Converting a Sony Playstation 2 into a Network Analyzer

Mayer Crystal
mc2087@columbia.edu

Abstract

This paper describes my experiences in converting a Sony Playstation 2 into a network analysis tool. The paper is divided into three logical sections which examine the various aspects involved in developing such a tool. The first two sections are concerned with the practical underpinnings of network analysis and the design and architecture of the Sony Playstation 2 (PS2) platform respectively. The third section is concerned with the specific challenges that were encountered while porting the network analysis application to the PS2.

1 Introduction

With the rapid expansion in recent years of internetworked applications, most notably the World Wide Web (WWW), individuals and companies have had to change the nature of their exposure to the outside world. Along with the move from a physical to a virtual marketplace, an additional technological need has arisen: to insure that the new virtual marketplace is secure, reliable, and responsive to client requests. In order to provide such services, companies have put in place measures to pre-emptively detect malicious behavior (be it from a person or a faulty piece of hardware equipment) and to protect their online assets with firewalls, cryptography, and other proprietary measures.

The focus of this paper is the pre-emptive detection of malicious users and faulty equipment related to the networking aspect of the system. Currently, there are two methods which are used to do this type of network monitoring. The first method is having a typical machine such as a UNIX workstation act as a monitor for other machines. Such an approach is described by Mogul [5]. An alternative method involves the use of a dedicated piece of hardware to perform the network monitoring. It is this second alternative that I will be exploring in porting the network monitoring application to the PS2.

There have been many works that deal with the analysis of TCP or network traffic in general [6, 5]. This paper presents a high level overview of such works in order to lay the theoretical foundation for the proposed application. In contrast to the cited works, however, this paper will also examine the feasibility of converting a relatively inexpensive embedded system - the PS2 - into a dedicated network analyzer.

By porting the network analysis software to an embedded system such as the PS2, many issues arise and must be dealt with. First and foremost, computational resources such as available memory are extremely limited, as they are with all embedded systems. Additionally, when porting to an embedded platform, it is necessary to understand the capabilities and limitations of the platform.

This paper presents the experiences of the author porting a publicly available network analysis tool to the PS2. Section 2 presents an overview of the current techniques and technologies used for network analysis. Section 3 presents an overview of the PS2 architecture, and Section 4 describes the goals for the project.

2 Network Analysis

The idea behind network analysis is relatively simple: listen to all the traffic traversing the network, categorize the traffic, and generate reports based on the observed patterns. The implementation of this idea, however, is significantly more complex. There are many issues which could cause the monitoring application to miss packets and hence misrepresent data. For instance, if the throughput of the network is greater than the throughput of the monitoring device, it is possible that packets will be dropped. Furthermore, aside from listening to the traffic, there is a significant amount of work expended in the categorization and processing of the packet itself which can cause additional delays and/or buffer overflows. It must also be noted that in certain conditions it is possible that packets will be counted more than once when they shouldn't be, such as when a packet is retransmitted due to checksum errors. Filtering out this data is time consuming and not necessarily the appropiate action, if, for instance, one is trying to determine network health based upon the number of packets retransmitted. In order to overcome these obstacles, a number of architectural designs have been put forward. One approach that has been suggested is to use a layered architectural approach in conjunction with protocol and/or packet filtering.

2.1 Layered Architecture and Filtering

In order to remove any extraneous processing from the execution path, a common theme that has been introduced into many network analysis systems is the use of a layered architecture. The architecture generally consists of the input/link layer, the protocol filtering layer, and the actual data processing layer. The idea is that if we are not interested in certain protocols (e.g., we only wish to examine IP traffic) then we should not waste any time with the processing of packets being used by other protocols and should instead discard them as closely to the link layer as a possible. This way, the monitor is free to examine a new packet without having had to go through the actual packet analysis. One such architecture is described by Ranum, et al in [6].

