Multi-threaded Packet Timestamping for End-to-End QoS Evaluation

Péter Orosz and Tamás Skopkó
Faculty of Informatics
University of Debrecen
Debrecen, Hungary
e-mail: {oroszp, skopkot}@unideb.hu

***Abstract*— In this work, we are focusing on the enhancement of end-to-end QoS evaluation by improving the performance and the functionality of packet timestamping. Accordingly, a new software-based multi-layer timestamping method is introduced, which implements a multi-threaded offloading mechanism. Compared to the available generic kernel-time based solutions, it provides not only higher precision but also lower kernel level overhead and thus lower packet loss without using any special hardware component. These improvements make the proposed method a more efficient basis for multi-layer QoS measurement performed on-the-fly on the communication endpoint, which may result in a better QoS-QoE correlation. The efficiency of the solution is validated against the generic, kernel-time based timestamping using their Linux implementations.**

Keywords-Timestamping; Media traffic; QoS evaluation; Next Generation Networking; Computer network management; Network measurement; Linux

I. INTRODUCTION

With the emergence of 1 Gbit/s and faster access networks and with the increasing demand for their packet level monitoring, the dominance of the purely software based network measurement tools, operating on generic desktop PCs, gradually decreased. Considering the high transmission rate on the monitored network link, the resolution and precision of software timestamping methods and the lossy packet procession provided by a generic Network Interface Card (NIC) became a serious bottleneck of traffic analysis. The drawbacks of the software-based packet capturing are already investigated by several papers (see Section II). As primary effect, using low resolution and precision software timestamps leads to an incorrect representation of the packet inter-arrival times, since the generation of these timestamps is performed within a shared resource environment, where the timestamping process, as any other process, is scheduled by the OS scheduler subsystem and competes for CPU time. The common kernel-time based solutions operate with large overhead and high variance, which could be a bottleneck of precise QoS measurement. The final 64-bit Time of Day (ToD) format of the software timestamp is calculated within the kernel space on-the-fly, based upon an arbitrary kernel clock-source (High Precision Event Timer (HPET), Advanced Configuration and Power Interface (ACPI), Timestamp Counter (TSC), etc.), which, by executing several conversion functions upon packet reception, results in a large packet processing overhead. Among other complex packet processing tasks, the timestamp calculation is implemented within the packet reception softIRQ, which, due to its complexity, produces excessive CPU load. The OS scheduling mechanism and the large timestamping overhead provides a very low precision timestamping output, while the intensive CPU usage of softIRQ results in packet loss. All these effects make end-to-end Quality of Service (QoS) evaluation that includes flow level delay, jitter, packet loss, and reordering measurement, very ineffective on high speed communication links. Nevertheless, today's hardware accelerated network monitoring devices are designed to operate on aggregated backbone link, not on an access link belonging to one single endpoint node. In contrast, monitoring QoS level of real-time media services on the communication endpoint itself should be a reasonable option and this is the point where the software-based on-the-fly QoS evaluation comes in. However, QoS evaluation (including timestamping) requires CPU and other resources and therefore it should be performed with low overhead without degrading the performance of the monitored real-time media communication.

We are facing three rudimental problems: low resolution and low precision of the timestamps, and large overhead. The resolution of the software timestamp, as in case of any timestamping mechanism, depends on the resolution of the applied hardware clock source, the length of the data structures that store the generated timestamp value, and the granularity of the clock-to-time conversion. Enhancing the microsecond resolution up to one nanosecond is not a particularly big challenge, since today's x86 and x64 CPUs operate at 1+ GHz and support the constant Time Stamp Counter (TSC) register, which acting as a kernel clock source enables the timestamping subsystem to generate timestamps with 1 ns resolution [1]. The constant rate property assures the register value to be incremented with a fixed rate according to the maximum CPU frequency, independently of the current operational mode.

