# PCAP Library Programming

## 1 Overview

The pcap (packet capture) format is a standard and portable representation of packet-level network traffic. The pcap library (http://www.tcpdump.org/) provides many routines1 to interface with both live (off the network card) and stored (previously captured) network traffic. You are already familiar with pcap from packet introspection lab both Wireshark and tcpdump store and read data in pcap format. This lab is designed to familiarize students with the pcap format and library as the basis for performing arbitrarily complex network traffic analysis tasks, especially across large data sets. For example, one might:

- Compute the fraction of web traffic on a link
- Measure the rate of traffic to a particular destination
- Discover anomalous packets
- Find scanning worm traffic
- Etc, etc.

by analyzing packet traces. This lab begins by investigating the pcap format and library/API. We assume basic familiarity with a UNIX environment, including programming tools. As a refresher, a reasonable UNIX tutorial is available at: http://www2.ocean.washington.edu/unix.tutorial.html. Remember, man pages are your friend[1]. Future traffic analysis labs build on the concepts here it is imperative that you understand these basics in order to be successful with subsequent lab work. There are 16 questions in the lab. Submit your written lab report, including your program code, to your instructor, e.g., via a CLE.

This lab includes software development in a UNIX environment. There exist pcap libraries for all major programming languages; developers of this lab actively use C++ and Python, and have included sample tutorial code for each. Many UNIX distributions include the pcap library and headers by default. Python users are recommended to use the dpkt library. dpkt is readily available via the package manager for most distributions of Linux, and is already installed on this Labtainer. There are a number of resources for understanding the dpkt API; it is recommended you start by reviewing the python tutorial code supplied in the home directory of this lab's computer. There also exist a number of online resources for API documentation. [2]

The language for completing this lab is up to you, however it is important that you become comfortable writing libpcap programs and are capable of running UNIX-style scripts and tools. Note that a C++ program may be significanly slower than a Pthon program, depending on what kinds of data structures you employ.

You may find the programs that you write for this lab are useful for future labs (or other exploration). You are encouraged to place your code and scripts in the `mystuff` directory as described below in 2 so that it persists independent of this lab.

### 1.1 Background

The student is expected to have an understanding of the Linux command line, and software development tools, e.g., Python or C++. The lab requires plotting of data, and thus some experience with with the python numpy and matplot packages will be helpful [3].

[1] man pcap

[2] `http://dpkt.readthedocs.org/en/latest/api/api_auto.html#module-dpkt.pcap`

[3] `https://matplotlib.org/tutorials/introductory/pyplot.html`

# 2 Lab Environment

This lab runs in the Labtainer framework, available at http://nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer pcap-lib
```

A link to this lab manual will be displayed.

The home directory of the resulting computer contains a directory named `mystuff`. That directory is shared with your Labtainers host, at:

```
labtainer-student/mystuff
```

Files and directories that you create in `mystuff` will persist independent of this lab (and other labs that make the mystuff directory available). Consider placing your code and scripts there.

# 3 Tasks

This lab includes some detailed instructions to facilitate grading. Failure to follow those instructions may result in your lab results not being properly recorded.

## 3.1 Unknown trace

We will be analyzing an anonymized packet trace taken from an Internet exchange point. This trace is in your home directory in: trace2.pcap. **Do not move this file.**

- Uncompress (this will take a couple minutes, so be patient): $ bzip2 -d ¡file¿
- Try viewing the trace in either tcpdump or Wireshark. Clearly, (if it opens at all) the packet trace is much too large for a human to make sense of it, draw conclusions, or understand the traffic statistics. We will write a program using a pcap library to analyze the trace. You can use either Python or C++ to complete this lab. The appropriate pcap manipulation libraries for both languages are already installed on the VM. Relevant libraries and documentation links:
- C++: #include ¡pcap/pcap.h¿ (http://www.tcpdump.org/manpages/pcap.3pcap.html)
- Python: import dpkt (https://dpkt.readthedocs.io/en/latest/api/api_auto.html#module-dpkt.pcap)

Start by answering the following questions:

1. [5 pts] What link-layer is included in the trace?
2. [5 pts] What is the snap length and what is the significance of the snapshot length? The link type defined in the packet trace header is important as we must skip over the correct amount of data to reach the IP packet (which is what were really interested in). Note that while pcap is the most popular and widely accepted packet capture format, it has several limitations, which have led to development of alternatives. For example, PcapNg, or next-generation pcap, is now the default format in Wireshark.
3. [5 pts] Find the documentation for PcapNg online. Briefly (no more than 2 or 3 sentences) describe the differences between pcap and PcapNg.

### 3.2 Basic traffic stats

Add to your pcap analysis program the ability to loop through all of the packets of the trace. Each packet in a pcap trace is preceded by a header as defined in pcap pkthdr. This header contains a UNIX timestamp, among other fields. To iterate through all packets of the trace, you may wish to use, in C, pcap loop() or pcap dispatch() with the appropriate callback (you must create the callback). With python's dpkt, `pcap.Reader` provides a similar iterator. Note you may wish to place a bound on the number of packets processed while you are developing your program – but do not forget to remove this bound to complete the assignment.

When answering the following questions, your program **must** output the desired value to stdout with the specified prefix (when present). For example, if the qustion is:

1. [5 pts] How many IPv4 packets does the trace contain (as IPv4 count:)?

then, your program should output a line containing:

```
Count IPv4:  343
```

or whatever the value is. Unless otherwise directed, output values as an integer with no formatting, e.g., no thousands commas. The order in which your program outputs the answers does not matter. Nor does it matter if different programs output results of different questions.

1. [5 pts] How many IPv4 packets does the trace contain (as IPv4 count:)?
2. [5 pts] How many non-IPv4 packets does the trace contain (as non-IPv4 count:)?
3. [5 pts] What is the timestamp of the first packet in the trace, including at least two decimal places. (as First timestamp:)?
4. [5 pts] What is the average packet rate (in packets per second to two decimal places) of the trace (as Avg packet rate:)?

After the per-packet pcap header is the traffic data. Improve your callback to decode IP packets. Your callback should obtain the source and destination IP addresses, the protocol (e.g. TCP, UDP, ICMP, etc), source and destination transport port (where applicable), etc. Several questions ask for an answer in the form of a distribution. Please note that a distribution must represent each value in the form of a fraction or percentage of the total. Answer the following questions:

5. [10 pts] What is the packet protocol distribution? (A table showing the 5 top protocols and their respective contributions is fine.)
6. [10 pts] Plot a histogram of the packet size distribution (the Python numpy and matplot packages are installed on the Labtainer).
7. [5 pts] How many unique IPv4 source addresses are present in the trace (as Unique sources:)?
8. [5 pts] How many unique IPv4 destination addresses are present in the trace (as Unique destinations:)?
9. [10 pts] Create a cumulative distribution function (CDF) plot. The x-axis is the number of bytes sent and the y-axis is the cumulative fraction of sources.
10. [5 pts] Which source sent the most bytes (as Source with most bytes:)?
11. [5 pts] Which source sent the most packets (as Source with most packets:)? Based on your analysis of the trace:

12. [10 pts] List 3 characteristics of the traffic that seem unusual to you.

13. [5 pts] Provide a reasonable explanation for what traffic the trace represents, taking into account the unusual characteristics you have identified.

# 4 Submission

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stoplab
```

When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.

This lab was originally created by faculty at the Naval Postgraduate School Department of Computer Science. It was adapted for the Labtainer framework by the NPS Center for Cybersecurity and Cyber Operations under sponsorship from the DoD CySP program. 