Computer Science Master's Project

A Network Packet Analyzer with Database Support

BY

Chi Yu Chan

chanc4@cs.rpi.edu

DEPARTMENT OF COMPUTER SCIENCE
RENSSELAER POLYTECHNIC INSTITUTE
TROY, NEW YORK 12180

August, 2002

Abstract

Network packets contain a lot of useful information about network activity that can be used as a description of the general network behavior. Network packet analyzers become a useful tool for system and network administrators to capture such kind of network information. In this report, an implementation of a network packet analyzer based on tcpdump[10], a popular network packet sniffer, will be described. This fully configurable tool concentrates particularly on its flexible input and output options so that it can easily be incorporated into other network tools to perform more complicated tasks, such as real-time or offline network intrusion detection. Database support is introduced in this tool as an output option for its well-known efficiency and convenience in handling huge amounts of information. A web front-end and the core packet analyzer can be integrated to become a database-backed web application as discussed in this report.

1 Introduction

In the modern society, computers are no longer treated as stand-alone machines. Instead, they are communicating to share resources and data through computer networks. Network packets are units of data traveling in these computer networks, and they carry all the important information from its source to its final destination. Besides the packet payload (the actual data) which contains lots of useful information, the packet headers themselves also contain a wealth of information about the network infrastructure, network topologies, and may also indicate some kind of general behavior of the network traffic. For example, the header information was used to discover the congestion sources in the network traffic in [5,6,7], and to analyze the quality of routing in the Internet in [9]. Another use of the packet header information is in Genesis, a distributed network simulation system [14,15,16], including wireless networks [11]. These pieces of information are thus important to the system and network administrators both from the statistical point of view and the network problem-solving point of view, as in [18]. In addition, they are useful to be the starting points in investigating possible intrusions by attackers who would like to compromise the computer systems through the fully connected computer networks. For example, in [4] the source-destination information was used to identify network attacks.

Thus, a network packet analyzer will be very useful to people who have the intentions to look into more details of what is actually going on inside the network. This report describes an implementation of a network packet analyzer based on the commonly used and freely distributed packet capturing tool tcpdump. This packet analyzer provides additional add-ons to the original sniffer which users may find helpful. These new features include a web front-end to allow users to configure this parser easily, a database backend which helps users organize their collected

information in a more systematic way and a flexible and fully configurable Java [8] parser which can easily be incorporated into other software with more complicated tasks. The system is intended for use in DOORS distributed network collection system [1,2,3].

This report serves as a starting point for users to know more about the functionalities and advantages of this tool. It describes the general usage information that users may have to know about this tool. Administrators will also benefit from this document as it indicates the requirements and installation procedures in setting up this tool. Chapter 2 describes the system architecture and some internal design issues. Chapter 3 is the user manual, a description of the system features from a user's point of view. Chapter 4 is designed for administrators who are willing to give a try to this tool. Chapter 5 includes a summary of this report.

2 System Architecture

2.1 Overview

A general overview of the system architecture and the interactions between the different system components are best described in Figure 1. Basically there is a web front-end which takes the user input and then sets up the required system environment, an analyzer wrapper which is responsible for starting and linking up the tcpdump and the parser programs, and also a database server which stores the packet information into database tables. This architecture requires the additional set up of a web server for hosting the HTML interface and a database server for data storage, which will be discussed later in the administrator manual. If the core analyzer component is incorporated into some other tools, the web front-end and the database backend setup can be omitted. A more detailed description of the functionality of each of the components is as follows.

2.2 Web Interface

This network packet analyzer emphasizes the ease of use and configuration of the several flexible options. Thus, a user-friendly interface is necessary to help users create their own settings. Screenshots and explanations of the options in the user interface will be described in more detail in the user manual section. In general, users can select whether the analyzer runs in real-time or offline mode, whether the output should be stored in the database server or simply a text file, and how network packet information is to be captured.

A perl [12] script will then receive the form input from the users and then do some error checking. If everything looks fine, the user settings will be saved into a configuration file and a

temporary directory will be created for storing this configuration file and other intermediate files. All the required files/information will also be set up in this temporary directory and when this is done, the analyzer wrapper will be called upon to continue processing the job.

