple platforms and a graphical user interface. It uses Win32 port, like WinDump requires the WinPCAP packet capture library in order to be functional. The primary interface for Ethereal is graphical; however, it does offer some command line support for data and capture manipulation. The significant advantage of the graphical interface is the ability to view the capture files within the same application. The GUI offers easy conversation navigation as well as color coded search and identification. The graphical interface does allow for both capture and viewing filtering allowing for quick and easy data viewing within the capture window. The strength of Ethereal includes;

1. Supports numerous packet capture software's files including tcpdump and a number of other commercial and freeware packages.
2. Dissection support of over 440 protocols.
3. Excellent platform support.
4. Packet capturing on real time basis.

However the shortcomings of Ethereal are in regards to its filtering capabilities. Though Ethereal is a graphical application, the filtering configuration is still handled through regular expression type syntax. This syntax can be confusing to a newcomer and consists of hundreds of possible combinations.

CHAPTER 3
METHODOLOGY/ PROJECT WORK

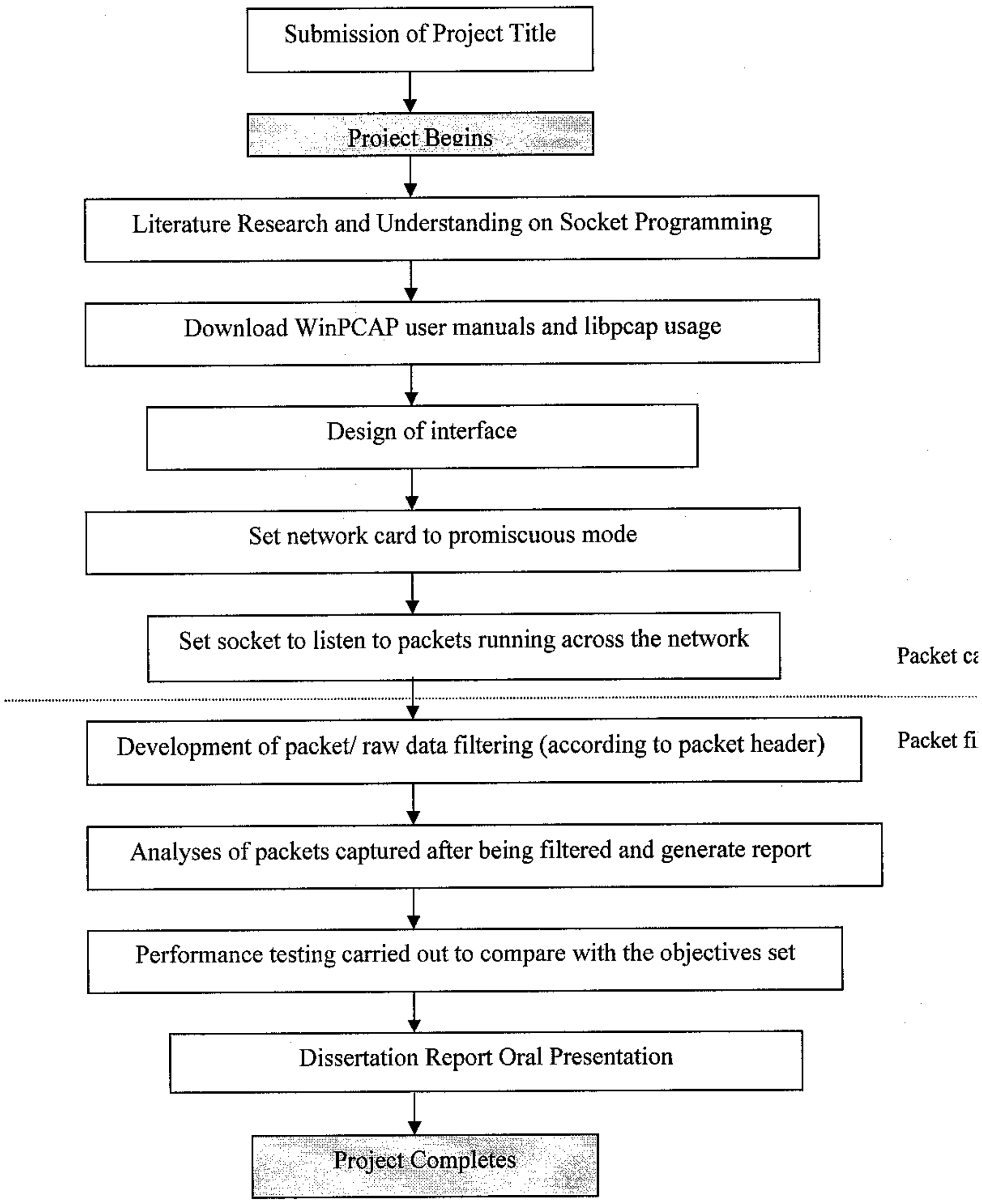
3.1 Procedure Identification

There are five phases in doing this project. The phases involved are preliminary study, analysis, design, development, testing and evaluation. Table 3.1 discuses on the activities carried out during each phases:

