Performance Evaluation of a High Precision Software-based Timestamping Solution for Network Monitoring

Peter Orosz, Tamas Skopko
Faculty of Informatics
University of Debrecen
Debrecen, Hungary
e-mail: oroszp@unideb.hu, skopkot@unideb.hu

***Abstract* — Widely used network measurement applications, such as tcpdump and Wireshark, use the same common libpcap packet capture library. Libpcap assigns a 10^{-6} second precision timestamp to all processed frames. Higher physical bandwidth implies shorter inter-arrival times between consecutive frames. Accordingly timestamp resolution must be proportional to the link speed. The latest version 1.1.x of libpcap provides 10^{-6} second native resolution, however pcap format supports a larger 2 x 32-bit timestamp value for each stored packet. On Gigabit Ethernet or faster networks, a timestamp resolution that works in the microsecond domain may not enable us to precisely reproduce the time-domain relation between consecutive frames. Therefore overall analysis of the data transmission could lead to a false result. For packet capturing with libpcap, it is assumed that the timestamp represents the time moment when a frame reaches the kernel's input packet queue. In an idealized case generated timestamps are always converging and suitably close to the real arrival or transmission time of each frame so as to conserve the original inter-arrival time values. The timestamp resolution of network measurement applications must be increased according to the requirements of advanced high speed data networks. In our paper, we are going to show and evaluate an alternative libpcap-based solution that features nanosecond precision timestamping.**

Keywords-libpcap; timestamp resolution; inter-arrival time; Linux kernel; high speed network.

I. INTRODUCTION

This paper is the extended version of our previous work [1] that focuses on a software solution based on the libpcap packet capture library and high resolution kernel-based timestamp generation. On Linux machines the libpcap library retrieves timestamps of captured frames from the kernel through some special kernel functions. Independently from one other, several impact factors could directly bias the generation of timestamps [2].

Some timestamp-related terms that will be used in the rest of this paper should be introduced here:

- Timestamp size *(TSS)*: bit length of the timestamp
- Timestamp precision *(TSP)*: sub-second resolution
- Timestamping time *(TST)*: time required to generate a timestamp value

High resolution timestamping of data packets on high speed networks is a challenging issue [3], which is even more critical on a software-based packet capture environment such as libpcap [4]. Libpcap relies on the operating system kernel to provide the arrival or transmission time moment of the processed data packets. Since timestamping is performed in the kernel space, several hardware and software factors impact the overall precision and accuracy of the generated timestamps. Furthermore, data structures in libpcap are designed for 32-bit TSS.

The precision requirement of TSP depends on:

- Link speed
- The minimum of packet inter-arrival times within a data stream

The following factors affect TST:

- Hardware architecture
- NIC driver design
- OS kernel (enqueing/dequeuing, handlers)
- Clock sources
- Libpcap

Let us suppose that two uniform sequences of minimum-sized and maximum-sized Ethernet frames are transmitted over Gigabit and Ten Gigabit Ethernet links at the theoretical maximum rate. Table 1 and Table 2 show the PHY (physical layer) level timing parameters of the Gigabit and Ten Gigabit Ethernet standards.

TABLE I. GIGABIT ETHERNET TIME PARAMETERS

Timing parameters 1 GbE	**Smallest Ethernet frame length: 72 Bytes**	**Largest Ethernet frame length: 1526 Bytes**
Bit time	1 ns	1 ns
Inter-frame gap	96 ns = 96 x bit time	96 ns = 96 x bit time
Δt between timestamps of two consecutive frames	576 ns + 96 ns = **672 ns**	12,208ns + 96ns = 12,304ns
Theoretical precision of NTP sync	≥1 msec	≥1 msec
Required time sync precision	≤600 ns (theoretical minimum)	≤12 µs (theoretical maximum)
Maximum number of frames per second	1,488,096	81,274

TABLE II. TEN GIGABIT ETHERNET TIME PARAMETERS

Timing parameters 10 GbE	Smallest Ethernet frame length: 72 Bytes	Largest Ethernet frame length: 1526 Bytes
Bit time	.1 ns	.1 ns
Inter-frame gap	9.6 ns = 96 x bit time	9.6 ns = 96 x bit time
Δt between timestamps of two consecutive frames	57.6 ns + 9.6 ns = **67.2 ns**	1,220.8 ns + 9.6 ns = 1,230 ns
Theoretical precision of NTP sync	≥1 msec	≥1 msec
Required time sync precision	≤60.0 ns (theoretical minimum)	≤1.2 µs (theoretical maximum)
Maximum number of frames per second	14,880,960	812,740

We assume that the indicated frame sizes include 8 bytes preamble, 6 bytes destination MAC address, 6 bytes Source MAC address, 2 bytes MAC Type/length, 4 bytes CRC and the payload (46-1500 bytes). The inter-frame gap is 12 bytes according to the Ethernet specification. Based on these values the minimum of packet inter-arrival time can be determined that is 672 ns for Gigabit Ethernet and 67.2 ns for Ten Gigabit Ethernet respectively.