2.2 *Network Flow Table Maintenance*

Protocol filtering not withstanding, it is still desirable to minimize the amount of processing time and, if possible, memory used by the monitoring application for packet processing. Since the monitor maintains a network flow in some data structure, it is desirable to minimize the amount of time spent locating the appropiate memory location for the given packet. In many works, as well as in publicly available tools, the authors have elected to use a hashtable to store the flow information [6, 5] . This allows for a fast table lookup via a hash function which uniquely identies the connection being monitored. Once this lookup is made, the processing of the received packet can be limited to a table entry update or addition.

Aside from adding information to the flow table, it is also necessary to purge data from the table. When a connection is closed, for instance, the entries related to that connection should be removed from the flow table. In normal, error free operation, the connection termination can be detected via the semantics of the protocols being used (i.e. the sending of FIN packets in the TCP protocol). In less than ideal conditions, however, additional measures such as inactivity timeouts must also be used to properly purge the flow table. The configuration of the inactivity timer must be carefully implemented, because a misconfiguration can cause a connection that is active but dormant to be purged from the flow table incorrectly.

2.3 *External Report Generation*

Although not integral to the monitoring system itself, it is important that the data collected be usable by network administrators. This requires some form of report generation over the collected data set. For performance reasons, it is usually not feasible to have this report generation run in the same process as the data collection process. Instead, many application developers have opted to run the report generation as an external task which can run on a distinct piece of hardware [6, 1].

2.4 *Existing Tools*

As the need for network monitoring has increased so too has the availability of technological tools to monitor the network. In this paper I have limited the scope of my discussion to software utilities which are open source and publicly available. Of those, perhaps the most famous and widely used are *libpcap* and *tcpdump*.[1] Although they are not directly useful for network analysis in and of themselves, they do provide a method for abstracting the low level network interface and presenting a unified view of network data to the user.

For more advanced network analysis, tools such as Argus [1] and Etherape[2] have built more complex monitoring structures on top of the libpcap interface. For the purposes of this paper, I have chosen to port the Argus software to the PS2 since the source code is structured in a clear manner and because Argus defines a clear separation between the data gathering/analysis and the reporting. This allows me to introduce my own front end to do the reporting.

[1]http://www.tcpdump.org

[2]http://etherape.sourceforge.net

3 The Playstation 2 Platform

The PS2 platform is an embedded platform specifically designed for multimedia applications. As described by R. Dubey, the architectural design for such a multimedia platform is radically different from the architectural design of general purpose processors (GPPs) [3]. Before porting an application to the PS2, it is important to understand the design benefits and limitations of such a platform.

3.1 *Differences Between GPPs and the PS2*

As described by R. Dubey and Jon Stokes, the goal of a multimedia system such as the PS2 is to process large amounts of constantly changing data with a relatively static set of instructions [3, 8]. GPP-based systems, on the other hand, are designed to handle relatively static data sets with a wide range of instructions. For example, GPP-based systems can easily be used to develop word processing applications which act on a small set of data (the document) but can perform many operations on that data (layout manipulation, spell check, etc...). Multimedia systems, by contrast, are designed to handle audio and video data streams which are constantly changing from one frame to the next, but commonly apply only a small set of operations to the data (rotation, zoom, etc...).

Based on the different usage patterns, it is no surprise that the internal processor designs of the two systems are dramatically different. Whereas the GPP-based systems make use of large data caches, the PS2 does not. Since the data is constantly changing for multimedia systems, the cache miss percentage would be too high to justify the inclusion of a data cache in the core processor. Additionally, the various data segments processed by the PS2 are normally independent of one another and can thus be manipulated in parallel. Because of this, the PS2 allows its vector units to be operated in a variety of modes allowing up to 16 8-bit data segments to be processed in parallel.

3.2 *The Emotion Engine*

The core of the PS2 platform is the Emotion Engine (EE). As illustrated in Figure 1, the EE consists of a MIPS III CPU core, an FPU, two vector units (VU0 and VU1), an image processing unit (IPU), a 10-channel DMA controller, a graphics interface unit (GIF), an RDRAM interface unit, and an I/O interface unit, all integrated on a single die. Furthermore, to increase the data throughput of the device, all of the components are connected via a shared 128-bit bus (excluding the FPU which is connected directly to the MIPS III core via a 128-bit bus).

