

Supporting Chat Protocols in PickPacket

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

by

K Anantha Kiran



to the

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

June, 2005

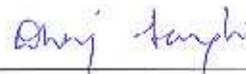
Certificate

This is to certify that the work contained in the thesis entitled "*Supporting Chat protocols in PickPacket*", by *K Anantha Kiran*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

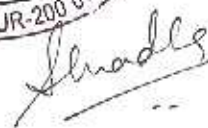
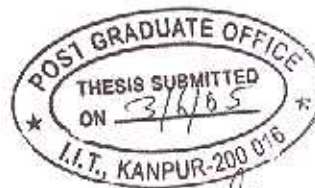
June, 2005



(Dr. Deepak Gupta)
Department of Computer Science &
Engineering,
Indian Institute of Technology,
Kanpur.



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



Abstract

Internet media is quite popular for the electronic transfer of both business and personal information. But, the same media can be and has been used for unlawful activities. This demands the need for highly customizable network monitoring tools to capture suspected communications over the network and to analyze them. However, electronic surveillance may violate the rights of privacy, free speech and association. PickPacket - a network monitoring tool, can handle the conflicting issues of network monitoring and privacy through its judicious use. PickPacket has four components - *Configuration File Generator* for assisting the users in setting up the filtering parameters, *Filter* for capturing the packets in the network, *PostProcessor* for analysing the output files and *Data Viewer* for interactive displaying of the captured sessions.

Earlier version of PickPacket had support for four application protocols - SMTP, HTTP, FTP and Telnet. Chat protocols, by which a group of users form a network to communicate information among themselves, have gained popularity in the last few years. Active use of these protocols on the Internet motivated the need for support of chatting protocols in PickPacket. This thesis discusses extension of PickPacket for two chatting protocols (IRC and Yahoo Messenger). All components of the PickPacket have been upgraded for the support of new protocols. PickPacket has been tested for correctness and performance measurement.

Acknowledgments

I take this opportunity to express to place on record my gratitude towards my thesis supervisors Dr. Dheeraj Sanghi and Dr. Deepak Gupta for their invaluable guidance throughout my thesis work. It would have never been possible for me to take this project to completion without their innovative ideas and encouragement. 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 Satya Srikanth. I never forget Srikanth's help in testing of *splitter* that was built in my second semester. He also introduced us architecture of PickPacket software. I also thank my project partner, Sudheer, not only for cooperating me during thesis but also for being my best friend. It is such a great learning experience working with him. I would like to thank Mr. Guru Preet Singh for making my visits to New Delhi comfortable and secure. Bharat and Puneet, my colleagues in Security Lab, are always there to share fun and problems in security lab. I will never forget those sleep less nights that we spent working and preparing things for the meetings. Devendar and Vinaya helped in changing the output structure of the post processor and developing a new web based GUI. I would like to thank Murthy for his valuable suggestions that he gave at the starting stages of my thesis.

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 am always indebted to the help extended by vipin and others while i was suffering from stomatch pain. I would like to thank all of my classmates for making my stay in kanpur homely. I never forget those moments i shared with them. 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 parents and sisters encouraging me all the times and taking me to this stage in life.

Contents

1	Introduction	1
1.1	Sniffers	2
1.2	PickPacket	3
1.3	Need for Chat protocols support in PickPacket	4
1.4	Organisation of the report	5
2	PickPacket: Architecture and Design	6
2.1	Architecture of PickPacket	6
2.2	PickPacket Design	7
2.2.1	Configuration File Generator	7
2.2.2	PickPacket Filter	9
2.2.3	PostProcessor	12
2.2.4	Data Viewer	13
3	Adding Support for IRC: Design and Implementation	15
3.1	Internet Relay Chat (IRC) Protocol	15
3.1.1	Protocol overview	15
3.1.2	Command Sequences	17
3.2	IRC Filter	18
3.2.1	Goals	19
3.2.2	Design and Implementation	19
3.3	IRC Metahandler and Viewer	21

4	Adding Support for Yahoo Messenger: Design and Implementation	24
4.1	Yahoo Messenger Protocol	24
4.1.1	Protocol Overview	25
4.1.2	Command sequences	26
4.2	Yahoo Filter	27
4.2.1	Objectives	28
4.2.2	Design and Implementation	28
4.3	Yahoo Metahandler and Viewer	31
5	Tests and Results	36
5.1	Correctness Verification	36
5.1.1	IRC Filter	37
5.1.2	Yahoo Filter	37
5.2	Performance evaluation	38
6	Conclusion	40
6.1	Further work	41
	Bibliography	43
A	Sample Configuration Files	44
A.1	Configuration File with Filtering Criteria (<i>.pcfg</i>)	44
A.2	Configuration File with Buffer Sizes(<i>.bcfg</i>)	55

List of Tables

List of Figures

2.1	Architecture of PickPacket	7
2.2	Filter design	10
2.3	The Basic Design of the PickPacket Filter	11
2.4	PostProcessor Design	13
3.1	An example of small IRC network	16
3.2	Working of IRC filter	22
3.3	ircinfo file format	23
4.1	Yahoo Command format	27
4.2	Working of yahoo filter	32
4.3	format of yahooinfo file	35

Chapter 1

Introduction

Usage of Internet for electronic transfer of both business and personal information is quite popular. As a result, Internet has become a key resource of information. But the same Internet can be and has been used by terrorists, criminals and others to communicate information about unlawful activities. This necessitates highly customizable network monitoring tools to capture suspected communications over the network and to analyze them later.

Companies too have to protect their intellectual property from falling into the hands of their competitors. Therefore, they resort to intelligence gathering over the network to check if any employee is sending such information illegally. Hence, there is a pressing need for development of tools that can monitor and detect undesirable communications over the network.

Monitoring tools perform their task by sniffing packets from the network and filtering them on the basis of user specified rules. The tools that provide the facility of specifying simple rules for filtering packets are called *Packet filters*. They use fixed offset packet information like IP addresses and port numbers for filtering. Tools that filter packets based on the complex rules and perform post-capture analysis of collected traffic are termed as *Network monitoring tools*. They understand applications, and can search through packet application data. The following section describes working of network monitoring tools in general and also mentions various commercial and non-commercial tools available publicly. However, electronic

surveillance conflicts with the rights of privacy, free speech and association. Section 1.2 explains how *PickPacket*, a network monitoring tool, works and how it addresses the conflicting requirements of privacy preservation and intelligence gathering. Section 1.3 gives the motivation for providing support for Chat protocols in PickPacket. Last section deals with the organisational flow of this report.

1.1 Sniffers

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 (now Network Associates incorporated). Network sniffers are software applications often bundled with hardware devices and are used for eavesdropping on the network. Akin to a telephone wire-tap that allows a person to listen in on to other person's communication, a sniffing program lets someone listen in on other computer conversations.

Generally sniffers work by putting the ethernet hardware (the standard network adapter) into promiscuous mode. The chip in the ethernet card, that is meant to ignore all the traffic not intended to this hardware, gets disabled in the promiscuous mode. This enables ethernet, and the sniffer consequently, to listen to all packets on that section of the network.

A simple sniffer just writes all the packets in the network onto disk. These sniffers will immediately fill up the entire disk space, if placed on the traffic bound segment of the network. Analysis of such a large database consumes considerable amount of resources. Also, such sniffers dump data belonging to the untargeted users who happened to access and transfer data through the network during the sniffing time. This may violate the privacy of users. Considering the above two issues, currently available sniffers are coming with three levels of filtering mechanisms. The first level of filtering is based upon network parameters like IP addresses, protocols and port numbers present in the packet. This level of filtering is generally supported at the kernel level also. The second level of filtering is based on application specific criteria like email id for SMTP, hostname for HTTP etc. The third level of filtering is based

on the content present in the application payload. Sniffers also come bundled with their own post-capture analysis and processing tools which extract meta information from the dump and present it in a user interactive manner.

Sniffers come in different flavors and capabilities for different Operating systems. WinDump [1] is a version of tcpdump [6] for Windows that uses WinCap, a library compatible to libpcap. This tool can filter packets based on the rules formed using network parameters. Also, it allows to make some statistical analysis out of the captured packets. Ethereal [2] is a UNIX-based sniffer program that also runs on Windows. This tool has equal filtering capability as that of tcpdump. It provides graphical interface for viewing captured data. EtherPeek NX [3] gives ability to troubleshoot and monitor network traffic quickly and effectively. Network Associates Incorporated [9] have a range of sniffers including VOIP sniffers. Carnivore [13] is a network monitoring tool developed by FBI with the sole purpose of directed surveillance. This tool can capture packets based on wide range of application-layer based criteria along with text strings match criteria. Carnivore is also capable of monitoring dynamic-IP address based networks. A more detailed survey on the currently available sniffer products can be found in [10].

1.2 PickPacket

PickPacket is a network monitoring tool, developed at IIT Kanpur [10, 7]. Its functionality is similar to that of Carnivore. PickPacket is a passive tool in the sense that it neither injects any packet into the network nor delays any packet to its destination. It supports various levels of packet filtering mechanisms while sniffing the packets. It can capture packets based on the network level parameters like IP-addresses, port numbers and protocols. It also supports filtering rules that are specific to certain application-layer protocols like SMTP, FTP, HTTP and Telnet. For example, a user can configure the filter to capture mails from or to certain users in SMTP sessions. Once the packets have been found based on the filtering criteria they are written on to the disk storage.

PickPacket addresses the conflicting issues of privacy preservation and network

monitoring in two ways. One, it has a rich set of configuration parameters which makes it easier to target very specific communication. Two, it provides two modes of packet capturing, "PEN" mode and "FULL" mode. In the "PEN" mode of operation, it only establishes occurrence of events based on the filtering criteria given, while in the "FULL" mode of operation it captures all the packets matching the criteria. Judiciously using these features can help in protecting the privacy of untargeted users.

PickPacket comprises of four components. *Configuration-File-Generator* is a JAVA based user interface for specifying the values of filtering parameters. *Filter* is an online component which selects the connections based on the criteria specified in configuration file. *PostProcessor* is an offline post-capture analysis tool that extracts per application protocol metadata information from the outputfile generated by *Filter* component. *DataViewer* is a webbased application to render the metadata generated by *PostProcessor* in a user interactive manner.

1.3 Need for Chat protocols support in PickPacket

The aim of PickPacket is to concentrate on those application layer protocols which form significant portion of the Internet traffic and are used to communicate information among users. In this view, earlier implementation of PickPacket had support for four application protocols named FTP, HTTP, SMTP and Telnet.

Chat protocols, by which a group of users form a network to communicate information among themselves, have gained popularity in the last few years. Chat protocols are generally used to establish collaboration among geographically distributed development teams. Active use of these protocols on the Internet motivated the need for support of chatting protocols in PickPacket.

As a step towards giving support for chatting protocols in PickPacket, we have considered two most popular protocols named IRC and Yahoo messenger. Internet Relay Chat (IRC) was one of the first chat protocols, and quickly gained the status of being the most popular one on the net. Yahoo messenger is another popular chat protocol which is proprietary.

