Collecting Packet Traces at High Speed

Gorka Aguirre Cascallana
Universidad Pública de Navarra
Depto. de Automatica y Computacion
31006 Pamplona, Spain
aguirre.36047@e.unavarra.es

Eduardo Magaña Lizarrondo
Universidad Pública de Navarra
Depto. de Automatica y Computacion
31006 Pamplona, Spain
eduardo.magana@unavarra.es

***Abstract*— In order to capture packet traces at high speed using a low-cost platform, we have to optimize the networking stack of a general purpose operating system. Different techniques are compared with the final objective of avoiding packet loss. Among those techniques we will study the performance of NAPI [6] and PF-RING [9]. Depending on the final application, we should tune certain parameters accordingly. We also present the advantages of a multiprocessor platform and the problematic of storing full packets directly to hard disk.**

Keywords: network monitoring, packet capture, interrupt coalescence, shared memory

I. INTRODUCTION

Broadband technology is changing our world. Continuously, network infrastructure is supporting more and more traffic. Network monitoring has to provide information about these networks so it has to support the latest 1 or 10 gigabit per second speeds. In addition, general purpose operating systems are not developed with an optimized networking stack. As a result, a high performance specific hardware is needed to monitor high speed links.

Nowadays there are different proposals to optimize networking procedures in order to prepare a network monitoring node based on a general purpose low-cost PC. This node will be able, for example, to capture all packets from a high-speed link and store them in hard disk for further post-processing. Some of these techniques are as follow:

- Improving operating system architecture: use less system resources (CPU power), for example reducing the number of memory copies [1].
- Improving hardware: network interface cards (NICs) designed specifically for network monitoring tasks [2]. They usually have bigger memory buffers, GPS synchronization and even a programmable processor to make some kind of pre-processing. However these special NICs area very expensive and then hard to apply for a low-cost platform.

There is another bottleneck that must be taken into account if we use a general purpose PC. PCI bus technology can be considered as a bottleneck depending on the version we are working on. Whereas PCI–X technology can manage a full 10 Gbps workload, a normal PCI (32 bit/33 MHz) Ethernet NIC will be a bottleneck for 1 Gbps traffic [3].

This work was supported by the Spanish Ministry of Education and Science (Project PINTA TEC2004-06437-C05-03).

In this paper will be show a comparison of the latest techniques that could be used to optimize a general purpose operating system. This will allow an easy way to make a powerful low-cost traffic monitoring node. Our final objective will be to prepare a monitoring node in charge of capturing packets to hard disk for later post-processing.

II. OPTIMIZING NETWORK STACK

As we stated before, nowadays the optimization in networking procedures is following two guidelines. The first one is based mainly on interrupt coalescence, polling and efficiency in copying to the user memory. Whilst, the second guideline is based on enhancements for latest NICs hardware, such as: scatter and gather, checksum offload, data alignment, packet split and jumbo frames. This paper is focused on the first guideline.

Interrupt coalescence [4] appeared as a solution for the old networking stacks. These systems launch make a single interrupt for each packet that was received. Therefore, we can reach a receiver livelock state without receiving too many packets. In this state, the CPU is using too much time in attending interrupts, with the overhead that it involves. Instead of making an interrupt call for each packet, interrupt coalescence makes an interrupt for a group of several packets. In fact, this technique looks at the CPU usage in order to decide whether it is reaching a livelock state. If the CPU usage by interrupts overtakes certain threshold, an optimal interrupt rate is calculated to avoid this state. By doing this, we obtain two advantages. In one hand, we have low packet latency and, on the other hand, we have a moderated CPU usage. Latest Gigabit Ethernet NIC drivers have worked out this feature, and now their advantages can be obtained by tuning a parameter of the driver [5].