High resolution time measurement is already supported by the Linux kernel from version 2.6.27 [2]. All we have to do is to provide an unconverted 64-bit path for these high resolution timestamps up to the user space monitoring application. To achieve this goal, we previously enhanced the common libpcap library to natively handle the nanosecond resolution timestamps provided by the kernel [3][4][5]. Unfortunately, the enhancement of the resolution alone does not imply high time precision. The primary bottleneck - the large deviation of the generation overhead - decreases the precision to an unacceptable level. Moreover,

 ISBN: 978-1-61208-305-6

the large CPU overhead of the built-in timestamp generator mechanism could lead to a serious amount of packet loss during the real-time conversation. To overcome this problem, a new timestamping method was designed and implemented, which is based on a multi-threaded offloading approach. The timestamping process is split into two separate phases. The primary goal was to achieve minimal timestamping overhead in kernel context with very low variation (see Section III). Even if we know that it is impossible to provide a precision close to the hardware-based timestamping solutions, a realistic goal was to significantly exceed the precision level of the existing software-based solutions available on generic multi-core architectures. A hardware accelerated approach of multi-layer timestamping is already presented in our previous paper [6]. However, in this proposal, we introduce a purely software based timestamping solution with low overhead and low variance, which enables to apply the method for multi-layer timestamping on generic PCs without any hardware acceleration.

Another critical aspect of packet processing is the packet loss ratio caused by the capturing itself. Beyond the benefits related to timestamp precision, the proposed timestamping method significantly decreases the packet loss compared to the large overhead kernel-time (ktime) based timestamping mechanisms. Active QoS monitoring of real-time media applications can therefore benefit from this method.

The rest of the paper is organized as follows. Section II gives a summary of related works in the field of high precision software-based packet timestamping. Theoretical background of our multi-threaded timestamping method is described in Section III and its Linux based implementation is presented in Section IV. We evaluated the performance of the introduced method using comparative laboratory measurements in Section V. Finally, Section VI concludes the presented work.

II. RELATED WORKS

In the last decade, several research projects focused on the challenges of high performance network monitoring, especially triggered by the emergence of the 1+ Gbps networking technologies [7][8]. While most of them are hardware accelerated solutions, some projects investigated the possible performance enhancement of the software-based network monitoring suites. Coppens et al. introduced a new scalable network monitoring platform called SCAMPI [9], which supports dedicated hardware accelerated monitoring boards as well as generic NICs for packet capturing. Heyde et al. investigated the loss property of the Intel NIC-based packet capturing in the context of Lawful interception [10]. Pásztor et al. presented a high resolution, low overhead timestamping method based on the CPU's TSC register [11]. Their timestamping proposal includes an offloading mechanism based on a post-processing phase. In our work, we defined two parallel goals: improving the performance of the offloading method proposed by Pásztor et al., and also decreasing the packet loss ratio during the capture process. The TSC clock source has high resolution as well as high precision, as already investigated in [11]. However, the software timestamping methods, relying on the TSC register as clock source, have a very limited overall precision due to the large generation overhead and the OS scheduling (within the shared resource environment). These bottlenecks do not enable these methods to provide adequate precision for high speed QoS evaluation.

III. THEORETICAL BACKGROUND

Our proposed timestamping mechanism's primary benefit is its ultra-low timestamp generation overhead and the low variance of this overhead on most of the generic purpose multi-core system. The applied clock source is the TSC clock cycle register, which is read by a custom, high priority kernel process at packet arrival, then, the obtained value is passed to the higher level packet processing application, which offloads the clock-to-time conversion. The original method proposed by Pásztor et al. performed the conversion in a separated post-processing phase and did focus neither on enhancing the packet processing performance nor its multi-layer application. In contrast, our overhead reduction and precision improvement is gained by the combination of following two ideas.

A. Decreasing timestamping overhead and packet loss ratio within the kernel space

The packet processing softIRQ, which implements kernel-level timestamping functionality, should be statically assigned to an otherwise idle CPU core by directly altering its core affinity. Instead of providing the final timestamp format, timestamping within the softIRQ context should include the acquisition of the TSC value only, which requires not more than 24 clock cycles to be performed. Then, this 64-bit clock cycle value, which represents the moment when a packet reaches the kernel's network stack, is preserved with the packet within its path up to the conversion thread. For further improvement of timestamp precision, all interrupts are disabled on the assigned CPU core during the execution of the timestamping code.

B. Offloading timestamp conversion to the user space

The cycle-to-time conversion should be done on-the-fly by a dedicated user space thread of the multi-thread capturing process. After conversion, the nanosecond resolution timestamps, which is now in time of day format, should be sent back to the main packet processing thread. The dedicated timestamp conversion thread is also running on an otherwise idle CPU core and therefore it provides a very low conversion overhead (Fig. 1).