My contribution includes extension of PickPacket for two chatting protocols (IRC and Yahoo Messenger). All components of the PickPacket have been upgraded for the support of new protocols. PickPacket has been tested for correctness and performance measurement.

1.4 Organisation of the report

This thesis describes design and implementation of support for two chat protocols, Internet relay chat [12] and Yahoo messenger [14], in PickPacket. Chapter 2 briefly describes the design of PickPacket tool. Chapter 3 and 4 discuss the design and implementation of two chat protocols, IRC and Yahoo messenger respectively. Chapter 5 explains the testing procedures employed for correctness, performance verification and results. Chapter 6 gives concluding remarks on the thesis with suggestions for future work.

Chapter 2

PickPacket: Architecture and Design

In this chapter, we first discuss the PickPacket architecture. This is followed by a brief explanation of the design of each component in the architecture. Detailed documentation about the design and implementation of PickPacket can be found in [7].

2.1 Architecture of PickPacket

PickPacket can be logically viewed as a composition of four independent components working in a pipeline. These components are - *PickPacket Configuration File Generator* deployed on a Windows/Linux machine, *PickPacket Filter* deployed on a Linux machine, *PickPacket PostProcessor* deployed on a Linux machine and *PickPacket DataViewer* deployed on a Windows/Linux machine. Ideally, it is possible to run all the four components on a single machine. Pictorial view of PickPacket architecture can be seen in Figure 2.1. *Configuration File Generator*, the first component, is a JAVA based user interface that can take filtering criteria at different levels. It generates configuration files which are input to *Filter*. *Filter*, an online component, reads all the packets and stores those packets which match the criteria in the configuration file. *PostProcessor* is an off-line capture analysis tool that accepts outputfiles of *Filter* and extracts meta information in a fixed directory structure. *DataViewer* is a web based user interface to render the *Postprocessor* outputted metadata in an

interactive fashion.

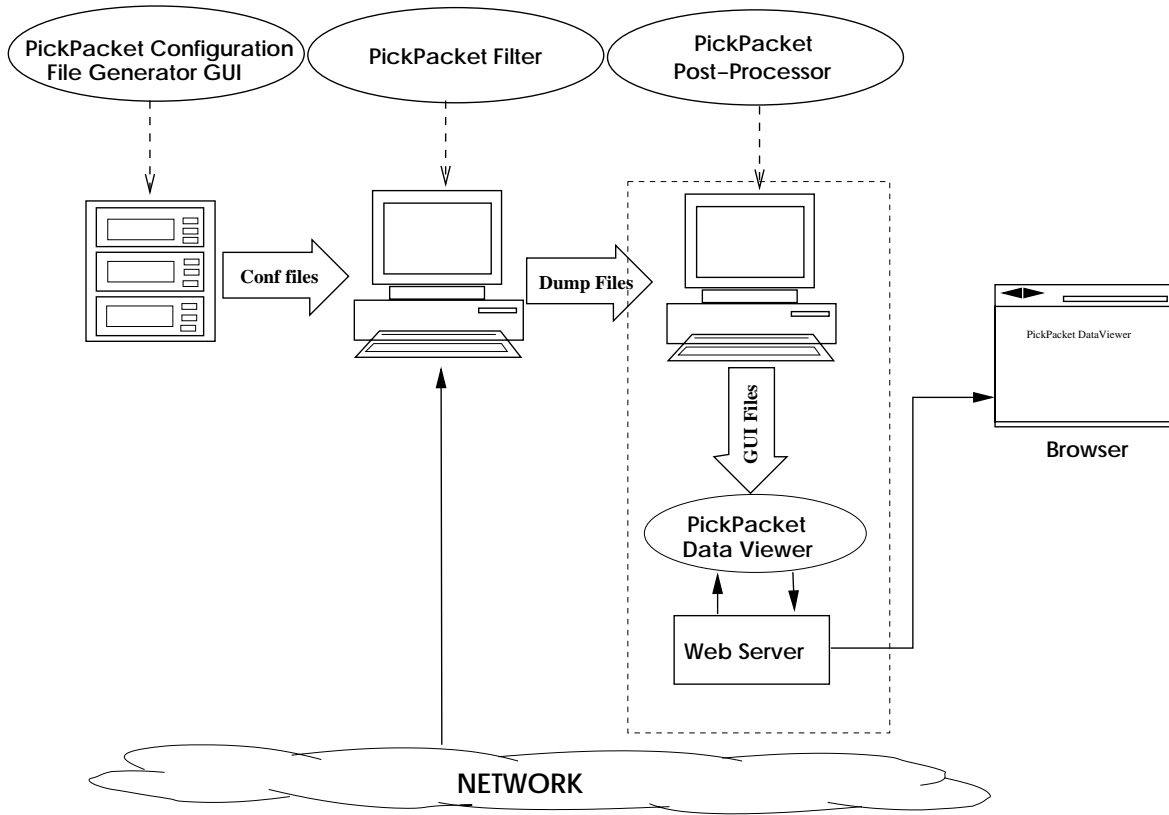


Figure 2.1: Architecture of PickPacket

2.2 PickPacket Design

In this section we discuss the design of four components of PickPacket separately in subsections.

2.2.1 Configuration File Generator

PickPacket Configuration File Generator generates two configuration files containing values of several parameters needed by *Filter*. One file stores filtering criteria given by the user. These criteria could be at different levels of protocol hierarchy.

The other file contains values for different buffering parameters. These parameters should be changed only by an advanced user, like a system administrator. The values of these parameters depend on hardware capability of the host machine. Earlier implementation of Configuration file generator had both the filtering and system parameters generated in a single file. It has been modified to generate two files so that a normal user is only concerned with filtering criteria. These files are written in HTML like syntax. An example configuration file set is given in *Appendix A*.

The configuration file containing filtering criteria has three sections.

- The first section contains specification of the outputfiles that would be created by Filter. This section gives information about how often, with what size, and by what name Filter should generate these files containing packets that it has captured.
- The second section contains criteria for filtering packets based on source and destination IP addresses, transport layer protocol, and source and destination port numbers. It also mentions the application layer protocol filter to be used for the packets belonging each such criteria. This information is used to demultiplex packets to the correct application layer protocol filter.
- The third section contains multiple subsections, each of which specifies a criteria corresponding to an application layer protocol. Application layer content of the packet is investigated for match against the corresponding protocol criteria.

The configuration file storing system parameters values has two sections.

- The first section comprises of entries giving a value to the maximum number of connections the filter should monitor simultaneously for each application protocol.
- The second section has one entry per application layer protocol specifying the number of history packets the filter should be able to keep while monitoring a connection of that protocol for criteria match.

The values in these sections are used to pre-allocate memory buffers in Filter.

2.2.2 PickPacket Filter

Filtering criteria in the configuration file, contains rules which conceptually fall into different levels. So, an ideal Filter design should accommodate various levels of packet filtering. Following are the various levels at which packets can be filtered.

1. Filtering based on network parameters (IP addresses, port numbers, etc.)
2. Filtering based on the application layer protocol criteria (user names, email-ids, etc.)
3. Filtering based on the content present in the application payload.

An efficient first level of filtering can be provided by using in-kernel filters [8]. Since the content of application layer can be best deciphered by the application itself, the second and third levels of filtering can be combined into a single application layer filter. For each application layer protocol, there is a separate filter module that understands that protocol. In this model of filtering, a *demultiplexer* is required that directs packets from the in-kernel filter to the appropriate application layer filter.

Finally, application specific filtering reduces to text search in the application layer data content of the packets. In case of communications over connection oriented protocol, the text search should handle situations where the desired text string is split across two or more packets before being transmitted on the network. As there may be losses or reordering of packets in the network, filter should also check for packets that are received out of sequence while performing the search for split text. Thus a module that does these checks is needed. This module is called the *TCP Connection Manager*. There is one such module for each application level filter that does the text string match in the application payload. All these modules and their relationships are presented in Figure 2.2.

All modules in the design are represented by ovals in the figure. *Basic Filter* module works at the first level of filtering, while the *Application Layer Filters* work at the second and third levels of filtering. *Demultiplexer* module directs the packets

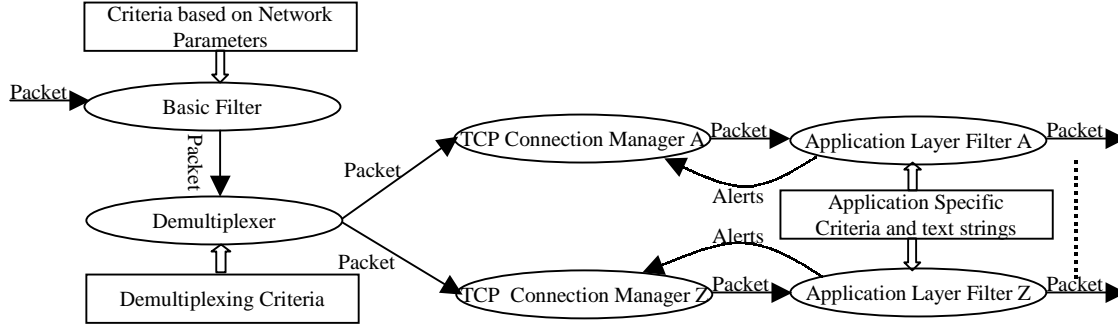


Figure 2.2: Filter design

to one of the application layer filters. There is a *TCP Connection Manager* for each application layer protocol. There are several issues that go into the design of connection manger. First the connection manager need not determine the sequencing of packets for all connections. Rather it should determine sequencing for only those connections that an application filter is interested in. For example, if an application layer filter has decided that a particular connection matched the criteria and to be stored, determining sequence of packets in this connection is not required. Communication between the application layer filter and the connection manager to indicate such interest is provided by means of *Alerts*. A second issue is the module which should store history data for an application protocol. Connection manager is anyway maintaining state for each connection till a criteria match for determining sequencing of packets. There is no advantage in duplicating this effort. Hence history packets can be stored at the connction manager. Alerts would also involve asking connection manager to either delete these packets or store them to disk.

