Investigating the Precision of the TSC-based Packet Timestamping

Tamas Skopko, Peter Orosz
University of Debrecen/Faculty of Informatics, Debrecen, Hungary

***Abstract*— With the emergence of gigabit per second and higher bandwidth networks, software based packet capturing faced severe performance challenges in two areas: lossless packet acquisition at high arrival rate and high precision timestamping. Many research projects proposed hardware based network monitoring solutions in order to eliminate these performance bottlenecks. In contrast, the common microsecond resolution software based packet processing has not been enhanced to meet the measurement requirements of high performance networks. In a previous paper, we already proposed an alternative packet capturing solution that is based on the libpcap library and supports 10^{-9} second resolution timestamping. We are now evaluating the performance of the proposed solution in practice. Experimental evidence shows that our approach represents the inter-arrival times of the incoming packets with a higher precision since the measured time values are generated with a lower overhead and stored in the native resolution.**

I. INTRODUCTION

Higher physical bandwidth implies the need of higher resolution and precision packet timestamps [1][2][3][4]. Because of missing software infrastructure and computing performance in the mainstream architectures, most high resolution timestamping implementations end up in dedicated hardware based solutions that support traffic analysis on 1 Gbps or faster network links [5][6].

The following two traffic analyzing techniques are applied for capturing packets: network link tapping or endpoint tapping. The former method means tapping at one point of the network path and collecting aggregated information of both directions. In the latter case, we capture incoming and outgoing packets at the communication endpoints.

The libpcap, the underlying packet capture library of traffic monitoring utilities such as tcpdump and Wireshark, supports 10^{-6} second resolution timestamping only [7][8]. In our paper, we present and evaluate an efficient high resolution software timestamping method supporting 10^{-9} second resolution, which is based on our modification of the libpcap library.

A. Timestamps

Timestamp is one of the most essential metadata assigned to a packet during the capturing process, which represents its arrival or departure time.

We define the following timestamp-related terms for later use throughout the paper:

- TSS: timestamp size (bit-length of the timestamp)
- TSR: timestamp resolution
- TSP: timestamp precision
- TST: timestamping time (timestamp calculation time)

Internal structures of the latest Linux kernel support 64-bit TSS. This data length provides enough storage for a 10^{-9} second resolution timestamp.

TSR requirements depend on the following parameters:

- physical bandwidth of the communication path
- the minimum of packet inter-arrival times within the monitored traffic

Both TSP and TST determined by the following factors:

- hardware platform architecture
- network interface card (NIC) architecture
- design and operation mode of the NIC driver
- operating system kernel (incoming packet queue, packet enqueueing and dequeueing, and interrupt handlers)
- clock source
- libpcap library

Using timestamps, the inter-arrival times for consecutive packets can be determined. Note that the minimum of packet inter-arrival times (Δt_i) depends on the physical bandwidth and the transmission rate and size of the packets. This minimum has to be expressed with the applied timestamping resolution to get valid measurement results.

II. PROBLEM DEFINITION

A. Link Speed

Assume that minimum-sized Ethernet frames are transmitted with the highest rate (i.e., line rate) on 1 Gbps and 10 Gbps links. Table I shows typical time parameters for these connection types.

B. NIC driver design and operation

The NIC driver controls data transmission between the MAC layer and the internal packet handling structures of the operating system. Two different approaches can be used to implement a Linux network device driver: non-NAPI and NAPI. A non-NAPI device driver can operate in interrupt-driven or in polling mode. Newer device drivers designed and implemented with NAPI in mind, which dynamically controls its operation mode: it applies interrupt-driven packet transmission method for low

traffic and it switches to polling mode when packet rate exceeds a driver-defined or pre-calculated threshold.