In terms of function, the core components are divided into three groups. The first, which consists of the MIPS III CPU, the FPU, and the VU0 unit, are normally used for the logical processing of the data. The VU1 unit is used to do the intense vector graphics manipulation, and the IPU is used for image decompression. Since all the components share a 128-bit bus, it is also possible to use the various components in other configurations which allow for more parallelism. For instance, the MIPS CPU and the VU0 could be manipulating a multimedia image at the same time that the VU1 is working on a differ-

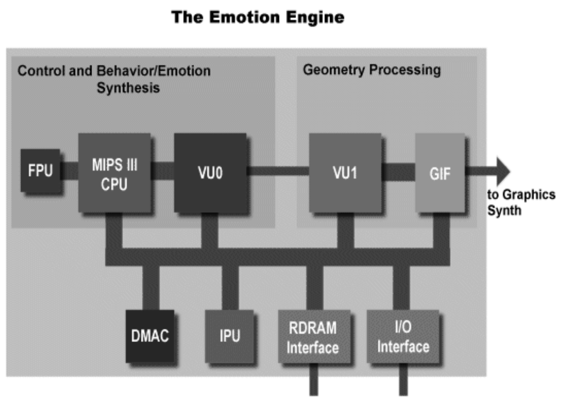

Figure 1: Emotion Engine Architecture

ent image. Alternatively, the two vector units can be 'locked together' to do 128-bit SIMD in the following configurations: 16 8-bit ops/cycle, 8 1-bit ops/cycle, or 4 32-bit ops/cycle [9].

In addition to the multimedia specific components, the MIPS III CPU component can perform most common risk operations including:

- MUL/DIV
- 3-op MUL/MADD
- Arithmetic ADD/SUB
- Pack and Extend
- Min/Max
- Absolute
- Shift
- Logical instructions
- Compare instructions
- Quadword Load/Store

The MIPS III CPU also includes a 16K instruction cache and a smaller 8K data cache and contains a 6 stage pipeline (select, fetch, decode and register read, execute, cache access, and writeback). For more efficient I/O, the CPU also contains 16K of 'scratchpad RAM (SPRAM)' which is accessible to the CPU for load instructions but is also directly accessible by VU0 and is also used as a staging area for placing data on the 128-bit shared bus.

4 Porting to the PS2

In order to convert the PS2 into a network analyzer I decided to port the Argus suite of applications to the PS2. However in order to keep the system as close to a generic PS2 as possible and maintain the advantages of standard embedded systems such as minimal user interaction, fast startup and automatic recovery, as described by Seltzer [7], I have opted to port the application in a manner that would allow it to run using a single PS2 memory card without requiring additional storage such as a hard drive. In order to accomplish this port, the following software and hardware pre-requisites were obtained:

- PS2 Network Adapter
- A Standard 8 Megabyte Memory Card
- The Linux port to the PS2
- Argus 2.0.5 Source Code

4.1 Obstacles to Porting

Although it should not have been difficult to port the desired software to the PS2 platform, there were many obstacles that were encountered along the way. The first and most daunting obstacle was that of the limited memory. Although the PS2 has 32 MB of RAM, both the OS kernel as well as the application and its libraries must be stored on an 8MB memory card. Because all applications are linked against the standard libc library, this library must be included. However, upon examination of the shared library, it becomes obvious that we have hit an apparent limitation; the libc library itself is approximately 8MB. Dealing with the size of the standard libc library, however, is not without a solution. In fact, because the size of the library makes it difficult to use in embedded systems, other alternatives have been developed. As described by Todd E. Sundsted, there are two common alternatives: uClibc and Dietlibc [10].

Although, the use of uClibc would alleviate the size problem, in the end I did not use it due to linker errors that were encountered while building the uCilbc development applications. It should be noted however, that for general development as described by Sundsted, uClibc is frequently used. Furthermore, there seem to have been successful ports of the uClibc environment to the PS2 [2]. Instead of uClibc, I took another approach similar to the one described by Patryk Laurent [4]. I created a 16MB disk image, copied glibc into that image and then compressed that image so that it would fit within the limits imposed by the available media.