The discussions above lay the foundation for a more detailed design of Pick-Packet Filter, which is shown in Figure 2.3. A module *Initialize* has been added for initialization of all filtering modules based on the configuration file. Another module, *Output File Manager*, is added for storing selected packets to the disk. A *Filter Generator* module is added for generating the in-kernel BPF [8] code. Hooks are provided for changing the BPF code generated. *Demultiplexer* is provided the facility of calling *Output File Manager* directly so that the basic filter can directly

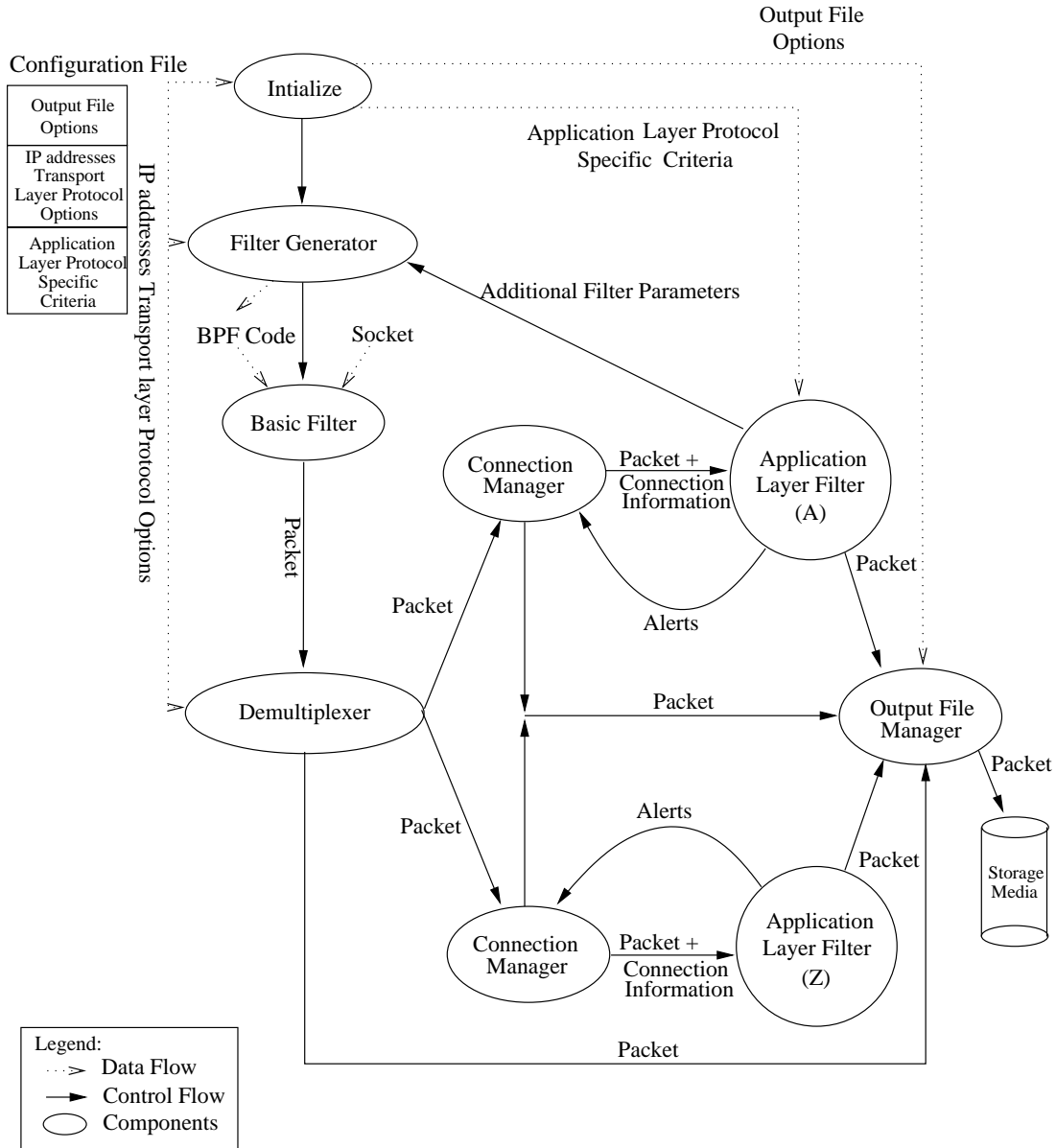


Figure 2.3: The Basic Design of the PickPacket Filter

store packets without resorting to application layer protocol based filtering, if necessary. *Connection Manager* can also directly store packets to the disk. This is required when a criteria has matched for a specific connection and the connection is still open. More details of these modules can be found in Reference [7].

2.2.3 PostProcessor

The PostProcessor is an offline analysis tool to extract the metadata corresponding to all the connections captured by the filter. Ideally PostProcessor should meet following objectives.

- **Session breaking:** A connection is identified by 4-tuple. There can be more than one connection existed at different time intervals but with same 4-tuple identifiers. Each of these connections is regarded as a session of corresponding 4-tuple. The filter output file contains packets belonging to several sessions. Before attempting to extract any data from these files, sessions need to be separated.
- **Metadata extraction:** Metadata includes important fields and entities present in the data content belonging to an application layer protocol. For example, it is email addresses and emails incase of SMTP, usernames and files incase of FTP. Metadata extraction from each session should be handled separately and should be stored in a fixed structure.

Figure 2.4 shows the design of *PickPacket PostProcessor* which captures above two objectives. The first module, *Connection breaker*, accepts the filter outputfile as input and produces set of files. Each file contains all the packets belongs to a 4-tuple. It works by reading each packet from the input dumpfile and writing it to a specific file that is named with the 4-tuple of the packet. This module writes all the packets sent by either entity of the 4-tuple into a single file.

The second module *Session breaker* reads each file generated by *Connection breaker* and splits that file into as many number of files as the sessions involved in the connection corresponding to that file. If the file belongs to a UDP connection, *Session breaker* directly produces same file as output. If the file belongs to a

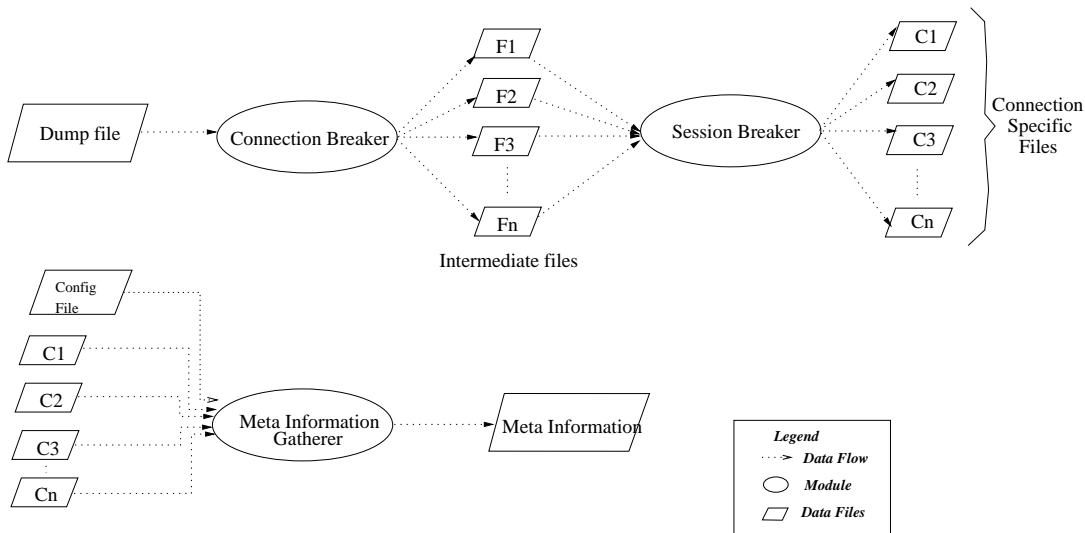


Figure 2.4: PostProcessor Design

TCP connection, it uses a TCP like engine to identify the sessions involved in that connection. Then, it produces one file for each session where packets are sorted by their time. Packets with same time are sorted by their sequence numbers.

The third module, *Metainformation gatherer*, employs one metadata extractor for each application layer protocol. It reads one session file at a time and directs it to appropriate metadata extractor depending on the application protocol of the session. These metaextractors store information in a fixed format. This format accommodates one directory per session which contains all the details regarding that session stored into separate files. File named “tcpipinfo” contains summary information of the session that includes network parameter details, application layer protocol and list of matched keywords given as criteria. File named “appinfo” contains the meta data of the session that is specific to its application protocol. All the files are stored in standard INI format.

2.2.4 Data Viewer

PickPacket DataViewer is a web based application that is aimed to render the post-processed information in user interactive manner. It is written in PHP [11] script.

It is deployed along with a web server. Web server access the data through PHP scripts and serve the user requests. Data viewer can be deployed either on the same machine where the post-processing is done or on a different machine.

PostProcessor output directory is placed in a fixed path that is known to DataViewer. At first, DataViewer lists all the directories present in that path. For any dumpfile directory selected by user, it lists summary of all the connections present in the directory. This summary includes the network parameters, application protocol and list of matched keywords. Connections can be sorted based on any one of summary fields. The DataViewer allows examining the details of a connection and can show the data for that connection through appropriate user agents.

The Data Viewer provides user with facilities like downloading the captured e-mails, viewing browsed web pages etc. The communication between connection entities can be seen in a separate dialogue window. The configuration file used for the filtering can be viewed. The connection-specific output file for each of the connection can also be downloaded separately.

User can search among the captured connections. The search criteria includes 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.

Chapter 3

Adding Support for IRC: Design and Implementation

The Internet Relay Chat (IRC) [12] was one of the first chat protocols on the Internet. This chapter discusses the design and implementation issues in adding support for IRC in *PickPacket*. PickPacket requires one new module in each of its components in order to support IRC. Section 3.1 gives brief overview of the IRC protocol. Section 3.2 explains the design and implementation of *IRC Filter*, an application protocol filter module in *PickPacket Filter*. Section 3.3 describes the design of *IRC Metahandler*, an application protocol metadata extractor in *PostProcessor*. It also describes *IRC Viewer* that has been added to DataViewer.

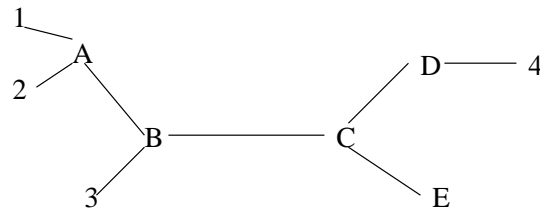
3.1 Internet Relay Chat (IRC) Protocol

The IRC has been designed over a number of years for use with text based conferencing. Following section explains the protocol in brief. This is followed by description of command sequences involved in the execution of the protocol.

3.1.1 Protocol overview

A typical IRC setup consists of group of servers and clients. Servers form the backbone of IRC, providing points to which clients may connect to talk to each

other. A server also forms a point for other servers to connect to, forming an IRC network. The only network topology allowed for IRC servers is that of a spanning tree, where each server acts as a central node for the rest of the net it sees. Figure 3.1 shows a sample configuration of IRC network.



Servers: A, B, C, D, E Clients: 1, 2, 3, 4

Figure 3.1: An example of small IRC network

An IRC server is a host that listens on TCP port 6667 for connections to other clients or servers. It is uniquely identified by a *servername*. It performs the required message delivery/multiplexing and other functions. An IRC client is any user connected to the server. So, an *IRC client* and *IRC user* are interchangeable. Each client is distinguished from the other client by a unique *nickname*. In addition to the nickname, all servers must have the real name of the host that the client is running on, the username of the client on that host and the server to which the client is connected.