The combination of these two ideas does not just reduce timestamping overhead and improve the precision, but significantly decreases the kernel-level packet loss ratio at high arrival rates: by offloading the cycle-to-time conversion, the low level processing of each incoming packet requires lower CPU resource and thus, more packets can be accepted by the kernel networking subsystem within the same time interval without resource exhaustion. To maintain the low loss ratio up to the capture application, large packet buffers should be applied at the higher levels of the kernel space. Our comparative measurements validate the

 ISBN: 978-1-61208-305-6

performance parameters of the proposed method using its Linux based implementation (see Section V).

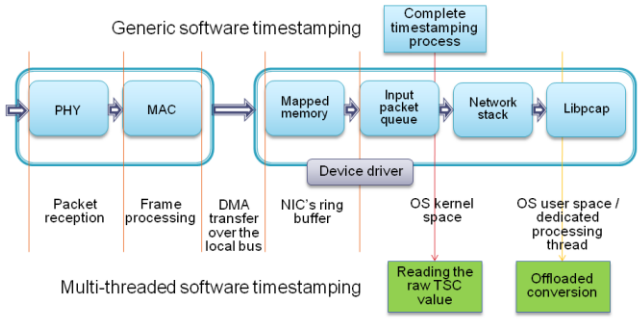

Figure 1. The new timestamping method produces a raw timestamp value in kernel space with very low overhead and implements the offloading of the cycle-to-time conversion by a dedicated processing thread in user space.

In network measurement, the precision of timestamping is a criterion more important than the low clock offset, especially for measuring packet inter-arrival times and round-trip delays at one single point of the network (e.g., active probing). On a generic purpose PC without any additional hardware, the TSC clock cycle register is the optimal choice for high precision packet timestamping, since it has a very low rate error (< 0.3 ppm) as presented in [11]. The multi-core architecture and the new multi-threaded offloading method together can provide enough processing capacity to fulfill the requirements of high precision packet timestamping.

The proposal of Pásztor et al. is based upon Fast Ethernet measurements. Today, Gigabit Ethernet is widely used for connecting endpoints to the network. Our multi-threaded timestampig solution was therefore validated using intensive traffic patterns on 1 Gbps network links. Intensive traffic on a Gigabit Ethernet link may involve high packet rate with inter-arrival times below 1 μs. The timestamping mechanism must provide enough resolution and precision to realistically interpret packet arrivals even at high rate. The resolution and precision capabilities of the software-based timestamping are determined by the generation overhead and its variance. Accordingly, the desirable overhead of packet timestamping should be well below the 1 μs order to represent the inter-arrival times.

On a single core system, even with our multi-threaded offloading method, it is hard to demonstrate such a low timestamp generation time since it is not possible for the packet processing softIRQ to be assigned to an idle CPU core. Accordingly, we assume, that a generic multi-core CPU with constant TSC register is available for the measurement task.

IV. Implementation of the Proposed Method

We decided to implement our method under Linux since it has sophisticated interrupt handling and scheduling mechanisms. As a first step, we modified the kernel's packet timestamping code by replacing the complex and relatively time-consuming calls by a simple RDTSC instruction. In the unmodified kernel, timestamps are generated by calling ktime_get_real() that queries the system's clock source and calculates a wall-clock time. RDTSC is an x86 CPU instruction that reads the CPU's TSC register. When compared to the built-in timestamping method, executing RDTSC takes shorter time, just about 24 clock cycles on a decent Intel CPU. Since the generated values are not represented in wall-clock format, they need to be post-processed later.

As a further optimization, we locked the packet processing softIRQ to a specific CPU-core. It ensures that no other interrups are intefering with timestamping and the packet processing functions. Multiple CPU cores don't necessarily keep their TSC counters in sync. Locking the softIRQ to a specific core also ensures that the read TSC values are acquired from only one CPU. The cycle to ToD conversion should be done in the user-space, utilizing another core. The off-line processing is elaborate when doing long-term capturing. Therefore, we modified the libpcap library to do the conversion at packet receiption. The capture application should query the CPU frequency and report it to libpcap to be able to the conversion. We locked the dumpcap application to another core. By using a multi-threaded design, libpcap could utilize more than one core for converting the timestamps.

