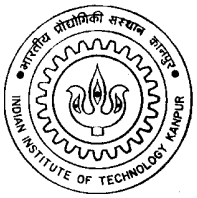
Supporting IPv6 in PickPacket

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

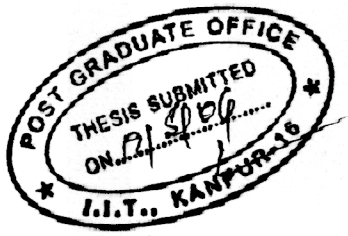
Devendar Bureddy



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2006



Certificate

This is to certify that the work contained in the thesis entitled "*Supporting IPv6 in PickPacket*", by *Devendar Bureddy*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2006

(Dr. Dheeraj Sanghi)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

Over the years, Internet has become a popular medium for communication. At the same time its use in illegal activities is also increasing. Therefore, there is a need to monitor network traffic. However, this monitoring should not compromise the privacy of individuals who are using the Internet for legal purposes. PickPacket - a network monitoring tool developed at IIT Kanpur, can handle the conflicting issues of network monitoring and privacy through its judicious use. It is a passive tool in the sense that it neither injects any packet into the network nor delays any packet. PickPacket comprises of four components – the *Configuration File Generator* helps the users in specifying the filtering parameters, *Filter* captures the packets from the network, *Post-Processor* analyzes the captured data and *Data Viewer* renders the captured sessions interactively. PickPacket has support for HTTP, FTP, SMTP, POP, IMAP, Telnet, IRC and Yahoo-messenger protocols. It can filter traffic belonging to these protocols, reconstruct the sessions and display it to the user.

IPv6 is the next generation protocol designed by the IETF to replace the current version Internet Protocol, IPv4. Most of today's internet uses IPv4, which is now more than twenty years old. The amount of IPv6 traffic is going to increase day by day. This thesis discusses an extension to PickPacket to support monitoring of IPv6 traffic. The work involved changing all components of the tool to support the new protocol. Various tests were conducted to verify the correctness of the tool and to measure its performance.

We have extended PickPacket in another way.The amount of compressed HTTP traffic in Internet traffic is also growing, since it saves the network bandwidth and speeds up the response time. This thesis also discusses on the fly decompression of compressed HTTP traffic and performing searching of strings searching in compressed data.This has been implemented both for IPv4 and IPv6.

Acknowledgments

I take this opportunity to express my sincere gratitude toward my thesis supervisor Dr.Dheeraj Sanghi for his invaluable guidance throughout my thesis work. It would have never been possible for me to take this project to completion without his innovative ideas and encouragement. I also thank whole heartily to Dr. Deepak Gupta for his valuable suggestions even though he was not available here. The thesis is for a project that is financially supported by Ministry of Communications and Information Technology, Government of India. The support of the Ministry of Communications and Information Technology is duly acknowledged.

I also thank the other team members involved with the development of PickPacket for their cooperation, especially Sudheer and Ananth who helped me initially while understanding the Architecture of PickPacket. I also thank my project partner, Vinaya, for her cooperation and innovative suggestions regarding the project. I also wish to thank whole heartily all the faculty members of the Department of Computer Science and Engineering, IIT Kanpur for enhancing my knowledge. I also wish to thank Navpreet Singh for his help in sniffing on the CC network. I would like to thank all my classmates for the moments I shared with them. Mtech2004 batch is one that I never forget in my life. I would also like to thank everyone in the Prabhu Goel Research Center for providing a nice and challenging work environment.

Finally, I would like to thank my family encouraging me at all times and taking me to this stage in life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Internet has become a major medium of communication all over the world. But Internet can also be and has been used for unlawful activities by terrorists and criminals to communicate information. Thus, There is a need for tools to monitor network traffic in order to prevent such illegal activities. Not just law enforcement agencies, but even companies need these tools to prevent their valuable data from falling into hands of their competitors. However, monitoring and analysis of Internet traffic may violate the privacy of individuals whose network communication is being monitored. PickPacket - a Network Monitoring Tool, can address the conflicting issues of networking monitoring and privacy through its judicious use.

PickPacket is a tool that monitors network traffic and captures those packets that the user is interested in. It provides a very powerful set of criteria for filtering the packets. It has support for many application protocols such as SMTP, HTTP, FTP, Telnet,POP,IMAP IRC and Yahoo-Messenger.

The previous version of PickPacket worked with the assumption that all network traffic uses IPv4. However, use of IPv6 is growing vary rapidly in the Internet. It is likely to become the dominant Internet Protocol in 3-5 years, replacing IPv4. Applications need to adapt to both IP versions. In this thesis, we describe the incorporation of support for IPv6 traffic handling in PickPacket.

Another problem we discussed in thesis is handling compressed HTTP traffic. Now a days most of the web servers are serving data in compressed form to improve

the web performance. If the HTTP traffic is in compress form, then it is difficult to apply string search algorithms to the compressed data. This violates the entire goal of PickPacket. We described the solution to decompress HTTP packets on the fly.

1.1 Network Monitoring Tools

Network monitoring tools are also called sniffers. Network sniffers are named after a product called Sniffer Network Analyzer, introduced in 1988 by Network General Corporation. These tools are used to monitor and analyze data flowing across the network. They capture the network traffic based on some rules specified by the user. Network monitoring tools usually contain some protocol analysis capabilities that allow users to decode the captured data and analyze it.

Generally sniffers work by putting the Network Interface Card into promiscuous mode. In this mode the Ethernet Card listens to "all the traffic which is coming in". If the Network Interface card is not in Promiscuous mode, it ignore all traffic which is not intended for it.

Filtering can be done in two modes, on-line filtering and off-line filtering. On-line filtering is implemented in kernel while capturing the traffic. Off-line filtering is done after the captured data is stored on a disk. Both approaches have their own advantages and disadvantages. On-line filtering needs machines with more memory and processing, but helps in reducing the storage requirements. Also, memory and processing power are not a strict limitation for today's computer systems. On the other hand, if off-line filtering is done at high speeds, a lot of disk space would soon get used up. Hence, on-line filtering is preferred. The sniffers should not copy the whole traffic which is appearing in the network. In such cases the sniffer also dumps data belonging to untargeted users who happens to access and transfer data through the network during the sniffing time. This violates the privacy of untargeted individuals. Another problem with whole traffic capture is that, it needs a huge disk space and analysis of such huge dumps consumes considerable amount of resources.

Sniffers filter packets based on various levels of criteria. The first level of filtering is based on parameters like IP address range, port number range and protocol present

in the packet. This level of can be done on-line at kernel level. BPF [14] (Berkeley Packet Filter) is an in-kernel packet filter that filters packets based on network parameters like IP addresses, port numbers, protocol types and other information at fixed location in a packet. The next level of filtering is based on application specific criteria like email-id, URL, host name, etc. The final level of filtering is based on keyword (string) match in application payload. Most sniffers also support post-capture analysis and provide processing tools which retrieve useful information from stored data and present it in an interactive manner.

Many sniffers are available commercially and publicly. They come with different capabilities, and for different operating systems. *Tcpdump* [10] is a UNIX based sniffer that uses lipcap[9] library. This tool captures the packets based on Network parameters like IP address and port numbers. Also, it allows user to make some statistical analysis on captured data. It has limited filtering capabilities. *WinDump* [2] is a version of TcpDump for Windows. *Ethereal* [4] is a UNIX based sniffer. It has a rich set of protocol analysis capabilities. However, It has limited filtering capabilities compared to TcpDump. It provides GUI for viewing captured data. *Etherpeek* [5] is a tool mainly used to troubleshoot networks problems. It also decodes some application level protocols off-line. It works only in Ethernet networks. *Carnivore* [6, 7, 20] is a packet capturing system developed by FBI. This tool monitors the traffic based on wide range of application specific filtering criteria. It is also capable of monitoring dynamic IP address based networks.

These monitoring tools mainly focus on network management and trouble shooting aspects. Though they have good protocol analysis capabilities, they have limited on-line packet filtering capabilities.

1.2 PickPacket

PickPacket is a network monitoring tool developed at Indian Institute of Technology Kanpur. PickPacket is passive tool which means it doesn't inject any new packets into network and doesn't delay any packet to its destination. PickPacket can store selected packets from network traffic for further analysis. The filtering criterion

can be specified at several layers of network protocol stack. PickPacket can filter packets based on IP address in Network Layer, Port numbers in Transport Layer and application-level parameters like email-ids, user names, URLs, and search strings.

PickPacket has support for application layer protocols like SMTP[12], Telnet-Neeraj2002, HTTP[16], FTP[16], RADIUS[11], IMAP[21], POP[21], IRC[13], Yahoo chat and Instant messages[13]. Users can specify criteria for each application protocol separately.

PickPacket can capture the packets in two modes. The two modes of operation are called "PEN" and "FULL". The Full mode of operation stores the whole connection, while in PEN mode a minimal amount of information about the connection is stored. Using these features judiciously will protect the privacy of users. The data stored on the disk is analyzed by the off-line component of PickPacket called *Post-Processor* and it is made available to the user with separate files for each connection. The Web based tool called *DataViewer* shows a summary of all connections as well as provides the capability to view the details of each connection

1.3 Organization of the Report

This report describes the extension of PickPacket to support Internet Protocol Version 6 and compressed HTTP. Chapter 2 discusses the architecture and design of PickPacket. Chapter 3 discusses the brief overview of IPv6 and implementation details of IPv6 support in PickPacket. Chapter 4 discusses testing and performance results. The final chapter concludes the thesis with suggestions on future work.