No.	Phase	Activities
1.	Preliminary Study	Topic Selection ▪ Studies carried out to find a suitable project title. ▪ Define the project's problem statement, objective, and scope of study. Literature Review ▪ Read and study the existing ideas and comments on IPv6, existing network monitoring tool, winpcap (packet capturing), and C#.net.
2.	Analysis	Information Gathering ▪ Looking up for sources of information such as the internet, written research papers and related books. Depth of analysis

		▪ Study on the design and development methods and concept by reflecting on the resources collected.
3	Design	Design the network monitoring tool for IPv6.
4	Development	Develop application: ▪ Network monitoring tool developed using C#.net ▪ Packet capturing using Winpcap ▪ Socket Programming to bind network card into a promiscuous mode.
5	Testing and Evaluation	Testing ▪ Conducted to test the system functionality and stability. ▪ Users to test run the system Evaluation ▪ Feedback from users after testing the application. ▪ Evaluate whether or not this project has met its objectives ▪ Suggest recommendations for the research project.

Table 1: Project Methodology



Submission of Project Title
Project Begins
Literature Research and Understanding on Socket Programming
Download WinPCAP user manuals and libpcap usage
Design of interface
Set network card to promiscuous mode
Set socket to listen to packets running across the network
Packet ca
Development of packet/ raw data filtering (according to packet header)
Packet fi
Analyses of packets captured after being filtered and generate report
Performance testing carried out to compare with the objectives set
Dissertation Report Oral Presentation
Project Completes

Figure 3: Flow Chart of project work for semester

3.2 Project Work

After the implementation of Winpcap in capturing the packets running on the network, the system used in packet filtering is shown below. The results are discussed in **Chapter 4**. The IP datagrams are transmitted by encapsulation in Ethernet packet (Medium Access Control (MAC) frames). An example of an ICMP packet that is encapsulated is shown **Figure 4** below.

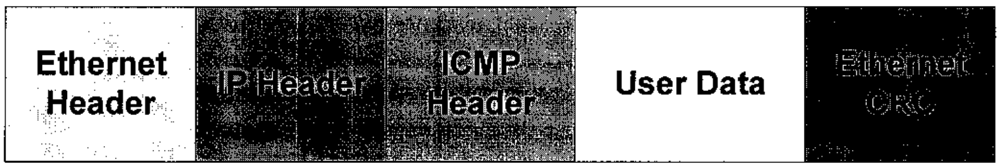


Figure 4: Example of an IP datagram carrying ICMP message for transmission over Ethernet

The first layer that the packet will pass through is the data link layer (MAC layer) before it is processed in the following layers. The protocol stack is shown in **Figure 5** below together with the position of each protocol in each layer within the Open System Interconnection (OSI) reference model.

