Comparing Network Traffic Probes based on Commodity Hardware

Luis Zabala(1)(2), Alberto Pineda(1), Armando Ferro(1), Daniel Fernández(2)
(1)Department of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao, Spain
(2)Stochastic and Operations Research – Networks (NET), Basque Center for Applied Mathematics (BCAM), Bilbao, Spain

Emails: {luis.zabala, alberto.pineda, armando.ferro}@ehu.es
Emails: {lzabala, dfernandez}@bcamath.org

***Abstract*—Due to the fact that, nowadays, it is possible to capture traffic in 1-10 Gigabit Ethernet networks using commodity hardware, many traffic monitoring systems, and especially capturing tools, have been proposed in recent years. This paper presents a comparison between two software probes named Adviser and Ksensor. Both of them are multi-processor systems and are built over conventional hardware. However, while Adviser is designed in user space, Ksensor runs in kernel space. This work compares the performance results of the two probes considering several capture engines (NAPI, PF_RING with DNA, PFQ) and, at the same time, different application or analysis loads. The evaluations of the probes with the different settings have been performed on the same hardware multi-core configuration. The results of the evaluations let conclude which solution is better in each situation and which solution must be discarded.**

Keywords-packet capturing; Ksensor; Adviser; NAPI, PF_RING; PFQ

I. Introduction

Nowadays, network traffic capturing and analyzing systems are becoming increasingly relevant. Different applications can be related to these traffic monitoring systems, for example, network antiviruses, Quality of Service monitoring, intrusion detection systems, traffic classification and balancing. They can also help administrators in network troubleshooting. As the speed of the network links increases, the performance requirements on the monitoring systems are more severe. In particular, in multi-Gigabit environments, overload situations can happen, reaching the system limitations in terms of memory occupation, CPU usage, system bus throughput, and having negative impact on the accuracy of the monitoring application. Therefore, as far as possible, unnecessary consumption of available resources must be avoided.

The packet capturing stage is an essential component of the traffic monitoring system. The evolution of commodity hardware has made possible the capture of network traffic to be a feasible task over high-speed networks, without using any neither specific nor expensive hardware [1]. This way, several research works [2][3][4] have arisen focused on the development of analysis systems that are able to process all the information carried by actual networks. Among them, our research group of the UPV/EHU, called Network, Quality and Security (NQaS), is working on software solution proposals for traffic analysis systems over multi-processor architectures.

Two traffic probes have been developed by NQaS. They are generic and flexible, and they allow doing any type of analysis on the captured traffic. Due to the fact that the monitoring application is over a multi-processor platform, the analysis can be done concurrently, obtaining a high performance. This paper presents a comparison between those two probes which have different view of design. On the one hand, Adviser is a generic multi-processor architecture which has been built in user space, it is portable and it can make use of different capturing systems. On the other hand, Ksensor is a kernel-space framework in which the processing modules have been migrated from user-level to the kernel of the operating system.

Many capturing tools and comparisons have made available in the literature. However, most of them do not asses how the packet capture is affected under different analysis or application loads. This work compares the performance results of the probes Adviser and Ksensor considering different capture engines and, at the same time, different analysis loads as will be seen below.

The rest of the paper is organized as follows. In Section II, a brief explanation about related work is introduced. In Section III, we describe the network traffic probes that will be compared later. Section IV presents the test setup for evaluating the performance of the traffic analysis systems. Section V shows the results of our measurements. Finally, Section VI remarks the conclusions.

II. Related Work

The improvement of packet capturing capabilities with commodity hardware has been an extensively covered research topic. Hardware and software solutions have been proposed.

Among the most recent software solutions, it is remarkable Luca Deri's numerous contributions within the project ntop [2]. In this project, an open source platform has been developed to monitor traffic in high speed networks and it has given rise to interesting works such as [5], which presents the network socket PF_RING, [6], in which nCap, a proposal based on commercial network cards was proposed, and [7], which deals with aspects related to the packet parallel processing in multi-core platforms, as well as with the driver called Threaded New API (TNAPI), which uses a multiqueue structure. The same research group has also presented the framework vPFRING for capturing packets on virtual machines running on commodity hardware [8].