Chapter 2

PickPacket: Architecture and Design

This chapter discusses the architecture and design of PickPacket. The design of each component is described briefly. Design and implementation issues are discussed in detail in References [1, 11, 12, 13, 16, 21]

2.1 Architecture

PickPacket can be logically viewed as aggregate of four components working in pipeline. These components are as follows.

- *PickPacket Configuration File Generator* is a JAVA based GUI for creating the configuration file. It can run either on a Windows or a Linux machine. The user can specify different filtering criteria for filtering the data which is subsequently written to configuration files in a format that is understood by the filter.

- *PickPacket Filter* is deployed on a Linux machine. It takes the configuration file as input. It reads packets from the network and stores those packets which match the criteria specified in the configuration file. Filtering is done at different levels based on criteria like IP addresses, port numbers and application layer information.

- *PickPacket Post-Processor* runs on a Linux machine. It processes the packets stored on the disk, and retrieves the meta-information from them and creates a directory structure which is used by the Data Viewer.

- *PickPacket Data Viewer* is a web based GUI. It can be deployed on the same machine where the post-processor is running. It takes the directory created by the post-processor as input and displays the data in an interactive manner. The user can access the captured data through a web server.

An architectural view of PickPacket is shown in Figure 2.1 in which each of these components along with data-flow is shown.

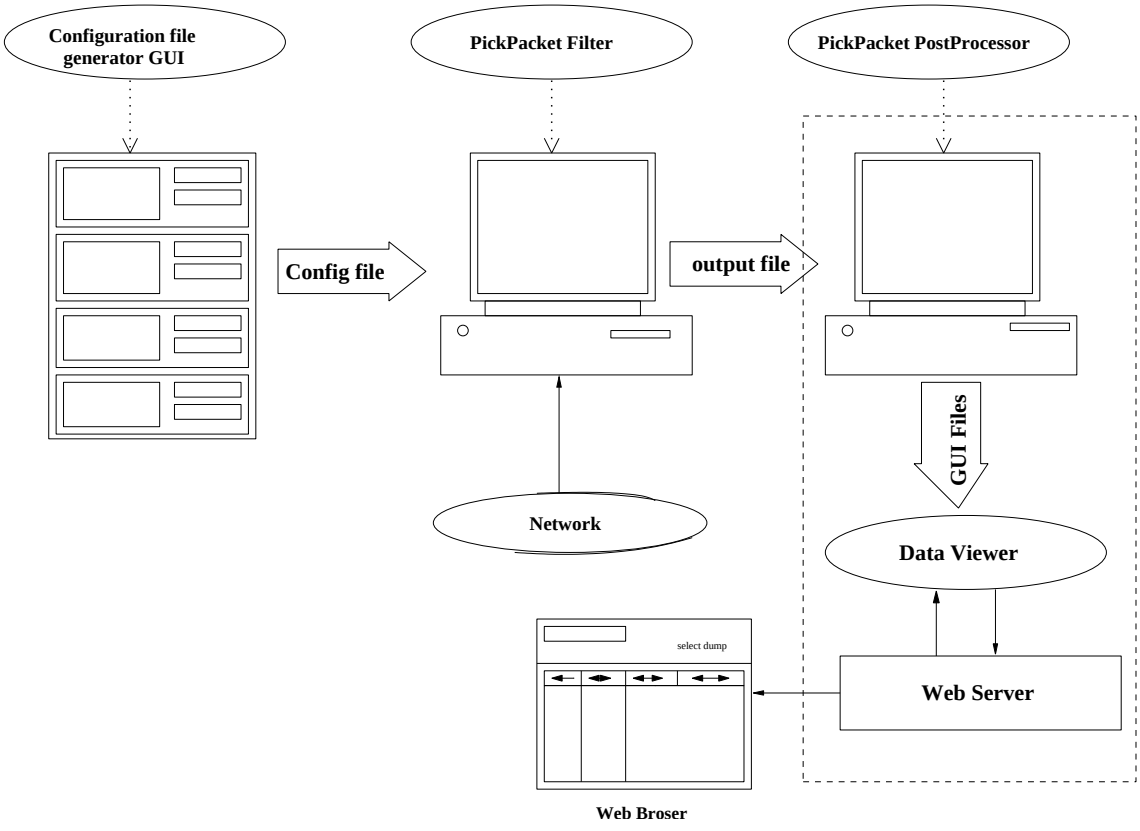


Figure 2.1: Architecture of PickPacket

2.2 Design

This section describes the design of each of the four components in PickPacket.

2.2.1 PickPacket Configuration File Generator

The PickPacket Configuration File Generator is a JAVA based graphical user interface. This helps the user to specify the criteria for filtering packets . This will generate two files, one containing the various levels of filtering rules given by the user and the other containing the configuration parameters. Both of these files have the same base name with a different extension. The format of these files is similar to XML. Sample configuration files are given in *Appendix A*.

The first file with an extension *.pcfg* contains parameters for filtering and output, and can be set by any normal user. It has three sections:

- The first section contains the details of output dump files generate by PickPacket Filter. It contains the output file prefix name. File-Prefix is suffixed with a time stamp at which the file is generated. The output file is changed periodically, so that the old output files can be transferred for further processing. The change of output file can be controlled either by specifying time or maximum size of the file.

- The second section contains criteria for filtering packets based on source and destination IP addresses, transport layer protocol, and source and destination port numbers. The application layer protocol that handles packets that match the specified criteria is also indicated. This information is used to filter based on network and transport layer information, and demultiplex packets among different application layer filters.

- The third section is divided into multiple subsections, each of which contains criteria corresponding to an application layer protocol. Based on these criteria the application layer content of the packets is analyzed. Here, the criteria for SMTP, HTTP, FTP, Telnet, IRC, IMAP, POP, YAHOO-chat and instant messenger protocols can be given. Each subsection contains application layer

protocol parameters like e-mail ids, URLs etc. The strings which will be matched in the payload of all packets and mode of operation of the filter, either PEN or FULL, is also mentioned for each application.

The second file with the extension *.bcfg* contains system related parameters to be set by an expert. It has two sections. The first section contains the number of simultaneous connections that can be monitored by the application filter. The second section contains the maximum number of packets that can be stored before the criteria matches. These values are used for the allocation of buffers by the PickPacket Filter. The default value is set to 500 for each application protocol. A very large value may cause the system to run out of memory. A small value may cause some connections to be dropped and not monitored. This value should be chosen based on filtering system capabilities.

2.2.2 The PickPacket Filter

PickPacket Filter reads packets from network and processes them to find whether they match the criteria specified by the user or not. Filter maintains the state for each connection until a criterion matches. Once a packet matches any criteria, it saves all the packets belonging to that connection. On the whole, filter stores the connections which match the criteria instead of individual packets. This section briefly describes the design of PickPacket Filter.

The PickPacket can filter packets at various levels.

- Basic filtering on network parameters (IP addresses, port numbers).
- Application level filtering based on criteria like host names, user names, etc.
- Filtering based on content present in the application payload.

The first level filtering is efficiently carried out by using in-kernel filters[14]. The in-kernel filter copies packets which match the network parameters from the kernel space to user space. Since the content of application can be best deciphered by the application itself, the second and third levels of filtering are combined. Figure 2.2 illustrates various levels of filtering.

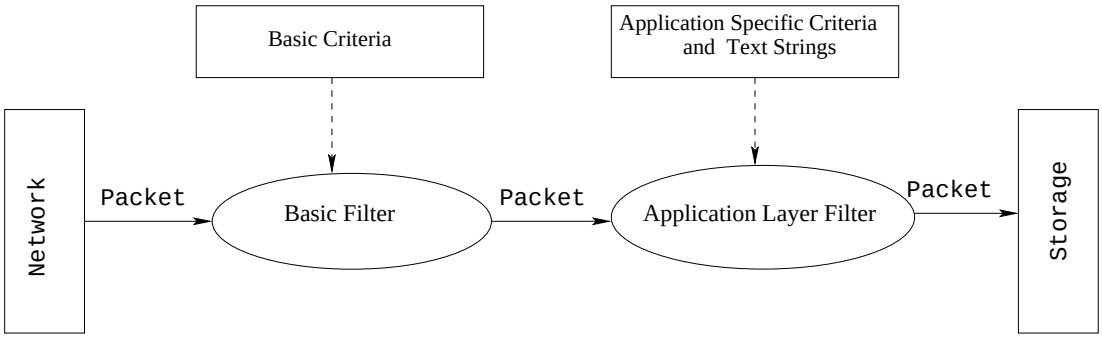


Figure 2.2: Filtering Levels

We have a separate software module for each application protocol. This design has the advantage that it is easy to add modules for new application protocols. In this design, a demultiplexer exists between the in-kernel basic filter and application level filters. Its main job is to direct the in-kernel filtered packets to one of the application level filters for further processing. This decision is made based on the rules present in the configuration file.