In addition to the required libc library, I also had to obtain a port of Argus and its required libraries. In the end, the requirements could be refined to the following components:

- libpcap
- bison[3]
- Argus

Compiling bison went smoothly. Compiling libpcap, however, proved a bit more challenging. The headers contained within the libpcap distribution were expecting certain kernel functions to be available. The kernel released by Sony for the PS2, however, did not contain the desired functions. This issue,

[3]Available at http://www.gnu.org/software/bison/bison.html

however, was circumvented by the use of the Xrhino kernel [4] which represents a community-supported effort to upgrade the PS2 Linux kernel to a more recent version. Upon compilation and successful installation of the latest kernel, I was able to get the libpcap library to compile. Once these libraries were compiled, the compilation of the Argus suite progressed without any issues.

The final issue to overcome for the port of the network analysis tool to the PS2, was the flawed implementation of the network driver for the PS2 Network Adapter. However, Sony has released a beta version of there smap driver (the driver which controls the PS2 network adapter) which has so far proven to be relatively stable.

5 PS2 Network Analyzer Runtime

After compiling the required programs, a 16MB RAM image was created using the following commands:

```
dd if=/dev/zero of=na.img bs=1k count=16384
mke2fs -i 16384 -b 1024 -m 5 -F -v na.img
```

This image was then mounted temporarily on the file system, and the following directory structure was created:

```
drwxr-xr-x    2 root     root         1024 Dec 14 22:54 bin
drwxr-xr-x    2 root     root         1024 Dec 14 22:49 dev
drwxr-xr-x    3 root     root         1024 Dec 15 10:31 etc
-rw-r--r--    1 root     root          218 Dec 15 12:18 flow.conf
drwxr-xr-x    3 root     root         1024 Dec 15 09:16 lib
-rwxr-xr-x    1 root     root           59 Dec 14 23:24 linuxrc
drwxr-xr-x    2 root     root        12288 Dec 14 22:48 lost+found
-rwxr-xr-x    1 root     root           89 Dec 15 12:18 na.sh
drwxr-xr-x    2 root     root         1024 Dec 15 09:37 sbin
drwxr-xr-x    5 root     root         1024 Dec 14 22:51 usr

bin/
-rwxr-xr-x    1 root     root       113304 Dec 14 22:50 ash
-rwxr-xr-x    1 root     root        78088 Dec 14 22:54 ls
lrwxrwxrwx    1 root     root            3 Dec 14 22:50 sh -> ash

dev/
(all standard dev devices created by MAKEDEV std and MAKEDEV console)

etc/
drwxr-xr-x    2 root     root         1024 Dec 15 09:35 init.d
-rw-r--r--    1 root     root           91 Dec 15 10:31 inittab
-rw-r--r--    1 root     root           16 Dec 15 09:34 modules.conf
-rwxr-xr-x    1 root     root          300 Dec 15 09:33 rc.modules

/etc/init.d/
-rwxr-xr-x    1 root     root          233 Dec 15 09:35 rcS

lib/
-rwxr-xr-x    1 root     root       607432 Dec 14 22:56 ld-2.2.2.so
lrwxrwxrwx    1 root     root           11 Dec 14 22:56 ld.so.1 -> ld-2.2.2.so
-rwxr-xr-x    1 root     root      7860856 Dec 14 22:54 libc-2.2.2.so
lrwxrwxrwx    1 root     root           13 Dec 14 22:55 libc.so.6 -> libc-2.2.2.so
-rwxr-xr-x    1 root     root      1192451 Dec 14 22:54 libm-2.2.2.so
lrwxrwxrwx    1 root     root           13 Dec 14 22:55 libm.so.6 -> libm-2.2.2.so
-rwxr-xr-x    1 root     root       983551 Dec 14 23:08 libpthread.so.0
-rwxr-xr-x    1 root     root       360644 Dec 14 23:09 librt.so.1
-rwxr-xr-x    1 root     root        16156 Dec 14 23:08 libtermcap.so.2
drwxr-xr-x    3 root     root         1024 Dec 15 09:16 modules

lib/modules/2.2.21-pre1-xr7/misc/
-rw-r--r--    1 root     root        43388 Dec 15 09:17 smap.o

sbin/
-rwxr-xr-x    1 root     root        72252 Dec 15 09:18 ifconfig
-rwxr-xr-x    1 root     root         1285 Dec 15 09:18 ifdown
-rwxr-xr-x    1 root     root         3331 Dec 15 09:18 ifup
-rwxr-xr-x    1 root     root        44292 Dec 14 22:57 init
-rwxr-xr-x    1 root     root       133772 Dec 15 09:36 insmod

usr/
drwxr-xr-x    2 root     root         1024 Dec 15 10:18 bin
drwxr-xr-x    2 root     root         1024 Dec 14 22:51 lib
drwxr-xr-x    2 root     root         1024 Dec 14 22:51 sbin

usr/bin/
-rwxr-xr-x    1 root     root       114248 Dec 14 22:52 bison
-rwxr-xr-x    1 root     root         5524 Dec 15 10:18 clear
-rwxr-xr-x    1 root     root       440230 Dec 14 22:52 ra
-rwxr-xr-x    1 root     root       435596 Dec 14 22:52 racount
-rwxr-xr-x    1 root     root       508494 Dec 14 22:52 ragator
-rwxr-xr-x    1 root     root       497114 Dec 14 22:52 ramon
-rwxr-xr-x    1 root     root       480928 Dec 14 22:52 rapath
-rwxr-xr-x    1 root     root       441452 Dec 14 22:52 rasort
-rwxr-xr-x    1 root     root       445298 Dec 14 22:52 raxml

usr/lib/
-rw-r--r--    1 root     root       184912 Dec 14 22:51 argus_common.a
-rw-r--r--    1 root     root       137704 Dec 14 22:51 argus_parse.a
-rw-r--r--    1 root     root       224594 Dec 14 22:51 libpcap.a

usr/sbin/
-rwxr-xr-x    1 root     root       676874 Dec 14 22:51 argus
```