IP Packet Delay Variation (IPDV) is an IETF RFC 3393 proposal [5][6]:

$$d_{T2} - d_{T1} = d_{\Delta T} \quad (1)$$

where
d_{T1}... delay of a packet sent at time T_1
d_{T2} ... delay of a packet sent at time T_2
$d_{\Delta T}$ ... type - P - one - way - ipdv from source to destinatio n

Delay per hop:

$$d_H = d_t + d_p + d_q \quad (2)$$

where
d_H ... delay per hop
d_t ... transmiss ion delay
d_p ... processing delay
d_q ... queuing delay

End-to-end delay:

$$d_T = \sum_{i=1}^{n} d_{H_i} \quad (3)$$

where
d_T ...end - to - end delay
d_H ...delay per hop
n... number of hops

We assume that (3) has a Gamma distribution function [7]:

$$f(x;k;\theta) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k \Gamma(k)} \quad (4)$$

where $x, k, \theta > 0$

A 64-Byte packet sequence has been generated according to (4) at a 1 Gbps transmission rate (Fig. 1).

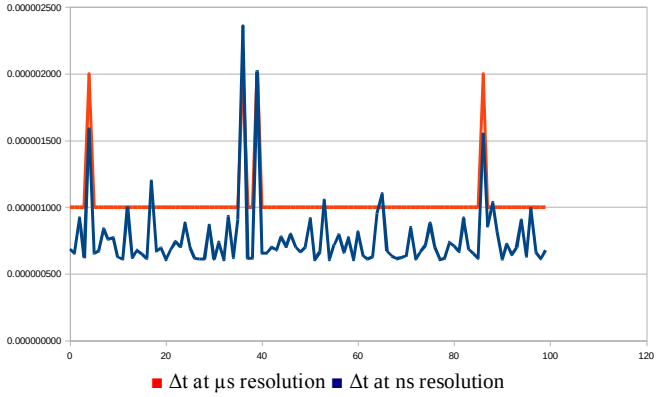

Figure 1. Gamma distributed PDV of 64-Byte frames at 1Gbps transmission rate

It can easily be shown that microsecond time resolution could be insufficient to describe the time domain relation (1)(2) between packet arrivals on Gbit/s or a higher speed network path [8].

II. RELATED WORK

In the last couple of years several research projects have realized the problem of the inefficient time resolution of packet timestamps [9][10][11]; most of their proposals and solutions resulted in hardware-based packet timestamping. For higher performance some of them integrated the entire capturing process into a dedicated hardware device [9][11][12][13]. However, none of them was focused on extending the resolution of software-based packet timestamping.

III. PROBLEM DEFINITION

A. NIC driver architecture

The NIC driver connects the physical layer and the internal packet structures of the operating system. A sophisticated network driver design combines interrupt and polling operation modes using the kernel feature NAPI (New API) [14]: at lower traffic it uses interrupts, while at higher loads it switches to polling mode [15].

Interrupt mode: When a frame arrives at the NIC, it generates an interrupt that calls a specific handler registered by the driver. The handler places the frame into the input packet queue and the kernel processes it thereafter. The handler is given priority over the kernel's processing code as long as frames are arriving at a higher rate (due to a high network load) than the kernel can handle them. High traffic results in a high number of interrupts that could consume hardware resources.

Some NIC drivers can support the passing of multiple frames within an interrupt.

- Polling mode: The kernel queries the driver about the arrival of new frames with a specific frequency. Resource consumption of this method is optimal at high network load.
- Timer-driven interrupts: The NIC asynchronously notifies the kernel about frame reception. The handler processes the frame, which has arrived since the last interrupt.

Since timestamping of the incoming packets is performed by the queue handler, the timestamp value does not necessarily depend on the operation mode of the NIC driver. Nevertheless, in order to determine the dependency of the timestamp value on the operation mode further analysis is required.