The project Ringmap [3] has certain similarities with ntop, since it also proposes to improve the performance of packet capture removing some packet copy operations and mapping the Direct Memory Access (DMA) buffer into the user space. Ringmap works with FreeBSD operating system, while ntop works with Linux. Another approach proposed to speed up the packet capturing capability is Netmap [4]. This is a BSD based project which integrates in the same interface a number of modified drivers mapping the NIC transmit and receive buffers directly into user space. [9] proposes a packet capturing engine with multi-core commodity hardware named PFQ, which allows parallel packet capturing in the kernel and, at the same time, to split and balance the captured packets across a user-defined set of capturing sockets. Even, there have been various works [1][10][11] in recent years looking at evaluating existing packet capture techniques. In particular, [11] evaluates and compares different capture solutions for Linux and FreeBSD operating systems. The evaluation shows that FreeBSD outperforms standard Linux PF_PACKET, Linux with PF_RING performs better than PF_PACKET and even better than FreeBSD if multiple capturing processes are run on the system. Another option analyzed is TNAPI, which achieves the best performance when it is combined with PF_RING.

III. Description of the Software Probes

As mentioned before, the performance of two software probes will be compared in this paper. The first one, named Adviser, is a user-level traffic probe, i.e., it has got the common structure with the analysis or monitoring application in user space and the capturing stage in kernel space. Adviser admits different configurations for the capturing as will be explained later. The second probe is called Ksensor and it is an entirely kernel-level probe. Both of them capture and analyze traffic in Gigabit Ethernet networks.

A. *Adviser. The User-Level Framework*

Adviser [12] is a multi-processor architecture able to capture network traffic and analyze it applying online complex algorithm. Since the architecture is built on top of the operating system, it is portable to several systems. Fig. 1 shows the block diagram of Adviser framework. It works essentially as follows.

First, the system parser interprets the configuration files and stores system logic in memory. Then, analysis engine processes captured packet from the network according to the logic stored in memory. After applying the rules, the engine stores the results of the analysis. Finally, offline processing module takes these results from memory and handles this information to provide traffic statistics or reports.

There is also a module called periodic action manager, which supports dynamic activation or deactivation of rules, modification of period time, etc.

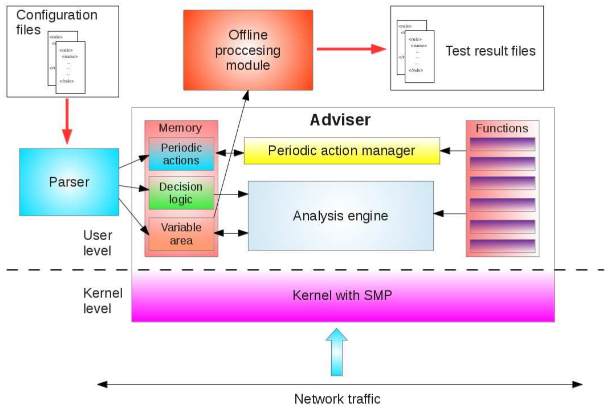

Figure 1. Adviser framework.

The traffic capturing system is in the kernel with Symmetric Multi-Processing (SMP). As Adviser can be configured with different capturing systems, in order to obtain Adviser's performance results with different configurations, we have integrated three capturing systems into Adviser, as follows.

1) *Adviser's capturing system with NAPI and LibPcap*

This first setting uses the network subsystem of standard GNU/Linux. It is New API (NAPI) [13] from kernel versions higher than 2.4. The link between the Linux networking subsystem and the user-space application Adviser is established by using the library LibPcap [14]. As can be observed in Fig. 2a, the application Adviser reads packets from the socket queue through Libpcap. Once Adviser's analysis engine receives the packet, it is decoded and the analysis logic is applied to it.

2) *Adviser's capturing system with PF_RING*

In order to reduce the number of copies from the moment that the packet arrives to the capture system until it is delivered to the application, we set out the use of PF_RING [3] as capturing system in Adviser. In this point, there are different options for doing the integration. One of them is the use of PF_RING with LibPcap and a PF_RING aware NIC driver. However, there is another one which provides a better performance and, for this reason, we select it for implementing. It is the integration of PF_RING with the driver Direct NIC Access (DNA) [2] into Adviser, which allows to map NIC memory and registers to the user space. This way, packet copy from the NIC to the DMA ring is done by the NIC Network Process Unit and not by NAPI, resulting in better line-rate captures. Fig. 2b shows Adviser with PF_RING DNA.

Some adaptation modules are needed to integrate PF_RING into Adviser. First, a new module is responsible for managing the operations of PF_RING, such as the creation of the capturing ring and the interaction with the network interface to set filtering rules or working modes.