As noted above, this image was then compressed into the na_glibc.img (for a total of 4,428,284 bytes). This was copied to the memory card in addition to the Xrhino kernel (which was also compressed to 2,678,535 bytes).

6 Additional Setup

In addition to the port and the compression onto the 8MB memory card, there were a few additional steps that had to be taken in order to have the system boot properly. The first requirement was that the Linux CD be placed in the CD drive. Although it would have been preferable to be able to boot directly off of the memory card (and hence require no additional media), the PS2 bootloader does not allow this. The CD must be in the drive in order to bootstrap the load process. The second additional requirement was the proper setup for the bootup and networking. In order to properly initialize the system, the following init scripts were added:

1. inittab:

   ```
   id:1:initdefault:
   si::sysinit:/etc/init.d/rcS
   1::sysinit:/usr/bin/clear
   2::sysinit:/na.sh
   ```

2. modules.conf

   ```
   alias eth0 smap
   ```

3. rc.modules:

   ```
   #!/bin/sh

   # Preload eth0 driver, if eth0 is not present.

   grep '[ \t]*eth0:' /proc/net/dev \
   2>/dev/null >/dev/null
   eth0_found=$?
   if [  $eth0_found -ne 0 ]; then
     echo "Preload eth0 driver"
     /sbin/ifconfig eth0 2> /dev/null > /dev/null
   fi
   ```

4. rcS:

   ```
   #!/bin/sh
   #

   export PATH=/sbin:/usr/sbin:/bin:/usr/bin

   # mount proc file system
   mount -t proc proc /proc

   # load PS2 modules
   /etc/rc.modules
   ```

[4] Available at http://playstation2-linux.com/projects/xrhino-kernel/

```
insmod smap

ifconfig eth0 192.168.1.92 \
  broadcast 192.168.1.255 \
  netmask 255.255.255.0 up
```

5. na.sh:

```
#!/bin/sh
/usr/bin/clear
/usr/sbin/argus -S 4 -w - | \
/usr/bin/ragator -f /flow.conf -r -
```

These scripts were added to properly initialize the networking and to start the system into the network monitoring mode immediately. This added a slight measure of security by not allowing any other commands to be executed when the PS2 is run as a network analyzer, thus making it harder to compromise the network monitoring server.