A channel is a named group of one or more IRC clients which will receive the messages addressed to that channel. An IRC client sends a message to its server to join any channel specifying its name. If the channel doesn't exist, the server creates a channel with the name specified by the IRC client. This client becomes channel operator. A channel's characteristics depends on its mode controlled by the operator. For example, if the channel is invite-only, an IRC client may join only if invited. Channel operators are endowed with certain powers which enables them to keep control and some sort of sanity in their channel. While a channel exists, any client can reference the channel using the name of the channel. Any member in

the channel can send a text message. The IRC servers will deliver one copy of this message to each member of the channel spread across the IRC net. The channel ceases to exist when all the members client leave it.

The IRC network can become disjoint when a connection between two servers breaks. In such a situation, the channel in each partition only consists of those clients which are connected to servers in that partition. It is possible that in one of the partition there is no client belonging to a particular channel. In this case, the channel will cease to exist in that partition. When the connection is re-established, the connecting servers exchange membership and modes of all channels. If a channel exists on both partitions, the join messages and modes are interpreted in an inclusive manner.

3.1.2 Command Sequences

In an IRC network, each server maintains a map of all of the servers on the net. A new server can connect to any server on the IRC net by sending a “SERVER” command. New server’s *servername* is the argument of command. If a “SERVER” message attempts to introduce a server which is already known to the receiving server, the message is ignored and the connection is closed. Accepting this request would have created an alternate path to that server, thereby breaking the acyclic nature of the IRC net. Whenever a new server is connected to the net, information about it is broadcasted to all the existing servers on the IRC network.

An IRC client can connect to any server by sending a “NICK” message that contains the nickname of user’s choice. The receiving server broadcasts the arrival of new nickname to all other servers. If a “NICK” message arrives at a server which already knows about an identical nickname for another client, a nickname collision occurs. As a result of nickname collision, all instances of nickname are removed from the server’s database and “KILL” command is issued to remove the nickname from database of all other servers. If the server receives an existing nickname from a client which is directly connected, it may issue a collision error to the client, and not generate any “KILL” commands. After “NICK” command, client sends “USER” command that is used to specify the username, hostname, and servername.

A client's nickname is a dynamic identity which can be changed at any time by resending "NICK" command.

A client can connect to any channel by sending a "JOIN" command that takes channel name as an argument. Whether a client is allowed to join a channel is checked only by the server to which the client is connected; all other servers automatically add the user to the channel when such a request is received from other servers. This allows each server to know where to find the users who are members of the channel. If "JOIN" is successful, the user is sent the channel's topic and the list of users on the channel. A channel operator is the user who joined the channel first. The operator of a channel can set the characteristics of a channel by using the "MODE" command. Any user on an invite-only mode channel can invite new members using "INVITE" command. A client can send a private message using the command "PRIVMSG". This command takes the receiver's name and the text to be sent as arguments. The receiver argument can be nickname of the receiver of the message. This can also be a list of names or channels separated with commas. The "PART" command causes the client sending the message to be removed from the list of active users for all given channels listed in the argument string.

A client session ends with a "QUIT" command. The server must close the connection to a client which sends the "QUIT" command. If a server wishes to break the connection to another server, it must send "SQUIT" command specifying the name of the other server as the parameter.

3.2 IRC Filter

The *IRC Filter* is an application protocol filter module in the component *PickPacket Filter*. This is meant for capturing selective data transferred in IRC protocol sessions. Section 3.2.1 gives the objectives of filter module. Section 3.2.2 discusses the design and implementation details of *IRC Filter*.

3.2.1 Goals

IRC channels are the entities of interest in an IRC session. Therefore, the filtering criteria for the *IRC Filter* should be able to specify details about a channel. A channel has a name, has members who are communicating with each other, and there is information that is being communicated. We should be able to filter based the values of channel names, nicknames, and keywords. Using channelnames and nicknames, targeted monitoring can be done for the channels having certain name and members. The ‘keyword’ field can be used to specify the strings for which *IRC Filter* should try to match in the text messages of channels.

The *Configuration File Generator* component has been updated to provide interface for accepting the values of IRC criteria. Objective of the *IRC Filter* is to capture only those conversations which match channel name, member nicknames and message strings.

3.2.2 Design and Implementation

In IRC protocol, communication on all channels is multiplexed onto same TCP connection. We want to monitor communication on selected channels and not store all the packets on that TCP connection. For each channel, we will maintain several flags. These flags will help us determine whether a packet needs to be stored or not. Since we do not know how many channels are present in an IRC connection, we will have a large array of data structure, each item corresponding to a channel. Everytime a new channel is noticed, another item in the array is initialized with that channelname. Other fields in that data structure include four flags. First flag, ‘channel_match_flag’, is set to “MATCHED” when the channel’s name matches with one of channelnames present in the criteria. Otherwise it is set to “NOMATCH”. If no channelname is given in the filtering rules, this flag is set to “MATCHED” for all channels. This flag is set only at the time of initializing the data structure. The other three flags are initialized to “NONE”. If the ‘channel_match_flag’ is being set to “NOMATCH” then ‘match_flag’ is set to “IGNORE”. Second flag, ‘nick_match_flag’, is set to “MATCHED” when nickname of one of the members of the channel matches with one of the nicknames present in the criteria. Third

flag, 'string_match_flag', is set to "MATCHED" when there is a channel message that has a keyword which matches with one of the keywords present in the criteria. Fourth flag, 'match_flag', is set to "MATCHED" when the channel has above three flags set to "MATCHED". If atleast one of the first three flags is set to "NOMATCH", the 'match_flag' is set to "IGNORE". The *IRC Filter* writes all the packets belonging to a channel which has 'match_flag' set to "MATCHED". If the 'match_flag' has neither of "MATCHED" or "IGNORE" values, the filter remembers packets in the history list of the channel matchstate.

The *IRC Filter* receives packets from the *TCP Channel Manager* module which also gives packet's connection information. So, the *IRC Filter* uses data structures belonging to the packet's connection while processing it. The *IRC Filter* first parses every packet to determine its command type and the command parameter values. The filter follows one of the following steps based on the command type.

If the command type is "NICK", the filter stores the nickname present in the packet. The filter tries to match this nickname with the list of nicknames present in the IRC criteria. If it matches, 'nick_match_flag' of every channel data structure is set to "MATCHED".

If the command type is "PRIVMSG", the filter extracts channelname, sender's nickname and message present in the packet. The filter considers data structure corresponding to the channelname. If the 'match_flag' is set to "IGNORE", this packet is ignored. If the 'match_flag' is set to "MATCHED", this packet is written to the disk. If this is the first packet of this channel then processing as discussed earlier. If the 'match_flag' is "NONE", the 'nick_match_flag' is considered. If it is set to "NONE", the filter tries to match the sender nickname with the list of nicknames present in the criteria and sets the flag to "MATCHED" if sender nickname matches. Then the filter considers 'string_match_flag'. If it is set to "NONE", the filter checks whether the message contains any of the keywords present in the criteria and sets the flag to "MATCHED" if keyword matches. If all three flags ('channel_match_flag', 'nick_match_flag' and 'string_match_flag') are set to "MATCHED", the filter sets the 'match_flag' to "MATCHED" and writes packets present in the history list including the current packet. Otherwise, the filter remembers current packet in the

history list of this channel.

If the command type is “JOIN”, the filter extracts channelname and new member’s nickname present in the packet. The filter considers data structure corresponding to the channelname. If the matchstate has ‘match_flag’ set to “IGNORE”, this packet is ignored. If the ‘match_flag’ is set to “MATCHED”, the filter writes this packet to outputfile. If this is the first packet of this channel then processing is as discussed above. If the ‘match_flag’ is set to “NONE”, the ‘nick_match_flag’ is considered. The filter tries to match the new member nickname with list of nicknames present in the criteria and sets ‘nick_match_flag’ to “MATCHED” if nickname matches. If the flag is set to “MATCHED” and the value of ‘string_match_flag’ is “MATCHED”, the filter sets the ‘match_flag’ to “MATCHED” and outputs packets present in the history list including the current packet. Otherwise, the filter adds current packet in the history list.

In the implementation of *IRC Filter*, we have a configurable limit on the maximum number of channels for which state information can be maintained at a time. The filter would clear the state information of the least recent channel to accommodate any new channel seen after the maximum limit is reached. It searches for substring match of keywords in the messages transferred.

In the “PEN” mode of packet capturing, the filter writes only the first packet of any channel that has matched the criteria. Before writing this packet, the filter fills the message part of the packet with ascii ‘X’ byte, and the ‘match_flag’ is set to “IGNORE” so that all future packets on this channel are not stored on disk.

3.3 IRC Metahandler and Viewer

The *IRC Metahandler* is a module in the component *PickPacket PostProcessor*. It extracts meta data from the session files belonging to IRC protocol. This module writes messages belonging to each channel in a separate file. These files are written in INI format. The Metahandler generates one “ircinfo” file for each session. This file contains a brief summary of every channel along with the list of nicknames and keywords that matched. An example file is shown in Figure 3.3.