V. Evaluation of the Timestamping Performance

Comparative measurements and statistical analysis of the measurement output data are used to validate the performance parameters of the new multi-threaded offloading method for software timestamping. The investigated parameters for both methods (the kernel-time based and the multi-threaded one) involve the per-packet overhead of the timestamping process, the variance of the overhead for the processed packets and the overall packet loss ratio within the system. A generic purpose PC with Intel Core i7 K-2600 2.93 GHz CPU and Linux non-preemptive kernel 2.6.39.2 was set up for all of our measurements. The used NIC was the common and well documented Intel 1000/PRO PT PCI Express card with the e1000e Linux device driver version 1.3.10-k2. This driver implements the New API (NAPI) operation mode introduced by the Linux kernel from version 2.4, which determines the low-level packet processing mechanism of the system [12]. All of the network stack related kernel buffer and driver parameters were optimized for intensive incoming traffic. With the following comparative measurement session, the characteristics of the timestamping overhead and its variation were investigated. In order to accurately measure the overhead of the timestamping mechanism, we modified the original Linux kernel. Two extra timestamping checkpoints had been inserted into the packet processing path, one just before the execution of the packet timestamping function __net_timestamp(), and another straight after its return point. These two timestamps are generated by the rdtscll() function call, which has a very low overhead, since it does not perform any conversion. The delta value minus the checkpoint generation overhead defines the per-packet timestamp generation overhead. This delta value is converted

 ISBN: 978-1-61208-305-6

to a real time value in a later evaluation phase. The built-in (kernel-time based) timestamping code performs real time cycle-to-time conversion and therefore, it produces an overhead with high variance, which results in low precision and also implies significant packet loss. Offloading this conversion task reduces timestamp generation time, which improves precision and reduces packet loss as well.

A. *Investigating precision: line-rate homogenous traffic pattern*

The first measurement scenario enables us to investigate the precision property of both timestamping methods. In this setup, we applied homogenous traffic patterns including fixed packet size and fixed inter-frame gap (IFG). This measurement type requires a high precision traffic generation device, which produces the line-rate packet stream in hardware. For this purpose, we applied a dedicated FPGA packet generator with Gigabit Ethernet interface. Since the generator and the measurement PC are directly connected, the packet inter-arrivals as seen on wire level are determined by the transmission rate of this device and therefore, it can be considered constant. The measurement PC, which was a general purpose desktop computer, performed software based packet capturing with a modified libpcap library supporting both timestamping methods: the generic ktime-based and the multi-threaded one. In both cases, the timestamping code is executed at a well defined point of the packet processing path. For non-NAPI supported device drivers, the software timestamping code is executed within interrupt context by the top half interrupt handler, when the packet is en-queued into the input packet queue of the operating system, while with NAPI support (it is our case), the timestamping is performed by the bottom half handler (softIRQ) [14]. In this latter configuration, the arrival time represented by a software timestamp indicates the moment when the packet is de-queued from the input packet queue, and therefore is affected by the scheduling mechanism of the Linux kernel and the intensity of the interrupts per second on the CPU core that the softIRQ is running on. According to the new method, interrupt handling was disabled during the execution of the kernel level timestamping code with the local_irq_disable() kernel function. This involves that no interrupt is generated for the affected CPU core until the interrupts are re-enabled. Since the execution overhead of the ktime-based method is higher than the multi-threaded one, the probability of an interrupt event (executing a top half handler) to happen during its execution is also higher. The execution of the top half handler puts the CPU in interrupt context, which causes jitter in the execution of the timestamping process.

The measurement (see Fig. 2 and Fig. 3) is performed with an artificially generated traffic pattern (pattern #1) that complies with one direction of a single VoIP conversation including 140 bytes packets transmitted by the traffic generator device in each 20 ms period, which is equivalent to 2,499,860 bytes of IFG. This is a typical packet size and transmission intensity provided by a VoIP audio codec, i.e., G.729.

The second measurement is also done by a synthesized traffic pattern of a HD video stream (see Fig. 4 and Fig. 5). Pattern #2 includes 1,368 bytes packets with a fixed 169,631 bytes of IFG value.

Since these traffics contain homogenous packet sequences, they are suitable to measure the timestamping methods in terms of the variance of the measured inter-arrivals, and the order and the variance of the generation overheads.

The synthesized VoIP traffic, due to its light packet-arrival intensity of 20 ms, does not imply high system load. However, during this measurement, the timestamping overhead provided by the generic ktime-based timestamping presents an average of 195 ns with a large deviation. In contrast, the multi-threaded timestamping method, by applying an effective timestamp acquisition method and offloading several conversion tasks to the user space, results in a low overhead, the half of that of the ktime-based one (Fig. 2).