Another improvement that uses polling has been developed for Linux systems after 2.4.20 release. This is called Napi [6]. Its purpose is again avoiding a livelock state. The way it works is similar to interrupt coalescence: whenever a livelock state is overtaken, Napi disables the interrupts subsystem and it starts polling packets which are held in the kernel memory. These packets have been transferred by DMA mechanisms, transparently for the CPU. Napi makes usage of SoftIrq, which is a new feature used by SMP (Symmetric MultiProcessor) systems, useful for example with Intel 4/Xeon processors.

Finally, one of the main problems in packet processing is the necessity of memory copies. For the entire processing of a single packet, first of all, it is necessary to copy the packet from the network card to the kernel memory. This is usually done by a DMA mechanism, and in most of the cases there are not further overheads [12]. However, another copy to user memory is necessary so that a user program can manage this data. To avoid these packet copies, a technique called zero-copy was developed [13]. In this case, page mapping moves data from kernel memory to user just by shared memory. The size of a memory page must be greater or equal to MTU in order to able to do the mapping. The packet size can be smaller than this page size.

III. LOW-COST HIGH-SPEED TRAFFIC GENERATOR

The design of a reliable traffic generator is the first step that is needed to make a testbed for network monitoring. This generator must be powerful enough to saturate the receiver (monitoring node) because we want to test the receiver in extreme conditions. In other words, we need to generate a huge amount of packet per second. The idea of using dedicated hardware to generate traffic is good, but rather expensive. Another way to obtain this traffic could be done by taking advantage of switching hardware technology. Generally, a single Gigabit switch has to support the maximum amount of packets stated in the gigabit technology standard with minimum packet size.

The basic idea to implement this generator is replicating a single traffic source into N free ports of a switch (replication switch). Then we would obtain the same single traffic N times in N different wires. Then, we need to add all this traffic in a single port and this can be done by another switch (adder switch).

Between the advantages of this configuration we have:

- Maximum speed (as much as gigabit technology can support).
- Low cost system: two switches, a computer (a source generator) and a couple of RJ-45 wires.
- We can obtain almost any amount of traffic just by plugging the proper number of wires. As an outcome we will have N (wires) times a single traffic.

And some disadvantages are also present:

- Duplicated traffic is obtained,
- In the receiver side, packets appear in bursts of the same replicated packet (worst case scenario for packet monitoring).
- No packets can go back to the traffic source. This situation would generate a packet loop between both switches.

The arrangement of the traffic generator is presented in Fig.1.

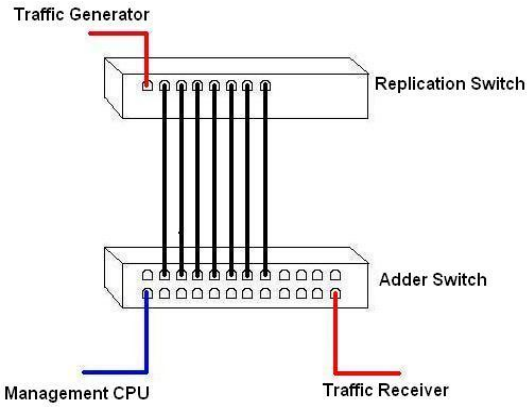

Figure 1. High-speed traffic generator

In this case, both switches are working in two different modes of operation: as a replication switch and as traffic adder switch.

The replication mode needs that the switch does not learn the MAC address of the receiver. By doing this, we obtain that the switch would work as a hub: it will copy input packets from one input port to all the other ports, because the switch does not know in which port the destination is attached. The idea of replace the switch with a real hub does not give good results, because in the hub we can have packet collisions.

The traffic adder switch needs the receiver attached to one of its ports and it has to learn the receiver MAC address. In the other ports we can connect the output ports from the replication switch. The adder switch will send all the traffic to only the port where the receiver is attached. Furthermore, this MAC address must be refreshed every certain minutes so the adder switch can remember the port of the receiver (the MAC table has to be refreshed).