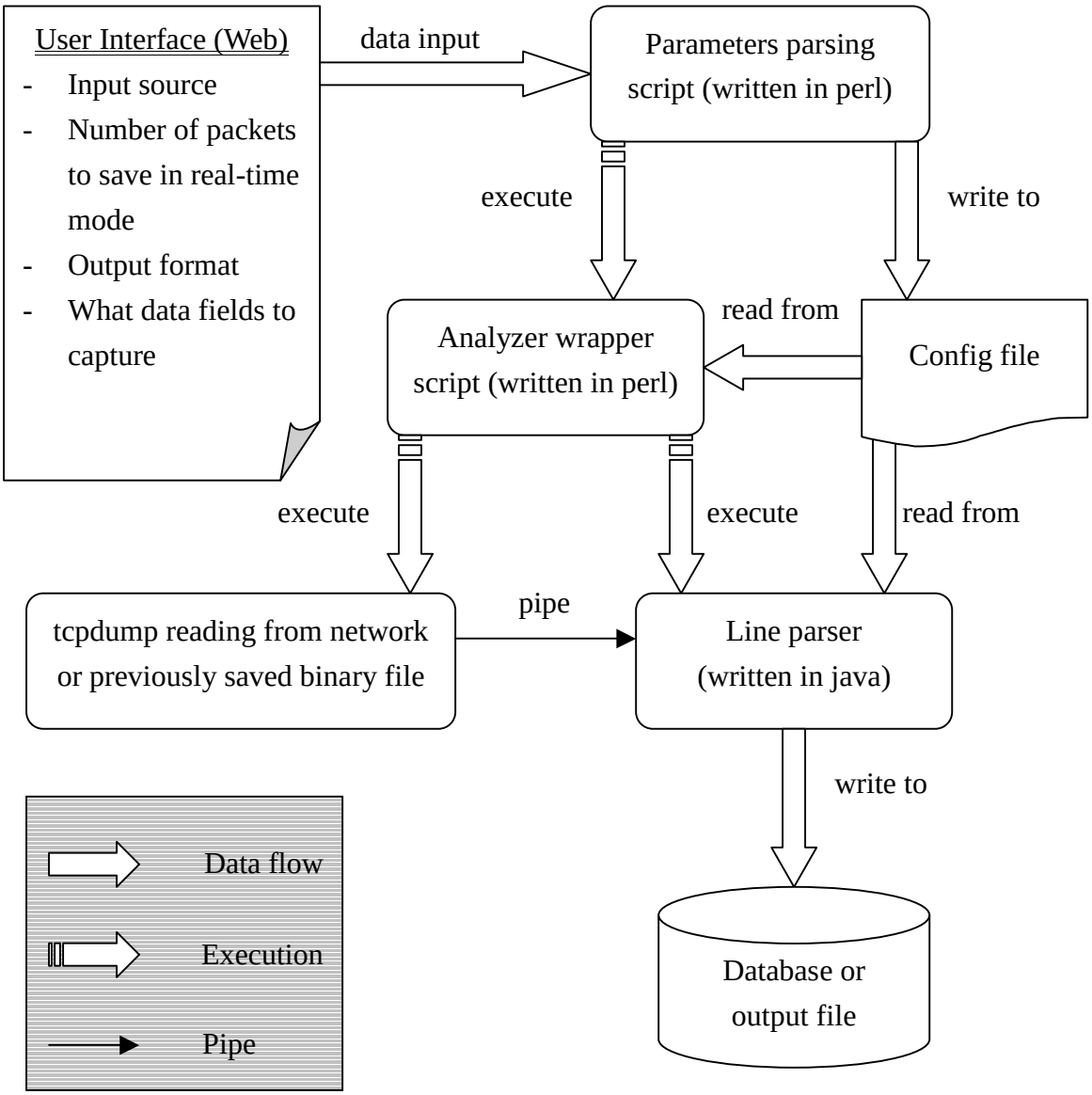

Figure 1 Interactions between the different system components

2.3 Analyzer Wrapper and Core Parser

The analyzer wrapper is actually a simple perl script that executes the tcpdump program, the java parser and then connects the output of the former and the input of the later using system pipes. The analyzer wrapper script is also responsible for sending email notifications to users when their jobs are completed. tcpdump is a commonly available software that does the actual packet sniffing in this tool. It can be run to capture real-time network packets or it can read from a previously saved tcpdump binary file to produce the corresponding outputs. The core parser, which is written in Java, then reads the packet information as a stream of hexadecimal digits from the tcpdump output, and will produce data output according to the output format specified in the configuration file by the user.

The Java parser is composed of 9 different classes. Six of them are for packet information storage, one is for user setting storage, one is for reading input and one is for printing output. The class hierarchy is shown in Figure 2.

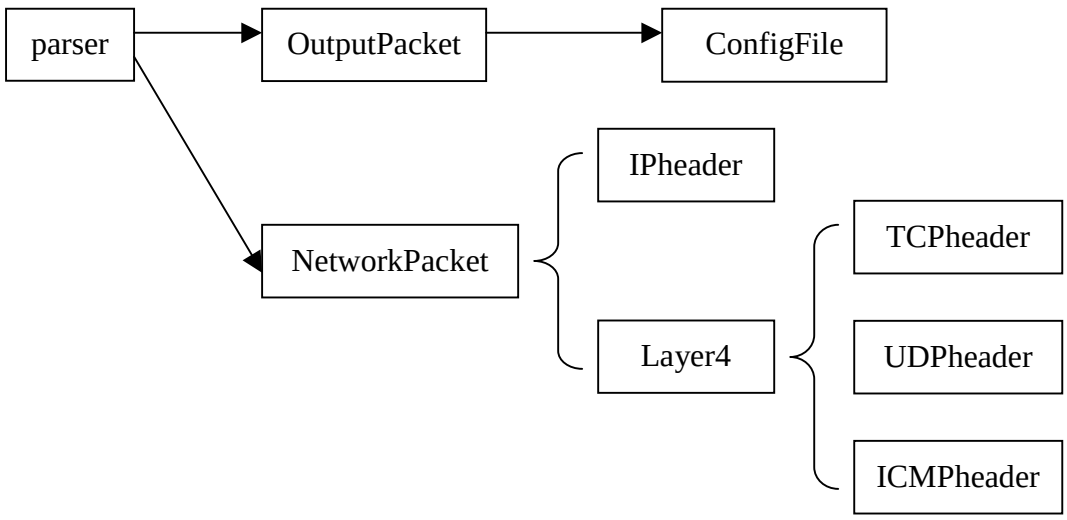

Figure 2 Class hierarchy of the Java core parser

A general description of the functionalities of each of these classes is shown below in Table 1.

Class Name	Functions
parser	➢ Read tcpdump output from standard input and process them line-by-line ➢ Create NetworkPacket for each packet and store data into this class ➢ Pass NetworkPacket to OutputPacket for printing to output
OutputPacket	➢ Read user settings into ConfigFile ➢ Read from NetworkPacket and do printout ➢ Insert packet information into database through JDBC [22] or print to standard output for text file output
ConfigFile	➢ Parse configuration file and store settings into variables ➢ Apply default settings if configuration file parsing fails
TCPheader	➢ Extract the 14 TCP header options [13,17] from hexadecimal packet payload and store them into variables
UDPheader	➢ Extract the 4 UDP header options [13,17] from hexadecimal packet payload and store them into variables
ICMPheader	➢ Extract the 3 ICMP header options [13,17] from hexadecimal packet payload and store them into variables
Layer4	➢ Collect a superset of all the available header options in TCPheader, UDPheader and ICMPheader
IPheader	➢ Extract the 16 IP header options from hexadecimal packet payload and store them into variables
NetworkPacket	➢ Read the actual packet payload from parser and create corresponding IPheader and Layer4 classes for data storage according to the protocol type ➢ Provide a generic interface to access options in packet headers even though the packet protocol is different

Table 1. Functionalities of all Java classes in the core parser

This layered structure is created to represent the data in network packets of different types because it provides a coherence interface for other classes to access the data, regardless of the actual packet type. An example would be that if a user wants to get the ICMP message type of all ICMP packets while many of the packets received are TCP packets, then this layered structure will automatically provide a default value (-1) for unavailable header option so that the same printout function can be used to generate the output without inserting some more conditional codes to check for the packet types. This structure thus provides an easy-to-use interface for other classes to include a section of codes without bothering too much about the internal packet type representation.