TABLE I.
TIMING PARAMETERS FOR SMALLEST FRAME SIZE ON 1 AND 10 GIGABIT ETHERNET LINKS

Timing parameters	Gigabit Ethernet	Ten Gigabit Ethernet
	Minimum sized (72 Bytes) Ethernet frames	
Bit time	1 ns	0.1 ns
Inter-frame gap	96 x bit time = 96 ns	96 x bit time = 9.6 ns
Δt between timestamps of two consecutive frames	576 ns + 96 ns = **672ns**	57.6 ns + 9.6 ns = **67.2 ns**
Maximum number of frames per second	1,488,096	14,880,960

1) Interrupt-driven operation mode

In this operation mode, an interrupt is generated after the reception of *k* consecutive frames (k≥1). It triggers interrupt handler of the operating system kernel to transfer received packets from the device driver and places them into the incoming packet queue. At low network traffic, this mechanism ensures that packets are immediately accessible in the incoming packet queue. This mode provides low latency for interactive and realtime applications. However, the utilization of the system resources is proportional to the packet arrival rate. Notably, high packet rate causes lossy packet processing, and at the extremes, it drives the system into a non-responsible state.

2) Polling operation mode

In polling mode, the kernel periodically queries the device driver about all of the packets received since the last query. The polling frequency is determined by the OS scheduler. At low traffic, this method is not efficient at all, since delay sensitive applications (i.e., realtime communication) receive their incoming packets with a delay inversely proportional to the polling frequency. The advantage of this operation mode develops under heavy network traffic of delay tolerant applications: scheduler controlled queries result in a lower overall system resource utilization.

3) Interrupt handling

Linux interrupt handling routines are built up into two levels to improve performance: a top half handler (TH) and a bottom half handler (BH) [9]. As an interrupt is triggered, the CPU executes the TH and disables other interrupts. TH saves any information required for a later execution. It uses a flag to notify the kernel about the interrupt event and its saved data. Before returning, it enables interrupts for local CPU again. Then the kernel checks for the flag and starts the execution of the assigned interrupt handler's BH (if exists), and clears the notification to enable a next call.

By using software interrupts (SoftIRQ), multiple BH processes can be executed concurrently to ensure low processing delay. The TH of the NIC driver transfers the received packets from the interface card to the kernel and schedules the BH, which will be called at a later "secure" time moment to process the enqueued packets.

Single frame per interrupt operation mode can easily exhaust system resources when intensive traffic contains small sized packets. Intel's e1000e driver is an excellent example of clever driver design [10]. It can be tuned to process the desired amount of packets per each interrupt, and to throttle interrupt requests in order to limit resource consumption under intense network load while maintains reasonable packet latency. This balancing behavior is based on some dynamically calculated thresholds that are determined upon packet counters and timers.

C. The operating system kernel

The kernel's incoming packet queue handler enqueues packets received from the network adapter's device driver (Fig. 1). The *sk_buff* structure stores all information about the enqueued packets. This data structure contains a 64-bit integer field called *tstamp*, which stores the packet's enqueueing or dequening time. Its upper 32-bit stores the time expressed in seconds, while the bottom 32-bit stores sub-second information of the timestamp. This representation length enables a TSR of 10^{-9} second.

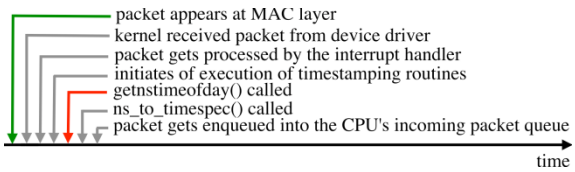

Figure 1. Timestamp generation process

Timestamps can be generated at several points of the data path: i.e., at frame reception, when it is copied into the system memory or at frame enqueuing when it is inserted into the incoming packet queue. First case is based on the APIC timer clock source with 10^{-3} second TSR, which is out of the scope. The latter case is more interesting since an incoming packet is directly accessible for the network stack (BH) after its enqueueing.

D. Clock sources

The Linux kernel supports various hardware clock sources. Their availability depends on the hardware architecture. They cannot only be used to determine short time intervals and for timing, but to calculate wall clock-time as well.

Classic PC timer has been available for a long time. It uses a crystal-controlled input oscillator (1.193182 MHz), and its counter has a 16-bit input register.