The points to be taken into account to carry out this traffic generator are:

- Switch off any spare network services in both computers (source and receiver-monitor).
- Use a single traffic generator (program) that doesn't need any response from the receiver. For example, a single UDP generator.
- Configure two subnets in the "generator – switches –receiver" system in order that the receiver can't send any packet to the generator.
- Define the ARP table in both ends, so that neither ARP replies nor requests are sent to the network.
- Spanning tree protocols must be disabled.

Finally, to make this system work properly we need to enable flow control in both NICs (in the source's and in receiver's card). In Fig.2 it is shown the consequences of switching off the flow control parameter.

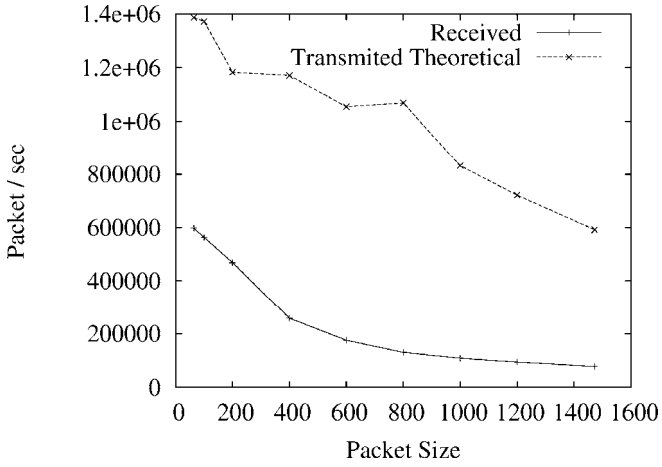

Figure 2. Theoretical transmission and reception rate

Fig.2 shows the theoretical transmission rate for each packet size. This is obtained multiplying N (wires) with the basic traffic source rate. Nevertheless this graphic is misleading due to the fact that Gigabit technology has its limit depending on packet size. For example, for a packet size of 1472 bytes the maximum number of packet per sec that Gigabit can support is nearly to 85,000 packets, whereas in Fig.2 appears 600,000 packets. The adder switch is not to transmit more than 85,000 packets/sec to the receiver so this is the real transmission rate. In any case the reception rate shown, is the number of packet per sec which are copied to kernel memory. The difference between the sending rate and receiving rate is equal to the number of packet dropped by the NIC's hardware.

In the next Fig.3 it is shown the same data but in this case flow control parameter is on. In this test, transmission rate and reception rate is nearly the same because with flow control we are reducing the sending rate. As a conclusion flow control is necessary in order to obtain reliable results and to avoid congestion in the adder switch (receiving much more packets that it is capable to transmit).

IV. NAPI SYSTEM VS INTERRUPT COALESCENCE SYSTEM

In the next comparative, both Napi and Interrupt Coalescence systems are going to be tested using the low-cost traffic generator.

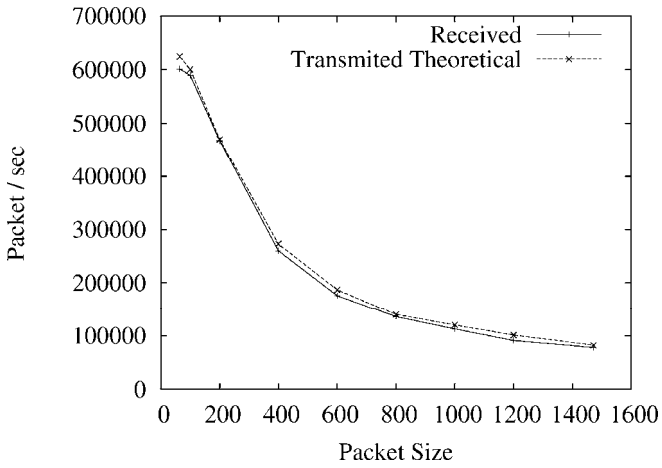

Figure 3. Theoretical Transmission and Reception rate

To carry on those tests, two different Linux kernel versions (2.4.18 and 2.6.9) are going to be used. The 2.4.18 kernel version is the last version where Napi was not implemented and therefore Interrupt Coalescence tests can be done. On the other hand, the second kernel version (2.6.9) has Napi features.