2.4 Database Server

One of the advantages of this tool is that the Java core parser has its built-in codes for communicating with the database server via the JDBC driver so that users are given an option to store their packet information in a more systematic way inside the database server for future references and analyses.

The data model used by this tool is pretty simple: There is a common table, called auth, that can be created by the following SQL commands:

```
CREATE SEQUENCE id_seq;

CREATE TABLE auth (
    id          integer primary key,
    password    text,
    email       text,
    columns     text
);
```

Each record in this auth table represents a single job submitted by the user, and each job has its unique id which is generated by the auto-incremental sequence id_seq. Columns store the header options that the user wants to capture, whereas password and email are used to authenticate the user when the user is retrieving the result from the web interface.

When a user submits a new job, a new id will be assigned to this job and a new record will be inserted into table auth. Another new separate table called n{id} will be created to store packet information in this job, where {id} is the newly assigned id for this job. Each column in this table represents a single header option selected to be captured by the user, whereas each row represents a single packet captured. When the user tries to retrieve the data from the web interface, a connection will be made to the database. A SELECT ALL command will be issued to retrieve all the records in the database and the records will be presented to the user in HTML format.

Instead of database output, another way of storage is plain text files. Each line in the text file represents a single packet and each line consists of all the field values separated by white spaces, and in the order selected by the user. This text file can easily be used as an input to some other programs that do further analyses on the network traffic.

2.5 Summary

In summary, some of the designs in this tool have advantages that make it possible to act as a suitable middleware in larger applications. These advantages include:

- The fully configurable parser with the use of a configuration file allows different user settings with no modifications to the source codes.

- Flexible input sources and output locations that make it easy to be incorporated into other software.
- Built-in database connecting codes provide an alternative way of storing and analyzing packet data.
- The layered structure in representing the packet information provides a coherence interface in accessing the information for different types of packets and it also provides scalability in extending the support of more different protocols in the TCP/IP protocol stack.

3 User Manual

3.1 Using the Web Interface

The web user interface is one that is written in HTML and Javascript to provide an easy-to-use and straightforward way in configuring the parser options. It also provides easy accessibility to users who just simply need to have a web browser. The web interface consists of two main sections: the job submission section and the result retrieval section. They will be discussed one by one in the following subsections:

3.1.1 Job Submission Page

A screenshot of the job submission page is shown in Figure 3. The descriptions of the fields can be found in Table 2.

Form Field	Description
Packet source	To select whether tcpdump should run in real-time mode to capture network data or offline mode to read from a previously saved tcpdump binary file
System path of tcpdump data file	When running in offline mode, this field is required to specify the location of the previously saved tcpdump file. The system path is required, meaning that you will need privileges to upload that file to the system before you can run this mode.
Maximum real-time packets stored	When running in real-time mode, this field is required to specify the stopping condition of the capturing process. The range of possible values is 1-10000. If you would like to run it continuously, you have to run the parser in command-line mode instead of using this web interface.
Storage location	To select whether the output should be directed to a text file or to a database
Fields to capture	To select the packet information that you would like to be shown in the output. The >> and << buttons are for selecting and deselecting

	fields respectively, whereas up and down control the order of appearance of the fields in the output.
Password to retrieve results	The password that you will be asked when you pick up the results. This is to protect your results from unauthorized viewers.
Re-enter password	To verify the correctness of the password that you have entered above.
Email address to receive notification	An email will be sent to you at this address to notify you of the completion of your job.