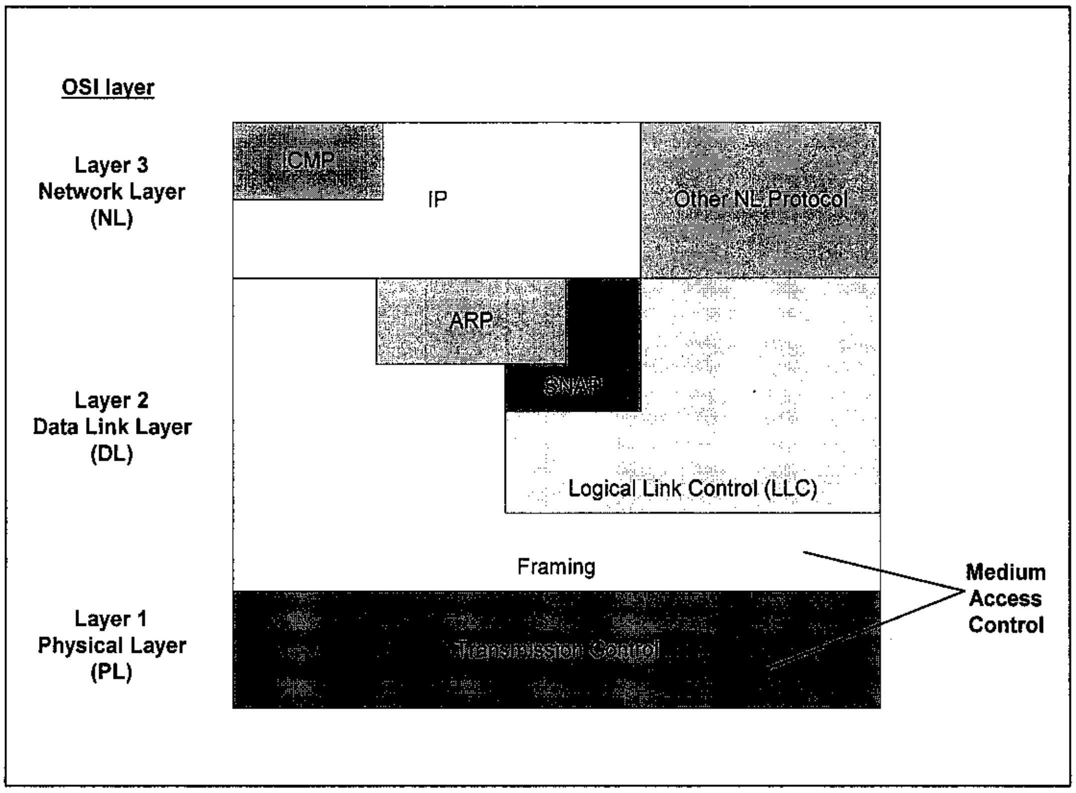


Figure 5: Protocol stack for IP over Ethernet

The following summary shows the processes performed by an Agent in an IPv6 network. It is assumed that the agent is residing on a node that is connected to an Ethernet network. The raw packets captured using the pcap library will be processed as follows (to filter for ICMPv6 packets)

i) MAC Protocol

- The source MAC address and the destination MAC address will be retrieved from the Ethernet packet.
- The type field in the packet will be checked. Only type field value of ***0x86dd*** (IPv6) will be processed. If the types are other than IPv6 such as value 0x0806 for Address Resolution Protocol (ARP), the packet will not be processed.
- The protocol type will be checked to confirm that it is IPv6 packet. It is stated in the version field in the IPv6 header (Deering, S., Hinden, R., 1998).

- The packet will be checked if the next header is the ICMPv6 header. This can be obtained by checking the next header field in the packet. The next header value for ICMPv6 is **58.** If it contains other protocol, the packet will be dropped.
- The IPv6 source and IPv6 destination address will then be retrieved from the packet.

Below are the descriptions of work done at each phase of the methodology:

3.2.1 Preliminary Study

At this stage, research is conducted to find a suitable project title. The selected title is then submitted to the **Final Year Project (FYP) Committee** in order to be approved. Once approved, the problem statement, objectives and scope of study of the project will be defined.

3.2.2 Analysis

During the analysis phase, researches are conducted in order to view and study on experienced people's comment on the particular topic and area of study. Apart from reading, reliable sources are gathered in order to further support the idea applied in developing the monitoring tool. Other developed application with similar concept is referred to improvise and strengthen the study done.

3.2.3 Design

At this phase of the methodology, the application will be designed. Scripting style, the user interface and methods used in order to capture the packets and analyze them will be identified and defined.

3.2.4 Development

In the development phase, the application designed will be developed. WinPCAP header files will be used in order to capture the packets running in a network. In order to capture the packets, socket programming will be needed to bind the network cards into promiscuous mode which is implied to listen for packets running in a network and receive and dissect the packet. WinPcap offers a kernel-level programmable monitoring module which is able to calculate simple statistics on the network traffic. The statistics can be gathered without the need to copy the packets to the application, which simply receives and displays the results obtained from the monitoring engine. This allows the avoidance of 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 that provides a configurable way to select a subset of the network 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. Buffers are not allocated at kernel and user level.

- **NPF and NDIS driver**

NDIS (Network Drive Interface Specification) is a standard that defines a communication between a network adapter and the protocol driver (that implements for example TCP/IP). The main purpose of NDIS driver is to act as a wrapper that allows protocol drivers to send and receive packets on to the drivers (either LAN or WAN). Whereas the **NPF** driver acts at the protocol driver allowing 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. **Figure 6** shows the position of NPF driver in the NDIS stack.

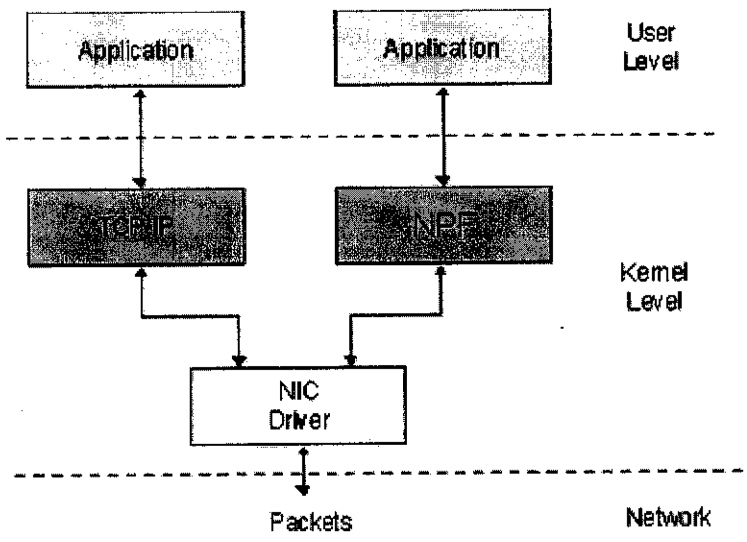


Figure 6: NPF inside NDIS

Events like the arrival of a new packet are notified to NPF through a callback function (Packet_tap()). Furthermore, the interaction with NDIS and the NIC driver takes place by non blocking functions: when NPF invokes a NDIS function, the call returns immediately; when the processing ends, NDIS invokes a specific NPF callback to inform that the function has finished. The driver exports a callback for any low-level operation, like sending packets, setting or requesting parameters on the NIC.

