



**Repositorio Institucional de la Universidad Autónoma de Madrid**

<https://repositorio.uam.es>

Esta es la **versión de autor** de la comunicación de congreso publicada en:  
This is an **author produced version** of a paper published in:

2014 IEEE Intl Conf on High Performance Computing and Communications,  
2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th  
Intl Conf on Embedded Software and Syst (HPCC,CSS,ICSS). IEEE, 2014.  
486-489

**DOI:** <http://dx.doi.org/10.1109/HPCC.2014.81>

**Copyright:** © 2014 IEEE

El acceso a la versión del editor puede requerir la suscripción del recurso  
Access to the published version may require subscription

# Packet storage at multi-gigabit rates using off-the-shelf systems

Victor Moreno, Pedro M. Santiago del Río, Javier Ramos, José Luis García-Dorado,  
Ivan Gonzalez, Francisco J. Gomez-Arribas, Javier Aracil

High Performance Computing and Networking Group, Universidad Autonoma de Madrid, Madrid, Spain  
email: victor.moreno@uam.es

**Abstract**—The use of closed solutions from most known vendors to carry out network-monitoring tasks has turned out to be a questionable option given their lack of flexibility and extensibility, which has typically been translated into higher costs. Consequently, we study whether high-performance monitoring tasks can be carried out using off-the-shelf systems, the alternative to these pitfalls from the research community, consisting in the combination of open-source software and commodity hardware. We focus on sniffing and storing network traffic as one of the major tasks in any monitoring architecture. Specifically, we first review the keys to sniff traffic at multi-gigabit rates, and then present an experimental evaluation of commodity hard drives. Finally, the lessons learned from such studies and the performed experiments have conducted us to the development of an open solution, namely **HPCAP**, which sniffs and stores multi-gigabit traffic using commodity hardware without packet losses in very demanding scenarios.

**Keywords:** Packet sniffing; packet storage; commodity hardware; open-source software; off-the-shelf systems.

## I. INTRODUCTION

To the historical fact that users' demands increase every day both in terms of more bandwidth and enhanced quality of experience, nowadays users are also very demanding in term of prices as the telecommunication market has progressively become a more mature market. In this challenging scenario, operators do not have another option but deploying extensive monitoring architectures to guarantee quality of experience, while network monitoring solutions must both be fast enough to deal with multi-gigabit rates and keep low costs —given that monitoring usually requires redundancy and multiple points of measurement. Nevertheless, even a simple monitoring action, such as sniffing traffic packets' headers, demands a high amount of resources and computational power. For instance, traffic monitoring at rates ranging from 100 Mb/s to 1 Gb/s was considered a challenging task a few years ago, whereas contemporary commercial routers typically feature 10 Gb/s interfaces, reaching aggregate rates as high as 100 Tb/s.

Traditionally, network managers have faced the monitoring problem by applying specialized hardware devices such as FPGA-based solutions, network processors and, above all, high-end (and closed) commercial solutions. While the achieved performance is outstanding, this tailor-made approach lacks either of flexibility, extensibility or both and tends to be expensive. Alternatively, the research community has recently paid special attention to the use of commodity hardware along with open-source software to carry out high-performance

traffic monitoring tasks [1]. Let us refer to such a combination of open-source software and commodity hardware as off-the-shelf systems. Off-the-shelf systems offer numerous advantages that overcome some of the limitations of ad-hoc hardware solutions, namely: flexibility, affordability, ease of maintenance, availability and scalability [2].

In this paper, we focus on the first task in any network monitoring system: sniffing and storing packet-level traces. Aggregate-level traces —e.g., Netflow or Multi Router Traffic Grapher (MRTG) records— may be helpful to monitor, diagnose, manage and operate networks, however they are often not enough to study the root cause of a given situation [3]. As an example, once a network intrusion detection system (NIDS) detects a network anomaly from MRTG records (e.g., a peak in the time series of active flows), inspecting packet payload during and after such anomaly (not necessarily over a long period of time) serves to identify the cause of the incident. To illustrate the very demanding requirements of such task, it suffices to say that a full-saturated 10GbE link carries more than 14 million packets per second in the worst-case scenario. Current commodity servers at hardware-level are potentially capable of sniffing (which does not mean storing) at such rates thanks to the use of multi-core architectures and modern network interface cards (NICs). On the other hand, the software side is not that fast. That is, by default network drivers, operating system's network stacks and user-level network applications present several architectural limitations, which impede fetching packets from the wire to the application layer at high rates. The storage process is even more challenging, given the I/O rates of commodity hard drives.