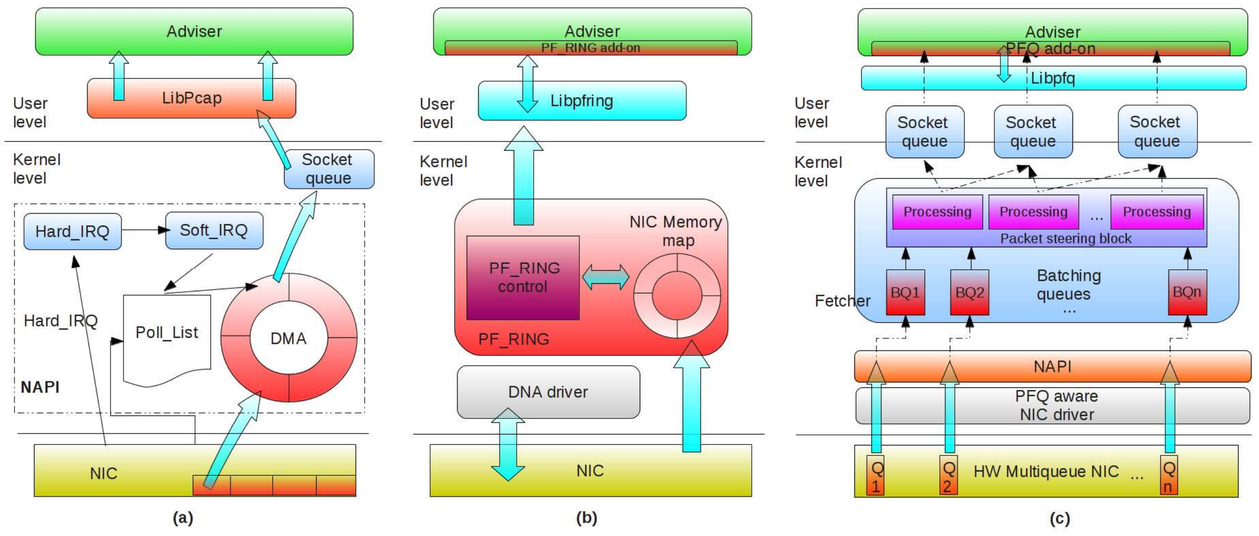

Figure 2. Adviser capturing packets (a) with NAPI and LibPcap (b) with PF_RING (c) with PFQ.

Once the capturing ring is created, a socket is enabled, the packet capturing starts and the application access to the ring through the socket. When the packet is captured, PF_RING places its contain in a data structure whose format is different from the one used by LibPcap. For this, a new module fits the format and the sizes of those data structures so that Adviser receives the data properly to be decoded.

The last adaptation is related to the concurrency system. Due to the design of PF_RING, the integration of Adviser and PF_RING has to be based on threads, instead of processes. For this reason, a new module is responsible for creating and managing threads to set an access control to the critical sections. The library Libpfring provides a control mechanism called spinlock, which allows one thread to access to the protected code, while the rest of the threads are blocked in an active-standby process.

3) *Adviser's capturing system with PFQ*

PFQ [15] is a network-capture engine designed for the Linux kernel 3.x and modern 64-bit architectures. It is optimized for multi-core processors, as well as for network devices supporting multiple hardware queues.

Adviser with PFQ is depicted in Fig. 2c. PFQ consists of the following components: the fetcher, the packet steering block and socket queues [9]. The fetcher dequeues the packet directly from the driver, which can be a standard driver or a patched "aware" driver, and inserts it into the batching queue. The next stage is represented by the packet steering block, which is in charge of selecting which socket needs to receive the packet. The final component is the socket queue, which represents the interface between user space and kernel space. Every kernel processing (from the reception of the packet up to its copy into the socket queue) is carried out within the NAPI context; the last processing stage is performed by Adviser at user space.

As in the case of PF_RING, an adaptation is necessary for the integration of Adviser with PFQ. To do this, using the tools provided by Libpfq [16], a PFQ add-on is created in Adviser. This access from Adviser to PFQ is based on threads.

B. *Ksensor. The Kernel-Level Framework*

Ksensor [17] is a kernel-level multi-processor monitoring system for high speed networks which uses commodity hardware. Its design (see Fig. 3) is based on the migration of the processing modules from user-level to the kernel of the operating system. Only system configuration (Parser) and result management (Offline Processing Module) modules remain at user-level.

First, the system parser interprets the configuration files and stores system logic in memory. Then, analysis engine processes captured packet from the network according to the logic stored in memory. After applying the rules, the engine stores the results of the analysis. Finally, offline processing module takes these results from memory and handles this information to provide traffic statistics or reports.

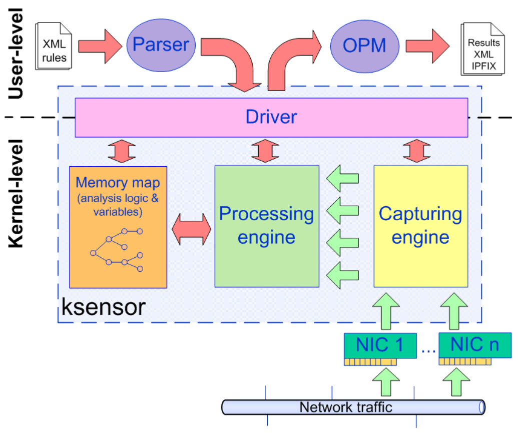

Figure 3. Ksensor framework.