With regard to the receiver, it is composed by a computer with 2 processors where each of them is a Pentium 1,7 GHz, and a 3Com 3C996B Gigabit Server NIC [7]. This card works with a compatible driver known as BCM5700 [8].

The BCM5700 driver has the following parameters that we will tune in order to obtain the combination which gives best results:

- rx_std_desc_cnt: number of descriptors in the kernel memory. It is related to the number of packet per sec that can be stored in the kernel memory.
- rx_max_coalesce_frames: this parameter sets the number of packet per sec that the driver will hold before a new interrupt call is generated.
- rx_coalesce_ticks: the times (in µs) that a driver holds on before generating a single interrupt.
- adaptive_coalescence: enables or disables the interrupt coalescence.

Several experiments were made with different combination of parameters. The best results for Interrupt Coalescence were for the next parameters: rx_std_desc_cnt=500 (in order to store more packets into the kernel memory), rx_max_coalesce_frames=0 (in order to disable it) and rx_coalesce_ticks=10 (10 µs that the driver will hold on before generating an interrupt). In Fig.4 it is shown the number of packets per second received by the NIC in which the packet size changes from 64 bytes to 1472 bytes. Fig.5 presents the packets drops profile.

According to Fig.5 a great amount of packets are dropped, especially for small packet sizes. Even if the network system is working with less interrupt calls, the copy process to the user memory consumes CPU resources.

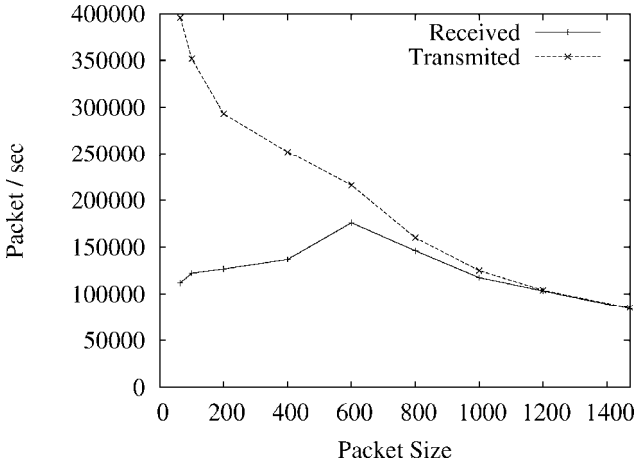

Figure 4. Received and transmitted packets

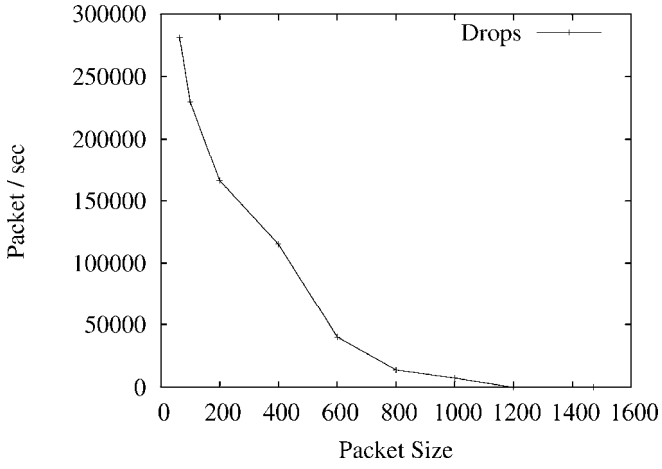

Figure 5. Packet drops

Besides the SMP feature is disabled whereas in systems with Napi (Kernel version greater than 2.4.20) SMP feature is exploited.