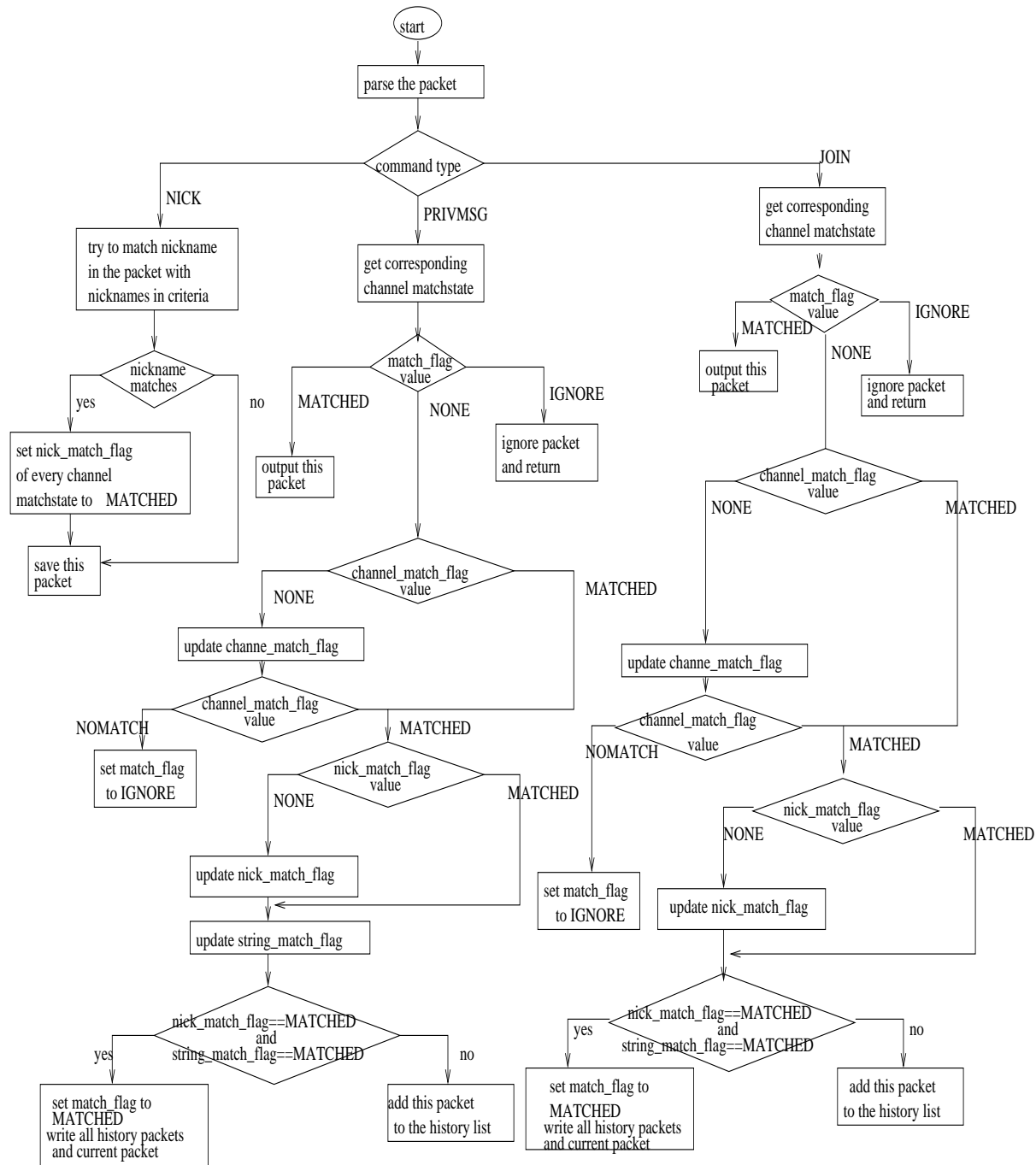


Figure 3.2: Working of IRC filter

```
nickname = "testnick"
username = "tempuser"
[channel1]
name = "testchannel1"
conversationfile = "channel1"
time = "Mon Apr 11 11:18:16 2005"
matchednicknames = "nick1 nick2"
keywords = "bomb terror"
```



Figure 3.3: ircinfo file format

The *Metahandler* maintains a table where each entry contains the name of the file corresponding to a channel along with list of matched nicknames and keywords in the IRC criteria. It parses each packet read from the session file to figure out the packet's command type, just like the way filter did. If the packet belongs to "USER" or "NICK" command, it stores the command argument. If it belongs to "PRIVMSG" or "JOIN" type, then it looks up the table to find the name of its file. If no entry is found in the table, then the handler would enter new entry into the table with a new file name. Therefore all the messages corresponding to a channel would be written into a separate file in a INI format. These messages are checked for updating the list of matched nicknames and keywords in IRC criteria.

The *IRC Viewer* is a module in *PickPacket DataView*. It is used to display the connections belonging to IRC protocol. This module uses the "ircinfo" file to display the channels present in a connection. It shows the channel conversations in a separate dialog box. User can search among the existing IRC sessions with different parameters.

Chapter 4

Adding Support for Yahoo Messenger: Design and Implementation

This chapter discusses Yahoo Messenger, a popular proprietary chat protocol, and the design and implementation issues in giving support to monitor Yahoo Messenger traffic in *PickPacket* tool. Provision of this protocol support in *PickPacket* requires implementation of one new module for each component of the tool. Section 4.1 gives a brief overview about the protocol and command transfers involved in protocol execution. Section 4.2 discusses the issues involved in the design and implementation of *Yahoo Filter*, an application protocol filter module in *PickPacket Filter* for yahoo messenger. Section 4.3 explains the design details of *Yahoo Metahandler*, an application protocol metadata extractor module in *PostProcessor*. It also describes *Yahoo Viewer* that has been added to the DataViewer.

4.1 Yahoo Messenger Protocol

Yahoo Messenger is a proprietary protocol used to provide instant messaging and chatting services among yahoo registered users only. As there is no official documentation available about this protocol on the net, we have taken the help of an

unofficial documentation [14]. We have also looked at the source code of Gaim [4], an open source multi-protocol instant messaging client supports Yahoo messenger protocol, to draw more information about the protocol. We have also conducted lab experiments to monitor protocol conversations and to verify its accordance with the protocol specifications that we have determined. Following subsection explains the specification of protocol inferred from above efforts. Section 4.1.2 explains commands involved in the protocol execution.

4.1.1 Protocol Overview

In Yahoo Messenger chat protocol, the server host listens on a standard TCP port 5050. The client host initiates the conversation by establishing a TCP connection with the server. Initial phase of the conversation tries to authenticate the client by exchanging appropriate information. In this state, the server sends a challenge based on which client computes hash response of its password. The server host authenticates client's hash response by sending a reply message. Once the authentication succeeds, server sends the user's preferences and friends list details to the client. Now, the user can communicate with the other users connected to the server through Instant Messages(IM) or Chatrooms.

Instant Messaging allows the user to exchange messages with any other user in real time. If a user writes an instant message to the other, it is first delivered to the server. If the destination user is already connected to the server, the server will immediately dispatch that message to him. Otherwise, server defers dispatching till the user gets connected to it.

Yahoo Chatrooms are similar to the IRC channels in concept except the fact that this service is limited to Yahoo registered users only. Yahoo Chatrooms are sessions in which a group of users can have real time message communication among themselves. Any user can start initiating a Yahoo Chatroom with a name of his choice by sending a request message to the server. The server checks uniqueness of the user's chatroom name and sends its acceptance reply to the client. Then the client can start inviting other users to join this room. All these invitation requests will first reach the server. The server sends a copy of the invitation message

containing the chatroom name and the list of current chatroom members to all the invitees separately. The server sends a “JOIN” message to each member of chatroom for every acceptance response it gets from the invitee. Once the chatroom is established, messages sent by any member will first reach the server. The server sends one copy of this message to each member of the room along with the sending user’s identity. Therefore, each member in the chatroom sees the messages given by any other member in the chatroom. The server will notify all the members if any of the users in the chatroom leaves the session.

4.1.2 Command sequences

Yahoo Messenger protocol classifies the commands involved in the protocol into several service types where each service type represents particular state of the protocol execution. All the commands follow a particular format shown in Figure 4.1. This format consists of two fields: a fixed-length *header* and a variable length *data*. The *header* field is a group of six fixed-length subfields. First subfield is always a string “YMSG”. The second subfield is the version of the protocol it is using. Third subfield gives the length of the *data* field. Fourth subfield specifies the service type of this command. Fifth and sixth subfields are specific to the user. They give the status of user and his id respectively. The *data* field is a collection of attribute-value pairs separated by a fixed two-byte separator.

The server uses *challenge-response* mechanism to authenticate its users. During the authentication phase of the protocol, both the server and client communicate with commands of service type “AUTH”. The client uses this service type to send its Yahoo username after it has established a TCP connection with the server. Then the server challenges with a random number. The client responds with an MD5 hash of the Yahoo user’s password using the random number sent by the server. The server authenticates client’s response after checking whether the user is already logged in or not.

For transferring Instant Messages (IMs), commands of service type “MESSAGE” are used. Fields in these commands include the receiver’s username and the message

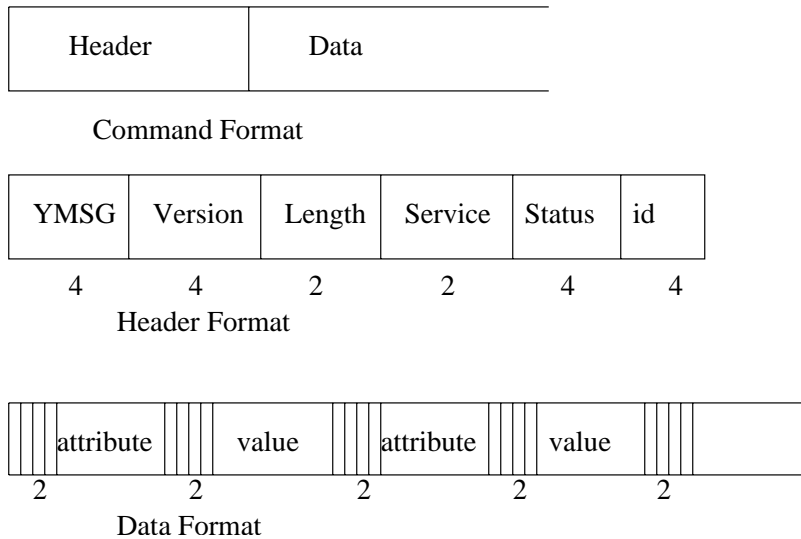


Figure 4.1: Yahoo Command format

to be sent. While initiating new chat sessions, commands of service type “CHAT-LOGON” are used. These commands allow the user to give his choice of chatroom name. The client uses commands of service type “CHATMSG” for sending messages to chatrooms. If a new user joins the chatroom, a command of service type “CHATJOIN” with the identity of the new user would be sent by the server to all the existing members of the chatroom. Similarly, a command of service type “CHATLEAVE” would be sent if any user leaves the chatroom.

4.2 Yahoo Filter

The *Yahoo Filter* is an application protocol filter module in *PickPacket Filter*. It is meant for capturing selective data transferred in Yahoo protocol sessions. In Section 4.2.1, we explain the objectives of *Yahoo Filter*. In section 4.2.2, the design and implementation details of *Yahoo Filter* are given.

4.2.1 Objectives

Entities of interest in a Yahoo session are IMs and chatrooms. Therefore, the filtering criteria for *Yahoo Filter* should be able to specify details about any IM or Chatroom. We have decided that filtering should be possible on the basis of usernames, chatroom names, yahoo-ids and keywords. Usernames are the identities with which a user tries to authenticate himself to the server. Using chatroom names and yahoo-ids, targetted monitoring can be done for chatroom sessions having certain name and members. Keywords are the strings for which the filter should try to find a match in both chatrooms and IM messages.

The *Configuration File Generator* has been updated to provide an interface for accepting the values of Yahoo criteria. The objective of *Yahoo Filter* is to capture only those conversations of chatrooms and IMs which have the name, members and message strings matching the criteria.

4.2.2 Design and Implementation

The *Yahoo Filter* design maintains matchstate of every chatroom/IM present in a Yahoo connection. A matchstate includes four flags. First flag, 'chatroom_match_flag', is set to "MATCHED" when the chatroom's name matches with one of chatroom names present in the criteria. Second flag, 'yahoo-id_match_flag', is set to "MATCHED" when yahoo-id of one of the members of the chatroom/IM matches with one of the yahoo-ids present in the criteria. Third flag, 'string_match_flag', is set to "MATCHED" when there is a chatroom/IM message that has a keyword which matches with one of the keywords present in the criteria. Fourth flag, 'match_state', is set to "MATCHED" when the channel has above three flags set to "MATCHED". If any of the four criteria, username, chatroom, yahoo-id, or keyword, is left blank in the configuration file, the corresponding flag in all matchstates is set to "MATCHED". If atleast one of the first three flags is set to "NOMATCH", the 'match_flag' is set to "IGNORE". The *Yahoo Filter* writes all the packets belonging to a chatroom/IM which has 'match_flag' set to "MATCHED". It ignores all the packets belonging to a chatroom/IM if the match-flag is "IGNORE". If the 'match_flag' has neither "MATCHED" nor "IGNORE" values, the filter remembers packets in the history

list of the matchstate. The filter keeps ‘username_match_flag’ for each connection. This flag is set to “MATCHED” when the username matches with one of the usernames present in the criteria.

