Using Java to Teach Networking Concepts With a Programmable Network Sniffer

Michael J Jipping
Department of Computer Science
Hope College
Holland, MI 49423
jipping@cs.hope.edu

Agata Bugaj
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
abugaj@andrew.cmu.edu

Liliyana Mihalkova
Department of Computer Science
Hope College
Holland, MI 49423
mihalkova@cs.hope.edu

Donald E. Porter
Department of Mathematics and Computer Science
Hendrix College
Conway, AR 72032
porterde@mercury.hendrix.edu

Abstract

A crucial part of the Networking course is the examination of and experimentation with network traffic data. Most standalone network traffic sniffers are quite expensive and those freely available on general purpose platforms (e.g., Linux or Windows) are quite cryptic. Because of this, we have developed NetSpy: a Java-based network sniffer that allows plug-in Java modules to analyze network data. These modules are written by students as part of their experimentation with traffic data. This paper describes the NetSpy system and the way we use this in Networking class.

1 Introduction

In the computer science curriculum, the Networking course stands out by virtue of its requirements. The course requires that a lot of difficult subject matter be delivered to the student. We believe that some of this material needs to be delivered via "hands-on" active learning activities. Active learning and experimentation in a Networking course is typically done by examining a network and interpreting the applications and protocols that run across that network.

Unfortunately, tools that facilitate network experimentation are either expensive or difficult to use in a class setting. Special purpose tools to sniff networks are quite expensive and do not allow general purpose programming to analyze network data. Tools available on Linux or Windows platforms generate cryptic output that is not easily deciphered or analyzed.

This paper documents a system designed to teach Networking concepts and analysis by using Java to program a plat form that delivers network data. We have developed a system called NetSpy that consists of a network sniffing platform coupled with a Java plug-in interface. Using this interface, students can write Java plug-in modules that can receive network data for analysis. NetSpy has been implemented for handheld computers, which extends its usefulness and experimentation ability.

This paper will present some background on NetSpy, followed by an examination of its system components. We will consider an example of a NetSpy Java module and conclude with some discussion on class use.

2 Background and Motivation

A key to the Networking course is network data analysis. In our course, we would like to perform activities like traffic measurement, observing and analyzing network protocols in detail, and developing solutions to network problems.

However, network analysis is difficult to do, especially if students need to develop all aspects of the analysis tools. If students program network sniffing themselves, for example, it is very easy to get mired in the data structures and system interfaces needed to properly fetch a network packet and extract data that can be analyzed. Students should focus on the analysis itself, not the mechanics of traffic sniffing.

Very little software exists to assist a student in this endeavor. For a general purpose platform, there are indeed tools that will watch network traffic. Applications such as snoop on Solaris platforms and tcpdump [3] on Linux are

very effective in fetching data traffic and displaying the contents of each network packet. Unfortunately, these tools display the data in a cryptic manner and are not at all programmable. Tools such as etherape [1,2] and ethereal [4] on Linux devices attempt to present network data in a format that is more easily analyzed and understood, but again do not allow for general analysis through custom designed programs

We seek a platform for network analysis that provides the following functionality:

- *Network data retrieval:* The platform should accurately read network packets and present these packets to the user in a clean, usable manner.
- *Ability to filter network data:* The platform should be able to filter the network packet data to present only the specific type of data requested by the user.
- *Network data object orientation:* For intuitive analysis, the platform should present network data to the user in an object-oriented manner, with a clean, usable method interface.
- *Progammable network data analysis:* The platform should allow the user to use a wide range of custom-built, programmable analysis tools to examine network data.

As an added bonus, the ability to use a handheld computer platform would be very beneficial. This would enable students to sample data from various sources all over campus – not just from a single computer laboratory or classroom.

3 The NetSpy System

The NetSpy system is a network data gathering platform, written in Java, that accepts Java programs as data analysis modules. Since we constructed it to abide by the criteria we established, the goals of the NetSpy system correspond to them: it accurately retrieves network data, allows the user to focus only on specific data, provides an object-based interface to Java-based analysis modules that the user provides. In addition, it runs on a Sharp Zaurus SL-5500 Linux handheld computer as well as other desktop platforms.