TABLE I. TIMESTAMPING OVERHEAD FOR PATTERN #1

Timestamping method[a]	Overhead average [ns]	Overhead variance [ns]
Generic ktime	195.2762	176.0954
Multi-thread TS	73.49326	3.793385

a. Homogenous traffic of 140-byte packets with 2,499,860 bytes of IFG

Moreover, the variance provided by the overhead values is also significantly lower (Table 1 and 2). Considering the measurement results, we can find out that both the effective resolution and the precision of the timestamps are significantly improved with the new timestamping method. Fig. 2 represents some extreme high values in the inter-arrival times measured by the ktime-based method (green bars), while these large values are not present in the result of the multi-threaded measurement. The smaller deviation of the multi-threaded solution implies higher timestamping precision.

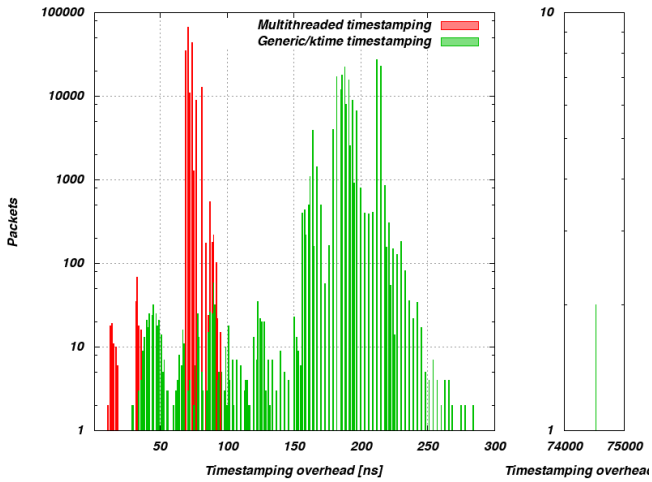

Figure 2. Density points of the timestamp generation overheads for the generic method and the multi-threaded one.

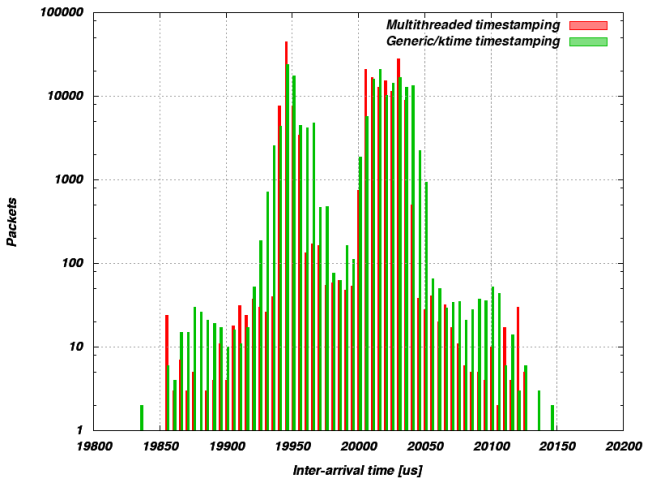

Figure 3. Density points of the packet inter-arrival times

We observed that the large inter-arrival values are in correlation with the large timestamping overhead values. Accordingly, we can assume that the large variance of the generation overhead drives to a very low precision representation of the packet inter-arrival times and could lead to false measurement results. Since there is a high correlation between the generation overhead and the measured arrival time, the low variance of the overhead is the key factor to get high timestamping precision. Whereas the overhead values of the generated traffic fall in a relatively small range, small histogram bins should be chosen.

This applies to the inter-arrival time graph, too. The histogram bars are spaced loosely and so, to make both measurement results represent on the same histogram.

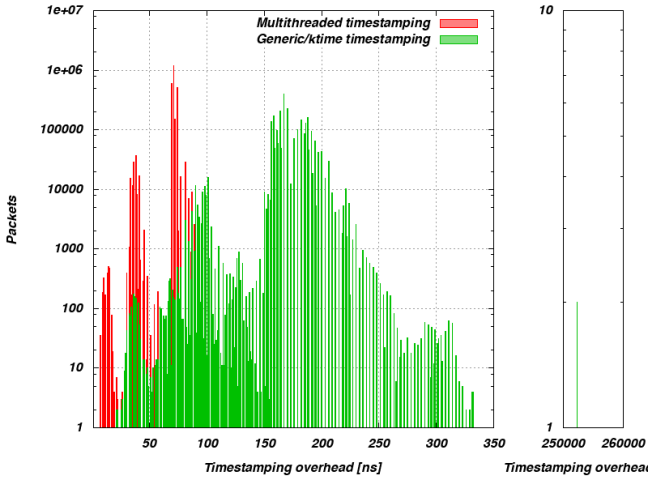

Figure 4. Density points of the timestamp generation overheads for the generic method and the multithreaded one.