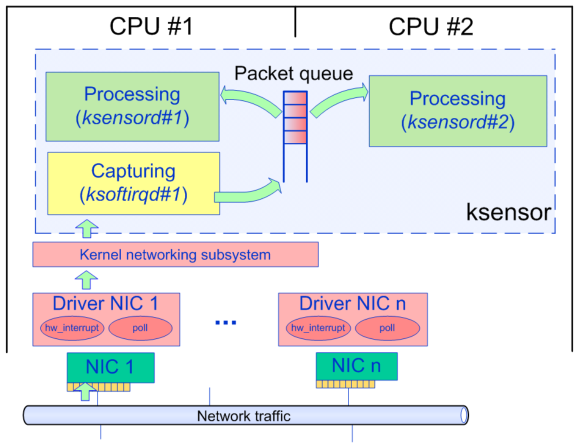

Figure 4. Execution instances in Ksensor with two processors and one NIC.

There are defined as many analyzing kernel threads (ksensord#n in Fig. 4) as the number of processors on the hardware. Each thread belongs to an execution instance of the system (capture and analysis). All threads share information through the kernel memory.

Regarding the capture, it is based on the kernel networking subsystem, i.e., NAPI. There are as many capturing instances (ksoftirqd#n in Fig. 4) as capturing NICs (IRQ affinity). A single packet queue is shared by all the analyzing instances (see Fig. 4).

In order to prevent livelock situations at high packet arrival rates, there is a congestion avoidance mechanism. It also prevents Ksensor from wasting resources in the capture of packets that the system will not be able to process later. When the packet queue reaches a maximum number of packets, this mechanism forces NAPI to stop capturing packets. This means that all the resources of all the processors are dedicated to analyzing instances. When the number of packets in the packet queue reaches a fixed threshold value the system starts capturing again.

IV. Test Setup for Comparing the Probes

The tests done in order to compare the probes are very important. Firstly, in order to automate the tests, a software architecture has been designed and implemented by NQaS research group. This architecture configures the tests, runs them and gathers the results automatically. It consists of four types of logical elements: manager, agents, daemons, and formatters.

A. *Software and Hardware Details*

The real environment where the different probes have been tested can be seen in Fig. 5. There are two networks. One is called management network and it is used for sending the configuration commands from the manager to the agents and the statistics of the test from the agents to the manager. The other one is called capturing network and it is used for testing the probes.

The machine called manager is the interface between the testing architecture and the administrator.

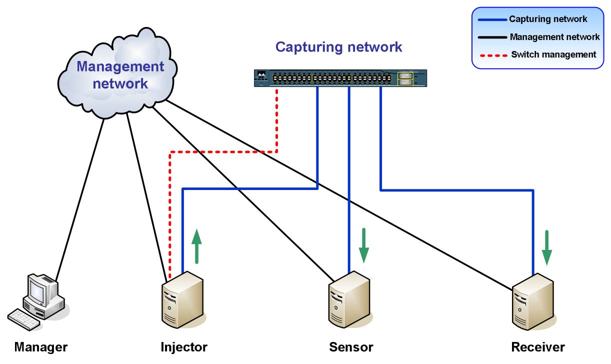

Figure 5. Network infrastructure to test the probes.

The injector is in charge of generating synthetic network traffic in order to simulate traffic load in the network. In order to do that, this machine has an Endace DAG 4.3GE card that allows injecting traffic rates up to 1 Gbps. The machine has got two processors Intel Xeon 5110 at 1.66 GHz and 2 GB of RAM memory. It runs a Debian GNU/Linux.

In the machine called Sensor run all the probes. The different implementations of Adviser are made using a Debian 7 with a kernel Linux 2.6.35. On the other hand, Ksensor is a modification of the kernel Linux 2.6.23 with a kernel module that implements the analysis tasks. The machine has got two processors Intel Quad Xeon 5420 at 2.5 GHz with 4 GB of RAM memory. Each processor has got four cores.

The receiver machine is the one that should receive the traffic. It is only used for extracting statistics.

These three machines, in order to configure the implied software and to collect the statistics, run an agent and several daemons of the testing architecture.

B. *Test Parameters*

In order to test each probe, some tests have been defined. Each test has got different configuration parameters in order to test the probes in different situations.

The parameters that can be configured are packet size, injection rate, analysis load, number of CPU cores and test duration.