Initially, matchstate of every chatroom/IM in a connection has all the four flags set to “NONE”. The ‘username_match_flag’ of the connection is also initialized to “NONE”. The *Yahoo Filter* receives packets from the *TCP Channel Manager* module which also gives packet’s connection information. So, the *Yahoo Filter* uses data structures belonging to the packet’s connection while processing it. The *Yahoo Filter* first parses every packet to determine its service type and the command parameter values. The filter follows one of the following steps based on the service type.

The filter first checks whether service type of packet is “AUTH”. If the service type is “AUTH”, the filter stores the username present in the packet. The filter tries to match the username with the list of usernames present in the Yahoo criteria. If it matches, ‘username_match_flag’ of the connection is set to “MATCHED”. Otherwise, the filter alerts the *TCP Channel Manager* to stop keeping information about this connection. If the service type is not “AUTH”, the filter processes the packet further only if the ‘username_match_flag’ is set to “MATCHED”.

If the service type is “CHATMSG”, the filter extracts chatroom name, sender’s yahoo-id and the message present in the packet. The filter considers matchstate corresponding to the chatroom name. If the matchstate has ‘match_flag’ set to “IGNORE”, this packet is ignored. If the ‘match_flag’ is set to “MATCHED”, this packet is written to the disk. Otherwise, if the ‘chatroom_match_flag’ is set to “NONE”, the filter tries to match the chatroom name with list of chatroom names present in the criteria. If the chatroom name doesn’t match, the filter sets the ‘chatroom_match_flag’ to “NOMATCH”, and the ‘match_flag’ to “IGNORE”. If it matches, the ‘yahoo-id_match_flag’ is considered. If it is set to “NONE”, the filter tries to match the sender yahoo-id with the list of yahoo-ids present in the criteria and sets the flag to “MATCHED” if sender yahoo-id matches. Then the filter considers ‘string_match_flag’. If it is set to “NONE”, the filter checks whether the message contains any of the keywords present in the criteria and sets the flag to “MATCHED” if keyword matches. If the matchstate has ‘chatroom_match_flag’,

'yahoo-id_match_flag' and 'string_match_flag' set to "MATCHED", the filter sets the 'match_flag' to "MATCHED" and writes packets present in the history list including the current packet. Otherwise, the filter remembers current packet in the history list of matchstate.

If the service type is "MESSAGE", the filter extracts yahoo-ids of sender and receiver and the message present in the packet. The filter considers matchstate corresponding to the IM. If the matchstate has 'match_flag' set to "IGNORE", this packet is ignored. If the 'match_flag' is set to "MATCHED", this packet is written to the disk. Otherwise, if the 'yahoo-id_match_flag' is set to "NONE", the filter tries to match the yahoo-ids of sender and receiver with list of yahoo-ids present in the criteria. If the yahoo-ids do not match, the filter sets the 'yahoo-id_match_flag' to "NOMATCH", and the 'match_flag' to "IGNORE". If they match, the filter considers 'string_match_flag'. If it is set to "NONE", the filter checks whether the message contains any of the keywords present in the criteria and sets the flag to "MATCHED" if keyword matches. If the matchstate has 'yahoo-id_match_flag' and 'string_match_flag' set to "MATCHED", the filter sets the 'match_flag' to "MATCHED" and writes packets present in the history list including the current packet. Otherwise, the filter remembers current packet in the history list of matchstate.

If the command type is "CHATJOIN", the filter extracts chatroom name and new member's yahoo-id present in the packet. The filter considers matchstate corresponding to the chatroom. If the matchstate has 'match_flag' set to "IGNORE", this packet is ignored. If the 'match_flag' is set to "MATCHED", the filter writes this packet to outputfile. Otherwise, the filter considers 'chatroom_match_flag'. If it is set to "NONE", the filter tries to match the chatroom name with one of the chatroom names present in the criteria. If it doesn't match, the filter sets 'chatroom_match_flag' to "NOMATCH", and 'match_flag' to "IGNORE". Otherwise, the filter considers 'yahoo-id_match_flag'. The filter tries to match the new member's yahoo-id with list of yahoo-ids present in the criteria and sets 'yahoo-id_match_flag' to "MATCHED", if yahoo-id matches. If the flag is set to "MATCHED" and the value of 'string_match_flag' is "MATCHED", the filter sets the 'match_flag'

to “MATCHED” and outputs packets present in the history list including the current packet. Otherwise, the filter adds current packet in the history list.

In the implementation of *Yahoo Filter*, we have given a configurable limit on the maximum number of Chat/IMs for which the state information can be maintained at a time. The filter would clear the state information of least recent chat session to accommodate any new chat session seen after the maximum limit is reached.

In the “PEN” mode of packet capturing, the filter writes only the first packet of any Chat/IM that has matched the criteria. Before writing this packet, the filter fills the message part of the packet with ascii ‘X’ byte. The filter sets the ‘match_flag’ to IGNORE to ignore future packets.

4.3 Yahoo Metahandler and Viewer

The *Yahoo Metahandler* is a module in *PostProcessor*. It extracts meta data from the session files belonging to Yahoo Messenger protocol. This module writes messages belonging to each chatroom or IM in a separate file. The Metahandler generates one “yahooinfo” file for each session. This file contains a brief summary of every chatroom or IM along with the list of yahoo-ids and keywords matched. An example file is shown in figure 4.3.

The *Metahandler* maintains a table where each entry contains the name of the file corresponding to a chatroom or IM along with the list of matched keywords in the Yahoo criteria. It parses each packet read from the session file to figure out the packet’s service type, just like the way filter did. If the packet belongs to “AUTH” service, it stores the username. If it belongs to “CONFMSG” or “MESSAGE” or “JOIN” type, it looks up the table to find the name of its file. If no entry is found in the table, the Handler would enter new entry into the table with a new file name. Therefore all the messages corresponding to a chatroom or IM would be written into a separate file. These messages are checked for matching with the yahoo-ids and keywords in yahoo criteria, then the list of matched keywords are updated.

The *Yahoo Viewer* is a module in *PickPacket DataViewer* to display the connections belonging to Yahoo Messenger protocol. This module uses the “yahooinfo” file