- **Packet Capturing**

During a capture, the driver sniffs the packets using a network interface and delivers them intact to the user-level applications. The packet capture process is the most important process of NPF. The capturing process using the NPF relies based on the packet filtering. Packet filter is programmed to decide whether an incoming packet has to be accepted and copied into the listening application. . 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"). The monitoring program is executed for every incoming packet, and only the conformant packets are accepted.

- **Packet filtering**

The Agent will capture IPv6 packets in the network and dissect the packet to get the necessary information. The IPv6 packet will be captured using the **pcap** library, which is defined in FreeBSD. **Pcap** provides high level interface to packet capture system. It is used to grab or sniff packets on a network. Some of the functions defined in **pcap** as shown in **Table 2** will be used in decoding the packets. The reason for capturing the raw IPv6 packets is to retrieve necessary information that will be used later for further requesting the nodes information such as the source and the destination address.

Function	Description
pcap_lookupdev(*char *errbuf*);	This is used to set the Ethernet card that will be used.
pcap_open_live(*char *device, int snaplen, int promisc, int to_ms, char *errbuf*);	This function is used to make the Ethernet card in promiscuous mode
pcap_compile(*pcap_t *p, struct bpf_program *fp, char *str, int optimize, bpf_u_int32 netmask*);	This is used to compile the string str into a filter program.
pcap_setfilter(*pcap_t *p, struct bpf_program *fp*);	This function is used to specify a filter program. We can filter the packets that need to be captured from the network.
pcap_loop(*pcap_t *p, int cnt, pcap_handler callback, u_char *user*);	It will continuously read packets from the network until someone terminate the application.

Table 2: Pcap functions used in the program

3.2.5 Testing and Evaluation

Testing is done to test the functionality and usability of the developed application. Thus, the application is tested by users (students) and feedback is given by the users whether the application has met the objectives of the research study. The system is tested by allowing user to test it on network by copying the .exe form into user's pc.

3.3 Tools Required

- Free BSD, NetBSD, OpenBSD
- Windows XP, Windows 2000, and Win95/98/ME with Winpcap library preinstalled into the Operating Systems
- NPF driver preinstalled (npf.sys installed into windows)
- C#.Net programming tool

CHAPTER 4
RESULTS AND FINDINGS

4.1 Winpcap library installed into developers computer

In order to allow packets to be listened across the network, Winpcap library needs to be installed into the computer. The main component in the Winpcap that needs to be installed is the NPF driver. NPF is installed as the protocol driver, though it is indicated as not the best solution in allowing packet capturing, but it allows reasonable independence in the MAC layer as well as complete access to the raw traffic. NPF driver is installed by downloading the setup file from Winpcap's official website. And NPF driver is installed into the network simply by running the .exe file from the Winpcap website.

4.2 Packet Capturing using the packet.dll function

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).

WinPcap is used more as an "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.

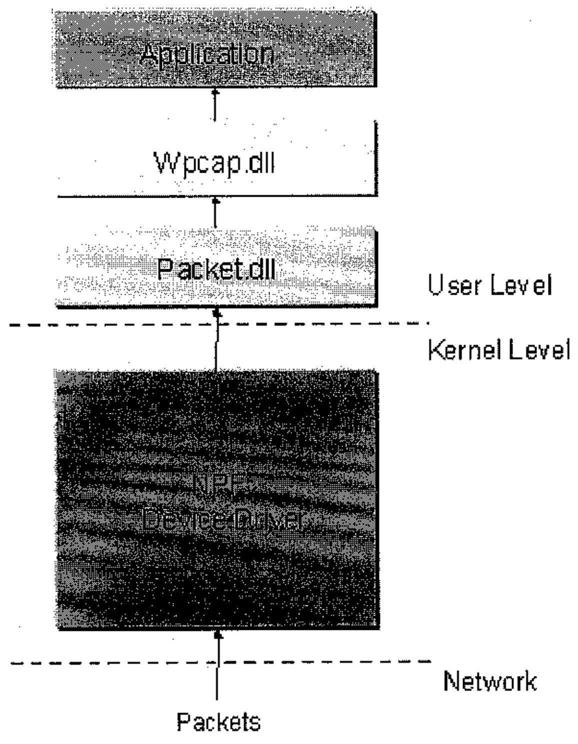


Figure 7: Components of Winpcap

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 it is known as a device driver, called Netgroup Packet Filter (NPF).NPF driver offers both basic features like packet capture and injection, as well as more advanced ones like a programmable filtering system and a monitoring engine (which will be very applicable in the IPv6 monitoring tool). In the IPv6 monitoring tool, the filtering system will be programmed to filter IPv6 headers (tcp, udp or ICMPv6). The first one is used to restrict a capture session to a subset of the network traffic (e.g. capturing IPv6 packets on the network), the second one provides a powerful but simple to use mechanism to obtain statistics on the traffic (e.g. 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.