Table 2 Field Descriptions of the Job Submission Page

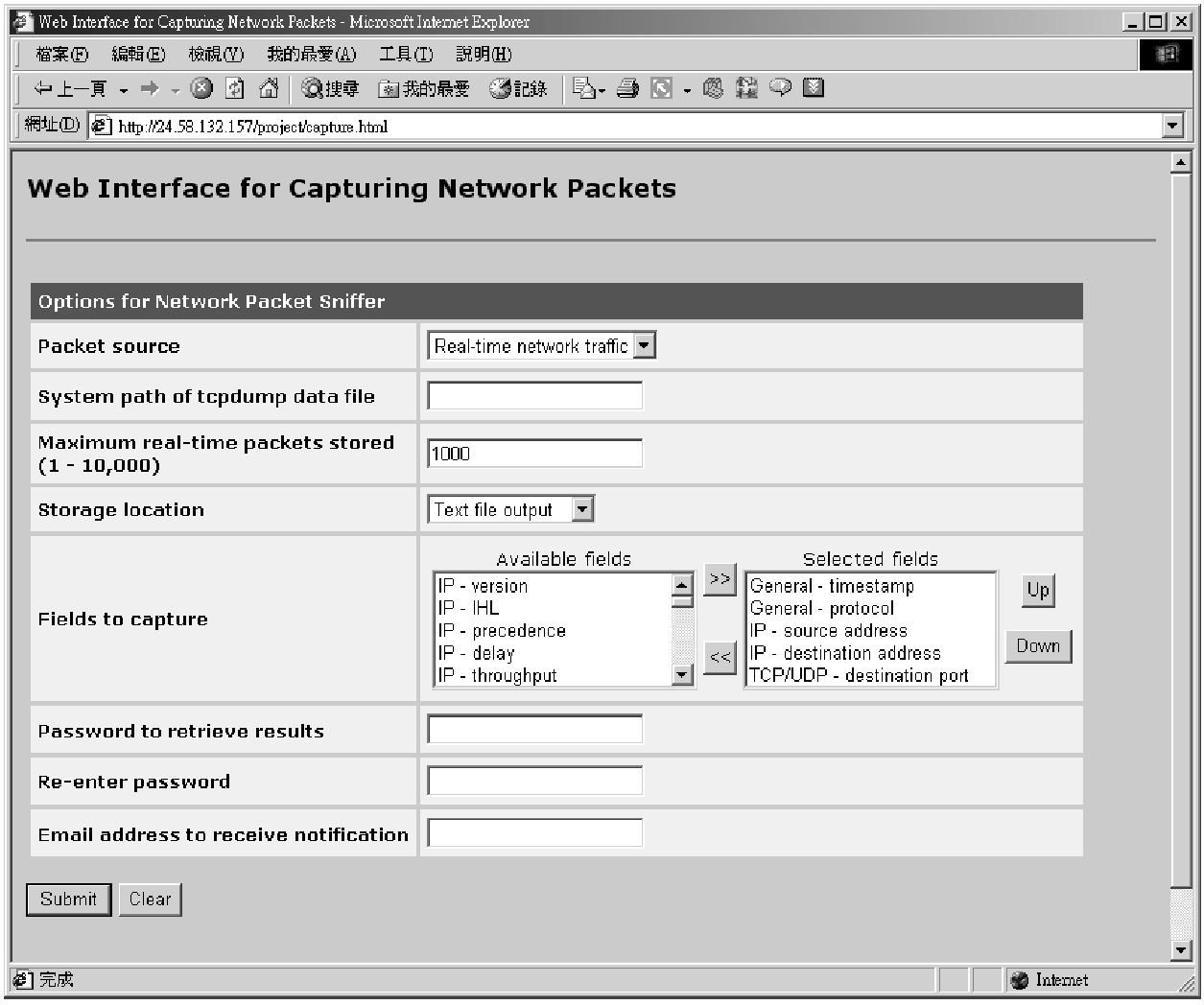

Figure 3 A Screenshot of the Job Submission Page

When this job submission form is filled out and submitted, you will get a notice about your job ID. With this ID, together with the password that you have entered, you can pick up the results of your job at a later time. In addition, you should be able to get an email notification when your job is done to remind you the job ID and the URL to pick up your results.

3.1.2 Job Retrieval Page

When you have got your email notification, you will be asked to come to this page to pick up your results. Figure 4 shows a screenshot of this job retrieval page.

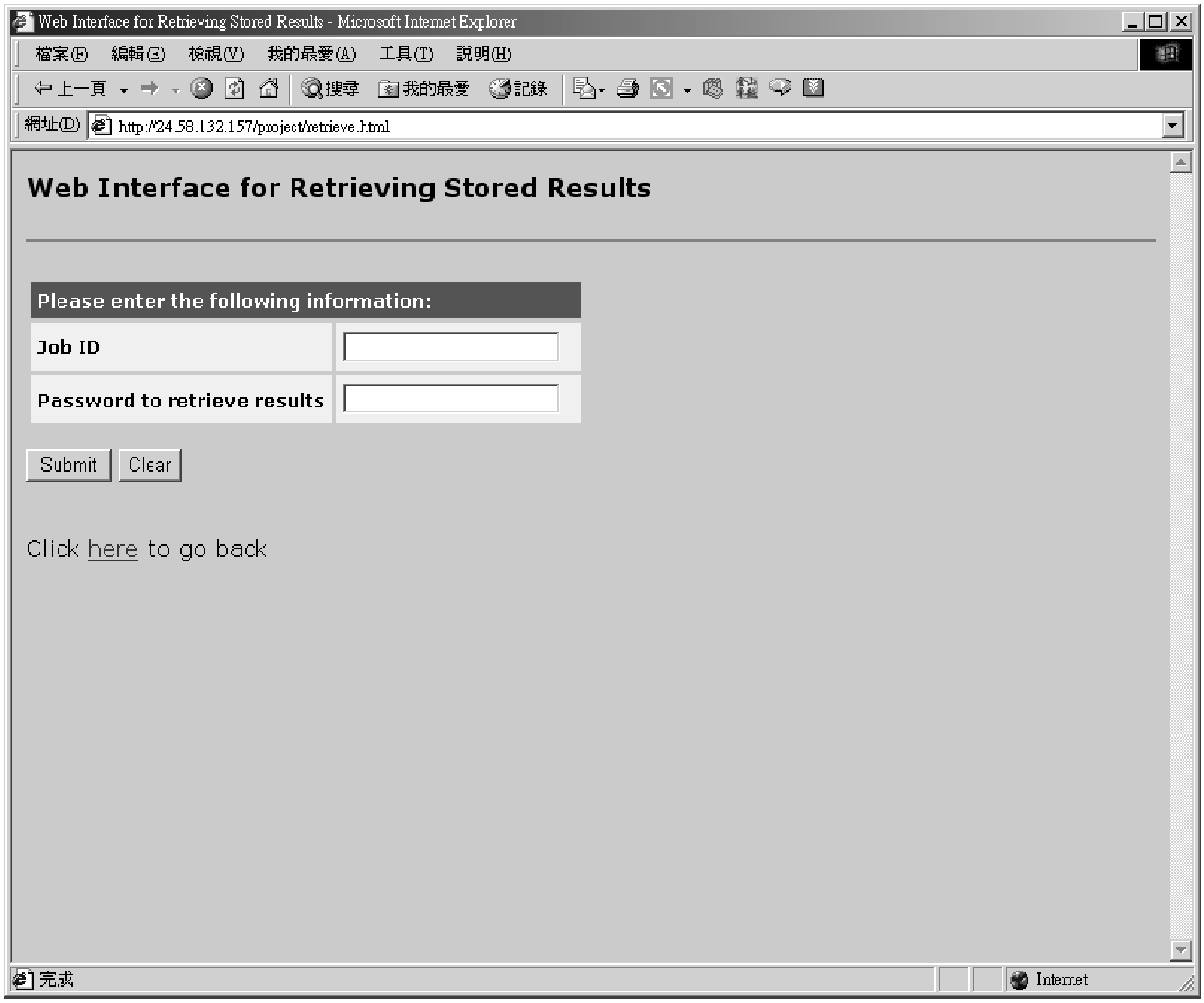

Figure 4 A Screenshot of the Job Retrieval Page

Enter the job ID and the password in this form and you should be able to get back your results. If yours is a database output, you should see something similar to Figure 5.