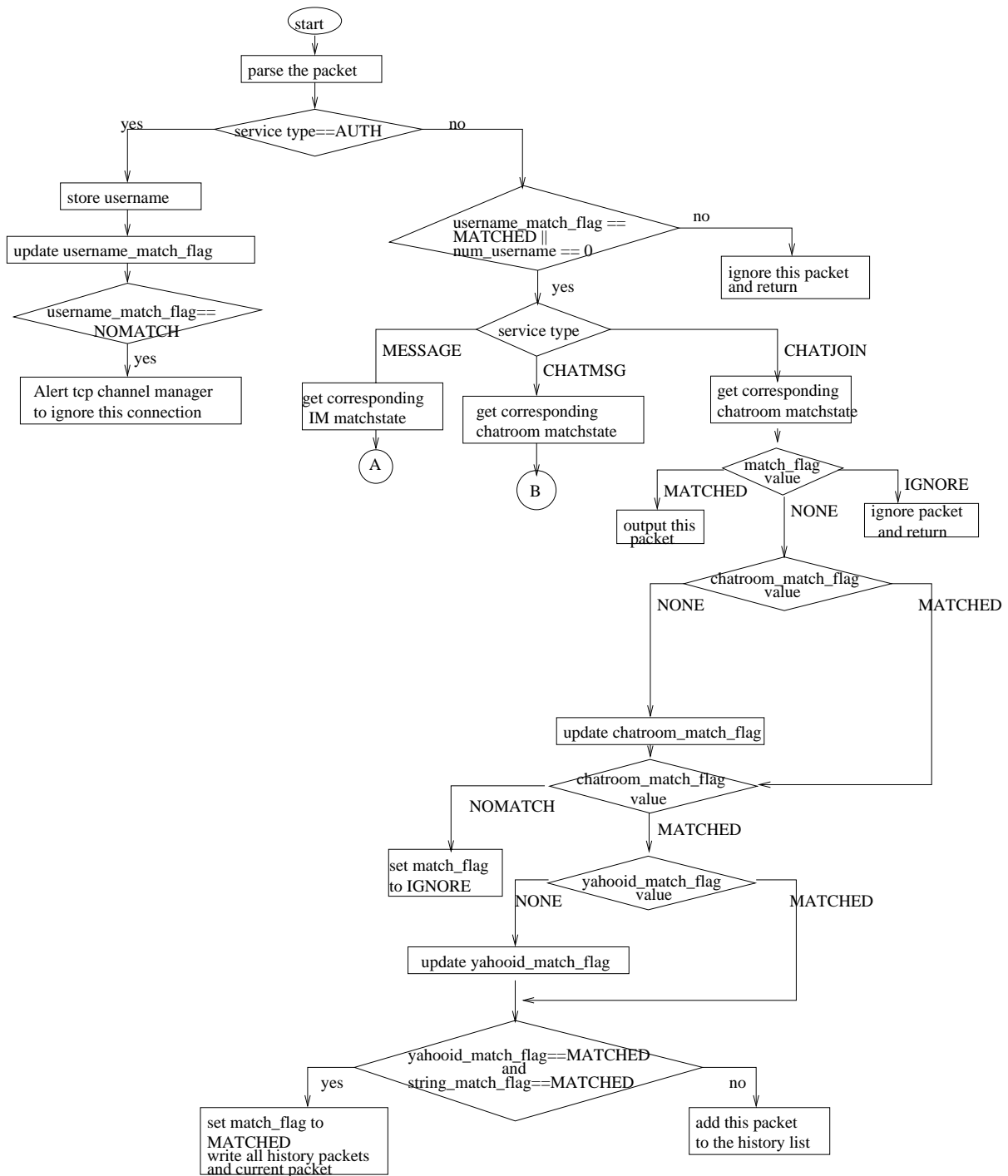
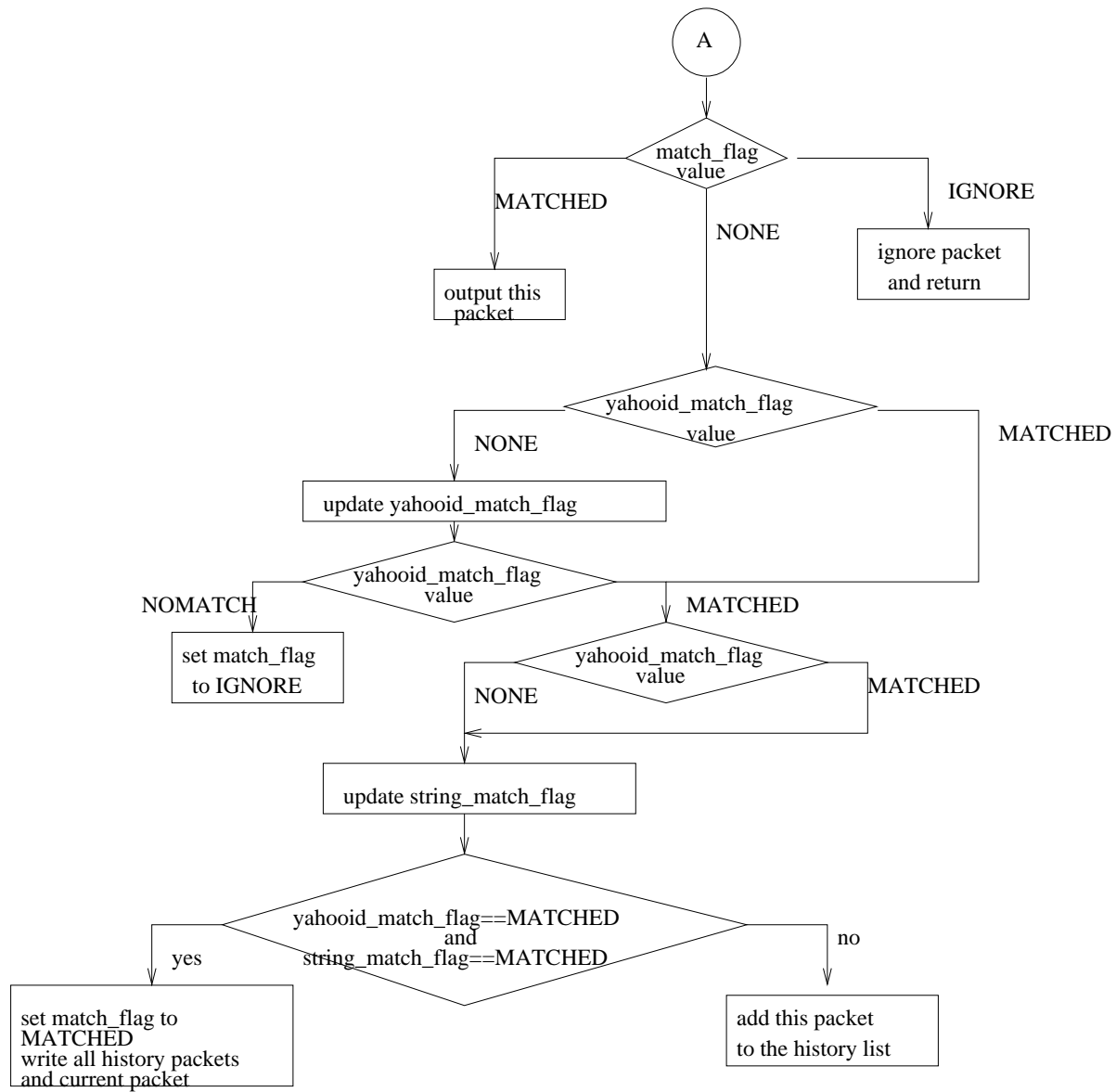
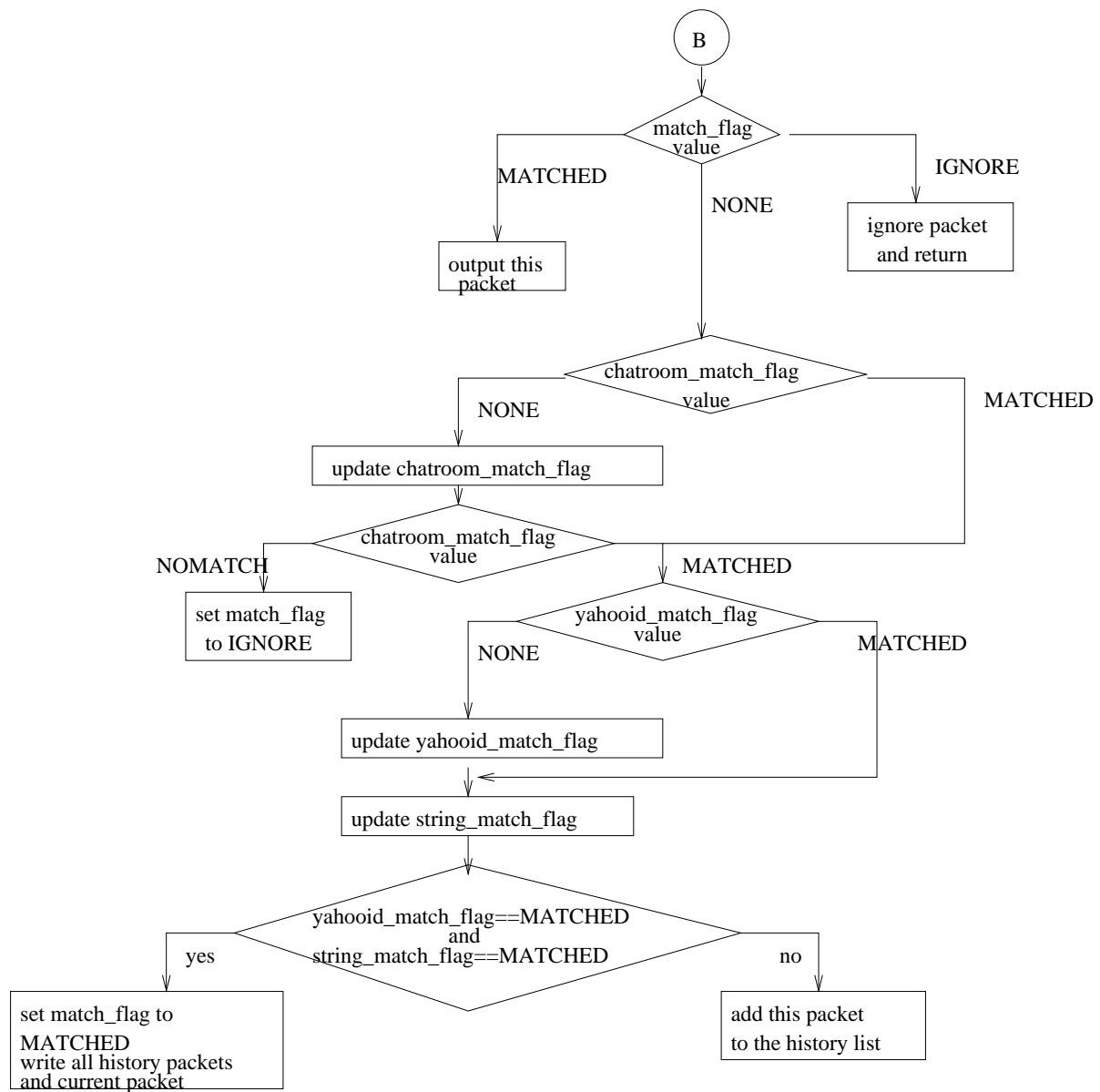


Figure 4.2: Working of yahoo filter





```
username = "tempuser"  
[yahoorecord1]  
type = "im"  
entryname = "imuser1"  
filename = "im0"  
keywords = "terror india"  
[yahoorecord2]  
type = "chat"  
entryname = "chatroom1"  
filename = "chat0"  
yahooids = "yahoo-id1 yahoo-id2"  
keywords = "bomb terror"  
  
_____  
_____  
_____
```

Figure 4.3: format of yahooinfo file

to display chat sessions and IMs present in a connection. It shows the chat sessions and instant message conversations in a separate dialog box. Users can search among the existing Yahoo sessions with different parameters.

Chapter 5

Tests and Results

In this chapter, we explain the experiments conducted to test the *PickPacket* with the new extensions in an actual network. The experiments are aimed to test the correctness and performance of the newly added modules for IRC and Yahoo Messenger. Section 5.1 deals with the correctness testing. The behaviour of the protocol filter has been tested in both “PEN” and “FULL” modes of capturing. Section 5.2 deals with the performance evaluation. The experiments for determining performance of the application layer filters are similar to experiments described in [10].

5.1 Correctness Verification

Goal of this experimentation 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 connection. An application protocol criteria contains values for several fields. An user can choose to store values for some of the these fields. A typical criteria may contain values for no fields which will cause the filter to write all the packets belonging to the corresponding protocol.

5.1.1 IRC Filter

An IRC criteria contains values for three fields - *channelname*, *nickname* and *keyword*. There are seven possible ways of preparing a criteria that contains values for atleast one field. We have created a test configuration file that contains seven possible IRC criterion. In our experimental setup we have used five machines with identical configuration. All these machines were connected to a HUB. The *PickPacket Filter* was run on one of the machines with above configuration file. IRC servers were started on two of the four machines. IRC clients were started on the remaining two machines. IRC traffic was created by creating a number of channels between these clients. Selective messages were transferred in these channels manually.

Above experiment was conducted for both “PEN” and “FULL” modes of packet capturing. The output files collected from the *IRC Filter* were post-processed to reconstuct the sessions. For each session, criteria match information generated by the *PostProcessor* was cross checked with the input configuration file. It is observed that *IRC Filter* stored those sessions which strictly matched the criteria. In case of “PEN” mode of packet captruing, the filter stored only the first packet of the channels matching the criteria.

5.1.2 Yahoo Filter

An Yahoo criteria contains values for four fields - *username*, *charoomname*, *yahoo-id* and *keyword*. There are fifteen possible ways of preparing a criteria that contains values for atleast one field. *Yahoo server* in Yahoo Messenger protocol communicates over HTTP protocol with the clients standing behind proxy servers. As HTTP protocol uses separate TCP connection for every message that is transfered between server and client, *Yahoo server* communication with these clients involves numerous HTTP connections. It is very difficult to correlate all these connections for YAHOO filtering. Therefore, we have used a host which connects *Yahoo server* over a direct link bypassing the proxy server. In our experimental setup we have used two identical machines connected to a HUB. The *PickPacket Filter* was run on one of the machines with the configuration file having fifteen possible criterion. The Yahoo client was

run on the other machine that is able to contact *Yahoo server* over the direct link. Yahoo protocol traffic was created by creating chatrooms and IM conversations with the people connected to the server.

Experiments were conducted for both “PEN” and “FULL” modes of packet capturing. The output files collected from the *Yahoo Filter* were post-processed to reconstruct the sessions. For each session, criteria match information generated by the *PostProcessor* was cross checked with the input configuration file. It is observed that *Yahoo Filter* stored only those sessions which strictly match the criteria. In case of “PEN” mode of packet capturing, the filter stored only the first packet of the chatrooms or IMs matching the criteria.

We have repeated above tests with configuration file containing criteria for every protocol supported by PickPacket. Traffic composing sessions belonging to various protocols was used for test. It is observed that the software is working properly even after addition of support for chat protocols.