B. The OS kernel

We must define the exact code point over the data path from the NIC to the kernel input queue where timestamping is performed. The Linux kernel puts timestamps onto each frame when they are enqueued to the input packet queue. This is the point where the kernel processes the frame (Fig. 1).

Since libpcap relies on kernel timestamps, we had to observe the latest Linux kernel functions, which could acquire 10^{-9} TSP from a high frequency and very accurate clock source.

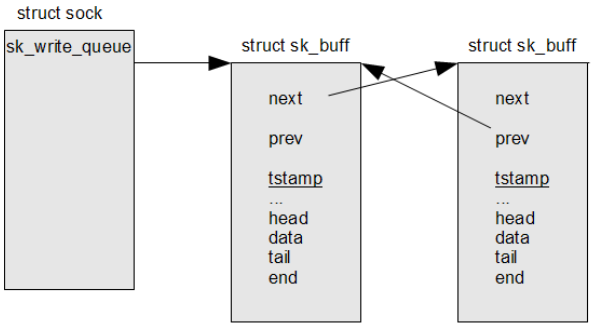

Figure 2. Sk_buff structure of the linux kernel

C. Clock sources

The Linux kernel supports multiple clock sources. Their availability depends on the underlying hardware architecture.

1) ACPI (Advanced Configuration and Power Interface) Power Management Timer

This clock, known as the RTC *(Real Time Clock)*, is usually integrated into the south bridge of the motherboard. Its 3.579545 MHz clock frequency limits its precision to the microsecond domain.

2) HPET (High Precision Event Timer)

This is available on most of today's PC architectures. HPET is a high precision clock source due to its very low jitter, which is within the nanosecond domain. However its clock frequency is about 10 MHz, which is not an appropriate for high resolution timestamping.

3) Jiffies

These are based on timer interrupt and are referred to as the kernel heartbeats. Jiffy frequency can be specified at compile time. Under recent 2.6.x Linux kernels it is set to 1/250 Hz (4 ms resolution) or its maximum 1/1000 Hz (1 ms) by default, which is far from the requirements of proper timestamping. There are plans to remove this timing method and move to tickless systems because of power saving considerations.

4) TSC (Time Stamp Counter)

A 64-bit CPU register that is present on all x86 processors since the Intel Pentium. It counts the number of ticks since boot or reset. The time stamp counter is an excellent high-resolution, low-overhead way of providing timestamps. The novel constant TSC feature ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. "This is the architectural behaviour moving forward for all Intel processors." [16]

Constant TSC operates at the CPU's clock speed from which the 10^{-9} second TSP can be easily derived.

D. Libpcap

The last stage of transmission just before getting to the capture application is the libpcap. Timestamp information received by the libpcap depends on the factors discussed in the previous sub-sections. The Linux-specific part of the libpcap is contained in the pcap-linux.c source file. The library captures the packets with the *pcap_read_packet()* function. Timestamping is handled either by the SIOCGSTAMP IOCTL call or by the TPACKETv2 structure.

IV. Implementation of High Resolution Timestamping

Our goal was to reveal and test all of the kernel functions and features that will be essential parts of our project to modify libpcap to a nanosecond-capable capture library. In this section, related source code snippets are presented in such a way that the beginning of deleted and inserted source code lines are marked with the '-' and '+' signs respectively.

A. Implementation

It is feasible to reach nanosecond TSP resolution purely on software-based tapping:

- The *tstamp* member of *sk_buff* structure is capable of nanosecond resolution
- The Linux kernel function *ktime_get_real()* to query the system clock is in nanosecond resolution
- This function is adequate to fill up nanosecond *tstamp* fields in *sk_buff*
- Accordingly user-space applications (such as libpcap-based ones) could display and process 10^{-9} second resolution timestamps

- The Linux kernel supports TSC as a clock source
- For efficient time synchronization dedicated LAN interfaces and a PTP timing protocol could be used within a low latency wired environment

The input queue handler within the 2.6 kernel puts a 64-bit timestamp onto each frame that is enqueued to the input_packet_queue. The Linux kernel API features the *ktime_get_real()* function that enables us to query nanosecond resolution timestamps from the kernel.

For nanosecond time resolution we assume that the kernel's clock source is a constant TSC that operates at ≥1GHz frequency.

The latest Linux kernels (v2.6.27+) introduce the SIOCGSTAMPNS call that returns with the nanosecond precision timestamp of the last incoming packet.

