Semester Thesis:

A Lightweight Packet Capturer for High-Speed Links
Gerry Zaugg <zauggge@student.ethz.ch>

Motivation

Distributed Denial of Service (DDoS) attacks are a threat to Internet services ever since the widely published attacks on ebay.com and amazon.com in 2000. ETH itself was the target of such an attack 6 months before these commercial sites were hit. ETH suffered repeated complete loss of Internet connectivity ranging from minutes to hours in duration. Massively distributed DoS attacks have the potential to cause major disruption of Internet functionality up to severely decreasing backbone availability.

In order to better understand traffic patterns in the rapidly growing Internet and consistently identify malicious traffic (like the one belonging to Slammer or the more recent Blaster and Sobig.F), there is an urgent need to design efficient network monitoring tools that can capture network traffic in a lightweight manner. Current packet capturing mechanisms, like tcpdump and ethereal are inefficient when working on high-speed links (like gigabit ethernet). They cannot keep up with the link speed and often end up dropping hundreds of packets, which results in loss of useful traffic information.

A consistent capturing mechanism of packet headers will provide the building blocks in helping network administrators to accurately capture and later classify data flowing through their network and make useful decisions like allowing or preventing certain kind of traffic.

Semester Thesis Task

The goal of this project is to design and develop a kernel daemon or program that is able to capture the first 40-60 bytes from every arriving packet on a gigabit interface so as to include TCP/IPv4 headers (without worrying about the extensive use of options fields on the IPv4 header). The collected data is stored in files for further analysis.

The daemon / program is for passive measurement only. We will not be attempting to alter any fields in the header of the packet.
Overall, the project is divided into three major tasks - design, implementation and testing. They all have several subtasks.

**Design**

**Daemon or program**

There are two ways to approach this problem - (1) Either write a program that does what is intended as the goal of this project or (2) create a daemon that does the tasks specified. The user should be able to issue commands to the daemon on a command line interface to start and stop the capture etc.

A final decision may be made by the student after carefully evaluating the trade offs of both techniques and discussing the same with the supervisors.

**Understanding the data capture mechanism**

Today, the *libpcap* library is widely used to snoop on an interface to capture raw traffic data. Additionally, libpcap provides statistical information that can be gotten using function interfaces. The same library should by used to collect the data and to do further processing on it. Open-source examples like *ethereal* can give an understanding on how the data is captured.

**Concepts to keep in mind**

- **Context Switches.** Copying headers from kernel to user level always involve a context switch into kernel mode and a copy of memory from the kernel to the user space. This "call-and-copy" approach is repeated for every packet, which becomes computationally expensive. The number of context switches and the number of copies have to be minimized.

- **Interrupt Coalescing.** Every captured packet leads to a hardware interrupt. Reducing these events by combining several packets as part of a single interrupt would yield an increase in efficiency.

- **Buffering.** Ensure that the headers are buffered correctly and efficiently when multiple packets arrive per interrupt.

It will be key to have a good design before moving to the implementation phase.

**Implementation**

In this thesis at least the following components of the framework will be developed:

- **Capture.** The capturing mechanism must be lightweight to avoid packet drops.
• **Command Line Interface.** Ability to *dynamically* say when to start and stop the data capture from user space and to set the number of bytes per packet to be captured. For TCP/IPv4 packets, it translates to at most 40 bytes of header from each packet excluding MAC header (54 bytes including MAC headers). This should be sufficient to capture most packet types.

IPv6 provides a number of functional enhancements over the existing IPv4. A packet includes the IPv6 header (fixed length of 40 bytes) and zero or more extension headers. The daemon / program must be able to handle these variable headers by dynamically extending the capture range.

• **Storage.** Write the data to files. The files must be compressed with bz2 or an equivalent program. Captured files do not exceed a certain size and capture has to be done in a ring-buffer mode. This means that once a certain number of files have been created, it should overwrite the file that was created the earliest. In case of an ongoing attack, one may not want a fixed number of files and continue capturing data by stopping the ring-buffer mode.

• **Analysis.** The user must be able to run the `tcpdump` utility on the captured files.

• **Statistics Component.** Keep a statistics file once a data file is written. This will allow the user to note if an unusual activity (like packet drops) occurred.

• **Performance Tuning.** All data structures in the code are cache aligned.

• **CPU Usage.** The daemon / program should not add significant overhead to CPU utilization of the system when the packet capture is turned on.

The capturing software is distributed under terms of **GNU GPL**.

**Testing**

The code should be tested using a gigabit interface running small UDP / TCP packets using `netperf` to ensure that all goals of the project are met. The results are compared against `ethereal` and `tcpdump` to show benefit.

We will run long time tests (at least 24 hours of duration) using synthetic data to measure the performance in a realistic environment.

The hardware requirements to perform the tests will be provided at an appropriate time.

**Documentation and Presentation**

A document that lists the steps taken in design, implementation, results and key lessons learnt along with future work must be written. The code should contain sufficient documentation so that a new student (if extending the work) can start working in a reasonable period of time.

At the end of the semester, the student is expected to make a short presentation which highlights the tasks accomplished and the results gathered.
Dates

The semester thesis will start on Oct 29th, 2003 and will be finished by February 20th, 2004.

Supervisors

Parminder Chhabra, chhabrap@tik.ee.ethz.ch, 001 408 394 3318, Rutgers Uni, NJ, USA
Thomas Dübendorfer, duebendorfer@tik.ee.ethz.ch, +1 632 7196, ETH Zürich, ETZ G64.1
Arno Wagner, wagner@tik.ee.ethz.ch, +1 632 7004, ETH Zürich, ETZ G64.1