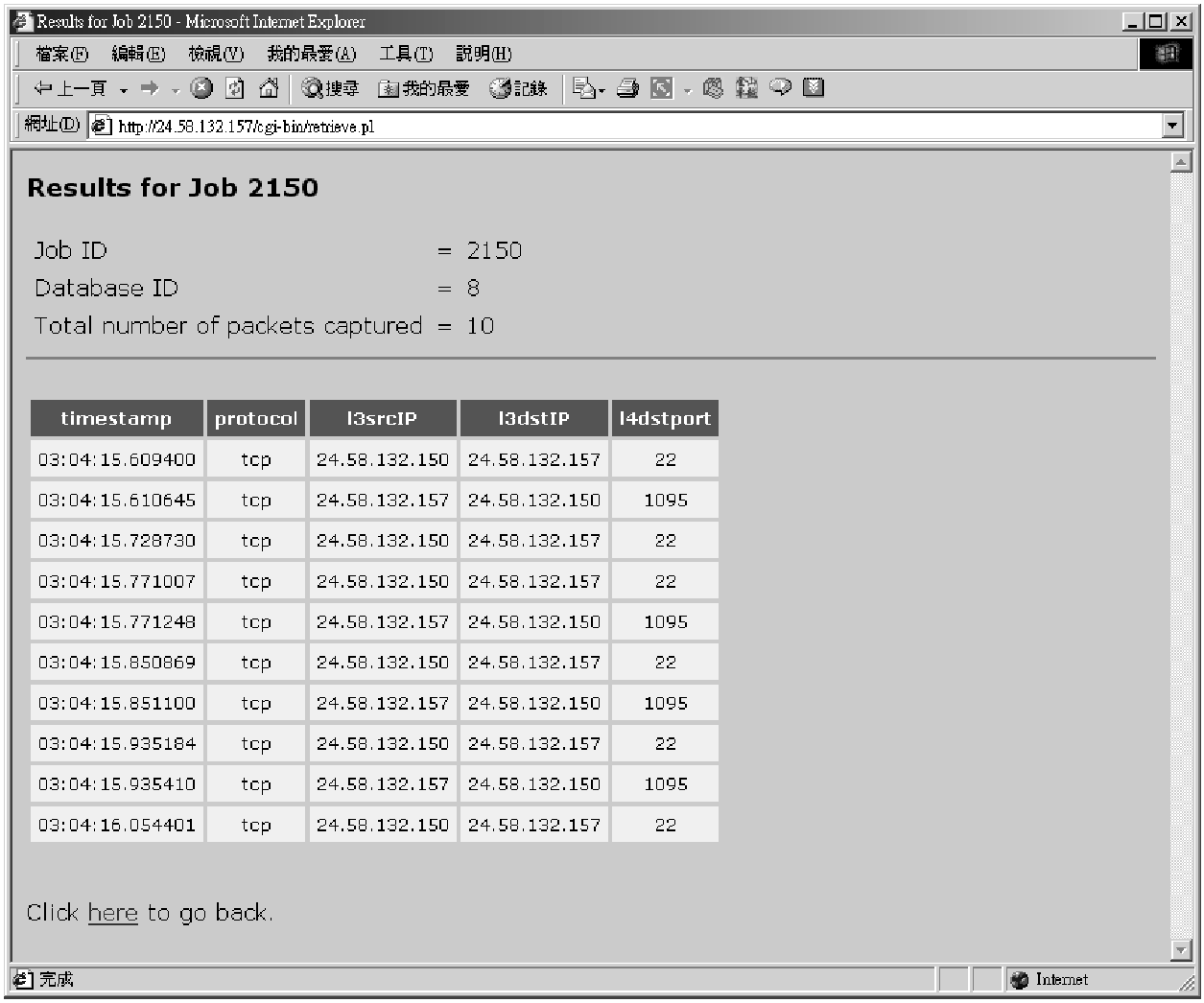

Figure 5 A Screenshot of the Database Output

This layout shows your job ID, the internal database ID represented for your database table, the number of packets captured in this job, and a list of all records and columns that you specified to record in your pre-defined order. If yours is a text file output, you should see something like Figure 6 instead.