Through this IOCTL call, we indirectly get to the *sock_get_timestampns()* function inside kernel's sock.c. This function relies on the *ktime_get_real()* for timestamp generation and uses the *ktime_to_timespec()* to convert it to *tv_nsec* format, which is a nanosecond capable time variable within the *timespec* data structure.

```
include/linux/sk_buff.h:

struct sk_buff {
/* These two members must be first. */
struct sk_buff        *next;
struct sk_buff        *prev;
struct sock           *sk;
ktime_t               tstamp;
struct net_device     *dev;
}


include/linux/ktime.h:

ktime_t
union ktime {
      s64     tv64;
#if BITS_PER_LONG != 64 &&
!defined(CONFIG_KTIME_SCALAR)
      struct {
# ifdef __BIG_ENDIAN
      s32     sec, nsec;
# else
      s32     nsec, sec;
# endif
      } tv;
#endif
};

typedef union ktime ktime_t;
```

Our first modification is replacing the SIOCGSTAMP IOCTL call with the more recent SIOCGSTAMPNS one:

```
pcap-linux.c, in function pcap_read_packet():

-       if (ioctl(handle->fd, SIOCGSTAMP,
&pcap_header.ts) == -1) {
-             snprintf(handle->errbuf,
PCAP_ERRBUF_SIZE, "SIOCGSTAMP: %s",
pcap_strerror(errno));
+       if (ioctl(handle->fd, SIOCGSTAMPNS,
&pcap_header.ts) == -1) {
+             snprintf(handle->errbuf,
PCAP_ERRBUF_SIZE, "SIOCGSTAMPNS:
%s", pcap_strerror(errno));
              return -1;
       }
```

Libpcap alternatively uses the *tpacket_hdr* structure to query packet description header information.

```
struct tpacket_hdr
{
      unsigned long   tp_status;
      unsigned int    tp_len;
      unsigned int    tp_snaplen;
      unsigned short  tp_mac;
      unsigned short  tp_net;
      unsigned int    tp_sec;
      unsigned int    tp_usec;
};
```

We would like to emphasize the limitation of this structure: content of the *tp_usec* element is always a microsecond precision sub-second time value (TSP). Latest linux kernels (2.6.27+) now feature the enhanced *tpacket_v2* structure:

```
struct tpacket2_hdr
{
      __u32         tp_status;
      __u32         tp_len;
      __u32         tp_snaplen;
      __u16         tp_mac;
      __u16         tp_net;
      __u32         tp_sec;
      __u32         tp_nsec;
      __u16         tp_vlan_tci;
};
```

The novel *tpacket_v2* is able to store nanosecond precision TSP as well as some VLAN information. We managed to maintain and adapt the benefits of *tpacket_v2* structure within the packet capturing process. Our next modification is to retain the nanosecond information provided by the tpacket_v2 structure:

pcap-linux.c, in function *pcap_read_linux_mmap*():

```
                case TPACKET_V2:
                        tp_len    = h.h2->tp_len;
                        tp_mac    = h.h2->tp_mac;
                        tp_snaplen = h.h2->tp_snaplen;
                        tp_sec    = h.h2->tp_sec;
  -                     tp_usec   = h.h2->tp_nsec / 1000;
  +                     tp_usec   = h.h2->tp_nsec;
                        break;
```

Using the default capture buffer size of libpcap (which is 2 Mbytes) disk I/O performance could lead to a serious number of packet drops at a high transmission rate. Unfortunately libpcap does not feature any option for adjusting capture buffer. In order to prepare libpcap for high speed packet procession, its capture buffer had to be increased. We made a series of measurements at 1 Gbps transmission rate so as to determine the optimal size of this buffer for capturing without packet loss. These measurements resulted in a capture buffer of 128 Mbytes, which is an empirical value. The last modification made to libpcap is to increase the default buffer size:

pcap-linux.c, in function *activate_mmap*():

```
        if (handle->opt.buffer_size == 0) {
  -             /* by default request 2M for the ring buffer
*/
  -             handle->opt.buffer_size = 2*1024*1024;
  +             /* request 128M for the ring buffer */
  +             handle->opt.buffer_size =
128*1024*1024;
        }
```

With this modification we enhanced the well-known libpcap (v1.1.x) library so as to be able to put nanosecond precision timestamps onto captured packets without packet loss. This feature relies on some novel features of the latest linux kernels that we effectively integrated into the libpcap library via our modifications. We made libpcap ready to operate with nanosecond precision timestamps; however its effective TST greatly depends on the performance of the underlying hardware.