NetSpy uses PCAP as the foundation for reading network data. It allows the user to build simple filters and combine them into more complex filtering objects. Its structure and use is quite intuitive. We will take a closer look at each of the aspects of NetSpy in this section.

3.1 The NetSpy Foundation

PCAP is a packet capture library that provides a uniform interface to the underlying packet capture applications, which vary with different operating system implementations. PCAP is used in packet analysis applications including tcpdump, ethereal, and etherape. PCAP is also the recommended starting point for the novice network analysis writer. It provides the programmer with a C-style character array containing the data of a packet, which the user then parses.

Although PCAP abstracts many details of packet capture nicely, it does not hide any facet of network data or cross-platform programming in C, especially the varying size of a given data type or even a byte. In all network protocols, all data sizes are in terms of bits or octets, which do not differ. In nearly all network protocols, data is always represented in big endian byte order. Thus, a language such as Java, which guarantees a constant size and byte order of its data types, is better suited for packet representation.

Packet retrieval in NetSpy is implemented by its Network class. This class of NetSpy gets a packet buffer from PCAP. The classe then parses the packet buffer using C/C++ native methods through Java Native Interface, placing the data into appropriately sized Java data types. Consider an Ethernet header, which consists of a Destination Address (six octets), Source Address (six octets), frame type (two octets), and a variable amount of data (ignoring the preamble and checksum, which are dropped by pcap). NetSpy provides the following interface to this data:

```
public class Ethernet{
    public Ethernet(Packet p);
    //constructor, which takes an unparsed
Packet, from the
    //Network class, as an argument
    public byte[] getDestinationAddress();
    public byte[] getSourceAddress();
    public short getType();
    public byte[] getDataBytes();
}
```

The user could then try to obtain the data of a higher level protocol, IP for example, from the byte array returned by *getDataBytes*(), which is essentially what PCAP requires of its users. Because network protocols are essentially static, NetSpy can perform the task of packet parsing for the user through object-oriented design. NetSpy's packet classes are designed so that each header packet takes a packet object of the immediately lower protocol as a constructor argument. In this case, class IP would take a Data Link Layer Header (Ethernet or 802.11 Data Frame) as an argument and then provide a similar interface to its contents.

The NetSpy packet system provides an intuitive and simple API to the writer of a module. If a student wanted to obtain the destination port of each TCP packet that came across the network, for example, the code would look something like this:

```
    short port;
    Packet p = _packetQueue.dequeue();
    Ethernet e = new Ethernet(p);
    if(e.getType() == 0x0800){
       IP ip = new IP(e);
```

```
        if(ip.getProtocol() == 6){
            TCP tcp = new TCP(ip);
            port = tcp.getDestinationPort();
        }
    }
```

3.2 Network Data Retrieval Components

The main components of the network listener core of NetSpy are the Filter, the Operation, the Agent, the AgentMap, and the Dispatcher.

The Filter class provides filtering functionality. Each Filter is defined through the graphical user interface and is constructed using a string of arguments that contains the filtering criteria. Users can filter by protocol, such as Ethernet, IP, TCP, and UDP, by host name, port number, packet length, and by any combination of these criteria. Moreover, individual filters can be combined using Boolean operations, such as AND, OR, AND NOT, and OR NOT, thus producing composite filters. The Filter class represents each composite filter as a tree structure. This model not only improves the efficiency of filter evaluation, but also preserves the correct order of evaluation of the constituent filters, which can be reproduced by a postorder traversal of the filter tree. This allows the Filter class to represent composite filters of any size and complexity. From a user's perspective, because the Filter object encapsulates both filters and composite filters, no difference exists between these two entities.

The Operation is an abstract class that hides the convoluted details of thread management from the user and provides a template for defining custom modules for packet analysis. By extending the Operation class and implementing its **void task()** method, users can easily program their own operations that define the actions to be performed on arriving packets.