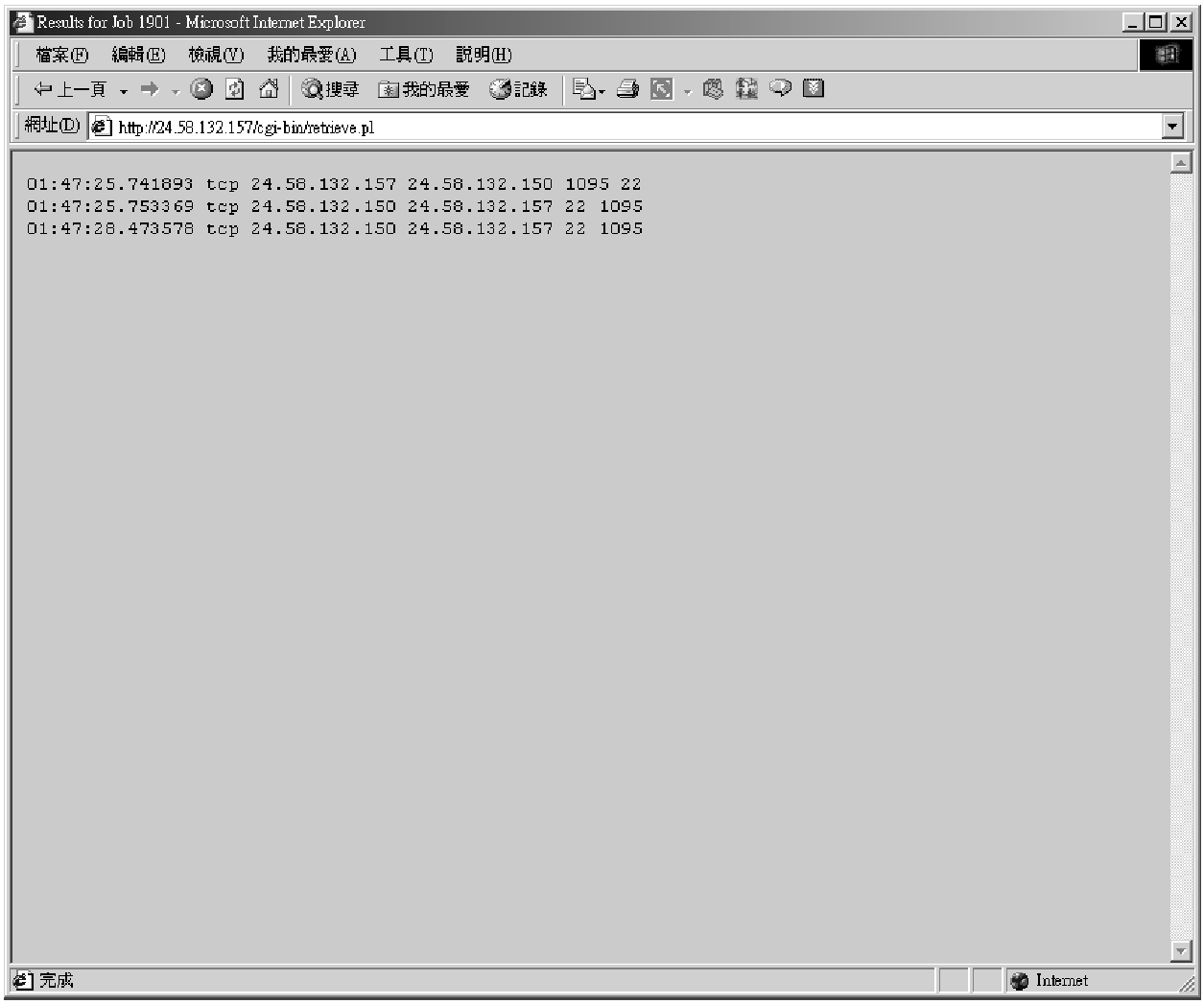

Figure 6 A Screenshot of the Text File Output

This is a much simpler output format, with each line representing a single packet. All the recorded header options are separated by white spaces. This simple layout allows you to copy and paste the results into a text file for further processing.

3.2 Running through the Command-line

If you prefer, it's also possible to run the Java parser directly from a command shell. All you have to do is to know about the configuration file format and the command-line options.

3.2.1 Configuration File

A sample configuration file is shown in Figure 7. Each line represents a single option, and every option comes in the `key=value` pair. The available keys include input, output, maxpackets, fields, password and email. The possible values and their meanings are shown in Table 3.

```
input = 0
output = 0
maxpackets = 1000
fields = timestamp,protocol,l3srcIP,l3dstIP,l4dstport
password = abc
email = chanc4@rpi.edu
```

Figure 7 A Sample Configuration File

Key	Possible Values and Meanings
input	➢ 0 means real-time network traffic (default) ➢ 1 means tcpdump data file
output	➢ 0 means text file output (default) ➢ 1 means database output
maxpackets	➢ Number of packets to be captured in real-time mode. The default is 1000.
fields	➢ Fields to be recorded. The default is `timestamp,protocol,l3srcIP,l3dstIP,l4dstport` ➢ Other possible values include: `l3version, l3IHL, l3precedence, l3delay, l3throughput, l3reliability, l3total_length, l3identification, l3DF, l3MF, l3fragment_offset, l3ttl, l3protocol, l3checksum, l4seqnum, l4acknum, l4header_length, l4URG, l4ACK, l4PSH, l4RST, l4SYN, l4FIN, l4window, l4urgent_ptr, l4srcport, l4checksum, l4length, l4type, l4code`
Password	➢ The password to retrieve the data from database. The default is empty.
email	➢ The email to get back the completion notification. The default is empty.

Table 3 Possible Key/Value Pairs in Configuration File

Whenever there is no value assigned to a particular key, the default will be used. In case the program is run without any configuration file, all the default values will apply.

3.2.2 Command-line Options

tcpdump provides the source of input to the Java parser in this tool. tcpdump can be run in many different options, but what is needed in this tool to run it continuously is simply:

```
tcpdump -x tcp or udp or icmp
```

where

- –x means printing out packet payload in hexadecimal digits
- tcp or udp or icmp means we just need TCP, UDP or ICMP packets. All the others will be ignored.

For the offline mode, when a previously saved tcpdump binary file is used as the source instead of the live network data, we need to add the –r option to make it become:

```
tcpdump -x -r tcpdump.file tcp or udp or icmp
```

where

- tcpdump.file is the location of the previously saved binary file

For the real-time mode, in case you do not want it to run continuously, but instead you want it to stop after a certain number of packets, the command line becomes:

```
tcpdump -x -c 1000 tcp or udp or icmp
```

where

- 1000 is the maximum number of packets that you want to record

For the Java parser itself, the simplest way to run it is:

```
java parser
```

where the default internal configuration will be used.

If you would like to use your own configuration file, the command will be:

```
java parser job.conf
```

where

- job.conf is the location of your own configuration file

If database output is needed and you don't have the JDBC jar file in your classpath, you may also have to add the -cp option:

```
java -cp {source}:{jdbc jar} parser job.conf
```

where

- {source} is the location of all the class files of the Java parser
- {jdbc jar} is the location of the JDBC jar file

Since the Java parser reads the input from the standard input, the easiest way is to pipe the output of tcpdump to the Java parser, i.e.,

```
tcpdump -x tcp or udp or icmp | java parser job.conf
```

If you select the text file output, you should get something like Figure 8. Redirecting the output to a file will provide you a saved result for future reference. If you select the database output, you will only get a database ID from the standard output as in Figure 9. With this database ID, you can connect to the database server and select the results for printout.

```
[root@joanne project]# tcpdump -x -c 10 tcp or udp or icmp | java parser job.conf
tcpdump: listening on eth0
17:50:44.777780 tcp 24.58.132.157 24.58.132.150 1095
17:50:44.778623 tcp 24.58.132.150 24.58.132.157 22
17:50:49.486433 tcp 24.58.132.150 24.58.132.157 139
17:50:49.486823 tcp 24.58.132.157 24.58.132.150 4401
17:50:49.673787 tcp 24.58.132.150 24.58.132.157 139
17:51:23.611997 tcp 24.58.132.150 24.58.132.157 139
17:51:23.612299 tcp 24.58.132.157 24.58.132.150 4401
17:51:23.799331 tcp 24.58.132.150 24.58.132.157 139
17:51:27.009984 udp 24.58.132.157 24.58.143.255 138
17:51:27.010031 udp 24.58.132.157 24.58.143.255 138
[root@joanne project]#
```