At this point it is important to note that the nanosecond resolution is largely theoretical. On a dedicated server-class machine with 2 x Intel Xeon dual-core 3GHz (Woodcrest 5160) CPUs and 8GB of RAM, we managed to get a TST of approx. 75 ns. Hence, we would like to emphasize that effective TST and its variance greatly depends on the hardware performance of the host computer, its current system load, various critical kernel parameters and the time data conversion overhead. Even so, in our result TST is close to the time precision requirement of packet capturing on 10 Gbps Ethernet since the inter-arrival time of minimum-sized consecutive frames is about 61ns.

System dependency and variance of TST are rooted in the software-based nature of the solution. Accordingly, an extensive series of comparative tests against hardware-based solutions is required for its validation (see Section IV for details).

B. *Application*

With the nanosecond capable libpcap, a wide range of network data traces can be captured and stored for subsequent analysis. Accordingly, we have made further developments to make the commonly used tcpdump and Wireshark capable of easily processing, displaying and storing these high precision timestamps in the quasi-standard pcap file format (Figs. 3, 4).

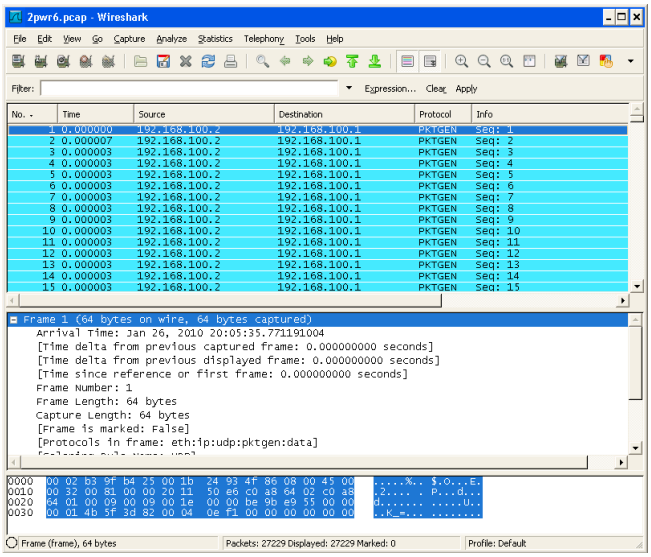

Figure 3. Output screen for microsecond timestamp resolution with the standard libpcap and Wireshark

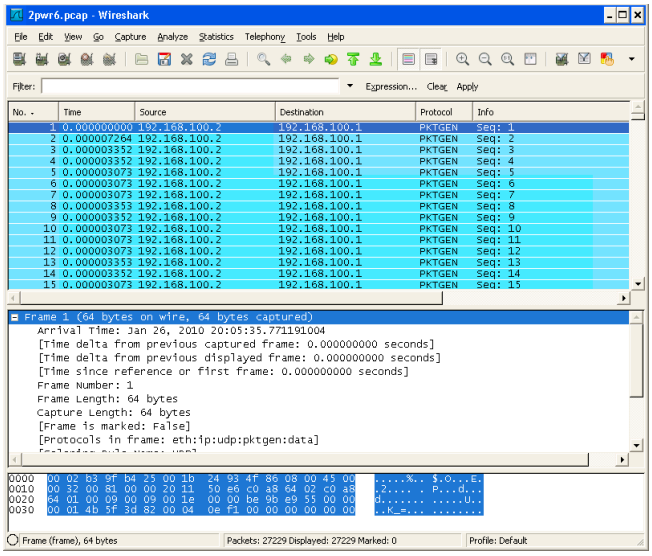

Figure 4. Output screen for nanosecond timestamp resolution with the enhanced libpcap and Wireshark

While tcpdump is not, Wireshark 1.2.x is indeed ready to process pcap trace files that feature nanosecond precision packet timestamps [17]. We had to apply a specific magic number (0xa1b23c4d) that registered for the nanosecond pcap format. Using this number, Wireshark could be made capable of identifying and capturing pcap files with alternative attributes and structures.

In the pcapio.c source file, in function libpcap_write_file_header():

```
-file_hdr.magic = PCAP_MAGIC;
+file_hdr.magic = PCAP_NSEC_MAGIC;
```

The magic number of 0xa1b23c4d stands for a subversion of pcap that includes nanosecond precision timestamps for each packet.