CMOS (Complementary Metal-Oxide Semiconductor) RTC is part of a battery-powered memory partition that also stores PC BIOS settings even when the machine is turned off. It is named after the low powered integrated circuitry technology. The RTC has two main functionalities: continuously runs the ToD (Time of Day) that stores time in *year/month/day hour:minute:second* format, and therefore can be read in 1 second resolution only. It also applies a timer that is capable of generating recurring interrupts in the range of 1 to 8192 Hz. This timer is write-only and its input value must be the power of two. Local APIC (Advanced Programmable Interrupt Controller) is part of the interrupt controller of modern

PCs. In multiprocessor systems, there is one APIC for each CPU, which is integrated on chip in recent processors. Its timer has a 32-bit counter and some input registers. The input frequency derived from the frequency of the system FSB (Front Side Bus). Its resolution is higher and its counter is wider compared to PIT and CMOS, but there is no way to reliably determine its operating frequency from software. Therefore, its frequency is computed by measuring it to PIT or CMOS, which results in an approximated value.

ACPI-PM (Advanced Configuration and Power Interface - Power Management Timer), often called Real Time Clock (RTC), is integrated into the south-bridge of the motherboards. It provides a 24-bit counter that increases by 3.579545 MHz (triple of PIT's frequency), and has a long access time (1-2 microseconds). It can operate even when the machine is suspended.

HPET (High Precision Event Timer) is available in most of the modern PC architectures. It has a central increasing counter that can be turned off and on by software and has 32-bit or 64-bit length. Its increment can be read through a register. It supports multiple timers each of them provides a timeout register that is compared with the central counter. When any of the timeouts is exceeded, the corresponding timer becomes activated. If the timer is set to recurring mode, HPET adds its increment to the corresponding register to automatically compute the moment of the next activation. One of its disadvantages is that its drift, precision and access time is not fixed by specification. HPET operates at 18 MHz typically, and it takes 1-2 microseconds to read it out. Its low jitter (10^{-8} second order) makes it a precise clock source but its frequency of ~20 MHz is too low for its application in high resolution timestamping.

TSC is a 64-bit counter register that can be found in x86 processors since the Intel Pentium. There is no input register, and it counts CPU ticks since system boot. TSC is an excellent, high resolution and low access overhead source, which makes it an optimal source for timestamping. Recent Intel CPU's also have the constant TSC feature that ensures the register value to be incremented always at the nominal processor frequency (even in power saving mode) [11].

E. The libpcap library

Common traffic analysis applications use the libpcap for packet capturing [7]. Its internal routines make possible to transfer packets and their corresponding metadata from the kernel to the user space. Unfortunately, this library internally represents the timestamp's sub-second information only in 10^{-6} second resolution. In a previous paper, we already introduced an alternate solution to improve its resolution [1].

On Linux systems MMAP (memory mapping) feature is available to reduce the number of memory copies. Otherwise, libpcap uses the IOCTL-call SIOCGSTAMP that has severe resource consumption.

III. IMPLEMENTATION

Our aim was to make libpcap capable of 10^{-9} second resolution timestamping and lossless capturing. As the first step, we had to investigate the timestamping mechanisms of the Linux kernel and test the relevant structures and functions with generated traffic.

Starting from Linux kernel 2.6.27, IOCTL-call *SIOCGSTAMPNS* is available, which returns 10^{-9} second resolution timestamps. However, it does not store the time of the packet reception, but the time of the last socket operation. *Tpacket_v2* structure appeared in the same version of the Linux kernel and its *tp_nsec* field stores timestamp's sub-second information in 10^{-9} second resolution. Accordingly, we modified libpcap to retain the resolution towards the related applications.

A. The timestamping process

To create adequately precise timestamps, we had to reveal the generation process and minimize the time consumption of each sub-process. After reception, the packet gets enqueued in the incoming packet queue by the kernel. As a sub-process, timestamp generation is initiated by calling the *getnstimeofday()* function, and *ns_to_timespec()* thereafter (Fig. 1).

To generate 10^{-9} second resolution timestamp, TST has also to be kept in nanosecond order. The minimum of TST overhead can be determined by measuring the process time of the functions *getnstimeofday()* and *ns_to_timespec()*.