In this scenario, we study whether such software side and commodity hard drives' access can be skilfully tuned so that packet storage at multi-gigabit rates using off-the-shelf systems becomes a reality. We divide this study in two parts. First, we review the key modifications and low-level tuning that can be made to the operating system's networking stack, the NIC driver and upper-layer applications to sniff at multi-gigabit rates. This is motivated by the impossibility to do so at multi-gigabit rates using standard capture tools —namely, `tcpdump`. We further test some research community's novel approaches (including our proposal), which follow such modifications and tuning principles, showing that full-packet capturing can be carried out with zero losses in several demanding scenarios. Second, we note that there is little knowledge about the viability to store traffic using off-the-self systems. This article addresses such issue showing the limitations of packet storage and highlighting the key parameters to achieve high-

performance storage. The results of the study of both the novel capture engines and the hard drive experiments guided us to the development of our own open-source tool, HPCAP, available under an open-source<sup>1</sup> license. Such tool spans both capture (with novel characteristics) and high-performance storage functionalities, while the experimental analysis assesses its capacity over 10 Gb/s using standard Linux commands and C functions.

## II. SNIFFING TRAFFIC FROM THE WIRE

The first objective that must be accomplished for multi-gigabit full data retention is sniffing at 10 Gb/s speeds and beyond. Such operation is not trivial (more than 14 million packets per second in the worst-case scenario) and requires multiple optimizations on off-the-shelf systems' architectures. Firstly, we wonder how much traffic may be captured in an out-of-the-box scenario —i.e., standard capture tools running on a commodity server. We performed several experiments of traffic sniffing using a Supermicro X9DR3-F commodity server and two 6-core Xeon E52630 processors running at 2.30 GHz and with 96 GB of DDR3 RAM at 1333 MHz. The server is equipped with an Intel 82599 10GbE NIC plugged into a Peripheral Component Interconnect Express (PCIe) 3.0 slot. From the software point of view, the system runs an Ubuntu 12.04 operating system, configured with the default network stack and default `ixgbe` network driver.

The leftmost 2-column group in Fig. 1 shows the percentage of captured packets using the vanilla `ixgbe` driver using the de facto standard capture tool `tcpdump`<sup>2</sup>. Those columns show two traffic injection cases, namely, synthetic 64-byte packets with CRC included (lighter bar) and a real backbone trace [4] (darker bar), both replayed at wire-speed. We observe that this out-of-the-box scenario is only able to sniff (disk storage is not yet considered) less than 15% and 60% out of the total sent packets, for synthetic and real traffic respectively.

In this light, the standard configuration has shown to be insufficient to capture full-rate 10 Gb/s traffic. Non-Uniform Memory Access (NUMA) architectures groups processing cores along with independent memory banks to create a NUMA node. Performance significantly increases if accesses to memory and PCIe devices are carefully planned. The interconnection between the processors and the PCIe bus that holds the NICs and the Redundant Array of Independent Disks (RAID) controllers determines the way that processes and memory should be distributed. Thus capture processes must be placed on the same NUMA node as the driver-level receiving threads, thus reducing the memory access overhead. If those affinity-related adjustments are performed in an out-of-the-box Linux machine with `tcpdump` a significant performance improvement appears, as shown in Fig. 1. In the vanilla case, the program runs in a random core and the number of reception queues is the default value (12 in our case, the maximum in the system). In the optimized case, the capture process runs in the NUMA node attached to the NIC with 4 reception queues whose interrupts were also attached to that node. This 4-queue configuration showed the maximum throughput in our setup. With such simple changes, the number of packets captured

rises up to 20% and 75% (from 15% and 60%), for synthetic and real traffic respectively.

To fully capture high-speed traffic, some capture engines [5], [6], [7], [8], [9] have been proposed in the literature, which apply some of the previous explained techniques over the driver and the network stack. Those capture engines make use of optimizations such as memory pre-allocation and reuse, prefetching packet data into system's caches, avoiding the system stack to exploit the parallel paths that multiple queues offer, etc. Specifically, `PF_RING` and `PacketShader` achieve wire-speed using few cores (even with only one reception queue) at the expense of flexibility. `HPCAP-driver`, our proposal, obtains similar results while additionally provides accurate timestamping and other features to maximize disk storage throughput. Note that `HPCAP-driver` uses two CPU cores per receive queue, while the other engines use one core per queue (see Sec. IV). `PFQ` needs a larger number of queues and cores to achieve an acceptable throughput but gives more flexibility and additional functionalities, such as customized packet aggregation and a socket-like API. To assess the outstanding capacity of such packet capture engines, we stress test `PF_RING` (as leading example due to its performance) and our proposal `HPCAP-driver` in the same two scenarios as `tcpdump` tool. In Fig. 1, we can see that both solutions present excellent results in both scenarios —i.e., 64-byte sized packets and real backbone traffic injected at wire-speed.