The default time precision of the Wireshark GUI must be set to a nanosecond:

In the gtk/recent.c source file:

```
- recent.gui_time_precision = TS_PREC_AUTO;
+ recent.gui_time_precision
=TS_PREC_FIXED_NSEC;
```

For comparison, two screenshots present the timestamp resolution enhancement. Microsecond scale TSP is the default time resolution for libpcap and Wireshark (Fig. 3), while our modified version is capable of capturing nanoscale TSP as displayed in the “Time” column (Fig. 4).

V. PRECISION EVALUATION OF THE SOFTWARE TIMESTAMPING

A. *The timestamp generation process within the kernel*

Timestamp put onto each packet must be adequately accurate. Hence, it is essential to investigate the process of its generation and to minimize CPU consumption of its sub-processes. Received packets are enqueued by the kernel in the CPU’s incoming packet queue. Packet enqueuing is performed within interrupt context while software timestamps are generated by the getnstimeofday() and ns_to_timespec() functions (Fig. 5).

For representing time domain relation of the successive packets in 10^{-9} second resolution, TST also has to be kept in this time domain. The minimum of this overhead can be predicted by measuring the execution times for getnstimeofday() and ns_to_timespec(). For this measurement we applied a clock source with the lowest and the most uniform overhead, called TSC. The CPU instruction used to read the TSC register value is RDTSC [18] since its execution time takes constant or shows a very low variation on most systems. Time consumption for the aforementioned functions are measured by inserting TSC checkpoints into the kernel code. The read TSC values are corrected with its overhead on the current system as well as taking the system CPU clock frequency in account.

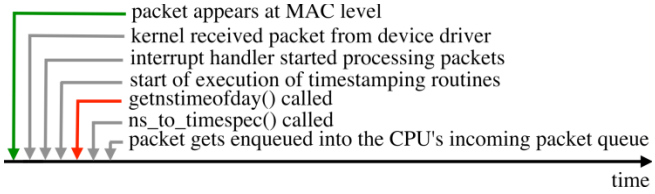

Figure 5. The sub-processes of software-based timestamp generation

The minimum overhead of TST for common CPU frequencies can be derived from mean results of a measurement series made on various hardware architectures (Fig. 6). For a CPU of 3 GHz only some 10 ns is the estimated minimum value of TST. Real execution times show some variation since several processes must share the hardware resources, in this case the CPU itself. In the extreme timestamp generation instructions are preceded or interrupted by the execution of other processes' instructions, which the kernel scheduler decides upon.

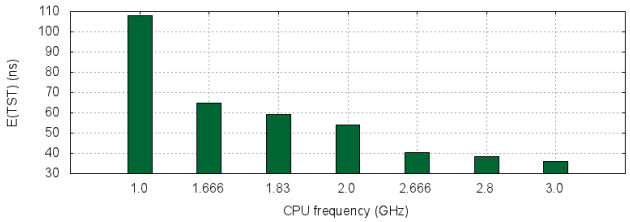

Figure 6. Timestamping overhead on different CPU speed

B. *Timestamping performance of the common kernel clock sources*

In order to compare timestamping capabilities of the mostly available clock sources, we set up a Gigabit Ethernet test network. A dedicated FPGA (Field-programmable Gate Array) packet generator had been set up and a continuous sequence of 72-Byte packets has been generated and captured. Since the PCI-based Netfpga-1G board [19] was applied as GbE NIC the inter-frame gap was adjusted to 1472 bytes due to the data transfer limitation of the PCI bus and the Netfpga device driver. Also note that its driver does not support NAPI, more packets per interrupt mode or other performance enhancing technologies. The inter-arrival times and packet losses were recorded for every clock source. Beside the software timestamp derived from the kernel clock source an additional 8 ns resolution hardware timestamp was inserted by the NetFPGA (Fig. 7).

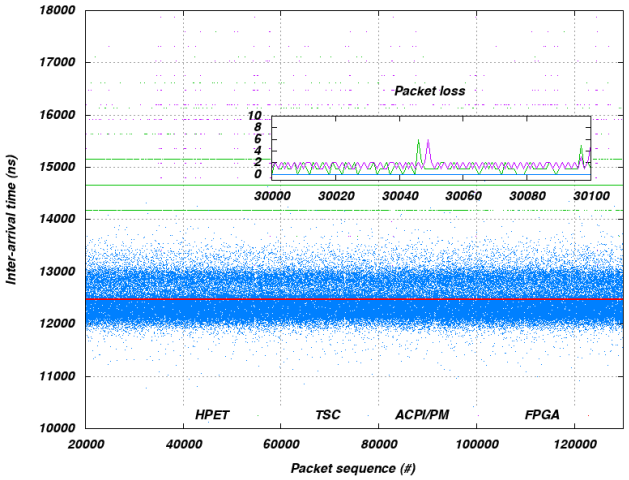

Figure 7. Precision evaluation of the timestamping

On Figure 7, one dot represents a successfully received and timestamped packet where the timestamp value is relative to the arrival of the previous packet.