In implementing Winpcap into the Network Monitoring Tool as a Windows Based Application, packet.dll is preferred. 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 netmasks
- Set various low-level parameters of an adapter

The other importance of the packet.dll function 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. In order to create an application that uses the packet.dll function;

- Include the file *packet32.h* at the beginning of every source file that uses the functions exported by the dll. *Packet32.h* is distributed both with the packet.dll source code and with the WinPcap developer's pack. It is platform-independent.

- Include *packet.lib* in the project. *Packet.lib* is generated compiling the packet driver and can be found in the developer's pack.

In the application however, packet.dll is called using the Packet32h.cs class. The class is then called in the main application form to allow the capture process to take place. An example of a .dll function being imported into a class is shown below.

```
[DllImport("kernel32.dll")] public extern static int

        GetVersionEx( ref Function.OSVERSIONINFO lpVersionInformation );
```

The kernel32.dll function is imported here to get the IP address version (version 4 or 6) to identify or differentiate IPv6 packets to IPv4 packets.

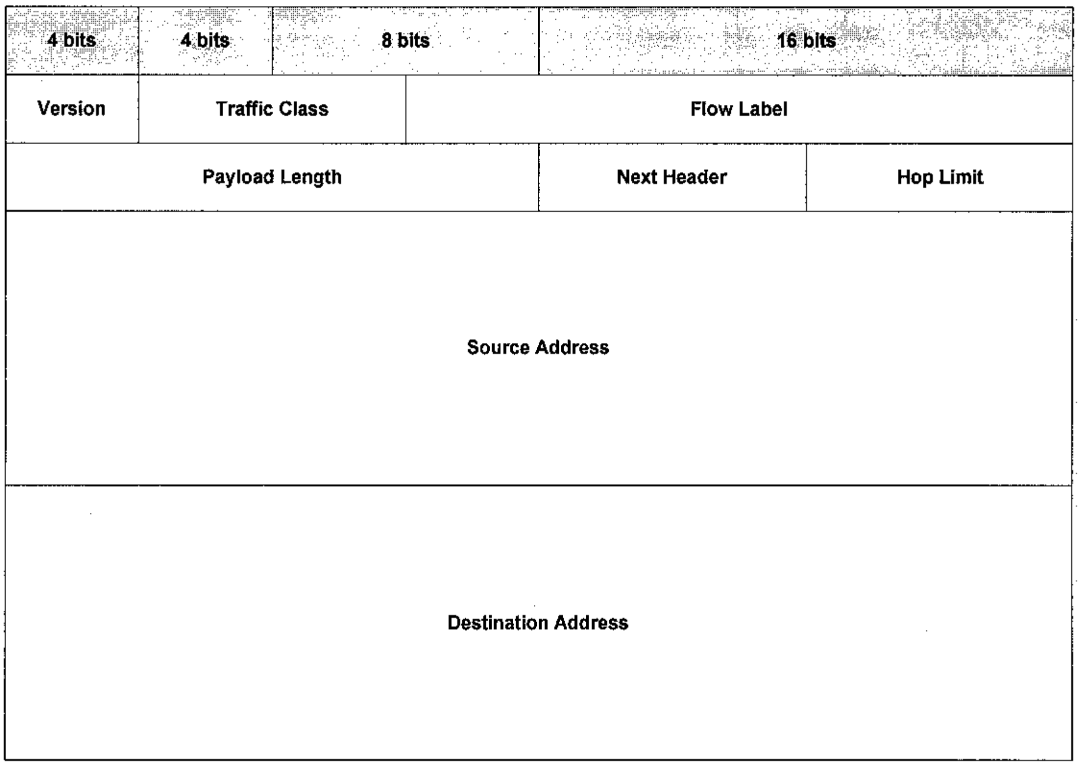
4.3 Packet filtering and analysing

The following summary shows the processes performed by an Agent in an IPv6 network. It is assumed that the agent is residing on a node that is connected to an Ethernet network. The raw packets captured using the pcap library will be processed as follows (to filter for ICMPv6 packets)

4.3.1 MAC Protocol

The source MAC address and the destination MAC address will be retrieved from the Ethernet packet. The type field in the packet will be checked. Only type field value of ***0x86dd*** (IPv6) will be processed. If the types are other than IPv6 such as value 0x0806 for Address Resolution Protocol (ARP), the packet will not be processed.