## III. STORING DATA

Once the traffic is sniffed, the storage process must be carried out. At this point traffic is captured at multi-gigabit rates, but, there is scarce knowledge about writing at multi-gigabit rates in general-purpose hard drives. Nowadays, a high-end SATA-3 mechanical disk allows theoretical rates up to 4.8 Gb/s for sequential reads and 1.2 Gb/s for sequential writes. A SATA-3 Solid-State Drive (SSD) may achieve speeds close to 3.2 Gb/s for both read and write operations, but its price per GB is 10 times greater. Consequently, no matter if the hard drive is SSD or conventional, a single disk is not enough to comply with the line rate and a disk RAID array is in order.

However, RAID configuration parameters are manifold and a wrong choice of values leads to severe performance degradation. We have evaluated the actual write throughput of a RAID volume composed of high-end mechanical hard disks<sup>3</sup> using the out-of-the-box Linux server described in the previous section. Specifically, we have conducted throughout testing with the following configuration parameters using an Intel RS25DB080 RAID controller:

*RAID level:* A RAID-0 configuration provides the maximum read/write rate at the expense of losing data redundancy.

*Number of disks:* We have assessed how performance differs with a varying number of disks from 1 to 12 in RAID-0.

*Strip size:* the amount of data per basic write operation. The default value is 128 KB, and we have tested strip sizes between 64 KB and 1MB.

*RAID write cache policy:* this parameter refers to the use of the RAID controller's cache memory. A *Direct* policy disables

<sup>1</sup><https://github.com/hpcn-uam/HPCAP>

<sup>2</sup><http://www.tcpdump.org/>

<sup>3</sup>We have used a Hitachi HUA723030ALA640 with SATA-3 interface and 3TB of capacity

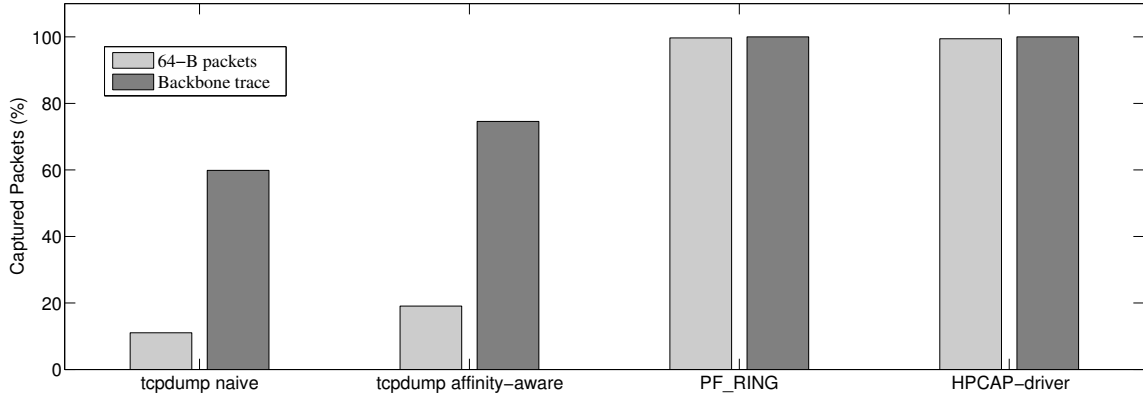


Fig. 1: Percentage of captured (stored into system RAM memory) packets for a full-saturated 10 Gb/s link for a synthetic 64-byte packet trace and a real backbone one (light and dark bars respectively), in a 60 second experiment

the cache and has a poor performance. The *Write Through Cache* (WTC) policy writes the cache content to disk and, then, a new cache write operation can proceed. Finally, the *Write Back Cache* (WBC) policy does not require that the cache is flushed to hard disk before a new write operation can be performed.

*Disk cache:* Some hard drives feature a cache that performs bundling of write operations to a given sector, thus saving disk head movements and minimizing mechanical operations.

*Filesystem (FS):* We have evaluated the `ext4` filesystem as it is the de facto standard for Linux systems. Additionally, we have tested the `xfs` and `jfs` filesystems, as a previous analysis pointed them as the best option for storing a large number of big files.