The analysis load is simulated implementing different loops that take different number of loops. In this paper, the results shown are made with 1000 processing loops and 25000 processing loops of analysis load.

Each test takes four minutes and it is made with the same traffic rate, packet size (54 bytes), analysis load and number of cores. A battery of tests is a group of tests with the same configuration parameters but the traffic rate that increases for each test from 50.000 packets per second up to 1.500.000 packets per second (1 Gbps with the fixed packet size).

There are tests for one, two, and four CPU cores. The machine used for running the probes in the tests has got two quad core processors. In the tests with two cores, there is one core running in each processor. On the other hand, in the tests with four cores, there are two cores running in each processor.

V. Test Results and Discussion

In order to test the probes, three test batteries have been done for each analysis load and for each probe and with different number of CPU cores. Each battery is composed of 21 tests of four minutes. Each test is done at a different rate.

Each graph in Fig. 6 shows the analysis throughput for the three probes with 1000 loops of analysis load and a fixed number of CPU cores. In Fig. 7, it can be seen, in each graph, the results for the three probes tested in this paper with 25000 loops of analysis and a fixed number of CPU cores. The graphs in both figures show the analysis throughput, that is, the throughput of the probe in packets per second. They have three series of data, one for each probe.

On the other hand, Fig. 8 and Fig. 9 show the capture throughput for the three probes with 1000 and 25000 loops of analysis load and a fixed number of CPU cores.

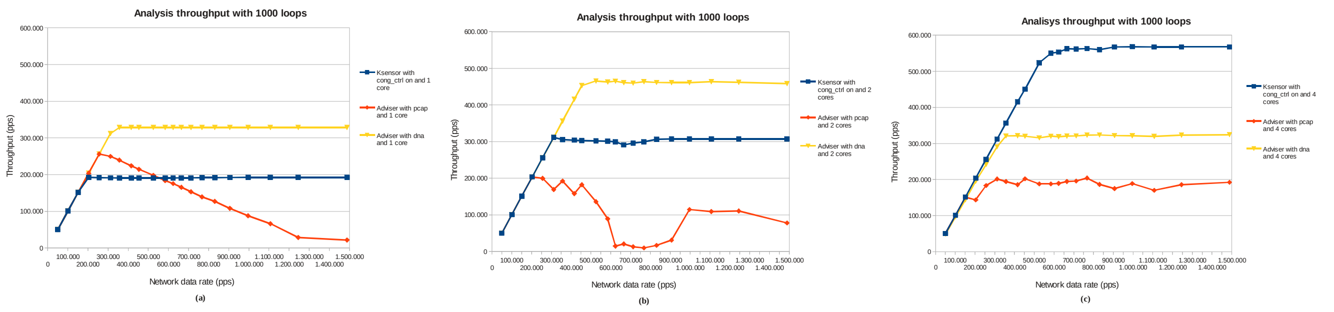

Figure 6. Comparison of analysis throughput with 1000 loops of analysis load. (a) With 1 CPU core. (b) With 2 CPU cores. (c) With 4 CPU cores.

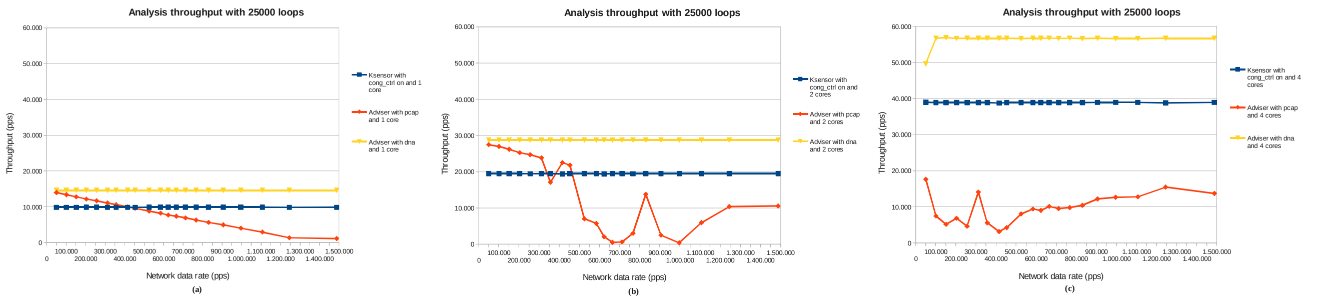

Figure 7. Comparison of analysis throughput with 25000 loops of analysis load. (a) With 1 CPU core. (b) With 2 CPU cores. (c) With 4 CPU cores.