An Agent is the union of an Operation and a Filter. Netspy supports multiple Agents simultaneously and allows users to activate and deactivate them as needed. To manage the Agents that are currently present, Netspy employs the services of the AgentMap class, which maintains a list of all Agents and Filters currently defined in the system. Each Filter and each Operation can be part of zero or more Agents, whereas each Agent should contain exactly one Filter and one Operation. The AgentMap provides methods

Figure 1. Main NetSpy User Interface

for managing a single agent, such as creating an agent or changing the filter of an agent in real time. It also provides functions for managing all the agents as a group, such as saving and retrieving agents.

The Dispatcher provides the link between the Network layer and the Agents. Its main function is to check each packet it receives from the Network against the Filter of each of the active Agents in turn, and if the packet passes, to forward a copy of it to the corresponding Operation.

3.3 Using NetSpy

NetSpy is used through its graphical user interface, which makes the program easy to use and understand. Not only is the GUI very 'user-friendly' but it also lets the user experience all of NetSpy's many features.

The main GUI of NetSpy, shown in Figure 1, contains five buttons entitled Filter, Agent, Start, Stop, and Help. The Filter and Agent buttons contain all functions that any Filter or Agent can have, respectively. The Start and Stop button control network packet flow. The Help button contains information about how to use NetSpy.

NetSpy provides many functions for the Filter. The user can define a filter, delete a filter and even combine two filters into a composite filter. Figure 2 shows a snapshot of the filter cration process.

The Agent also has many functions. An agent can be created, deleted, changed, stopped, restarted, renamed, and saved. Each of the Agent and Filter functions can be performed through a simple step-by-step procedure provided by the GUI.

One of the major advantages of using NetSpy compared to other sniffing programs is the option to create Operations. Operations represent data analysis in any way that the user deems fit. We have tried to minimize the work a user must do to provide an Operation; all that needs to be done is the writing of a single function. The Operation class is an abstract class that extended the system Thread class. A data analysis module, then, must extend the Operation class, and a single function called by Operation's **run()** method is what needs to be defined. The Operation begin extended has access to a drawing surface, all network data objects, and the main program's GUI objects.

Once defined, a user-supplied Operation is executed by pairing it with a filter and defining an agent. When the agent is started, the operation starts to analyze all the packets that are passing through the filter. The analysis can consist of a bar graph, a simple counter, writing information to a file, creating a diagram of some kind or any other object which can be used to analyze network traffic.

4 Using NetSpy as a Teaching Tool

Because of the GUI as well as the opportunity to create operations, NetSpy provides much more flexibility than other network sniffing programs. It has many unique applications that others do not. There are several ways that we have and will be using NetSpy in a classroom setting.

Network analysis and experimentation: Through NetSpy, a user has access to a complete set of network data – all layers in the ISO network model stack. This means that protocol analysis and statistics calculation are possible. We are using the following projects with NetSpy:

- *Counting packets:* As a first module, students are asked to write a class that simply counts packets and categorizes the protocols. Students are surprised as they do the calculations on just how much data goes across a network. Figure 3 holds an example of this type of NetSpy module.
- *Protocol reconstruction:* Students write modules that reconstruct a network session of some sort based on the exchange of protocols. One example is the client/server conversation in Web page requesting. This is done by simply writing lines of client request and server response to the screen. Another example (that intrigues students) is the reconstruction of a Telnet protocol session *in real time* by displaying a "terminal window" and echoing the textual traffic.

Network data sampling: For this exercise, students use a protocol counting module – one that simply writes numbers to a log file – and uses a handheld computer to do *protocol walking*. They sample data from all over campus, noting the different types of network applications in different areas. In their writeups and subsequent discussion, we examine differences and make predictions of network requirements based on type of usage and time of day.

Network administration: There are many network administration tools that claim to detect certain network conditions – like traffic congestion or uninvited intruders. How these tools work is interesting and NetSpy can be used to detect the conditions that these administration tools do. Many of these tools are open-source, which allows students to duplicate some of their functionality in a NetSpy module.