Figure 8 A Sample Text File Output Running in Command-line mode

```
[root@joanne project]# tcpdump -x -c 10 tcp or udp or icmp | java parser job.conf
tcpdump: listening on eth0
9[root@joanne project]#
```

Figure 9 A Sample Database Output Running in Command-line mode
(9 is the database ID in this case)

3.2.3 Connecting to the Database to View Results

PostgreSQL [19] is the database used in the sample tool. If other databases are used in your system, the corresponding documentations about using the database may have to be consulted. To connect to the PostgreSQL database, type:

```
psql -U httpd tcpdump
```

where

- –U specifies the username to be used to connect to the DB (httpd in this case)
- tcpdump is the database that you would like to connect to

Enter the password (if needed) for this user to log on to the database. Execute the SQL

command:

```
select * from n9;
```

to list all the results from your job (n9 is the database table in this case, where 9 is the database ID returned by the Java parser). A sample output can be found in Figure 10.

```
[root@joanne root]# psql -U httpd tcpdump
Password:
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

tcpdump=> select * from n9;
    timestamp    | protocol |    l3srcip     |    l3dstip     | l4dstport
-----------------+----------+----------------+----------------+-----------
 17:57:02.090059 | tcp      | 24.58.132.157  | 24.58.132.150  | 1095
 17:57:02.090982 | tcp      | 24.58.132.150  | 24.58.132.157  | 22
 17:57:02.767843 | tcp      | 24.58.132.157  | 24.58.132.150  | 1095
 17:57:02.978673 | tcp      | 24.58.132.150  | 24.58.132.157  | 22
 17:57:04.867521 | tcp      | 24.58.132.150  | 24.58.132.157  | 139
 17:57:04.867897 | tcp      | 24.58.132.157  | 24.58.132.150  | 4401
 17:57:04.868059 | tcp      | 24.58.132.157  | 24.58.132.150  | 4401
 17:57:04.868146 | tcp      | 24.58.132.150  | 24.58.132.157  | 139
 17:57:05.517429 | tcp      | 212.171.19.219 | 24.58.132.157  | 1105
 17:57:05.517471 | icmp     | 24.58.132.157  | 212.171.19.219 | -1
(10 rows)

tcpdump=> \q
[root@joanne root]#
```

Figure 10 A Sample Output Returned by the Database Server

3.2.4 Continuous Running of this Tool

As described earlier, the tool can be run continuously to capture network data without stopping by:

```
tcpdump -x tcp or udp or icmp | java parser job.conf
```

But it is advised that you should set a stopping condition for the tool. If it is really necessary to run this command continuously, you should not choose to save all the data output as it will

finally eat up all the disk space in the system and the tool will finally generate error messages reporting disk full. The best way to handle this is to pipe the output of this tool to another piece of software that is capable of parsing the output of this tool and to generate useful information when some of the conditions in the output are met. This avoids filling up the whole system disk space in a short period of time due to the large amount of network data captured by this tool.

4 Administrator Manual

4.1 System Requirements

This network packet analyzer requires the background support of many different separate pieces of software. The following is a list of the system environments used in the development platform of this software:

Operating System	Red Hat Linux 7.3 kernel version 2.4.18-3
Java Compiler and Interpreter	Java 2 Platform, Standard Edition version 1.4
Database Server	PostgreSQL version 7.2.1 with JDBC support
Web Server	Apache version 1.3.23 [20]
Perl Interpreter	Perl version 5.6.1 Perl DBI version 1.21 [23] Perl DBD-Pg version 1.01
sudo [21]	sudo version 1.6.5p2
tcpdump	tcpdump version 3.6.2
Mail Transport Agent	sendmail version 8.11.6

For installation instructions of each piece of software, please refer to their individual websites for more detailed documentations. It is assumed from this point onwards that all the above software has been successfully installed and configured to run normally.

4.2 Additional Configurations

After successful installation of the above software, there are a few more additional steps to configure them so that they can run properly for the network analyzer tool.

4.2.1 PostgreSQL

- Enable TCP/IP sockets: The JDBC driver uses the TCP/IP protocol to communicate with a PostgreSQL server. However, this is not used by default. Edit the file postgresql.conf under the database root directory and add the line:

  ```
  tcpip_socket = true
  ```

  to enable this option. Restart the database server to make this effective.

- Allow access to PostgreSQL from the network: PostgreSQL does not allow access from locations other than local sources by default. Edit the file pg_hba.conf under the database root directory and add the following lines:

  ```
  host   all    127.0.0.1     255.255.255.255 password
  local  all                                  password
  ```

  This should allow access via TCP/IP.

- Create a new user for the application to connect as: Execute the wrapper script createuser to perform this task.

- Create a new database for use by the application: Execute the wrapper script createdb to perform this task.

4.2.2 Java

- Add the JDBC jar file to the environment variable CLASSPATH so that the Java interpreter knows where to find the JDBC-related codes. On the development platform, the location is: /usr/share/pgsql/jdbc7.2dev-1.2.jar

- Download all Java source codes to the same directory.

- Edit OutputPacket.java and insert the PostgreSQL username, password and database that you would like to connect as.

- Compile the source codes using javac *.java.

4.2.3 Apache Server

- Download all perl scripts and put them into the cgi-bin directory. On the development platform, this directory is /var/www/cgi-bin. Edit all scripts to adjust some of the system-dependent parameters.
- Download all HTML files and put them into the HTML root directory. On the development platform, this directory is /var/www/html. It is advised to password-protect these pages as unauthorized users will be able to get sensitive network traffic information through these pages. Documentations about how to add passwords to your site can be found at Apache's website at http://www.apache.org/
- Put the script capturebgjob.pl into the directory where you store the Java source codes.
- Since the web server is usually run as a separate user who has no root privileges while the execution of the program tcpdump requires root access, the solution in this tool is to use sudo to execute the actual tcpdump statement. Before you can do that, you have to grant access to this program to the web server user. Run visudo to edit the sudo configuration file and add the line:

  ```
  apache  ALL=NOPASSWD:    /usr/sbin/tcpdump
  ```

  where apache is the user who owns the web server processes and /usr/sbin/tcpdump is the location of the tcpdump program. After doing this, perl scripts run by the user apache will be able to execute tcpdump. These steps, to a certain extent, open a security hole to the system where other scripts run by the web server will also have access to the tcpdump program. In order to block this hole, it is recommended that only authorized users can edit and add perl scripts to be run by the web server. Under this condition where no other

unauthorized users can run jobs as apache, the system should be safe.