7 Results

After setting up the scripts and installing the image, the system was booted and began to monitor the network. For this test, the machine was put on a small private network which was running a few applications (web browser, gnutella, email, and other common web based applications). The resulting output is shown in Figure 2.

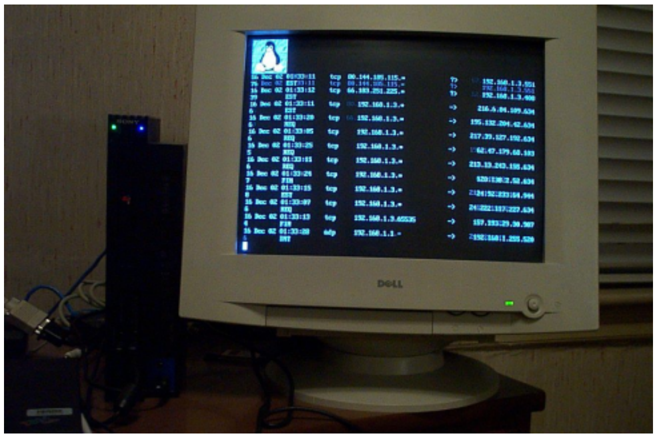

Figure 2: PS2 Network Analyzer in Action

8 Future Work and Considerations

This project can be considered a proof of concept. There are many other ways in which the PS2 Network Analyzer can be made more complete. Some of the areas which may be of interest are: minimizing the amount of memory used by the kernel and the runtime. When the kernel was compiled, it was mainly a stock kernel with some minor patches. There are many areas in which the kernel could be trimmed down. Furthermore, as mentioned above, the use of uClibc instead of glibc would greatly improve the memory footprint. In addition to the memory enhancements, there is also room for user interface/presentation enhancements. At the time of this writing, Sony had just recently released a library to allow PS2 developers access to DMA channels, the scratchpad memory, and the GIF. By programming to this library, a very sophisticated user interface could be created. Lastly, since the Playstation 2 is running a full Linux kernel, it is possible that the PS2 could be a security risk to the network which it is monitoring. This can be contained by severing the transmit wires in the RJ45 cable (as is done in many security conscious institutions), and can also be accomplished by tweaking and securing the kernel.

References

[1] Argus. Documentation at http://www.qosient.com/argus/.

[2] Nohdd (ucilbc) ps2 bootable memory card instructions: (revision 2). Documentation at http://playstation2-linux.com/download/nohdd/uclibnohdd.html.

[3] K. Diefendorff and R. Dubey. How multimedia workloads will change processor design. *IEEE Computer*, 30(9):43–45, September 1997.

[4] Patryk Laurent. Ps2 nohdd howto. Documentation at http://playstation2-linux.com/download/nohdd/ps2nohdd.html.

[5] J. Mogul. Observing TCP dynamics in real networks. *Proc. SIGCOMM ’92 Symposium on Communications Architectures and Protocols*, pages 305–317, 1992.

[6] Marcus J. Ranum, Kent Landfield, Mike Stolarchuk, Mark Sienkiewicz, Andrew Lambeth, and Eric Wall. Implementing A generalized tool for network monitoring. In *Proceedings of the Eleventh Systems Administration Conference (LISA ’97)*, San Diego, CA, 1997.

[7] Margo I. Seltzer and Michael A. Olson. Challenges in embedded database system administration. pages 103–110.

[8] Jon Stokes. The Playstation 2 vs. the PC: a system level comparison of two 3D platforms. Documentation at http://www.arstechnia.com/cpu/2q00/ps2/ps2vcpc-1.html, 2000.

[9] Jon Stokes. Sound and Vision: A technical overview of the emotion engine. Documentation at http://www.arstechnia.com/reviews/1q00/playstation2/ee-1.html, 2000.

[10] Todd E. Sunsted. Lightweight linux, part 1. Documentation at http://www-106.ibm.xom/developerworks/linux/library/l-lwl1/?t=gr,lnxw12=LLP1.