Application specific filtering extracts the application specific parameters from the packets and checks for matches. Keywords are matched in the application payload of packets. In case of communication over connection oriented protocol, this text search should handles situations where the desired text is split across two or more packets. The Filter also handles the case of out of order reception of packets. *TCP Connection manager* module exists between demultiplexer and application filter. This module handles out of order packets as well as deals with packet loss. Application layer filter can alert the Connection manager to maintain the sequence information for connections it in interested in.

Figure 2.3 shows the major modules in the PickPacket Filter.

The module *Initialize* is used for initialization of all the modules before filtering based on input configuration file. The *Output File Manager* module is responsible for storing packets on the disk. The *Filter Generator* module is used for generating the in-kernel BPF code based on network parameters present in configuration file. Application filters can call the functions to generate the filter code on-the-fly based

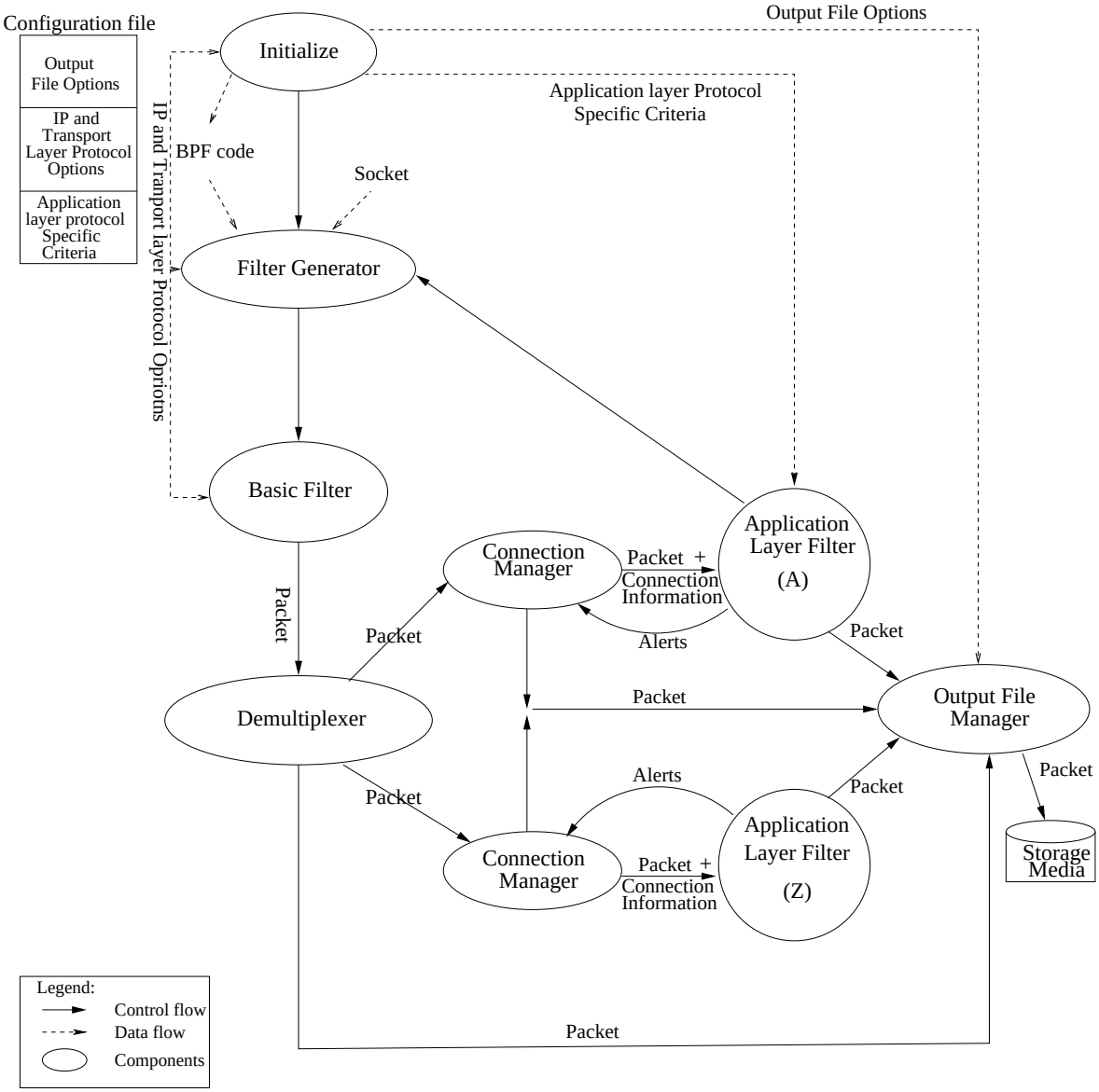


Figure 2.3: Basic Design of the PickPacket Filter

on changing parameters. For example in FTP during "PASSIVE" mode of file transfer, FTP filter[16] changes the BPF code. The *Demultiplexer* can also call the *Output File Manager* directly so that the filter can directly store packets without sending them to any application filter, if necessary. For example, if only IP addresses and port numbers are specified. The *Connection manager* can also directly store packets to the disk. This is needed if a criteria has already matched for a connection. More details of these modules can be found in [12]

The *output file manager* stores the output packets in *pcap* [9] file format. The packets stored in pcap file format can also be viewed using utilities like tcpdump. This standard format allows a user to use other tools for analysis of captured data.

The PickPacket Filter contains a text string search library. Application filters use this library for string matching in application payload. This library uses the *Boyer-Moore*[17] string matching algorithm.

2.2.3 PickPacket Post-Processor

PickPacket Post-Processor is an off-line component. It processes the packets stored by filter in output files. It separates the packets in output file into different connections. It extracts application protocol specific information from connections to display in data viewer. Post-Processor has to meet the following objectives.

- A connection is identified by 4-tuple. With the same 4-tuple there can exist more than one connection at different time intervals. Filter output file contains packets belonging to several sessions. Before attempting to extract any data from these files, sessions need to be separated.

- Information should be extracted from each connection. This information includes meta data from application payload. Meta data includes important fields and entities present in the data content belonging to an application layer protocol. For example Email-ids from SMTP, POP, IMAP and user name, filename from FTP connections.

PickPacket Post-Processor has three modules: the *Connection Breaker*, the *Session Breaker* and the *Meta Information Gatherer*. These are shown in Figure 2.4.

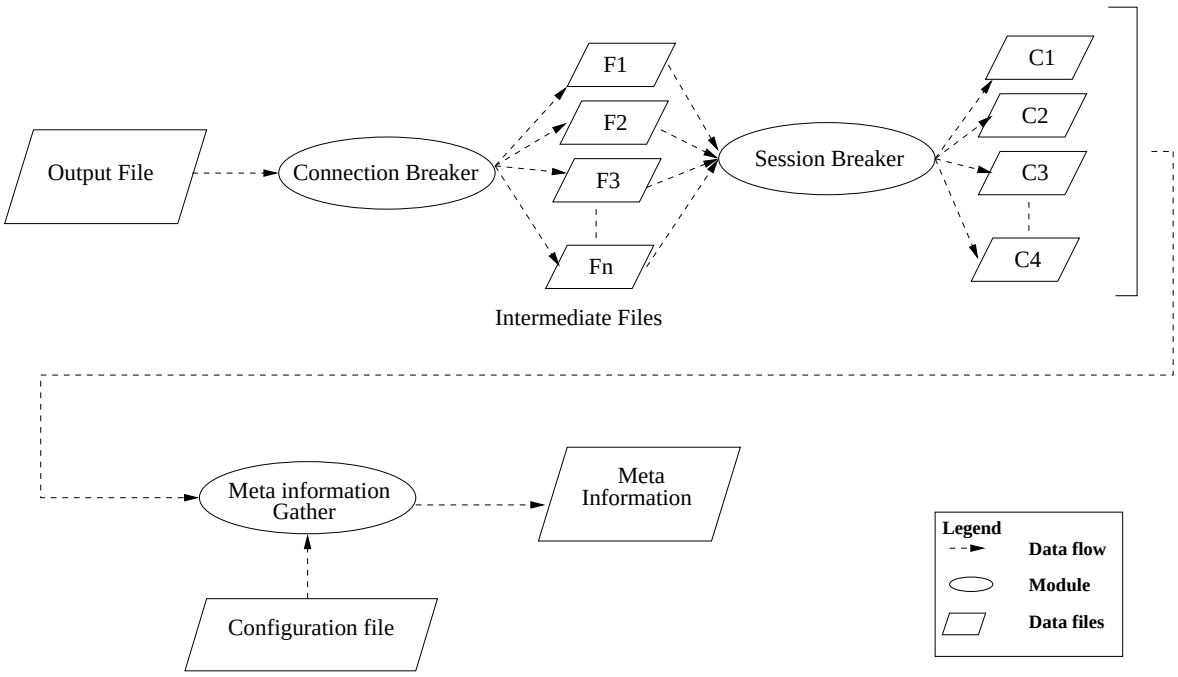


Figure 2.4: Post-Processing Design

Connection Breaker module reads packets from output file and separates them into different files based on the 4-tuple. The Connection breaker generates set of files with name as the 4-tuple, i.e., source and destination IP addresses and port numbers.