5.2 Performance evaluation

Each packet received by the *PickPacket Filter*, is processed by the application protocol filter for criteria match. If the protocol filter is slow in filtering packets, the kernel may start dropping packets when its internal buffers gets filled up. This may result in not capturing of some packets even though they may be meeting user specified criteria. But, the Linux kernel does not provide statistics of packet loss happened due to application level filtering. Therefore, we use an instance of *PickPacket Filter* without application filtering to find the number of packets read. This count can be used to find the amount of packet loss. The earlier version of *PickPacket Filter* works at line speed in case of 100Mbps Ethernet segment. We have done experiments to ensure that the addition of chat protocol filters to PickPacket doesn't slow it down, and it still works at line speed.

Two identical machines with Intel Pentium 3.6GHz CPU, 2GB RAM and running linux kernel version 2.4.20-8 were used on a 100Mbps Ethernet segment. Both these machines were connected to a Hub. One instance of *PickPacket Filter* with no

application level filtering was run on a machine. The output file for this filter was specified as `/dev/null`. Thus, this instance of the packet filter reads all packets and writes them to the NULL device. The other instance of *PickPacket Filter* with application layer protocol specific criteria was run on second machine. The output file for this filter was a normal file. For simplicity the former packet filter is referred to as *counting sniffer* while the other as *filtering sniffer*. Filtering was stopped by setting a timer.

The *filtering sniffer* was run with configuration file containing criteria for all protocols supported by PickPacket. The criteria for IRC protocol includes 50 channel names, 50 nicknames and 50 keywords. The criteria for Yahoo protocol includes 50 usernames, 50 chatroom names, 50 yahooids and 50 keywords. Traffic containing SMTP, FTP, HTTP, POP, IMAP, IRC and Yahoo protocol sessions was generated to Hub. Significant portion of the traffic included data belonging to HTTP, SMTP, FTP protocols. Both the sniffers were started manually and ran for same time (6 minutes). In our experiment, *counting sniffer* processed 3533100 packets and *filtering sniffer* processed 3527900 packets. The small difference in number of packets is due to delay while starting the sniffers manually.

Thus the PickPacket Filter can work at line speed with out loss of infomation. We have measured the usage of processor and memory while running the filter and found that they were less than 50% used. Therefore, the filter can run at higher traffic speeds.

Chapter 6

Conclusion

This thesis discussed the filtering of packets flowing across the network by PickPacket with a special focus on filtering packets based on the IRC and Yahoo Messenger application level protocols criteria. 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.

PickPacket is a useful tool for gathering and rendering information flowing across the network. The design of PickPacket is modular, flexible, extensible, robust and efficient. Judicious use of the system can also help protect the privacy of individuals and can dump only necessary data to the disk. Tools for Post-processing and subsequent rendering make the tool easy to use. The universality of the capture file formats offer the user a choice of using “rendering and post-processing tools” other than those provided by PickPacket.

PickPacket is architecturally divided into four components the *PickPacket Configuration File Generator*, the *PickPacket Filter*, the *PickPacket Post Processor*, and the *PickPacket Data Viewer*. Design of each of these components were briefly discussed. PickPacket uses in-kernel filtering to capture packets at the network level. The packets filtered by the in-kernel filter are passed to the application level filter for further processing.

Modules for filtering IRC and Yahoo Messenger protocol packets have been further discussed in this thesis. Users of PickPacket can specify names of channels,

nicknames and text search strings for filtering packets belonging to IRC sessions. Usernames, chatroom names, yahoo-ids and keywords can be specified for filtering packets belonging to Yahoo Messenger sessions.

Experiments were conducted to check the performance of the IRC and Yahoo filters of PickPacket. These experiments show that these filters can successfully capture and filter packets on the basis of several criteria at reasonably high network loads.

6.1 Further work

PickPacket currently supports SMTP, POP, IMAP, Telnet, FTP, HTTP, IRC, and Yahoo Messenger application level protocols. There is always scope for extending PickPacket to support other application level protocols. PickPacket doesn't have support for decompressing the compressed data to do string matching. This would be required as electronic transfer of data in compressed format is popular.

Due to recent concerns over the impending depletion of the current pool of Internet addresses and the desire to provide additional functionality for modern devices, a new version of Internet Protocol(IP) called IPv6 [5] is in the process of standardization. This version resolves unanticipated IPv4 design issues and is poised to take the Internet into the 21st Century. Therefore, PickPacket would need changes for compatibility with IPv6.

References

- [1] Loris Degioanni, Fulvio Rizzo, and Piero Viano. “Windump”. <http://netgroup-serv.polito.it/windump>.
- [2] Gerald Combs et al. “Ethereal”. Available at <http://www.ethereal.com>.
- [3] “Etherpeek nx”. <http://www.wildpackets.com>.
- [4] “Gaim: A multi-protocol instant messaging (im) client”. “<http://gaim.sourceforge.net/>”.
- [5] “Ipv6: The Next Generation Internet!”. “<http://www.ipv6.org>”.
- [6] Van Jacobson, Craig Leres, and Steven McCanne. “tcpdump : A Network Monitoring and Packet Capturing Tool”. Available via anonymous FTP from <ftp://ftp.ee.lbl.gov> and www.tcpdump.org.
- [7] Neeraj Kapoor. “Design and Implementation of a Network Monitoring Tool”. Technical report, Department of Computer Science and Engineering, IIT Kanpur, Apr 2001. <http://www.cse.iitk.ac.in/research/mtech2000/Y011111.html>.
- [8] Steve McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture”. In *Proceedings of USENIX Winter Conference*, pages 259–269, San Diego, California, Jan 1993.
- [9] “Network Associates Incorporated”. <http://www.sniffer.com>.
- [10] Brajesh Pande. “The Network Monitoring Tool - Pickpacket : Filtering ftp and http packets”. Technical report, Department

of Computer Science and Engineering, IIT Kanpur, Sep 2002.
<http://www.cse.iitk.ac.in/research/mtech2000/Y011104.ps.gz>.

- [11] “Php Site”. <http://www.php.net>.
- [12] J. Oikarinen D. Reed. “Internet Relay Chat Protocol”. Technical report, 1993.
<http://www.faqs.org/rfcs/rfc1459.html>.
- [13] Stephen P. Smith, Henry Perrit Jr., Harold Krent, Stephen Mencik, J. Allen Crider, Mengfen Shyong, and Larry L. Reynolds. “Independent Technical Review of the Carnivore System”. Technical report, IIT Research Institute, Nov 2000. http://www.usdoj.gov/jmd/publications/carniv_entry.htm.
- [14] Venkat. “Yahoo Messenger Protocol(unofficial documnetation)”.
“<http://www.venkydude.com/articles/yahoo.htm>”.

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 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=generaltest
```

```

# 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>
*****Second Section*****
# The basic criteria here are for the Device and
# SrcIP1:SrcIP2:DestIP1:DestIP2:SrcP1:SrcP2:DestP1:DestP2:ProtoA:App
# Should be read as For the range of source IP from SrcIP1 to SrcIP2
#         For associated ports from SrcP1 to SrcP2
#         and For the range of destination IP from DestIP1 to DestIP2
#         For associated ports from DestP1 to DestP2
#         and FOR Protocol ProtoA
#         monitor connections according to Application App
# Protocols can be UDP or TCP
# Applications for TCP are
#     SMTP, FTP, HTTP, TELNET, POP, IMAP, IRC, YAHOO,
#     RADIUS, TEXT, DUMP_FULL, DUMP_PEN
# Applications for UDP are
#     DUMP_FULL, DUMP_PEN
# No further specs are required for DUMP kind of applications.
# Do not mix too many applications for clarity
# Take care that IPs Ports and applications do not conflict
<Basic_Criteria>

```

```

    DEVICE=eth0
Num_Of_Criteria=10
    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=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:110-110: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=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:23-23: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=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:143-143:TCP:TEXT
    Criteria=0.0.0.0-0.0.0.0:0.0.0.0-0.0.0.0:1024-65535:1024-65535:TCP:DUMP_FULL
</Basic_Criteria>
#*****End of Second Section*****>

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

#*****IRC Specifications*****
<IRC_Configuration>
    <IRC_Criteria>
        NUM_of_Criteria=1
            <Search_NickName>
                Num_of_nicknames=3
                Case-Sensitive=yes
                NickName=AVE
                NickName=ananth
                NickName=ramu
            </Search_NickName>
            <Search_Channel>
                Num_of_channelnames=1

```

```
        Case-Sensitive=yes
        ChannelName=VENHQ
        ChannelName=codevi
    </Search_Channel>
    <Search_Text_Strings>
        Num_of_Strings=4
        Case-Sensitive=no
        String=enti
        String=Exploiting
        String=gathulu
        String=unnava
    </Search_Text_Strings>
</IRC_Criteria>
<Port_List>
    Num_of_Ports=1
    IRC_Server_Port=6667
</Port_List>
Mode_Of_Operation=full
</IRC_Configuration>
#*****END of IRC Specifications*****
```

```
#*****Yahoo Specifications*****
```

```
<YAHOO_Configuration>
    <YAHOO_Criteria>
        NUM_of_Criteria=1
    <Usernames>
        Num_of_Usernames=5
        Case-Sensitive=yes
        Username=feel
        Username=india
        Username=delhi
```

```
        Username=fire
        Username=ricky
    </Usernames>
    <Chatrooms>
        Num_of_Chatrooms=5
        Case-Sensitive=yes
        Chatroom=Delhi
        Chatroom=Admirer
        Chatroom=Calcutta
        Chatroom=India
        Chatroom=Pakistan
    </Chatrooms>
    <Search_Email_ID>
        Num_of_email_id=50
        Case-Sensitive=yes
        E-mail_ID=fire
        E-mail_ID=balu_balu
        E-mail_ID=mumbai
        E-mail_ID=india
        E-mail_ID=private
    </Search_Email_ID>
    <Search_Text_Strings>
        Num_of_Strings=50
        Case-Sensitive=no
        String=invisible
        String=private
        String=defence
        String=india
        String=chat
    </Search_Text_Strings>
</YAHOO_Criteria>
```



```
Mode_Of_Operation=full
</YAHOO_Configuration>
#*****END of YAHOO Specifications*****
```

```
#*****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>
```

```
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*****
*****TEXT SEARCH Specifications*****
<TEXT_Configuration>
  <Search_Text_Strings>
    Num_of_Strings=1
    Case-Sensitive=no
    String=timesofindia
  </Search_Text_Strings>
  Mode_Of_Operation=pen
</TEXT_Configuration>
*****END of TEXT SEARCH Specifications*****
*****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=500
</NUM_CONNECTIONS>
    Num_of_IMAP_Stored_Packets=100
    Num_of_POP_Stored_Packets=100
    Num_of_IRC_Stored_Packets=100
    Num_of_IRC_Channels=10
    Num_of_YAHOO_Stored_Packets=100
    Num_of_SMTP_Stored_Packets=100
    Num_of_FTP_Stored_Packets=100
    Num_of_HTTP_Stored_Packets=100
    Num_of_TEXT_Stored_Packets=100
```