The experiments have been carried out by taking all possible combinations of the parameters. For each combination, one hundred 2 GB-sized files have been written using the Linux `dd` tool. This size has been chosen as a tradeoff between write performance and later read accessibility for packet traces.

Interestingly, the results have shown that there is not a general best solution regardless the number of disk. While the best solution for 1 disk was found with the cache policy of WTC and disk's cache disable, and `jfs` filesystem, the configuration for 9 and 10 disks was WBC cache policy, disk's cache on operation and `xfs` as filesystem. In these three cases the optimal strip size was 1 MB.

By paying attention to the average write throughputs, we have found that a single disk has an average throughput of 1.26 Gb/s for its best configuration. Then, a 9-disk RAID has shown an average write throughput of 11.53 Gb/s, and by using 10 disks RAID has achieved as much as 12.60 Gb/s. However all these results are averages, and in the case of 9 disks in several experiments the throughput was below 10 Gb/s whereas using 10 disk all experiments show throughput over such target threshold. To prevent from such excursions below 10 Gb/s write speed, we choose the disk array setup that makes the entire confidence interval for the average lie above 10 Gb/s. To this end, we recommend an array of no less than 10 disks.

#### IV. SNIFFING AND STORING

In the light of all the lessons and results shown in the previous sections, we proposed the HPCAP system [9]. HPCAP is a modular system which is able to capture and store network traffic at wire-speed. HPCAP is composed of an improved driver and a user-level application, named `hpcapdd`, which is an adaptation of the standard `dd` tool to work with our driver. Such driver follows the design concepts presented in Section II to capture traffic, and has been further optimized with some novel features for improving the traffic storage performance, namely:

*Byte-stream oriented:* Packets can be stored and processed as a continuous stream rather than packet by packet, thus reducing the amount of system calls from user space.

*Huge intermediate receiving buffers:* Receiving buffers must be large enough to cope with the variable disk write throughput, in order to guarantee a sustained throughput of 10 Gb/s. The maximum memory that the HPCAP driver can allocate for such buffers is 1 GB, which is the default size. Those buffers will be mapped to user-level memory, improving performance because the captured traffic will not copied from kernel to user space.

*Memory-alignment:* Page-aligned data transfer operations enables the use of the `O_DIRECT` system flag. With `O_DIRECT` the kernel carries out Direct Memory Access (DMA) transfers from system memory to the target device. Consequently, no CPU nor cache management nor memory bandwidth is spent in the data transfers and thus write performance is increased.

*Receive and storage overlapping:* HPCAP instantiates one kernel-level thread which is in charge of polling for new packets and fetching them from the NIC descriptors to an intermediate buffer. A different user-level thread (`hpcapdd`) reads the packets stored in such buffer and writes them into disk. This way capture and storage processes are isolated and their execution is pipelined.

*Affinity planning:* Kernel-level threads and their associated user-level threads are attached to different CPUs but to the

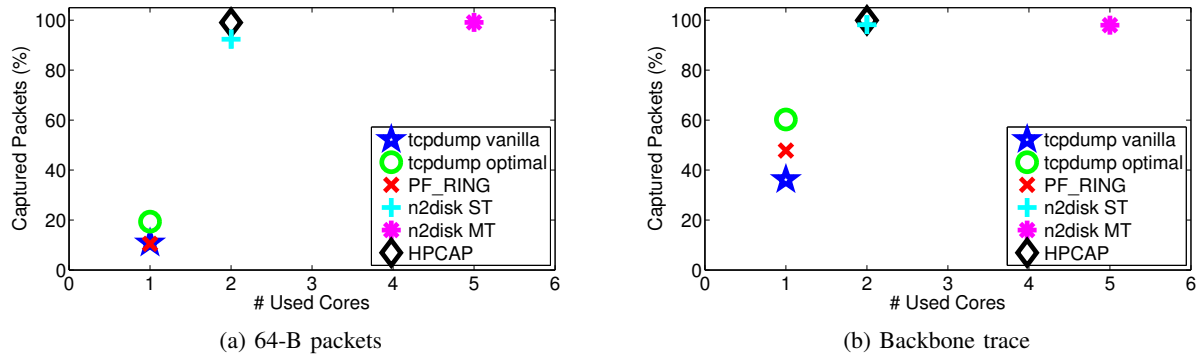


Fig. 2: Percentage of stored packets (into a RAID-0 volume) for a full-saturated 10 Gb/s link versus the number of occupied CPU cores for a 60-second experiment

same NUMA node, and such NUMA node has to be the one associated to the PCIe slot where the NIC has been plugged.