The Session Breaker module takes each file produced by Connection Breaker and splits it into multiple files, if it contains multiple sessions with the same 4-tuple. It then writes the packets in sorted order based on time stamp value, i.e., the time at which the packets were read from the network. Packets with same time stamps are sorted by their sequence number.

The *Meta Information Gather* module contains one meta data extractor for each application layer protocol. This module creates a separate directory for each connection. The connection directory contains several file with meta-information about connection. File named "tcpipinfo" contains the summary information about the session that includes IP addresses, port numbers, application level protocol and

matched keywords in the connection and count of each keyword. File named "appinfo" contains the meta data of the session that is specific to its application protocol. Example of this information includes host names and URL in case of HTTP connections.

2.2.4 PickPacket Data Viewer

PickPacket Data Viewer is a web based application used to show the PostProcessed information in interactive manner. This module is designed using PHP[8]. It is deployed on a web server. Web server accesses the data through PHP scripts and serve the user requests. Data viewer can be deployed either on the same machine where post-processing is done or on a different machine.

Data viewer is provided with an authentication screen. The user can login in two modes “ADMIN mode” and “User mode”. Admin can add users, delete users and change their passwords. Users can only change their passwords. Post Processor output directory is placed in a fixed path that is known to Data Viewer. After a user has logged in, DataViewer lists all the directories present in that path. For any output directory selected by user, it lists summery of all the connections, including MAC addresses, IP addresses, port numbers, Transport Protocol, Application Protocol, RADIUS User and keywords matched from the output of Post-Processor. Connection list can be sorted on any of these fields. User can change his configuration to show or remove some of these fields.

User can search among the captured connections. The search criteria include network parameters, application parameters and keywords. User can search on all the fields that he specified in the configuration file. When a user sets a connection filter for displaying the connection only those connections that match the criteria will be displayed. User can get back all the connection by setting the connection filter to null.

On selecting a connection from the list of connections, the details of the connection are shown. The details include network parameters and application parameters. The Data Viewer provides user with facilities like viewing captured e-mails, web pages accessed, etc. The dialogue between communicating hosts can also be seen in

a dialogue window. The configuration file used for the filtering can also be viewed.

Chapter 3

Implementation of IPv6 in PickPacket

This chapter discusses the design and implementation of the IPv6 traffic handling in PickPacket. First, a brief overview of the IPv6 protocol is given, with a focus on those features that are important for designing and implementing the filter. Then the modification in different PickPacket components to support IPv6 are discussed.

3.1 IPv6 Protocol Overview

IPv6 [19], also referred to as IP Next Generation, is designed as the successor of IPv4 [3]. IPv4 was designed more than twenty five years ago. During this period, Internet has grown beyond anyone's expectations and the requirements of users have changed. To satisfy increased demand for IP addresses and newer user requirements, in 1990, Internet Engineering Task Force (IETF) started to work on a new version of IP. The idea was to design a protocol which would never run out of addresses, would solve several other problems, and be more flexible and efficient. The major changes from IPv4 to IPv6 are expanded addressing capabilities (128 bit address size), simplified header structure, better support for options, QoS and security.

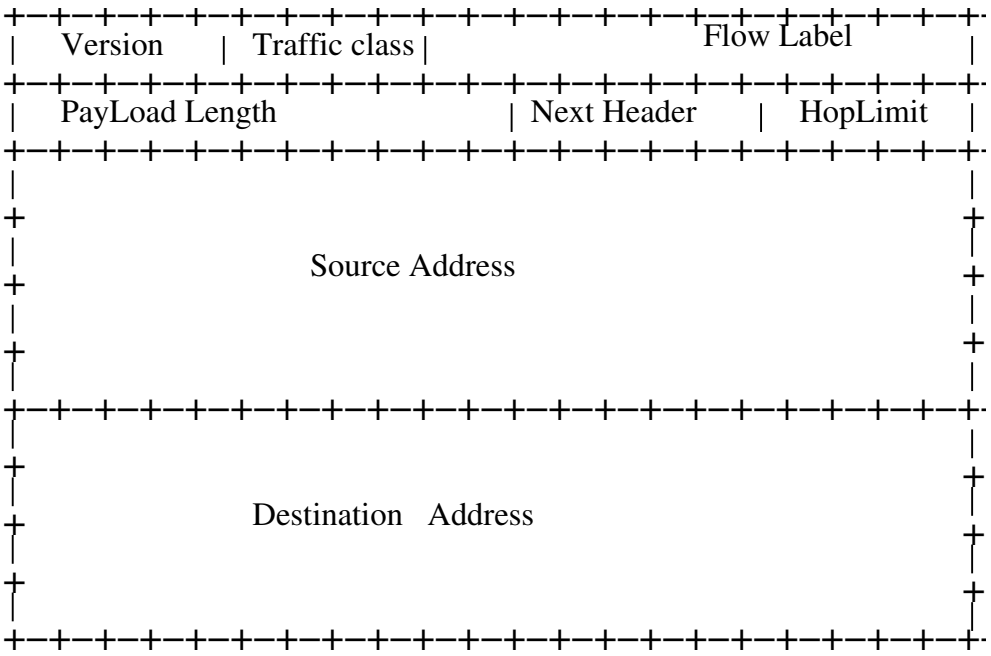


Figure 3.1: IPv6 header

3.1.1 IPv6 header

The basic header used in all IPv6 packets is shown in Figure 3.1. The following are the main features of IPv6:

- Fixed 40 bytes header size - whereas the IPv4 header-length is variable, but is at least 20 bytes without extra options.
- Source and destination addresses are 128-bit compared to 32-bit addresses in version 4.
- Six fields in IPv6 header compared to ten fields in IPv4 header.
- No checksum
- 64-bit aligned fields

IPv6 header structure is very flexible because it can include extra essential information very easily. When more information is needed another header is linked

through the next header field as an extension header. Currently there are six types of extension headers defined, but more can very easily be added in the future. The size of all extension headers must be integer multiple of 8 bytes long, so alignment is optimal for most platforms.

The interested reader who need more detail can find it in the IPv6 specification, [19] and the documents that describe transmission of IPv6 packets across diverse network technologies [15].

3.2 Upgrading Configuration File Generator

The configuration file is a text which contains the rules specified by the user in a fixed format. The details of configuration file are explained in Section 2.2.1. To specify IPv6 criteria a new section has been added to this file. A sample file with new section is given in *Appendix-A*.

This new section contains rules for filtering packets based on source and destination IPv6 addresses, transport layer protocol , and port numbers. These rules are referred to as *basic_criteria_6* in the rest of this report. Every rule in basic_criteria_6 also specifies the application layer protocol to which the packets belongs. Configuration file has support for multiple such IPv6 rule tuples for different application layer protocols in the same configuration file so as to simultaneously filter packets belonging to these protocols.

IPv6 addresses can be either specified as individual addresses, or as a prefix. In case of individual addresses, user can use any legal format of IPv6 addresses representation. In case of IPv6 address prefix a length is also to be provided by the user.

3.3 Upgrading Filter

PickPacket Filter has to capture IPv6 packets which are flowing across the network according to user specified IPv6 criteria. Provision has been made in the Configuration File Generator to specify proper IPv6 criteria which include network parameters

like IPv6 address prefix and port numbers. Filter will monitor IPv6 traffic based on this criteria.

Presently a very small portion of network traffic uses IPv6. In the network both IPv6 and IPv4 traffic will appear. PickPickat should handle both these protocols simultaneously. The application layer protocols won't change in IPv6 traffic. Application filters designed in PickPacket for IPv4, can be directly used for IPv6 communication with little modifications.

The upgraded PickPacket filter can read both IPv4 and IPv6 traffic and matches them against the user specified criteria. The modified architecture of PickPacket Filter is shown in Figure 3.2

During *initialization* phase IPv6 criteria are passed to other modules of the system. *filter generator* is responsible for generating BPF filter code and passing it to the kernel for in-kernel filtering.

In-kernel packet filtering for IPv4 traffic using network parameters is simple because generating BPF expressions for IPv4 traffic is very easy. IPv4 packets have their network parameters values at fixed byte offset in the packets. These data values can be extracted from packet by using following syntax

proto [expr : size]

proto can be one of IP,TCP,UDP or Ether and indicates the protocol layer for the index operation. Expressions involving TCP and UDP and other upper-layer protocols can not be applied to IPv6 because of uncertainty in its packet structure. There is a chance of having extension headers in between IPv6 header and TCP or UDP header. TCP, UDP or upper-layer protocol information cannot be extracted easily because of this uncertainty. Because of this reason, in the newly designed architecture, basic filtering of IPv6 traffic is shifted to user level. The *demultiplexer6* reads IPv6 packets transmitted on the network and passes them to *application protocol filter.* The behavior of *application protocol filter* is same for both IPv4 and IPv6 connections.