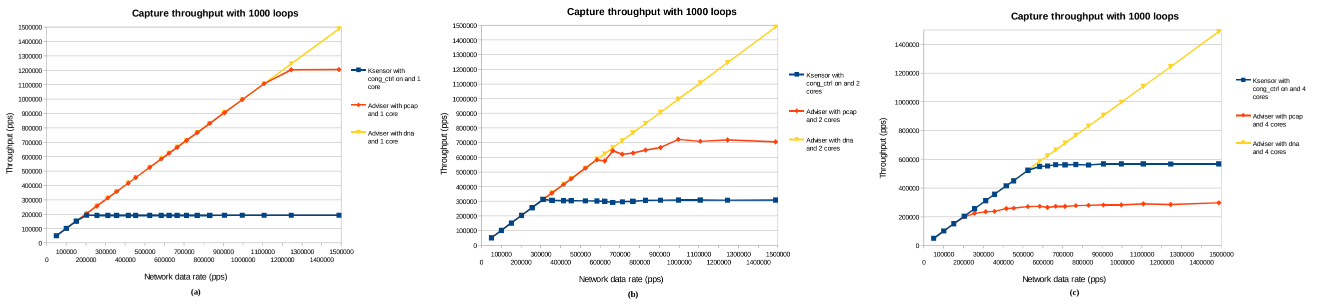

Figure 8. Comparison of capture throughput with 1000 loops of analysis load. (a) With 1 CPU core. (b) With 2 CPU cores. (c) With 4 CPU cores.

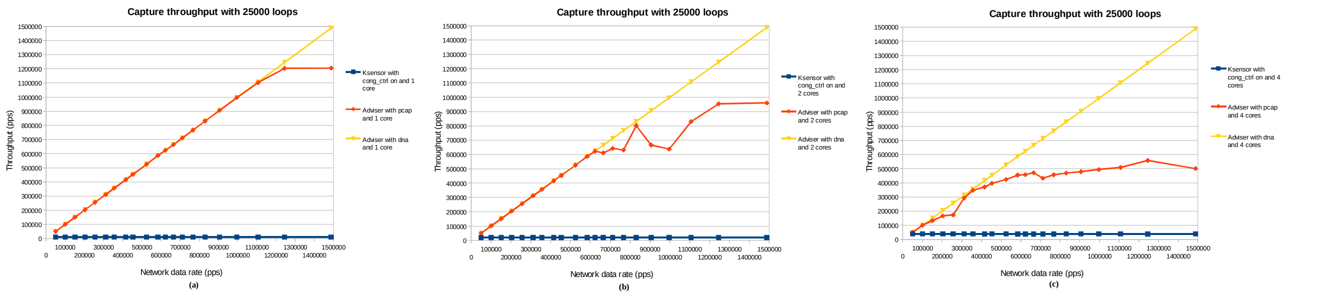

Figure 9. Comparison capture throughput with 25000 loops of analysis load. (a) With 1 CPU core. (b) With 2 CPU cores. (c) With 4 CPU cores.

It is remarkable that all these tests are done with 54 byte packets, the minimum sized ones that work in Ethernet networks. This means that, with a data rate of 1 Gbps the probes receive the maximum number of packets as possible. The system allocates its buffers taking into account the number of received packets and not the size of them.

This paper shows results of two prototypes with Adviser. One of them uses Libpcap as interface to capture packets, while the other prototype uses PF_RING_DNA. It is worth mentioning that the prototype with PF_RING DNA uses threads in order to implement the analysis task while the prototype with Libpcap uses processes. We also show results from Ksensor, the kernel-level probe presented before.

As we can see in Fig. 6-9, the prototype that has the worst performance is Adviser with Libpcap. With one CPU core it has a stable behavior. The analysis throughput is the lowest one although the capture throughput is nearly the same as with PF_RING DNA, the highest one. This happens because the capture processes have higher priority that the analysis ones. Besides, the packets are captured with all the infrastructure of the operating system. The packets are disassembled and treated as normal packets. Because of all this, the capture takes a lot of time.

When the system is capturing packets the analysis processes are slept and are not analyzing packets because there is only one CPU. Because the system takes more time capturing packets and the capture processes have more priority than the analysis ones, there are more captured packets than analyzed ones. This means that the system has to drop packets without analyzing them so the analysis throughput is lower than the capture one. There is a lot of CPU usage lost capturing packets that the system is not able to analyze. With more than one CPU core the behavior of Adviser with PCAP has the same problems that have been explained in the previous paragraph. Moreover, the design of this prototype has not resolved well the multiprocessor execution. It has two problems. The first one is that the design is done with processes. The system can execute only one process at a time so the system cannot execute more than one analysis task at the same time although the analysis processes have affinity with one CPU core. The second problem is that there is only one packet queue and the processes have to compete in order to take a packet from the queue. Because of all this, the behavior of the probe is not very stable and the performance is not good.