The performance evaluation for the different solutions available in the state of the art for storing traffic is shown in Fig. 2. Specifically, we measure the percentage of incoming traffic stored versus the number of cores used in our testbed, i.e., an optimized 10-disk RAID-0 volume—as discussed in Section III. We note that standard solutions such as `tcpdump` are not capable to store all the incoming traffic, neither high-performance sniffing solutions, such as `PF_RING`. Remarkably, due to the afore-mentioned features, `HPCAP` is capable of storing all of the incoming full-rate traffic for both synthetic 64-byte (CRC included) and real-traffic experiments. This is achieved at the expense of using two cores instead of only one, but `HPCAP` is still the solution capable of storing incoming traffic in worst-case scenarios with minimum number of cores used. Interestingly, the authors of `PF_RING` have latterly implemented `n2disk` [10], a user application that follows most of the concepts reviewed along this article to store traffic. We have evaluated its performance, resulting in similar multi-gigabit rates to those `HPCAP` had achieved. This certainly proves that beyond the particular way of coding the concepts explained in this article, the combination of commodity hardware and open-source software suffices for high-performance network tasks. Results for `n2disk` are shown in Fig. 2 for the single-thread (ST) and multi-thread (MT) versions of this tool following the instructions given by their developers. As `HPCAP`, the single-thread version uses one thread for packet sniffing and one more thread for storage whereas the multi-thread version uses one thread for sniffing and four threads for processing and storing.

## V. CONCLUSIONS

This work has evaluated the performance levels provided by a set of network monitoring tools when it comes to capture and store traffic. This evaluation showed that standard tools have proven to be unable to cope with line-rate traffic in 10 Gb/s scenarios, and novel capture engines are needed. While network traffic may be fetched using those novel capture engines, they may not suffice for storing traffic at such rates. The storage process implies optimization both in the storage device and in the network capture system. In this light, we

have released `HPCAP` system, which is an open-source high-performance capture engine that both fetches and stores traffic in 10 Gb/s scenarios thanks to exploit all of the lessons and findings we have learned and explained throughout this paper.

## ACKNOWLEDGEMENTS

This research was carried out with the support of the EU FP7 *OpenLab* project (Grant No. 287581), the Spanish National I+D *Packtrack* project (TEC2012-33754) and Universidad Autónoma de Madrid’s multidisciplinary project *Implementación de Modelos Computacionales Masivamente Paralelos* (CEMU-2013-14).

## REFERENCES

- [1] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle, “Comparing and improving current packet capturing solutions based on commodity hardware,” in *Proceedings of ACM Internet Measurement Conference*, 2010, pp. 206–207.
- [2] J. García-Dorado, F. Mata, J. Ramos, P. Santiago del Río, V. Moreno, and J. Aracil, “High-performance network traffic processing systems using commodity hardware,” in *Data Traffic Monitoring and Analysis*. Springer Berlin Heidelberg, 2013, ch. 1, pp. 3–27.
- [3] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider, “Enriching network security analysis with time travel,” in *Proceedings of ACM SIGCOMM*, 2008, pp. 183–194.
- [4] C. Walsworth, E. Aben, k. claffy, and D. Andersen, “The CAIDA anonymized 2009 Internet traces,” [http://www.caida.org/data/passive/passive\\_2009\\_dataset.xml](http://www.caida.org/data/passive/passive_2009_dataset.xml).
- [5] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: a GPU-accelerated software router,” in *Proceedings of ACM SIGCOMM*, 2010.
- [6] F. Fusco and L. Deri, “High speed network traffic analysis with commodity multi-core systems,” in *Proceedings of ACM Internet Measurement Conference*, 2010, pp. 218–224.
- [7] L. Rizzo, “Revisiting network I/O APIs: the netmap framework,” *Communications of the ACM*, vol. 55, no. 3, pp. 45–51, 2012.
- [8] N. Bonelli, A. Di Pietro, S. Giordano, and G. Prociassi, “On multi-gigabit packet capturing with multi-core commodity hardware,” in *Proceedings of the Passive and Active Network Measurement Conference*, 2012, pp. 64–73.
- [9] V. Moreno, “Development and evaluation of a low-cost scalable architecture for network traffic capture and storage for 10Gbps networks,” Master’s thesis, Universidad Autónoma de Madrid, 2012.
- [10] L. Deri, A. Cardigliano, and F. Fusco, “10 Gbit line rate packet-to-disk using `n2disk`,” in *Proceedings of Traffic Monitoring and Analysis Workshop*, 2013, pp. 441–446.