When a packet is received, based on the type of packet it is either passed to *demultiplexer* or to IPv6 demultiplexer(IPv4) i.e *demultiplexer6(IPv6).* The PickPacket filter supports filtering of packets belonging to different IP versions and

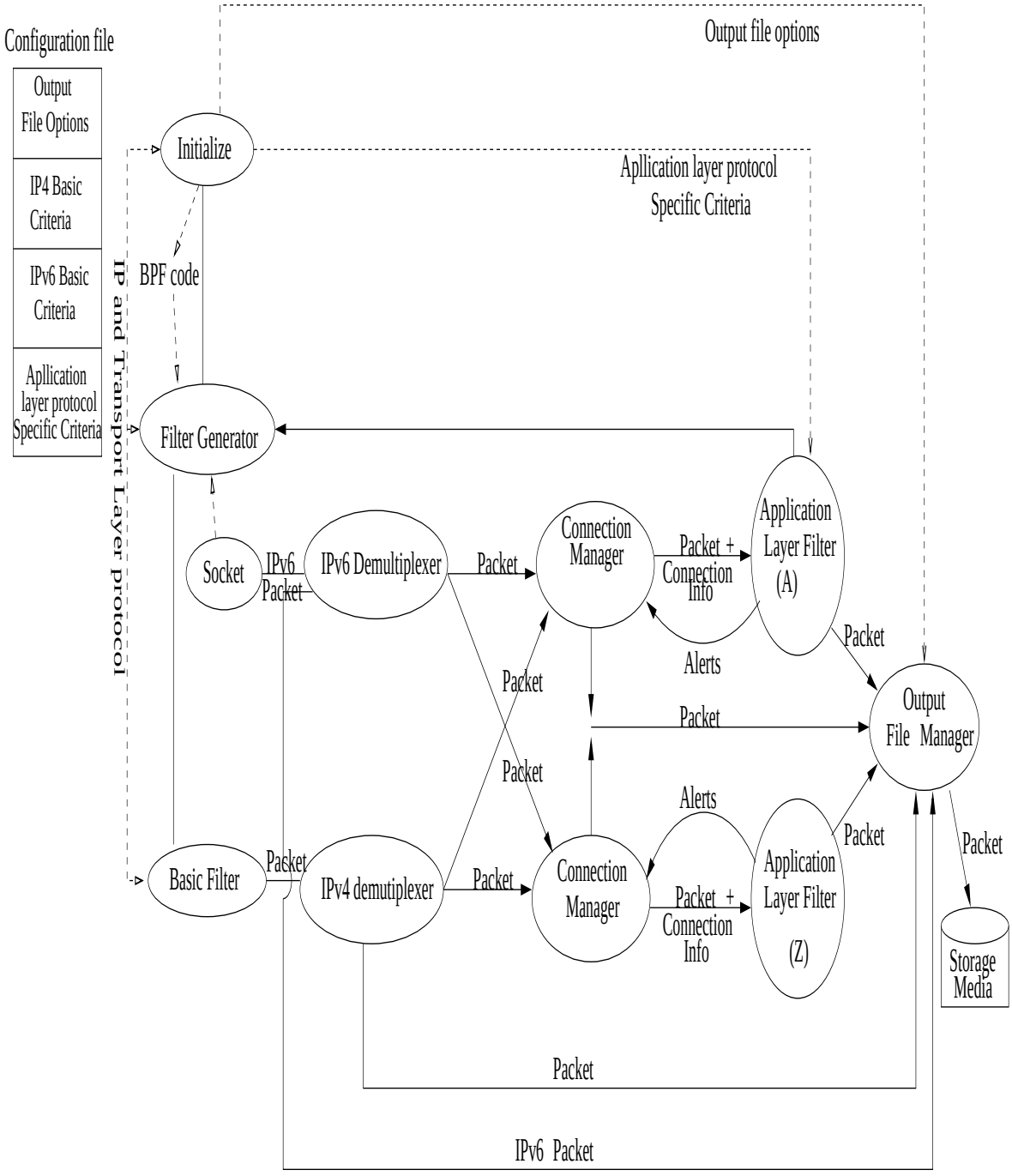


Figure 3.2: New Design of the PickPacket Filter

different application protocols simultaneously. Based on IPv6 information present in the configuration file, the *demultiplexer6* determines the application protocol to which this packet belongs and passes it to the connection manager corresponding to the transport protocol header present in the packet.

The rest of the section provides a detail description of changes made to existing modules, newly created modules and their roles in filtering IPv6 packets.

3.3.1 Initialization

During initialization the packet filter reads the configuration file and initializes the various modules of the system. At this time *basic_ criteria_ 6* are read from the configuration file. Based on the application protocol name specified in these criteria, initialization functions of corresponding application protocols are called if they have not already been initialized by IPv4 criteria. For example, if a user specifies HTTP protocol criteria for both IPv4 and IPv6, then initialization functions of HTTP are executed at the time IPv4 basic criteria is read. If any application protocol rule is specified in IPv6 and not specified in IPv4 rules then that application protocol functions are called at IPv6 initialization time. These functions allocate memory required for maintaining state information about the connections of interest. Also, the initialization functions of the connection managers corresponding to the transport layer protocol specified in the basic_criteria_6 are called. These functions allocate memory required for maintaining connection information for connections that are of interest to application protocol filter. After this initialization, the IPv6 addresses and prefixes, transport layer protocol name and port number range is passed to IPv6 demultiplexer, i.e., *demultiplexer6*. The *Demultiplexer6* saves this information by making an entry into its table.

3.3.2 The IPv6 Demultiplexer

Packets after being copied from the network are checked for their version of internet protocol. Based on its version, packet is sent to either IPv4 demultiplexer or IPv6 demultiplexer. These demultiplexers determine application protocol the packet

belongs to and pass it to the respective filtering modules.

For demultiplexing IPv6 packets the demultiplexer maintains a table of tuples containing the IPv6 basic criteria specified in the configuration file. If any processing of data content is desired (for example, keyword search) then the packet is passed to connection manager routine for the corresponding transport layer protocol. Otherwise packets are directly written to disk by passing them to output file manager. IPv6 demultiplexer checks every packet against the entries in the table. Packet which do not match any of these criteria are discarded by it. In IPv6 case, entire basic filtering is being done at IPv6 demultiplexer, whereas in IPv4 some part of basic filtering is done as in-kernel filtering and other part of filtering at demultiplexer.

After reading the packet from the network, it is saved in a *My_ Packet* structure. The size of packet in bytes and time at which packet was received is also saved in this structure. Based on the *Type* value in the Ethernet header, the packet is sent to one of the demultiplexer. If type value is equal to 0x0800 (packet belongs to Internet Protocol version 4), then packet is sent to IPv4 demultiplexer. If type value equals 0x86dd (packet belongs to Internet Protocol version 6), then packet is sent to IPv6 demultiplexer. IPv6 demultiplexer maintains a table of tuples in structure *dmplx_ table6*. Every tuple comprises of the following fields - 128-bit source and destination IP addresses, port numbers, transport layer protocol name, a reference to a connection manager's function, and a reference to the memory allocated to the application protocol filter. This table is maintained in two parts, *static_ table* and *dynamic_ table*. The *static_ table* stores the IPv6 basic criteria specified in the configuration file. The *dynamic_ table* stores those IPv6 basic criteria which may be added by other modules while filtering.

When IPv6 demultiplexer receives a packet, it retrieves 128-bit source and destination IP addresses and port numbers from packet. In IPv6 packet 128-bit source and destination addresses are available at fixed location from the beginning of packet. The demultiplexer searches for a matching tuple in its static and dynamic tables. A match occurs if source and destination IP addresses and port number present in the packet lie in the corresponding ranges present in the tuple, with transport layer protocol being the same. A reverse check is also made for capturing packets sent by

either of the communication hosts. On successful match, the packet is either passed to the TCP connection manager or is passed to the default output file manager based on application protocol name specified in the configuration file.

The IPv6 demultiplexer also allows addition and removal of filtering tuples from its dynamic table through the *dmplx_add_to_dynamic_table6* and *dmplx_rem_from_dynamic_table6* functions. These function can be called from other modules when they require modifications to basic IPv6 filtering.

The IPv6 packet filter processes transport layer protocol headers of two protocols- TCP and UDP. The connection manager for TCP has been modified to handle IPv6 packets. Packets belonging to UDP are written to disk by the demultiplexer in either *Pen* or *Full* mode.

3.3.3 Connection Manager

The functionality of Connection manager of IPv6 connections is similar to Connection manager of IPv4 connections. This Connection manager maintains state specific to the transport layer protocol of the currently monitored connections . The packet filter can filter packets belonging to two transport layer protocols - TCP and UDP. UDP is a connection-less protocol hence no connection information needs to be maintained for packets belonging to it. Thus a connection manager for UDP has not been implemented. State information for connections belonging to TCP needs to be maintained as it is a connection oriented protocol. We have modified the connection manager for handling IPv6 and TCP packets are filtered based on application layer data content present in them.