Later in this section, we will investigate the root cause of the large variance represented by the generic ktime-based method.

TABLE II. TIMESTAMPING OVERHEAD FOR PATTERN #2

Timestamping method[a]	Overhead average [ns]	Overhead variance [ns]
Generic ktime	173.3606	177.1197
Multithread TS	70.84912	8.197374

a. Homogenous traffic of 1368-byte packets with 169,631 bytes of IFG

Nevertheless, the new multi-threaded timestamping method introduced in this paper, is a dedicated software-based method, that improves the resolution as well as the precision of the traffic measurement. Besides its primary benefits, its low generation overhead has also a positive side-effect: since it is less CPU intensive than the ktime-based one, capturing intensive network traffic will result in lower packet loss rate on any generic PC (see Fig. 6a and Fig. 6b). The auto-correlation analysis of the overhead series showed that the system with the new method has a high memory. Accordingly, based on the current overhead value, future overhead values can be predicted with a higher probability, which implies higher system stability and improved timestamp precision. If the intensity of the packet arrivals is higher, as with the second measurement, the difference in the measured properties becomes more obvious (see Fig. 4 and Fig. 5).

B. Investigating packet loss ratio within the system

We also investigated the packet loss rate during the capturing process in case of both methods. Extreme low overhead values (< 25 ns) derive from the instruction caching mechanism of the applied CPU architecture (Fig. 4). Each measurement round was performed with fixed packet size and fixed IFG combination. 20,000 packets were transmitted each time by the hardware accelerated traffic generator presented earlier, which was directly connected to the measurement PC. The generated packets were captured by the dumpcap application. According to the results presented in Fig. 6a and Fig. 6b, the loss ratio with the multi-threaded timestamping method is improved. Since this method uses a low overhead access to the TSC register and offloads all of the conversion tasks from the kernel space to a dedicated user thread.

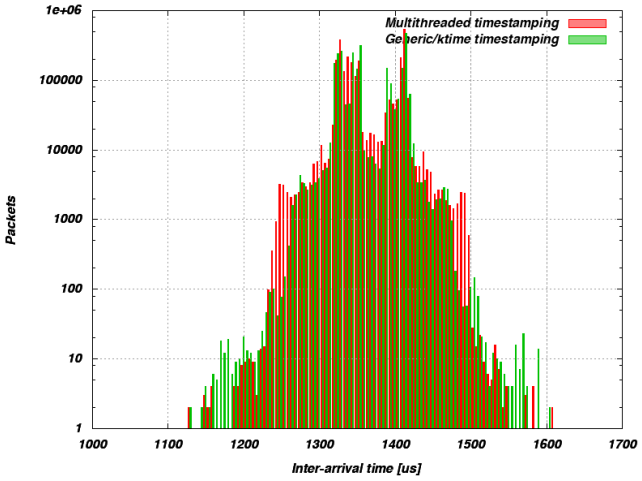

Figure 5. Density points of the packet inter-arrival times

This free time becomes available (within the kernel space) to the kernel scheduler to give it away to another CPU intensive softIRQ tasks that are parts of the packet processing subsystem.

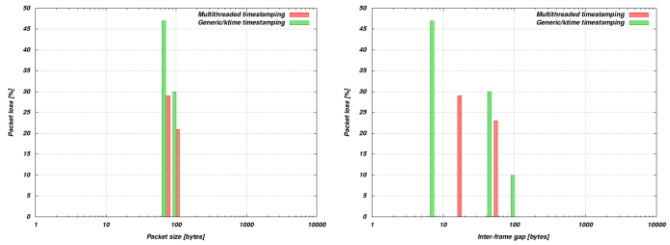

Figure 6. a) Packet loss ratio in function of the packet size b) Packet loss ratio in function of the inter-frame gap

Moreover, the execution of the conversion code in user space is more tolerant to delay. These benefits directly affect the packet processing performance and thus the loss ratio. When the CPU core, dedicated for the execution of the packet processing softIRQ, gets full load for a relatively long time period then the NIC's private queue reaches its maximum length and packet loss will happen.