The parameters which were chosen for the Napi system to obtain the best results were: rx_std_desc_cnt=500, rx_max_coalesce_frames=0, rx_coalesce_ticks = 50 and adaptive_coalescence=0 (this is switched off by default when Napi system is detected). The Napi system has a dropped packet subsystem were any dropped packet is processed and therefore any dropped packet is counted.

In Fig.6 we can see the number of packet per sec received using the Napi system. In this case, there are no dropped packets at kernel level, however packets are dropped at NIC's memory level.

In Fig. 7 CPU usage for both systems is shown. Napi uses more CPU because is using both processors (softirq). However Interrupt Coalescence is using only one processor, obtaining 50% of total CPU power.

As a conclusion, Napi is a good alternative in resource consumption and packets processing power.

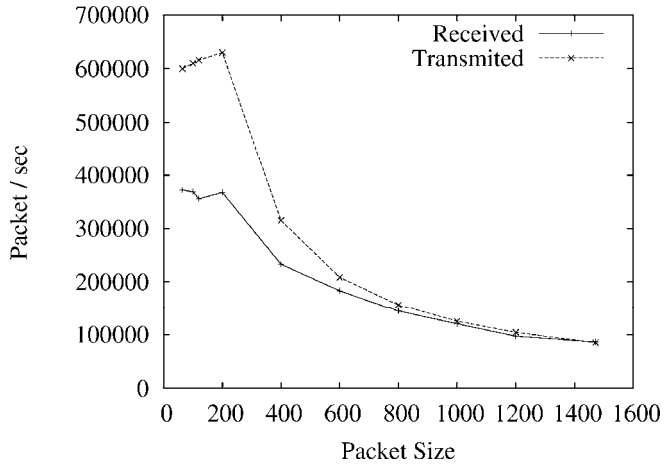

Figure 6. Received and transmited Packets

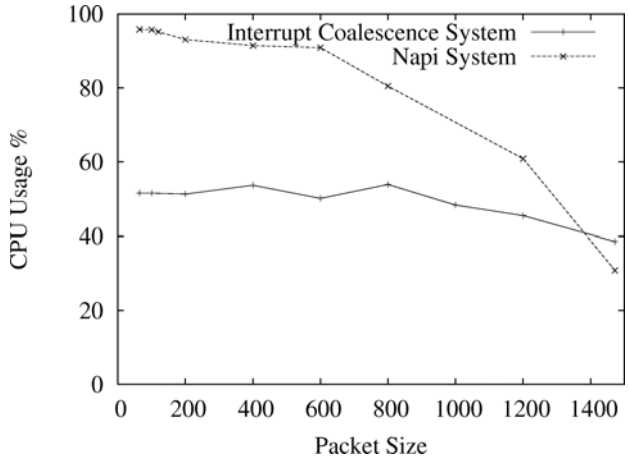

Figure 7. CPU usage for both systems

V. THE PF-RING MODULE

The first step for capturing a network packet is storing packets into the kernel memory. This is a task that generally DMA does. Whatever, copying from the kernel memory to the user memory is necessary for an application, in which software is easier to develop. A system which needs to make two copies for each packet is really inefficient. However there is a proposal that, with the proper setting, avoids the second copy by using shared memory mapping. The module is known as PF_RING [9]. In addition, a modified libpcap [10] library is designed by the same author in order to build libpcap applications with this feature enabled.

The following experiment presents the difference between a standard built libpcap application, which counts the number of packet per sec received, and the same application with the modified libpcap + PF_RING. The difference meanly is that one makes copies from the kernel memory to the user memory and the other doesn't (a shared memory is available for intercommunication). In Fig.8 it is shown the packets per second with a standard libpcap and the enhancement with PF_RING and modified libpcap.