The TCP connection manager is modified to check for out of sequence IPv6 packets and pass this information to the application protocol filters. For this, it maintains connection information similar to IPv4 connections in memory. The reference to this memory is received from demultiplexer. Each application layer protocol has separate memory area as shown in table 3.1. Each of these areas consists of the following fields - a pointer to application filter function, a pointer to write-packet function, a pointer to free_function which removes connection information, a pointer to active_list which points to connection information of different connections, a pointer

to free_list which points list of free connection information nodes, a variable called last_flush_time to store the time of when flush function was last called.

filter_func
write_packet
free_func
last_flush_time
active_list
free_list

Table 3.1: TCP Connection Manager's Memory

Connection manager retrieves this information when packet belonging to this connection is received. Connection manager releases connection information when FIN packet is received or when the timer expires. Every connection information node has a *last_used* variable to remember time when the packet belonging to this connection was read by the packet filter. Connection information contains some other information as shown in table 3.2. This information includes 128-bit source and destination addresses, source and destination port numbers, source and destination sequence numbers of the source and destination packets of last received, connection state, application protocol state information, and dump flag.

The *dump_flag* value is set or unset by application layer filters. If this value is set by any application layer filter, then packet is sent to output file manager by calling the function pointed to by *write_packet*. When application layer filter needs to monitor the connection then it alerts the connection manager by calling functions *tcp_alert_current* and *tcp_alert_new*. Similarly, if the application layer filter determines that connection information for the connection should be released then the functions *tcp_forget_current* or *tcp_forget_conn* provided by Connection manager is called.

3.3.4 Application Protocol Filters

There are a few changes common to all application protocol filters. In these filters small changes are made where there is a need to store source and destination IP

128-bit source IP address
128-bit destination IP address
source port number
destination port number
source sequence number
destination sequence number
highest source sequence number
highest destination sequence number
connection state
last_used
Application layer protocol filter state information
dump_flag

Table 3.2: TCP Connection Information

addresses. In all application protocol filters there is need to find the application data offset and source and destination sequence number in the packets. Basically packets are processed to find data offset and TCP headers by skipping any extension headers if present.

While analysing IPv6 HTTP traffic we found that almost all the traffic is compressed. Use of compression in IPv4 is also increasing. Hence we modified the HTTP filter to handle compressed HTTP for both IPv4 and IPv6. The following subsection describes changes made to filter to handle compression.

3.3.5 HTTP Filter

- *What is HTTP Compression*

The volume of data on the web is increasing very rapidly because of increasing functionality in web-pages. With greater quantities of Images, Java script and DHTML than ever, the HTML Payload per-page is growing significantly. Data and content

will remain the largest percentage of web traffic and majority of this information is dynamic so it one can't use conventional caching technologies. Compression is now being used on HTML to address this and is implemented by encoding the Body of the HTTP message. This is supported by HTTP 1.1 [18] but not HTTP 1.0[22].

The main use of introducing HTTP compression is to improve web performance by having the server send compressed files to clients and having the browser uncompress before displaying. The compressed data at server can be generated in two ways, dynamically and pre-compressed. Dynamic Content Accelerator compresses the data on the fly (useful for database-driven sites, etc). Pre-compressed text data is pre-generated and stored on the server (.html.gz files, etc.,).

Since a majority of network traffic these days is HTTP, compressed files in HTTP improves usage of network bandwidth and user ends up seeing the document very fast as compared to sending uncompressed HTML.

HTTP compression uses standard gzip and deflate compression algorithms to compress XHTML, CSS, and JavaScript to speed up web page downloads and save bandwidth. HTTP compression, also known as content encoding, is a publicly defined way to compress the content transfered from web servers to web browsers. HTTP 1.1 protocol is well designed with standard methods to deliver the compressed content from web server to browser. The modern browsers that support HTTP 1.1 can decompress compressed files automatically

Browsers and servers have a brief conversation before actual transfer of compressed data. This conversation is done using HTTP headers. The following HTTP request header message shows how a compression-aware browser informs server that it prefers to receive compressed content.

GET / HTTP/1.1
Host: www.cricinfo.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.5)
Gecko/20031007 Firebird/0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Encoding: **gzip,deflate**
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

In the above header, the header *Accept-Encoding: gzip,deflate* tells the server about the browser's preference and capability about the compressed data.

When web server receives request header with *Accept-Encoding : gzip*, it delivers the requested document with the encoding accepted by the client. The following is a sample response header message for the compressed content.

HTTP/1.1 200 OK
Date: Thu, 04 Dec 2003 16:15:12 GMT
Server: Apache/2.0
Vary: Accept-Encoding
Content-Encoding: gzip
Cache-Control: max-age=300
Expires: Thu, 04 Dec 2003 16:20:12 GMT
Content-Length: 533
Content-Type: text/html; charset=ISO-8859-1

Now the client knows that the server supports gzip content encoding by the header line *Content-Encoding: gzip* and it also knows the size of the file *content-length* . The client downloads the compressed file, decompresses it, and displays the page.

Any text string search algorithm will works only on the uncompressed document. One solution to this problem is to store all the packets into a buffer until the whole file is captured. We can then decompress this file by using standard decompression or unzip tools and then apply the string searching algorithms on decompressed data. This method is inefficient and not compatible with architecture of PickPacket.

PickPacket needs an efficient solution which decompresses the packets on the fly and is able to perform string search in decompressed content.

We now describe the implementation details of compressed HTTP handling.

HTTP application layer filter parses the response and request headers separately. When it comes to parsing of response header, it first checks whether there is any *Transfer-Encoding = chunked* header in the response header. If it exists then PickPacket calls *parse_transfer_encoding* to unchunk the chunked data. Otherwise directly goes to *parse_content_encode*

In *parse_content_encoding* it checks whether *Accept-Encoding : gzip* header field is present. If this header field is found, it initializes the current HTTP connection's *gzFile* structure object by calling *gzopenstr* method with first packet's compressed content. For every packet in the connection it calls *gzreadstr* methods with the compressed content as parameter. This function takes part of compressed data and its length as input parameters and returns equivalent uncompressed data and its length. It remembers the data structures to decompress next part of file coming in next consecutive packet. It successfully decompresses packet only if they are received in order. It fails to decompress a packet if packets before it have been lost.

gzopenstr method is invoked only for the first part of compressed content received. This method initializes the data structure with default values and checks for gzip file header values. It check the presence of header values one after another and finds offset of compressed data block. *Appendix-B* shows sample gzip file header content.

On successful decompression of compressed HTTP data, PickPacket sends uncompressed data to string search methods to match user specified keywords. If a match occurs, then the packet is stored to disk for further analysis.

3.3.6 Unmodified Components

Filter Generator module generates the in-kernel BPF filter code for basic criteria and attach this code to socket. In case of IPv6, the whole basic filtering is done by the demultiplexer. There are no changes in filter generator module. The same set of *Application layer protocol filters* handles both IPv4 and IPv6 traffic. The *Output File Manager* module writes packets to disk in *pcap* format.

3.4 Upgrading PostProcessor

PickPacket PostProcessor is modified to group the IPv6 packets into connections. It also collects the IPv6 meta information like 128 bit address and stores it in metafile which is understandable to PickPacket DataViewer. It also finds any data loss occurred in an IPv6 connection by using sequence number information.

PostProcessor also needs to handle compressed HTTP data. Dataviewer need some meta-information to show the connection data along with criteria because of which connection is selected. If that criteria is keyword, then it displays that keyword and frequency of that keyword in the connection. In order to get keyword and to find its frequency in compressed HTTP data. PostProcessor has to uncompress the packet data and apply string search algorithm. Finally after processing of compressed HTTP data, PickPacket stores it in compressed form only. In Web based DataViewer, When we request that content, browser will get that from the disk and uncompress it and displays it to the user.

3.5 Upgrading DataViewer

There are no changes to this component of PickPacket. In case of IPv6 connections, the PostProcessor writes 128 bit address into the meta information file. Since DataViewer treats addresses as strings, no changes are needed to display IPv6 connections. For compressed HTTP, since the PostProcesor stores uncompressed version of compressed content, the DataViewer remains unaffected.

Chapter 4

Correctness Verification and Performance Evaluation

In this chapter we discuss the experiments conducted to test the PickPacket after adding supporting for IPv6 traffic and for handling compressed HTTP. In the performance evaluation we have tested whether the software is able to handle traffic at line speed on 100 Mbps Ethernet.

4.1 Correctness Verification

The main aim of this experiment is to test whether the filter and other components are working correctly. That is, the packets stored on the disk must correspond to a session which has matched a criteria mentioned in the configuration file, and one should be able to post-process and view such connections.

4.1.1 IPv6 Filtering

In testing for IPv6 monitoring, we first verified that the filter captures IPv6 packets according to the criteria specified in the configuration file and then, the whole software was tested for the correctness.

In initial phase, we tested IPv6 filter by specifying various criteria for filtering IPv6 connections. These criteria includes various IPv6 address prefix values and

application specific criteria. To test IPv6 filter, we have taken IPv6 connection traces from traffic using IITK IPv6 Proxy server. The test setup in the lab included two machines. One machine ran PickPacket Filter, and the other machine generated IPv6 traffic by replaying packets taken at IITK IPv6 Proxy server. We ran the filter with each of the configuration files for the same traffic. For each configuration, we knew what connections match the criteria and thus should be stored. We found that the filter was storing the expected data in each test. In the next phase, we tested PickPacket with combined traffic of both IPv4 and IPv6 to check whether it is filtering both types of packets simultaneously or not. The behavior of filter on combined traffic with the specified configuration was as expected. From this we concluded that PickPacket filter is working properly. Post-processor is tested to ensure that the reconstruction of IPv6 connection and meta data extraction is done correctly.