VI. CONCLUSION

A multi-threaded timestamping method for end-to-end QoS evaluation has been introduced, which provides low kernel context overhead by eliminating the built in clock source API inside the kernel and offloading the conversion tasks to a dedicated processing thread in the user space. In contrast to the solution of Pásztor et al., this method implements a multi-threaded design that supports on-the-fly timestamp conversion and multi-layer application. The main goal of the design was to minimize kernel context timestamping overhead during packet capturing in order to improve timestamping precision and increase effective resolution. The performance properties of the new solution have been evaluated and compared against the generic kernel time based timestamping mechanism available in the Linux kernel. We showed that the overhead of our new method is half of the generic one enabling multi-layer timestamping and the variation of the overhead that determines the precision property of the timestamping is also significantly improved. Over the main benefits, this solution has a positive side effect: the packet loss rate, while capturing high intensity network traffic, is decreased, since the new method requires less CPU resource in kernel context, furthermore the execution of the offloaded timestamp conversion tasks can be delayed due to the scheduling policy of the user space. Though our method was implemented on Linux system, it is not necessarily limited to this OS and the x86-based processors. It can be adapted to environments built on multi-core processors with counting registers (linear increment) and kernels capable of binding packet processing threads to specific cores (affinity).

ACKNOWLEDGMENT

The publication was supported by the TAMOP-4.2.2.C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

This research was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4.A/2-11-1-2012-0001 'National Excellence Program'.

REFERENCES

[1] TSC, Intel64 and IA-32 Architectures SoftwareDeveloper's Manual. Online. Available from: http://download.intel.com/products/processor/manual/325462.pdf, [retrieved: august, 2013]

[2] Linux kernel source: Linux/arch/x86/include/asm/timer.h. Online. Available from: http://www.kernel.org/, [retrieved: august, 2013]

[3] P. Orosz and T. Skopko, "Software-based Packet Capturing with High Precision Timestamping for Linux," 5th International Conference on Systems and Networks Communications, August 22-27, 2010, Nice, France, Proceeding pp. 381-386.

[4] P. Orosz, T. Skopko, and J. Imrek, "Performance Evaluation of the Nanosecond Resolution Timestamping Feature of the Enhanced Libpcap," 6th International Conference on Systems and Networks Communications, ICSNC 2011, October 23-28, 2011, Barcelona, Spain, ISBN 978-1-61208-166-3, Proceeding pp. 220-225.

[5] P. Orosz and T. Skopko, "Performance Evaluation of a High Precision Software-based Timestamping Solution for Network Monitoring," the International Journal on Advances in Software, ISSN 1942-2628, 2011 Vol 4. No. 1 & 2 pp. 181-188.

[6] P. Orosz, T. Skopko, and J. Imrek, "A NetFPGA-based Network Monitoring System with Multi-layer Timestamping: Rnetprobe," NETWORKS 2012, 15th International Telecommunications Network Strategy and Planning Symposium, October 15-18, 2012, Rome, Italy, Proceeding pp. 1-6.

[7] J. Micheel, S. Donnelly, and I. Graham, "Precision timestamping of network packets," 1st ACM SIGCOMM Workshop on Internet Measurement, November 1-2, 2001, San Francisco, California, USA, Proceeding pp. 273-277.

[8] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," 1st ACM SIGCOMM Workshop on Internet Measurement, November 1-2, 2001, San Francisco, California, USA, Proceeding pp. 267-271.

[9] J. Coppens, E.P. Markatos, J. Novotny, M. Polychronakis, V. Smotlacha, and S. Ubik, "SCAMPI - A Scaleable Monitoring Platform for the Internet," 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004), Budapest, Hungary, 22-23 March 2004

[10] A.A. Heyde, "Investigating the performance of Endace DAG monitoring hardware and Intel NICs in the context of Lawful Interception," CAIA Technical Report 080222A, august 2008.

[11] A. Pásztor and D. Veitch, "PC Based Precision Timing Without GPS," ACM SIGMETRICS 2002, Proceeding pp. 1-10.

[12] Linux NAPI device driver packet processing framework. Online. Available from: http://www.linuxfoundation.org/collaborate/workgroups/networking/napi, [retrieved: august, 2013]

 ISBN: 978-1-61208-305-6