4.3.2 IPv6 Header



Fields:

Version	4-bit Internet Protocol Version = 6
Traffic Class	8-bit
Flow Label	20-bit
Payload Length	16-bit Length of the IPv6 payload
Next Header	8-bit Identifies the header immediately following the IPv6 header
Hop Limit	8-bit Decremented by 1 by each node that forwards the packet
Source Address	128-bit Address of the source node
Destination Address	128-bit Address of the destination node

Figure 8: IPv6 Header

The protocol type will be checked to confirm that it is IPv6 packet. It is stated in the version field in an IPv6 header, shows a 6 or a 4 depending on the type of protocol that it carries. (Version 4: IPv4 and Version 6: IPv6)

4.3.3 Next header

The packet will be checked if the next header is the ICMPv6 header. This can be obtained by checking the next header field in the packet. The next header value for ICMPv6 is **58**. If it contains other protocol, the packet will be dropped. A node that sends an ICMPv6 message has to determine both the Source and Destination IPv6 Addresses in the IPv6 header before calculating the checksum. If the node has more than one unicast address, it must choose the Source Address of the message as follows:

- If the message is a response to a message sent to one of the node's unicast addresses, the Source Address of the reply must be that same address.

- If the message is a response to a message sent to a multicast or anycast group in which the node is a member, the Source Address of the reply must be a unicast address belonging to the interface on which the multicast or anycast packet was received.

- If the message is a response to a message sent to an address that does not belong to the node, the Source Address should be that unicast address belonging to the node that will be most helpful in diagnosing the error. For example, if the message is a response to a packet forwarding action that cannot complete successfully, the Source Address should be a unicast address belonging to the interface on which the packet forwarding failed.

- Otherwise, the node's routing table must be examined to determine which interface will be used to transmit the message to its destination, and a unicast address belonging to that interface must be used as the Source Address of the message.

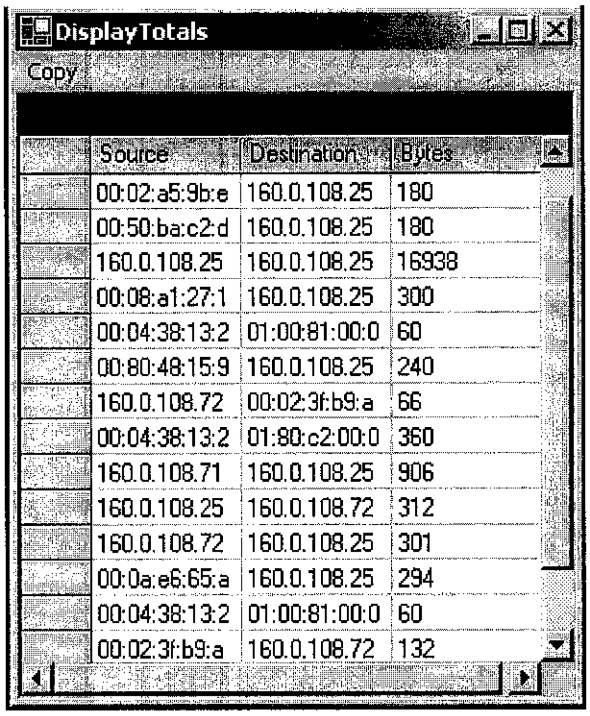
Type	**Meaning**
1	Destination Unreachable
2	Packet Too Big
3	Time Exceeded
4	Parameter Problem
128	Echo Request
129	Echo Reply
130	Group Membership Query
131	Group Membership Report
132	Group Membership Reduction
133	Router Solicitation
134	Router Advertisement
135	Neighbor Solicitation
136	Neighbor Advertisement
137	Redirect

Table 3: Type of ICMPv6 messages

The IPv6 source and IPv6 destination address will then be retrieved from the packet.

4.4 Application of NPF driver in the network monitoring tool

NPF driver allows not only the filtered packets to be displayed but it also determines the amount of bytes to be kept. Thus, the application of the NPF driver in developing the monitoring tools allows the application to display the source and the destination address as well as the bytes transferred by the packets. **Figure 9** shows the source and the destination address of the captured packets as well as the bytes transferred.



Source	Destination	Bytes
00:02:a5:9b:e	160.0.108.25	180
00:50:ba:c2:d	160.0.108.25	180
160.0.108.25	160.0.108.25	16938
00:08:a1:27:1	160.0.108.25	300
00:04:38:13:2	01:00:81:00:0	60
00:80:48:15:9	160.0.108.25	240
160.0.108.72	00:02:3f:b9:a	66
00:04:38:13:2	01:80:c2:00:0	360
160.0.108.71	160.0.108.25	906
160.0.108.25	160.0.108.72	312
160.0.108.72	160.0.108.25	301
00:0a:e6:65:a	160.0.108.25	294
00:04:38:13:2	01:00:81:00:0	60
00:02:3f:b9:a	160.0.108.72	132

Figure 9: Display of total packets captured

CHAPTER 5
CONCLUSION

The implementation of a network monitoring tool that can be extended to capture IPv6 packet on the network is applicable with the combination of Winpcap and NPF driver being preinstalled in the operating systems. The application of NPF driver 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. Though in developing the network monitoring tool, NPF driver is used to capture and monitor packets running over the network.