HTTP module is tested after adding the module that handles compressed HTTP traffic. To test this, on one machine we installed a web server with gzip-enabled option. From another machine, we requested compressed content from web server. We captured these packets and displayed the uncompressed content of each connection. We then collected two kinds of keywords from the uncompressed page. In one case, the keywords appeared in one packet and in the other case, the keywords were split across consecutive packets. We again ran the above test with these keywords as criteria. We found that HTTP compression module is working correctly by dumping only those connections which have the given keywords in compressed form. This means that it is able to perform on the fly uncompress and string searching in uncompressed content.

4.2 Performance Evaluation

The experiments conducted for performance evaluation are similar to experiments described in [12, 21]. PickPacket filter is run on two machines simultaneously on the same network. First one simply counts the number of packets and second one filters based on the user specified criteria. Two instances of PickPacket filter were

run on two identical machines. The configuration of both PCs was: Intel Pentium 3.6 G Hz CPU, 2GB RAM , Red hat Linux 9 with 2.4.20-8 kernel and connected to 100Mbps Ethernet port. On one machine, PickPacket was run without any application filtering criteria. It simply wrote the packets to /dev/null. This is referred to as *counter sniffer*. On second machine we ran PickPacket with configuration file containing filtering criteria. This is referred to as *filtering sniffer*. On this machine PickPacket was configured to write packets to disk, if a criteria is matched.

The *filtering sniffer* was run with configuration criteria for all protocol supported by PickPacket and basic IPv6 criteria. Traffic containing all protocols was generated and significant portion of the traffic included compressed data. Almost all IPv6 HTTP traffic is compressed traffic. Both the sniffers were started manually and run for some time. In our experiment, we observed that the number of packets captured by the two sniffers differed by a small value. This small difference in number of packets is due to delay while starting the sniffers manually. In this way, we verified that the modified PickPacket Filter was able to capture traffic at 100 Mbps Ethernet speed.

Chapter 5

Conclusions and Future Work

PickPacket is a useful tool that can capture packets flowing across the network and store some of the packets which match the user specified criteria. The criteria for filtering of packets ranges from network parameters like IP addresses and port numbers to application level parameters like User names, Email-Ids, URLs, etc. The design of tool is modular, flexible, and efficient. Judicious use of PickPacket can help protect the privacy of individuals by capturing only the packets which match the user-specified criteria. The storage format of packets is standard *pcap* format which is used by many freely and commercially available tools. This adds the flexibility of using other processing and rendering tools.

This thesis discusses the filtering of IPv6 packets flowing across the network by PickPacket with a special focus on IPv6 traffic. PickPacket allows the filtering of packets on the basis of criteria specified by the user both at the network and the application level of the protocol stack.

Also discussed is the need of handling compressed HTTP traffic in PickPacket and the solution, which is to decompress the compressed data on the fly to do string matching, is described.

Experiments are conducted to check the performance of IPv6 filter of the PickPacket. These experiments show that IPv6 filter successfully captures and filters packets on the basis of user specified criteria. We have also conducted experiments

on HTTP filter with compressed packets. PickPacket software, after incorporating all these extensions, have been shown to perform at line speed using 100Mbps Ethernet.

5.1 Future Work

PickPacket currently supports SMTP, POP, IMAP, IRC, Yahoo, HTTP, FTP and Telnet application level protocol. There is always scope for extending PickPacket to support other application level protocols. Presently VoIP is also used for illegal activities. Current PickPacket does not support VoIP interception.

Currently PickPacket does in-kernel filtering for IPv4 traffic, while IPv6 basic filtering is implemented in demultiplexer. Moving basic filtering of IPv6 to kernel will improve the performance of PickPacket.

Bibliography

[1] Aditya, S. P. "Pickpacket: Design and Implementation of the HTTP postprocessor and MIME parser-decoder", Dec 2002. BTP, Department of Computer Science and Engineering, IIT Kanpur, http://www.cse.iitk.ac.in/research/btp2003/98316.html.

[2] Degioanni, L., Risso, F., and Viano, P. "Windump". http://netgroup-serv.polito.it/windump.

[3] DoD. "Internet Protocol Specification". Tech. rep., 1980. http://www.ietf.org/rfc/rfc760.txt.

[4] et al., G. C. "Ethereal". Available at http://www.ethereal.com.

[5] "Etherpeek nx". http://www.wildpackets.com.

[6] Graham, R. "carnivore faq". http://www.robertgraham.com/pubs/carnivore-faq.html.

[7] "How Carnivore Works". http://www.howstuffworks.com/carnivore.htm.

[8] "PHP:Hypertext Preprocesor". http://www.php.net.

[9] Jacobson., Leres, and McCanne. *"pcap - Packet Capture Library"*, 2001. Unix man page.

[10] Jacobson, V., Leres, C., and McCanne, S. "tcpdump : A Network Monitoring and Packet Capturing Tool". Available via anonymous FTP from ftp://ftp.ee.lbl.gov and www.tcpdump.org.

[11] Jain, S. K. "Implementation of RADIUS Support in Pickpacket". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2003. http://www.cse.iitk.ac.in/research/mtech2001/Y111122.html.

[12] Kapoor, N. "Design and Implementation of a Network Monitoring Tool". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Apr 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html.

[13] Kiran, A. "Supporting Chat Protocols YAHOO ,IRC in PickPacket". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, June 2004.

[14] McCanne, S., and Jacobson, V. "The BSD Packet Filter: A New Architecture for User-level Packet Capture". In *Proceedings of USENIX Winter Conference* (San Diego, California, Jan 1993), pp. 259–269.

[15] M.Crawford. "Transmission of IPv6 Packets Over Ethernet Networks". Tech. rep., 1998. http://www.ietf.org/rfc/rfc2464.txt.

[16] Pande, B. "Design and Implementation of a Network Monitoring Tool". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, Sep 2002. http://www.cse.iitk.ac.in/research/mtech2000/Y011104.html.

[17] R., B., and Moore, J. "A fast string searching algorithm". In *Comm. ACM 20* (1977), pp. 762–772.

[18] R.Fielding, Irvine, G. M., and Frystyk. "Hypertext Transfer Protocol - HTTP/1.1". Tech. rep., 1999. http://www.ietf.org/rfc/rfc2616.txt.

[19] S.Deering. "Internet Protocol, Version 6 (IPv6) Specification". Tech. rep., 1998. http://www.ietf.org/rfc/rfc2460.txt.

[20] Smith, S. P., Jr., H. P., Krent, H., Mencik, S., Crider, J. A., Shyong, M., and Reynolds, L. L. "Independent Technical Review of the Carnivore System". Tech. rep., IIT Research Institute, Nov 2000. http://www.usdoj.gov/jmd/publications/carniv_entry.htm.

[21] SUDHEER, V. "Supporing Mail Protocols POP , IMAP in PickPacket". Master's thesis, Department of Computer Science and Engineering, IIT Kanpur, June 2004.

[22] T.BERNERS-LEE, R.FIELDING, I., AND FRYSTYK. "Hypertext Transfer Protocol - HTTP/1.0". Tech. rep., 1996. http://www.ietf.org/rfc/rfc1945.txt.