Obviously, the performance of the analysis with higher analysis load is lower. The system takes more time in analysis per packet so it analyzes fewer packets per second.

Regarding Ksensor, its congestion avoidance mechanism guarantees that all the packets that are captured are analyzed. Because of this, the capture throughput (see Fig. 8-9) and the analysis throughput (see Fig. 6-7) are the same.

On the other hand, we can observe in Fig. 8-9 that PF_RING DNA captures all the packets that are sent. In this case, the CPU does not execute anything because PF_RING DNA works with memory mapping.

If we compare the capture throughput of Adviser with PF_RING DNA and the capture throughput of Ksensor, we can see that PF_RING DNA has a better performance on capture terms. Moreover, the prototype with PF_RING DNA does not use CPU resources in order to capture packets so all the resources can be used to analyze them.

The comparison of the analysis throughput is not so easy. There is only one packet queue in both cases. Both prototypes have implemented threads for the analysis. So, with more than one core there are many consumers of the packet queue. There are many threads competing to access to the queue.

The higher the analysis load is the fewer accesses must be made to the packet queue. With high analysis loads the system analyzes fewer packets than with a lower analysis load. This means that, with a higher analysis load, the analysis threads make fewer accesses to the queue so there are fewer concurrency problems.

If we compare the analysis throughput we can see that, with 1000 loops of analysis, the performance of Ksensor with 2 cores is lower than the performance of the prototype with PF_RING DNA. On the other hand, with 4 cores, the performance of Ksensor is higher. With 25000 loops of analysis, the performance of the prototype with PF_RING DNA is higher in both cases, with 2 and with 4 CPU cores. One of the differences between 1000 and 25000 loops is that, with 1000, there are more accesses to the queue so the analysis threads have to wait more time in order to take a packet. Both prototypes work with as many analysis threads as CPU cores.

Ksensor has a better design for the multiple accesses to the packet queue with more than one thread at the same time but PF_RING DNA has a better performance in packet capture. With 1000 loops there are many accesses to the queue but the performance in analysis of Adviser with PF_RING DNA is higher with one and two CPU cores. But with four cores the performance of Ksensor is higher. With one and two cores the performance of the capture of Adviser with PF_RING DNA makes the analysis performance higher but with four cores the low performance in multiple access of the prototype Adviser makes the analysis performance be low. With 25000 loops there are fewer accesses to the queue so there are not as many problems as before with the multiple accesses to the queue.

Obviously, with more CPU cores the performance of the probes is higher.

VI. CONCLUSIONS

This work sets out to evaluate two software probes based on commodity hardware under different configurations. On the one hand, Adviser, a user-level framework, is evaluated with several current capturing systems (NAPI with LibPcap, PF_RING with DNA, PFQ) and several analysis loads (1000, 25000 processing loops). On the other hand, Ksensor, a kernel-level framework, uses NAPI in the capturing stage and it is tested for different analysis loads (1000 and 25000 processing loops too). It is worth mentioning that all the evaluations have been performed on the same hardware platform. It has got two quad core processors. When it is configured with one or two cores it uses one core per processor, but with more than two cores it has to use more than one core per processor. It is also remarkable the use of a testing architecture which configures the tests, runs them and gathers the results automatically.

The results indicate that Adviser with NAPI-Pcap is not a good solution. Its behavior is not predictable and its performance is lower than the performance of the other probes. With low analysis load, the performance of Adviser with PF_RING-DNA with four cores is lower than the performance of Ksensor and, even, the performance of Adviser with PF_RING and DNA and two cores. With high analysis load, the performance of Adviser with PF_RING-DNA is higher than the performance of Ksensor.

All these results have their corresponding explanation. The numerous copies in the capturing process and the absence of a congestion control mechanism between the capturing and the analysis stage are the main reasons of the unstable behavior of Adviser with NAPI-Pcap. However, Adviser with PF_RING-DNA provides a higher performance due to the improvement that it offers in the capturing stage, although there could be concurrency problems. We are referring to the problems between the capturing and analysis instances when both of them try to access the same packet queue. Finally, Ksensor does not provide a capturing performance as good as PF_RING-DNA, but it incorporates elements of control to solve concurrency problems, as well as a congestion control mechanism. For this reason, under certain circumstances (for instance, the case of 4 CPU cores with 1000 loops analysis load), Ksensor can offer a better performance than PF_RING-DNA.