The author proposes the development of a network monitoring tool that extends to capture the latest Internet Protocol which is referred to as IPv6 using the inner functions of Winpcap (NPF driver). The extension of the network monitoring tool to capture IPv6 packets as well over the network is relevant as many network users have changed their IP addresses to IPv6, which compared to the previous IP address (IPv4), provides more network facilities. Current Network Monitoring Tools designed does not cater to the need of users to capture IPv6 packets running over the network.

The network monitoring tool proposed by the author allows users to capture the packets running over the network and hence display statistics on the percentage of the particular header running over the network. The captured packets are then displayed by indicating the source and the destination address of the packet as well as the amount of bytes carried by the packet. This complies with the objective of the project.

From the discussion in Chapter 4, it can be deduced that NPF driver is the most suitable protocol driver that is chosen by author to be used in capturing packets over the network. With reference to the results obtained, it can be said that NPF driver is an attractive tool to be used to capture IPv6 packets over the network. With further study and research on IPv6 implementation and transition of IPv4 to IPv6, the packets captured can be decoded so that user can view the transmitted packets messages and also detect the packet loss over the network.

Two objectives have been outlined for the semester. First is the development of an agent that captures IPv6 packets over the network. Secondly to develop an agent that filters the captured packets into packet headers, source address and the destination address carried by the packets over the network. Both the agent were developed using C#.Net as the programming tool and NPF driver is used as the protocol driver to allow the network card to be in a promiscuous mode.

In future the network monitoring tool that has been designed by author can be further developed to meet the standards of the growing network technology. Expansions that can be done to the monitoring tool include the ability of the monitoring tool to observe any packet loss during the packet transmission over the network. User can also be given capture options, in order to choose the type of packet filtering (choose the protocol options that user wants the tool to filter) and also the maximum number of packets to be captured at a given time set by the user.

In addition to that, future expansion can also be done by adding buffer decoding to the network monitoring tool. Buffer Decoding does additional clustering, reassembly and decoding routines to offer a broad overview over the Network usage and Network captures currently within the Buffer.

Thus, referring to the study done by the author, it is essential that applications such as the network monitoring tool for IPv6 needs to be implemented in the near future as IPv6 will be taking over IPv4. Network users should be provided with proper application and tools to implement and use the latest Internet Protocol – Ipv6.

REFERENCES

1. S. Deering and R. Hinden. Internet Protocol, Version 6. RFC 1883, December 1995.
2. J. McCann, S. Deering, and J. Mogul. Path MTU Discovery for IP version 6, RFC 1981, Aug. 1996.
3. R. E. Gilligan, S. Thomson, J. Bound, and W. R.Stevens. Basic Socket Interface Extensions for IPv6. Work In Progress.
4. Wireless Monitoring Tool in Linux Platform (research paper): Chua Kim Yong, Lim Siew Ching, Phang Tze Shu (BScCS), School of Computer Science, Universiti Sains Malaysia.
5. Deering, S. and R. Hinden, "Internet Protocol, Version 6, (IPv6) Specification", December 1998.
6. S. McCanne and V. Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture. Proceedings of the 1993 Winter USENIX Technical Conference (San Diego, CA, Jan. 1993), USENIX.
7. A. Begel, S. McCanne, S.L.Graham, BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture, Proceedings of ACM SIGCOMM '99, pages 123-134, Conference on Applications, technologies, architectures, and protocols for computer communications, August 30 - September 3, 1999, Cambridge, USA

8. Eric Marin, EMEA Senior Consulting Engineer, "The Challenges of Filtering IPv6 Packets", RSA Conference 2003.
9. http://www.ipv6.org/
10. http://www.ipv6forum.com/
11. http://www.bieringer.de/linux/IPv6/
12. http://www.nwfusion.com/research/ipv6.html
13. http://nrg.cs.usm.my/~tcwan/