Appendix A

Sample Configuration Files

A.1 Configuration File with Filtering Criteria (*.pcfg*)

```
# This is a sample configuration file with filtering criteria
# A hash(#) is used for comments
# This file has several sections
# Sections start and end with tags similar to HTML.
# Tags within sections can start and end subsections or can be tag-value pairs.
# All the tags that are recognized appear in this file.
# First Section specifies the sizes and names of the dump files
# The Second Section specifies the source and destination IP ranges
# the source and destination ports, the protocol and the application
# that should handle these IPs and ports
# The third Section specifies the IPv6 basic criteria, source and destination
#  IPv6 address prefix values,port numbers,tcp and application layer protocols
# The next sections describe the application specific
# input criteria.

#**************First Section*****************************

<Output_File_Manager_Settings>
```

```
   <Default_Output_File_manager_Settings>
# File_Prefix is the name used to generate the dump filename suffixed with
# the time stamp at which the file is created
       File_Prefix=livedump
# If the dump file has to be changed based on size then this field is having
#     value yes
       Size_Based=yes
# This field exists when the Size_Based is yes this tell the size of dump
#         file in Mega Bytes
       File_Size=100
# Time_Based attribute tells if the change of dump file is based on time also
       Time_Based=yes
# This field exists when the Time_Based is yes this tell the time period in
#  minutes
       Time_Period=60
   </Default_Output_File_manager_Settings>
</Output_File_Manager_Settings>
#**************End of First Section************************

#**************Second Section*****************************
# The basic criteria here are for the Device and
# SrcIP1:SrcIP2:DestIP1:DestIP2:SrcP1:SrcP2:DestP1:DestP2:ProtoA:App

<Basic_Criteria>
  DEVICE=eth0
  Num_Of_Criteria=9
  Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:25-25:TCP:SMTP
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:20-20:TCP:FTP
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:21-21:TCP:FTP
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:23-23:TCP:TELNET
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:80-80:TCP:HTTP
```

```
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:110-110:TCP:POP
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:143-143:TCP:IMAP
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:5050-5050:TCP:YAHOO
Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:6667-6667:TCP:IRC
</Basic_Criteria>
#**********End of Second Section***************************>

#**************Third Section*****************************
# The IPv6 basic criteria here are in the form
# SrcIP1:SrcIP2:DestIP1:DestIP2:SrcP1:SrcP2:DestP1:DestP2:ProtoA:App

<Basic_Criteria6>
  Num_Of_Criteria=9
Criteria=::/0;::/0;1024-65535;25-25;TCP;SMTP
Criteria=::/0;::/0;1024-65535;23-23;TCP;TELNET
Criteria=::/0;::/0;1024-65535;80-80;TCP;HTTP
Criteria=::/0;::/0;1024-65535;3128-3128;TCP;HTTP
Criteria=::/0;::/0;1024-65535;110-110;TCP;POP
Criteria=::/0;::/0;1024-65535;143-143;TCP;IMAP
Criteria=::/0;::/0;1024-65535;5050-5050;TCP;YAHOO
Criteria=::/0;::/0;1024-65535;6667-6667;TCP;IRC
Criteria=::/0;::/0;1024-65535;1111-1111;TCP;IRC
</Basic_Criteria6>
#**********End of Third Section***************************>


#**************Application Specific Specifications******
# Here the criteria corresponding to different application level
# protocols are specified

#**************IMAP Specifications******
```

```
<IMAP_Criteria>
        NUM_of_Criteria=2
        <Usernames>
              Num_of_Usernames=1
              Case-Sensitive=no
              Username=sudheerv
        </Usernames>
        <Search_Email_ID>
              Num_of_email_id=2
              Case-Sensitive=yes
              E-mail_ID=ananth
              E-mail_ID=deshaw
        </Search_Email_ID>
        <Search_Text_Strings>
              Num_of_Strings=0
        </Search_Text_Strings>
        <Usernames>
              Num_of_Usernames=1
              Case-Sensitive=no
              Username=sudheer
        </Usernames>
        <Search_Email_ID>
              Num_of_email_id=1
              Case-Sensitive=yes
              E-mail_ID=deepak
        </Search_Email_ID>
        <Search_Text_Strings>
              Num_of_Strings=2
              Case-Sensitive=no
              String=pickpacket
              String=IMAP
```

```
        </Search_Text_Strings>
</IMAP_Criteria>
#**************END of IMAP Specifications******

#**************POP Specifications******
<POP_Criteria>
        NUM_of_Criteria=2
        <Usernames>
              Num_of_Usernames=1
              Case-Sensitive=no
              Username=ananth
        </Usernames>
        <Search_Email_ID>
              Num_of_email_id=2
              Case-Sensitive=yes
              E-mail_ID=sudheer
      E-mail_ID=sybase
        </Search_Email_ID>
        <Search_Text_Strings>
              Num_of_Strings=0
        </Search_Text_Strings>
        <Usernames>
              Num_of_Usernames=1
              Case-Sensitive=no
              Username=jainbk
        </Usernames>
        <Search_Email_ID>
              Num_of_email_id=1
              Case-Sensitive=yes
              E-mail_ID=dheeraj
        </Search_Email_ID>
```

```
        <Search_Text_Strings>
              Num_of_Strings=2
              Case-Sensitive=no
              String=sachet
              String=POP
        </Search_Text_Strings>
</POP_Criteria>
#**************END of POP Specifications******

#**************SMTP Specifications******
<SMTP_Configuration>
   <SMTP_Criteria>
        NUM_of_Criteria=2
        <Search_Email_ID>
            Num_of_email_id=1
            Case-Sensitive=yes
            E-mail_ID=sudheerv@cse.iitk.ac.in
        </Search_Email_ID>
        <Search_Text_Strings>
            Num_of_Strings=1
            Case-Sensitive=yes
            String=PickPacket
        </Search_Text_Strings>
        <Search_Email_ID>
            Num_of_email_id=2
            Case-Sensitive=yes
            E-mail_ID=ananth@iitk.ac.in
            E-mail_ID=jainbk@hotmail.com
        </Search_Email_ID>
        <Search_Text_Strings>
            Num_of_Strings=0
```

```
        </Search_Text_Strings>
    </SMTP_Criteria>
    Mode_Of_Operation=full
</SMTP_Configuration>
#**********END of SMTP Specifications******

#**********FTP Specifications******
<FTP_Configuration>
    <FTP_Criteria>
        NUM_of_Criteria=1
        <Usernames>
            Num_Of_Usernames=2
            Case-Sensitive=no
            Username=puneetk
            Username=jainbk
        </Usernames>
        <Filenames>
            Num_Of_Filenames=1
            Case-Sensitive=no
            Filename=test.txt
        </Filenames>
        <Search_Text_Strings>
            Num_Of_Strings=1
            Case-Sensitive=yes
            String=book secret
        </Search_Text_Strings>
    </FTP_Criteria>
    Monitor_FTP_Data=yes
    Mode_of_Operation=full
</FTP_Configuration>
#**********END of FTP Specifications******
```

```
#*************HTTP Specifications******
<HTTP_Configuration>
   <HTTP_Criteria>
       NUM_of_Criteria=1
      <Host>
           Num_Of_Hosts=1
           Case-Sensitive=no
           HOST=http://www.rediff.com
       </Host>
       <Path>
           Num_Of_Paths=1
           Case-Sensitive=yes
           PATH=cricket
       </Path>
       <Search_Text_Strings>
           Num_of_Strings=1
           Case-Sensitive=no
           String=neutral venu
       </Search_Text_Strings>
   </HTTP_Criteria>
   <Port_List>
       Num_of_Ports=1
       HTTP_Server_Port=80
   </Port_List><Port_List6>
Num_of_Ports=2
HTTP_Server_Port=80
H TTP_Server_Port=3128
</Port_List6>

   Mode_Of_Operation=full
```

```
</HTTP_Configuration>
#**********END of HTTP Specifications******

#**********TELNET Specifications******
<TELNET_Configuration>
   <Usernames>
       Num_of_Usernames=1
       Case-Sensitive=yes
       Username=ankanand
   </Usernames>
   Mode_Of_Operation=full
</TELNET_Configuration>
#******END of TELNET Specifications******
#****** IRC Specification ***************
<IRC_Configuration>
<IRC_Criteria>
NUM_of_Criteria=0
</IRC_Criteria>
<Port_List>
Num_of_Ports=1
IRC_Server_Port=6667
</Port_List>
<Port_List6>
Num_of_Ports=2
IRC_Server_Port=6667
IRC_Server_Port=1111
</Port_List6>
Mode_Of_Operation=full
</IRC_Configuration>
#****** End of IRC Configuration *********
```

```
##########End Application Specific Specifications****
```

A.2 Configuration File with Buffer Sizes(*.bcfg*)

```
# The file contains the number of connections to open simultaneously
# for some applications
# and the number of packets to be stored per connection before a match
#  occurs
<NUM_CONNECTIONS>
NUM_CONNECTIONS=10
NUM_SMTP_CONNECTIONS=500
NUM_FTP_CONNECTIONS=500
NUM_HTTP_CONNECTIONS=500
NUM_TELNET_CONNECTIONS=500
NUM_TEXT_CONNECTIONS=500
NUM_RADIUS_CONNECTIONS=500
NUM_POP_CONNECTIONS=500
NUM_IMAP_CONNECTIONS=500
NUM_IRC_CONNECTIONS=500
NUM_YAHOO_CONNECTIONS=100
</NUM_CONNECTIONS>
Num_of_SMTP_Stored_Packets=0
Num_of_FTP_Stored_Packets=0
Num_of_HTTP_Stored_Packets=0
Num_of_POP_Stored_Packets=0
Num_of_IMAP_Stored_Packets=0
Num_of_YAHOO_Stored_Packets=0
Num_of_IRC_Stored_Packets=0
Num_of_IRC_Channels=10
```

Appendix B

GZIP File format

B.1 File format

A gzip file consists of a series of "‘members" (compressed data sets).
The format of each member is specified in the following section.
The members simply appear one after another in the file with no additional information before, between, or after them.

B.2 Member format

Each member has the following structure:

```
+---+---+---+---+---+---+---+---+---+---+
|ID1|ID2|CM |FLG|     MTIME     |XFL|OS | (more-->)
+---+---+---+---+---+---+---+---+---+---+

(if FLG.FEXTRA set)

+---+---+=================================+
| XLEN  |...XLEN bytes of "extra field"...| (more-->)
+---+---+=================================+
```

```
(if FLG.FNAME set)

+========================================+
|...original file name, zero-terminated...| (more-->)
+========================================+

(if FLG.FCOMMENT set)

+==================================+
|...file comment, zero-terminated...| (more-->)
+==================================+

(if FLG.FHCRC set)

+---+---+
| CRC16 |
+---+---+

+=======================+
|...compressed blocks...| (more-->)
+=======================+

  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|     CRC32     |     ISIZE     |
+---+---+---+---+---+---+---+---+
```