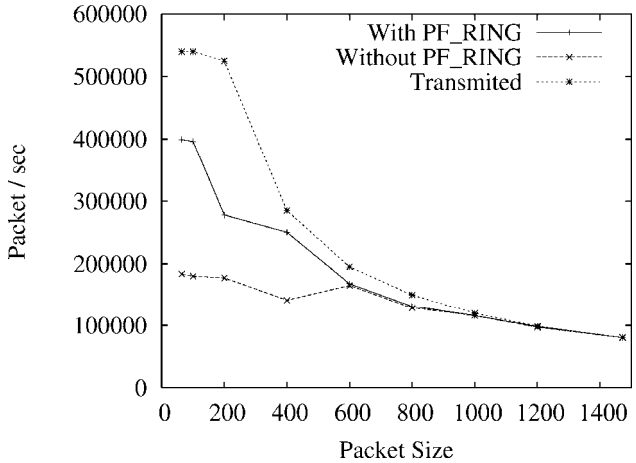

Figure 8. Performance with and without PF_RING

If the final objective is to obtain general network statistics, there is no need to work with full-sized packets. For the vast majority of monitoring applications, the headers of a packet provide enough information. If the idea is storing packets in hard disk, storing only the headers will reduce the amount of information to process. The next experiment will obtain the packet headers and will store them in hard disk.

First of all, there is an issue that must be dealt with. The data rate of a hard disk can be a bottleneck when storing packet traces. A standard hard disk can keep up with 20 Mbyte/sec, SATA-SCSI hard disks are near to 40 – 60 Mbyte/sec [11]. A great number of small packets (from 64 to 200 bytes) can be saved as a huge amount of small packets and it does not mean a high data transfer.

On the other hand, there will be a bottleneck when using a big packet size. But, if we only store packet headers (about 20-100 bytes in size), a high data transfer is not necessary. In order to save packets to hard disk, tcpdump [10] was built with standard libpcap and modified libpcap + PF_RING. Fig.9 shows the results.

In Fig. 9 hard disk transfer rate is the bottleneck and the reason why no more packets are saved. Whatever, when saving packets between 64 to 200 bytes it is shown an improvement with regard to standard tcpdump (now the hard disk is not the bottleneck).

In Fig. 10 we only save packet headers for each packet. In this way, hard disk speed won't be the bottleneck and therefore more packets will be processed. We choose 60 bytes for header length (typical 20 bytes basic IP header, 20 bytes basic TCP header and 20 bytes for application level). In Fig.10 a huge improvement is shown if we only store packet headers.

We measured the average packet size over a week for the Internet access link of the Public University of Navarra and we obtained 592 bytes. Taking advantage of Fig.10, for a 592 bytes packet size, we would obtain around 5% packet loss at gigabit speeds.