APPENDICES

Code Values

net.sourceforge.jpcap.net.EthernetProtocols

public static final int AARP	33011
public static final int ALL	3
public static final int ARP	2054
public static final int ATALK	32923
public static final int AX25	2
public static final int BPQ	2303
public static final int CONTROL	22
public static final int CUST	24582
public static final int DDCMP	6
public static final int DEC	24576
public static final int DECDNS	32828
public static final int DECDTS	32830
public static final int DIAG	24581
public static final int DNA_DL	24577
public static final int DNA_RC	24578
public static final int DNA_RT	24579
public static final int ECHO	512
public static final int INFTH	34925
public static final int IP	2048
public static final int IPV6	34525
public static final int IPX	33079
public static final int IRDA	23
public static final int LANBRIDGE	32824
public static final int LAT	24580
public static final int LOCALTALK	9
public static final int LOOP	96
public static final int LOOPBACK	36864
public static final int MASK	65535
public static final int MOBITEX	21
public static final int N802_2	4
public static final int N802_3	1
public static final int N8021Q	33024
public static final int NS	1536
public static final int PPP	34827
public static final int PPP_MP	8
public static final int PPPOED	34915
public static final int PPPOES	34916

public static final int PPPTALK	16
public static final int PUP	1024
public static final int RARP	32821
public static final int SCA	24583
public static final int SNAP	5
public static final int SPRITE	1280
public static final int STBPDU	38
public static final int TR_802_2	17
public static final int TRAIL	4096
public static final int VEXP	32859
public static final int VPROD	32860
public static final int WAN_PPP	7
public static final int X25	2053

net.sourceforge.jpcap.net.IPProtocols

public static final int AH	51
public static final int COMP	108
public static final int DSTOPTS	60
public static final int EGP	8
public static final int ENCAP	98
public static final int ESP	50
public static final int FRAGMENT	44
public static final int GRE	47
public static final int HOPOPTS	0
public static final int ICMP	1
public static final int ICMPV6	58
public static final int IDP	22
public static final int IGMP	2
public static final int INVALID	-1
public static final int IP	0
public static final int IPIP	4
public static final int IPV6	41
public static final int MASK	255
public static final int MTP	92
public static final int NONE	59
public static final int PIM	103
public static final int PUP	12
public static final int RAW	255
public static final int ROUTING	43
public static final int RSVP	46

```
public static final int TCP         6
public static final int TP         29
public static final int UDP        17
```

net.sourceforge.jpcap.net.IPVersions

```
  public static final int IPV4  4
  public static final int IPV6  6
```

Class Address

Field Summary

static java.lang.String	ATM
static java.lang.String	E_MAIL
static java.lang.String	IPV4_ADDR
static java.lang.String	IPV4_ADDR_HEX
static java.lang.String	IPV4_NET
static java.lang.String	IPV4_NET_MASK
static java.lang.String	IPV6_ADDR
static java.lang.String	IPV6_ADDR_HEX
static java.lang.String	IPV6_NET
static java.lang.String	IPV6_NET_MASK
static java.lang.String	LOTUS_NOTES
static java.lang.String	MAC
static java.lang.String	SNA
static java.lang.String	UNKNOWN
static java.lang.String	VM

PCAP Function

int **pcap_findalldevs_ex** (char ***host**, char ***port**, SOCKET sockctrl, struct **pcap_rmtauth** *auth, **pcap_if_t** **alldevs, char ***errbuf**)
It creates a list of network devices that can be opened with ***pcap_open()****.*

int **pcap_createsrcstr** (char *source, int type, const char ***host**, const char ***port**, const char *name, char ***errbuf**)
Accepts a set of strings (host name, port, ...), and it returns the complete source string according to the new format (e.g. 'rpcap://1.2.3.4/eth0').

int **pcap_parsesrcstr** (const char *source, int *type, char ***host**, char ***port**, char *name, char ***errbuf**)
Parses the source string and returns the pieces in which the source can be split.

pcap_t * **pcap_open** (const char *source, int **snaplen**, int flags, int read_timeout, struct **pcap_rmtauth** *auth, char ***errbuf**)
It opens a generic source in order to capture / send (WinPcap only) traffic.

int **pcap_remoteact_accept** (const char ***address**, const char ***port**, const char ***hostlist**, char *connectinghost, struct **pcap_rmtauth** *auth, char ***errbuf**)
It blocks until a network connection is accepted (active mode only).

int **pcap_remoteact_close** (const char ***host**, char ***errbuf**)
It drops an active connection (active mode only).

void **pcap_remoteact_cleanup** ()
Cleans the socket that is currently used in waiting active connections.

int **pcap_remoteact_list** (char ***hostlist**, char sep, int **size**, char ***errbuf**)
Returns the hostname of the host that have an active connection with us (active mode only).

char **fakeerrbuf** [PCAP_ERRBUF_SIZE+1]
activehosts * **activeHosts**
Keeps a list of all the opened connections in the active mode.

SOCKET **sockmain**
Keeps the main socket identifier when we want to accept a new remote connection (active mode only).

Packet types

Network Layer protocol

----

1. IPv4
2. IPv6
3. ARP
4. x75
5. Rev ARP
6. DEC LAN Bridge
7. Apple Talk
8. Apple Talk ARP
9. MPLS
10. IPX
11. Spanning Tree
12. NetBIOS
13. Others

IP based protocol (transport)

----

1. TCP
2. UDP
3. ICMP
4. IGMP
5. EGP
6. IGRP
7. RSVP
8. GRE
9. ESP
10. VINES
11. OSPF
12. SCTP