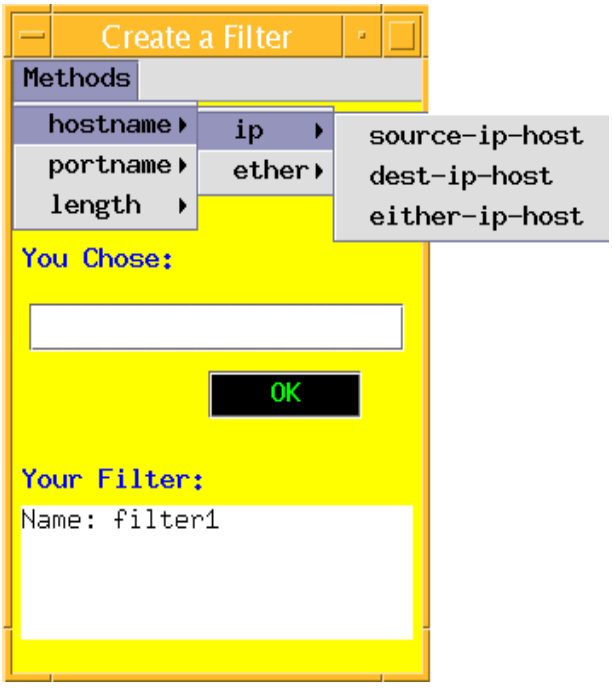

Figure 2. Filter Creation Interface

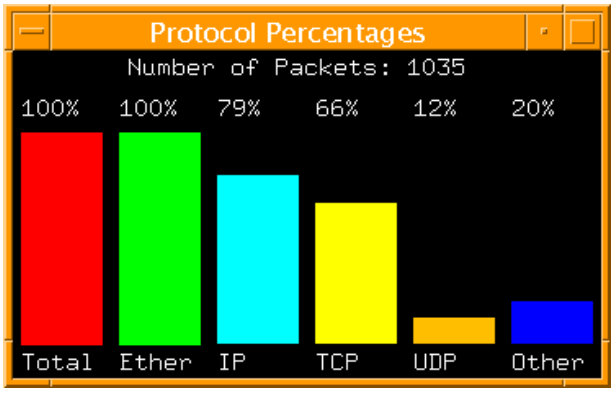

Figure 3: Student Project on Protocol Counting

Application debugging: Many network-based applications are difficult to write because the protocol exchanges are hard to watch and debug. Since filtering is built into NetSpy, modules can be written to track and interpret specific network protocols connected with certain applications.

5 Caveats and Conclusion

NetSpy is an effective tool for allowing students to do protocol analysis and network experimentation. It allows users to write their own analysis tools and protocol collectors and, in doing so, students are able to understand and analyze network data.

We have learned much in our experience with snooping on live network data. There are two caveats that users of NetSpy – and all network snoopers – should be aware of.

- *Students require stern warnings about privacy.* Obviously, privacy issues feature prominently in any activity that involves network snooping. Students tend to focus on the joy of possibly finding some illicit data rather than on protecting a network user's rights. This gives the instructor a great opportunity for discussions on network privacy, both from a snooping perspective and from a more global perspective.
- *Inform the campus Computer Center of your experimentation activities.* In all likelihood, a computer center will have snooping detection programs running. NetSpy projects may even violate campus security policies. It is a good idea to discuss projects with computer center personnel before engaging in them.

For the future, we wish to add wireless protocol objects to NetSpy. Currently, NetSpy focuses on wired connectivity – specifically on Ethernet connections. Since there are so many wireless opportunities with handheld devices, we hope to expand the reach of NetSpy to incorporate 802.11x protocols.

References

[1] Cota, J.T., *Implementicion de un Monitor Analizador Grafico de Reden el Entorno Gnome*, Final Report on the Etherape Project, University of Seville, Spain, July 2001.

[2] *Etherape Software Repository*, available online: http://etherape.sourceforge.net.

[3] *TCPDUMP Public Repository* available online: http://www.tcpdump.org.

[4] *The Ethereal Network Analyzer*, available online: http://www.zing.org.