As a future work we plan to migrate the prototype Ksensor to a recent Linux version in order to take advantage of the improvements that this recent kernel offers in capturing performance. In this way, the adaptation of the probe to the Generic Receive Offload (GRO) and Receive Packet Steering (RPS) techniques, which are included in recent kernel versions, can bring benefits for the system performance. On the one hand, GRO implies to change the processing of the packets in the capturing stage, by grouping packets which belong to the same flow. On the other hand, RPS proposes to increase the number of packet queues, by having one packet queue for each processor and by creating a NAPI virtual interface for each processor. This will imply to reduce the concurrency problems between the capturing and the analysis instances.

As explained in Section III, PFQ has been integrated into Adviser. This has been validated by using a conventional NIC (in particular, the model Intel 82574L) and the results obtained have been similar to native PF_RING (without DNA). But PFQ needs a multiqueue NIC in order to obtain an optimal performance. As the test scenario described in Section IV does not have any NIC of this type, Adviser with PFQ has not been tested under the optimal conditions. For this reason, there is not any result of PFQ in the comparison of Section V. In the future, we plan to obtain a multiqueue NIC to test Adviser with PFQ properly.

Finally, we want to mention that, once the migration of Ksensor is completed, we also plan to make a new comparison among the new Ksensor, Adviser with PF_RING-DNA and Adviser with PFQ

Acknowledgment

We gratefully acknowledge support from the Basque Government funding the VMAT project within the SAIOTEK 2012 initiative in the scope of which this research work has been conducted.

References

[1] F. Schneider, “Packet capturing with contemporary hardware in 10 Gigabit Ethernet environments," Proc. Passive and Active Measurement Conference (PAM 2007), Springer-Verlag Berlin Heidelberg, Apr. 2007, pp. 207-217.

[2] ntop project, http://www.ntop.org, 14.10.2013.

[3] A. Fiveg, “Ringmap capturing stack for high performance packet capturing”, http://wiki.freebsd.org/AlexandreFiveg, Sept. 2010.

[4] L. Rizzo, “Netmap: a novel framework for fast packet I/O,” Proc. 2012 USENIX Annual Technical Conference, USENIX Association, Jun. 2012, pp. 9-20.

[5] L. Deri, “Improving passive packet capture: beyond device polling,” Proc. 4th International System Administration and Network Engineering Conference (SANE), vol. 2004, Oct. 2004, pp. 85-93.

[6] L. Deri, “nCap: Wire-speed packet capture and transmission,” IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON), IEEE, May. 2005, pp. 47-55.

[7] F. Fusco and L. Deri, “High speed network traffic analysis with commodity multi-core systems,” Proc. Internet Measurement Conference (IMC 2010), ACM, Nov. 2010, pp. 218-224.

[8] A. Cardigliano, L. Deri, J. Gasparakis, and F. Fusco, “vPF_RING: Towards wire-speed network monitoring using virtual machines,” Proc. Internet Measurement Conference (IMC 2011), ACM, Nov. 2011, pp. 533-548.

[9] N. Bonelli, A. Di Pietro, S. Giordano, and G. Procissi, “On multi-Gigabit packet capturing with multi-core commodity hardware,” Proc. 13th Passive and Active Measurement Conference (PAM), Springer, Mar. 2012, pp. 64–73.

[10] T. Mrazek and J. Vykopal, “Packet capture benchmark on 1 GE”, CESNET technical report 22/2008, http://www.cesnet.cz, Dec. 2008.

[11] L. Braun, A. Didebulidze, A. Kammenhuber, and G. Carle, “Comparing and improving current packet capturing solutions based on commodity hardware,” Proc. Internet Measurement Conference (IMC 2010), ACM, Nov. 2010, pp. 206-217.

[12] A. Ferro, F. Liberal, A. Muñoz, I. Delgado, and A. Beaumont, “Software architecture based on multiprocessor platform to apply complex intrusion detection techniques”, Proc. 2005 IEEE International Carnahan Conference on Security Technology (CCST’05), IEEE, Oct. 2005, pp. 287-290.

[13] C. Benvenuti, Understanding Linux Network Internals, O'Reilly Media, 2005.

[14] LibPcap, http://www.tcpdump.org, 14.10.2013.

[15] PFQ Homepage, http:// netserv.iet.unipi.it/software/pfq, 14.10.2013.

[16] PFQ: Accelerated packet capture engine for multi-core architectures, http://pfq.github.com/PFQ, 14.10.2013.

[17] A. Munoz, A. Ferro, F. Liberal, and J. Lopez, “A kernel-level monitor over multiprocessor architectures for high-performance network analysis with commodity hardware,” Proc. SensorComm 2007, IEEE, Oct. 2007, pp. 457-462.

[18] A. Pineda, L. Zabala, and A. Ferro, “Network architecture to automatically test traffic monitoring systems,” Proc. Mosharaka Int. Conference on Communications and Signal Processing (MIC-CSP2012), Academy, Apr. 2012, pp. 18-23.