For this dual timestamping operation mode, we modified the Linux kernel as well as the libpcap library to be able to store both timestamps side by side. Since our FPGA-based packet generator injects a 32-bit serial number into each packet, losses were easy to detect.

In this case beside that the inter-arrival times show large variation using the ACPI-PM and HPET clock sources, serious packet drops were present due to the high overhead of accessing these clock sources.. ACPI-PM based timestamps show the highest overhead. The three nearly solid lines of HPET show that TST variance is lower than that of ACPI-PM. Since the packet sending rate was constant a higher inter-arrival time value indicates a higher execution overhead.

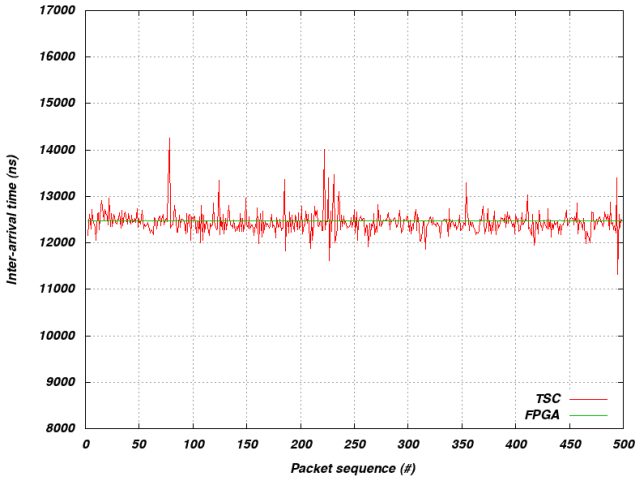

Figure 8. TST performance comparison (TSC and hardware timestamping)

Based on the hardware timestamps we can assume that the time values derived from TSC show a more realistic representation of the inter-arrival times. Moreover, it features the lowest TST overhead among the supported kernel clock sources.

Accordingly, it is reasonable to compare the variance of inter-arrival times gained from the TSC and the NetFPGA (Fig. 8). It is important to note that hardware timestamps show inter-arrival times of the packets in the MAC layer in contrast to the software based timestamps that represent the time moment of enqueuing the received packet.

Nevertheless, there is an obvious difference between the hardware and software timestamps in absolute time since their generation occurs at different points of the data path. They are not related very closely but the comparison shows that the relative inter-arrival times derived from TSC can be fairly close to the hardware-based values. However, even with TSC the TST variation of software timestamps has a significant extent since the execution of the generator functions is triggered by the kernel's scheduler subsystem (Fig. 8). Software timestamps are generated by kernel space functions, thus it is easy to see that kernels with pre-emption enabled are not eligible for high precision timestamping and high performance packet capturing.

C. *Timestamp generation using TSC*

To generate a timestamp based on TSC its value has to be converted, since the register value read by RDTSC is not represented in natural time but in the CPU frequency. Linux kernel has the cyclecounter/timecounter/timecompare framework. It makes possible to use independent cycle counter running on an arbitrary frequency to convert it to natural time. The cyclecounter structure has to be initialized by storing the counter's current value (in this case the register value of TSC) and letting it know its ticking frequency. Since there is no floating point support in the kernel, this frequency conversion is described by a mask, a shift and a mult member. Mask describes the size of the counter (64 bit for TSC). Shift is fixed at 22 (based on an algorithm in arch/mips/kernel/time.c from the Linux kernel) and clocksource_khz2mult() helps to convert CPU frequency into a multiplier. Then the timecounter has to be fixed to a base time (using ktime_get_real() and another clock source such as HPET) and to be stored the counter's current cycle counter value.

Timecounter_cyc2time() function is applied to convert counter value to natural time and timecompare_update() to update offset since last conversion, furthermore to handle possible counter turnaround. Downside is that the usage of these function calls adds significant processing overhead.