4.3 Summary

After doing all the above configurations, open a web browser and connect to your web server. You should be able to get the web interface for this network packet analyzer. If in any case the system does not work as expected, please always go to /var/log to check the system logs, web server logs and also the database server logs for further explanation of the problems.

5 Conclusion

This project implemented successfully original concepts in designing a network packet analyzer that is easy-to-use and suitable for further information analyses. The fully configurable core parser has also been applied to another intrusion detection project to act as the Data Filtration Unit of the intrusion detection system, and it proves the effectiveness and correctness of the results generated by this packet analyzer. The fully configurable parser with the use of a configuration file allows different user settings with no modifications to the source codes. The flexible input sources and output locations make it easy to be incorporated into other software. The layered structure in representing the packet information in the Java source provides a coherence interface in accessing the information for different types of packets and it also provides scalability in extending the support of more different protocols in the TCP/IP protocol stack. Future developments in this project may include the addition of packet filtering conditions on different packet header information, the addition of the data portions in network packets as one of the parameters to be collected and the addition of other protocols to be supported by this analyzer.

References

[1] A. Bivens, L. Gao, M. F. Hulber and B.K. Szymanski, ``Agent-Based Network Monitoring," *Proc. Autonomous Agents99Conference*, Seattle, WA, May 1999, pp. 41-53.

[2] A. Bivens, P. Fry, L. Gao, M.F. Hulber, Q. Zhang and B.K. Szymanski, ``Distributed Object-Oriented Repositories for Network Management," *Proc. 13th Int. Conference on System Engineering*, pp. CS169-174, Las Vegas, NV, August, 1999.

[3] A. Bivens, R. Gupta, I. McLead, B. Szymanski and J. White, ``Scalability and Performance of an Agent-based Network Management Middleware," *International Journal of Network Management*, submitted, 2002.

[4] A. Bivens, M. Embrechts, C. Palagiri, R. Smith, and B.K. Szymanski, ``Network-based Intrusion Detection using Neural Networks," *Intelligent Engineering Systems through Artificial Neural Networks*, Vol. 12, Proc. ANNIE 2002 Conference, November 10-13, 2002, St. Louis, MI, ASME Press, New York, NY, 2002, pp. 579-584.

[5] A. Bivens, M. Embrechts, and B.K. Szymanski, ``Forecasting and Mitigating Network Congestion using Neural Networks," *5th Online World Conference on Soft Computing in Industrial Applications,*, September 4 - 18, 2000 http://wsc-virtual.hut.fi/.

[6] J. Bivens, M. Embrechts, and B.K. Szymanski, ``Network Congestion Arbitration and Source Problem Prediction Using Neural Networks," *Smart Engineering System Design*, vol. 4, 2002, pp. 243-252.

[7] J. Bivens, B.K. Szymanski, and M. Embrechts, ``Network Congestion Arbitration and Source Problem Prediction using Neural Networks," *Proc. Artificial Neural Networks in Engineering, ANNIE'2000*, ASME Press, Fairfield, NJ, 2000, pp.489-494.

[8] M. Campione and K. Walrath, *The Java Tutorial Second Edition*, Object-Oriented Programming for the Internet, Addison-Wesley, 1998.

[9] S. Gurun and B.K. Szymanski, ``Automating Internet Routing Behavior Analysis Using Public WWW Traceroute Services," *Proc. IFIP/IEEE MMNS'2000 Conference*, Fortaleza, Brazil, September 2000, Kluwer Academic Publishers, Boston, MA, 2000, pp. 47-59.

[10] V. Jacobson, C. Leres, and S McCanne, “Tcpdump,” June 1989, Available via anonymous FTP from ftp.ee.lbl.gov.

[11] Mandani and B.K. Szymanski, ``Integrating Distributed Wireless Simulation Into Genesis Framework," *Summer Computer Simulation Conference*, Montreal, Canada, 2003, to appear.

[12] Randal L. Schwartz, *Learning Perl*, Second Edition, O’Reilly, 1997.

[13] W. Richard Stevens, *TCP/IP Illustrated*, vol. 1, Addison-Wesley Publishing Company, One Jacob Way; Reading, Massachusetts 01867, 1994.

[14] B.K. Szymanski, Q. Gu, and Y. Liu, ``Time-Network Partitioning for Large-Scale Parallel Network Simulation under SSFNet," *Proc. Applied Telecommunication Symposium*, San Diego, CA, April 14-17, 2002, SCS Press, pp. 90-95.

[15] B.K. Szymanski, Y.Liu, A. Sastry, and K. Madnani, ``Real-Time On-Line Network Simulation," *Proc. 5th IEEE Int. Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001*, IEEE Computer Society Press, Los Alamitos, CA, 2001, Cincinnati, OH, August 13-15, 2001, pp. 22-29.

[16] B.K. Szymanski, A. Saifee, A. Sastry, Y. Liu and K. Madnani, ``Genesis: A System for Large-scale Parallel Network Simulation," *Proc. 16th Workshop on Parallel and Distributed Simulation*, Washington, DC, May 12-15, 2002, IEEE CS Press, pp. 89-96.

[17] Andrew S. Tanenbaum, *Computer Networks*, Third Edition, Prentice Hall International Editions, 1996.

[18] T. Ye, S. Kalyanaraman, B. Mo, B.K. Szymanski, D. Harrison, B. Sikdar, H. Kaur, and K. Vastola, ``Network Management and Control Using Collaborative On-line Simulation," *Proc. IEEE Int. Conference on Communications ICC2001*, IEEE Computer Science Press, Los Alamitos, CA, 2001, Helsinki, Finland, June 2001.

[19] http://www.postgresql.org/

[20] http://www.apache.org/

[21] http://www.apache.org/

[22] http://www.j-elite.com/pgprimer/index.jsp

[23] http://www.perldoc.com/cpan/DBI.html