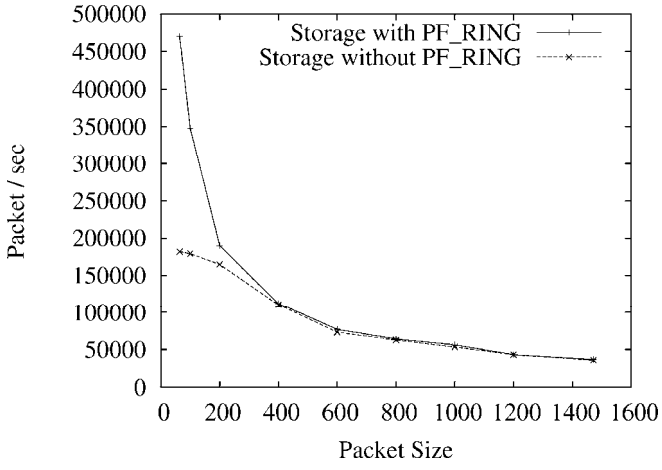

Figure 9. Packets stored in Hard Disk using PF_RING

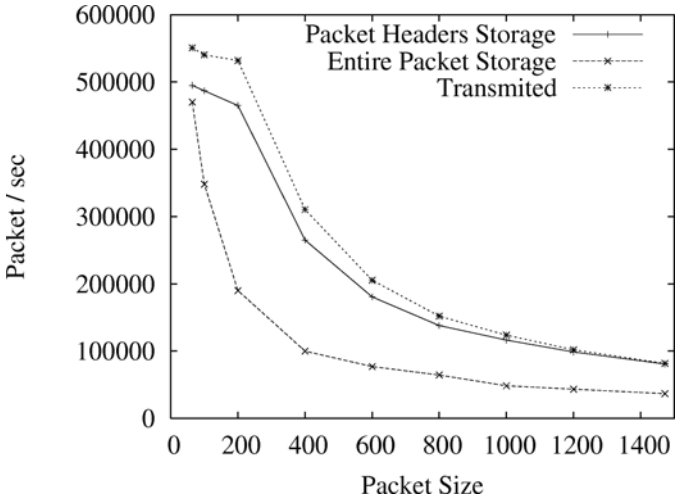

Figure 10. Outcome saving Headers and whole packets

VI. CONCLUSIONS

Nowadays, there are many techniques that have improved the packet capture speed and that can be classified in two types: improving NICs hardware and improving operating system architecture. In this paper we have focused in the second one.

The proposed traffic multiplier is a cheap traffic generator based on combining Gigabit Ethernet switches. The multiplier can obtain 1 Gbps test traffic without problems, and it would scale with 10 Gbps switches accordingly.

New Napi features and PF_RING proposal are the best choices to optimize the packet capture subsystem of a full monitoring system. On the other hand, packet storage on hard disk is a typical option for later post-processing. For this case, packet headers can be saved instead of storing the whole packets. By doing this, we can increase the packet capturing rate.

As future work, the capability of generic NICs for only capturing packet headers will be revised. This would allow reducing the transfer size between the NIC and kernel memory.

REFERENCES

[1] Jeffrey S. Chase, Andrew J. Gallatin, and Kenneth G. Yocum. "End System Optimizations for High-Speed TCP". IEEE Communications, Special Issue on TCP Performance in Future Networking Environments. 39 (4), pp. 68—74, April 2001.

[2] H.-W. Jin, P. Balaji, C. Yoo, J.-Y. Choi, and D.K. Panda. "Exploiting NIC Architectural Support for Enhancing IP based Protocols on High Performance Networks". Journal of Parallel and Distributed Computing, 65(11):1348--1365, November 2005.

[3] "PCI-SIG developers homepage".*(http://www.pcisig.com/home).*

[4] R. Prasad and C. Dovrolis. "Effects of Interrupt Coalescence on Network Measurements". In proceedings of Passive and Active Measurement (PAM) Workshop, April 2004, France.

[5] J. C. Mogul and K. K. Ramakrishnan. "Eliminating receive livelock in an interrupt-driven kernel". In Proceedings of the 1996 Usenix Technical Conference, pages 99--111, 1996.

[6] J. H. Salim, R. Olsson, and A. Kuznetsov. Beyond softnet. In proceedings of USENIX 2001, pp.165-172, November 2001.

[7] V. Pai, S. Rixner, H. Kim. ”Isolating the Performance Impacts of Network Interface Cards through Microbenchmarks”. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), New York, NY, (July, 2004)

[8] “Broadcom Homepage”. *(http://www.broadcom.com).*

[9] Lucas Deri. “Improving Passive Packet Capture: Beyond Device Polling”. 15th NMRG, Bremen, Germany, January 2004. *http://www.infosecwriters.com/text_resources/pdf/passive_packet_capture.pdf).*

[10] “Tcpdump homepage”.*(http://www.tcpdump.org).*

[11] “SATA Technology Homepage”. (*http://www.serialata.org/)*

[12] David D. Clark, John Romkey and Howard Salwen, "An Analysis of TCP Processing Overhead", proc. IEEE Conference on Local Computer Networks 1988, pp. 284-291

[13] James P.G. Sterbenz and Gurudatta M. Parulkar,"Axon: A Distributed Communication Architecture for High-Speed Networking", proceedings of IEEE INFOCOM 1990, pp 415-425