Nevertheless, TSC is actually the most adequate clock among the commonly available Linux kernel clock sources. On high performance and carefully tuned systems, its precision is sufficient for generating timestamps in the nanosecond time domain for certain traffic patterns, however for intensive traffic (high packet rate) hardware-based solution has to be applied.

As a future project, a post-processing of TSC data could be implemented to get the potential benefits of offloading register data conversion to time of day format.

VI. CONCLUSION

Version 1.1.1 of libpcap provides a 10^{-6} second native resolution, however pcap format supports a larger 2 x 32-bit timestamp value for each stored packet. On Gigabit Ethernet and faster networks, a timestamp resolution that works in the microsecond domain may not enable the precise reproduction of the time-domain relation between consecutive frames. Therefore overall analysis of the data transmission could lead to a false result.

For packet capturing with libpcap, it is assumed that timestamping is performed when a frame is enqueued to the kernel's input packet queue. Accordingly libpcap must retrieve timestamps from the kernel.

In this paper, we showed our alternative libpcap-based network monitoring solution for Linux systems, which features nanosecond resolution timestamping. Our primary goal was to test and evaluate all of the clock sources and kernel functions and features that are essential parts of our project to turn libpcap into a nanosecond-capable capture library. With the presented modifications and additions to the original codes, we managed to maintain and adapt the benefits of *tpacket_v2* structure within the entire packet capturing process, which resulted in our enhanced libpcap solution. In Section V, the precision of the applied software-based timestamping was analyzed and evaluated. We showed that the variation of the TST derived from two factors: the retrieval overhead of the applied clock source and the kernel's scheduler that commands the execution of running processes.

ACKNOWLEDGMENT

The work is supported by the TÁMOP 4.2.1./B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

REFERENCES

[1] Peter Orosz and Tamas Skopko, "Software-based Packet Capturing with High Precision Timestamping for Linux," August 22-27, 2010, 5th International Conference on Systems and Networks Communications, Nice, France

[2] Peter Orosz and Tamas Skopko, Timestamp-resolution problem of traffic capturing on high speed networks, January 28-30, 2010, ICAI international conference, Eger, Hungary

[3] Gianluca Iannaccone, Christophe Diot, Ian Graham, and Nick McKeown, "Monitoring very high speed links," Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[4] Libpcap, a common open source packet capture library for Unix/Linux systems. [Online]. Available: http://www.tcpdump.org/, 29/07/2011

[5] IETF RFC2679, A one-way delay metric for IPPM. [Online]. Available: http://www.ietf.org/rfc/rfc2679.txt, 29/07/2011

[6] IETF 3393, IP Packet Delay Variation Metric for IPPM. [Online]. Available: http://www.ietf.org/rfc/rfc3393.txt, 29/07/2011

[7] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwal, and P. Van Mieghem, "Analysis of End-to-end Delay Measurements in Internet," submitted to PAM 2002

[8] Jörg Micheel, Stephen Donnelly, and Ian Graham, "Precision timestamping of network packets," Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, November 01-02, 2001, San Francisco, California, USA

[9] The DAG project. [Online]. Available: http://www.endace.com, 29/07/2011

[10] Attila Pásztor and Darryl Veitch, "PC based precision timing without GPS," Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, June 15-19, 2002, Marina Del Rey, California, USA

[11] Cace TurboCap network interface card. [Online]. Available: http://www.cacetech.com/products/turbocap.html, 29/07/2011

[12] Nethawk iPro traffic analysis appliance. [Online]. Available: https://www.nethawk.fi/products/nethawk_ipsolutions/ipro/, 29/07/2011

[13] Cascade Shark Appliance. [Online]. Available: http://www.riverbed.com/us/products/cascade/cascade_shark_appliance.php, 29/07/2011

[14] Linux NAPI device driver packet processing framework. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/napi, 29/07/2011

[15] Christian Benvenuti, Understanding Linux Network Internals, O'Reilly, 2006

[16] TSC, Intel 64 and IA-32 Architectures Software Developer's Manual. [Online]. Available: http://developer.intel.com/Assets/PDF/manual/253667.pdf, 29/07/2011

[17] Wireshark Network Protocol Analyser. [Online]. Available: http://www.wireshark.org/, 29/07/2011

[18] Performance monitoring with the RDTSC instruction, [Online]. Available: http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf, 29/07/2011

[19] NetFPGA project. [Online]. Available: http://www.netfpga.org/, 13/01/2011