For this calculation, we used the fastest accessible clock source, TSC. The execution time of TSC reading instruction *RDTSC* is near constant or shows low variance on most systems. We can measure the execution time of the aforementioned functions by inserting TSC reading checkpoints within the timestamp generation code and compensating the read values by the process time of the RDTSC instruction itself.

Figure 2 shows mean minimum TST overhead at typical CPU frequencies, which is derived from the average of the measurements performed on different architectures. For the CPU frequency of 3 GHz, the mean value is only 45 nanoseconds. Real execution times show some variance, since multiple processes share the same hardware resource, i.e., the CPU. Timestamp generating instructions can be preceded or interrupted by other higher priority processes. This decision is made by the kernel's scheduler.

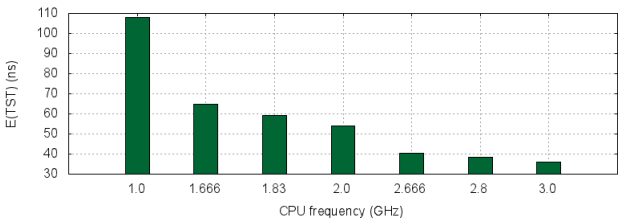

Figure 2. Mean time of timestamp generation on different CPU frequencies (Intel Core architecture)

B. Choosing the optimal clock source

As a minimum requirement for 1 ns resolution timestamping, the operation frequency of a clock source has to be at least 1 GHz. For low TST, the clock source has to be accessed as fast as possible. TSC is the only clock source among the discussed ones (see II. D), which fulfills the mentioned requirements. Furthermore, TST and its variance can be kept low by avoiding the usage of preemptive kernel.

C. NIC selection

In an ideal scenario, timestamps on successive incoming packets would show very small relative difference between their arrival times at the MAC layer and their enqueueing times at kernel level. However, this demand would strongly require a single frame per interrupt operation mode. Nevertheless, no system can tolerate an interrupt intensity of 1,488,096 Hz (Table 1.). Apart from this fact, its high processing performance and comprehensive parametrization make e1000e NIC driver an optimal choice for packet capturing purpose. For our measurements, we configured it to *InterruptThrottleMode = 0, RxIntDelay = 0* and *RxAbsIntDelay = 0*. Nevertheless, even these settings do not force it into a single frame per interrupt operation mode at high arrival rate.

D. Buffers

In order to provide sufficient memory for incoming packets during a long-term capture session, some kernel buffer parameters had to be increased (Table II). Kernel parameter *net.core.rmem_max* sets the maximum receive buffer size for all types of network connections. The value of *net.core.rmem_default* defines the default receive buffer size of a single connection. The maximum buffer size per socket can be set via the n*et.core.optmem_max* variable. The parameter value of *net.core.netdev_max_backlog* is also essential, since it specifies the maximum number of packets to be enqueued when the packets are received on the interface faster than the kernel can process them.

TABLE II.
DEFAULT AND TUNED VALUES OF NETWORK BUFFER SPECIFIC KERNEL PARAMETERS

Kernel parameter	Default value	Tuned value
net.core.rmem_max	131071	16777216
net.core.rmem_default	108544	108544
net.core.optmem_max	10240	20480
net.core.netdev_max_backlog	1000	50000

Since frequent I/O operations can reduce packet processing performance, we also had to increase the size of the capture buffer within the libpcap. The particular value depends on the link speed and the system architecture. At 1 Gbps and a system with at least Intel Core CPU and 7200 RPM SATA disks, an empirical value of 64 MB seems to be sufficient for capturing without packet loss. However, extended buffers do not compensate missing computing power. At 1 Gbps line rate, we could run short-term measurement sessions until a maximum of 2000 small sized packets without loss. For longer-term measurements, this system was capable of lossless capturing a sequence of 424-Byte packets separated by an inter-frame gap (IFG) of 96-Byte. Packet processing performance depends on many factors. The evaluation of processing power requirements for a line rate lossless capturing capable system is beyond the aims of this paper.

IV. EVALUATION

An FPGA-based packet generator device was dedicated to generate synthesized network traffic with high precision. It transmits a preconfigured amount of packets of a specific size with a constant IFG. Packets were captured on an Intel PRO/1000 dual-port Gigabit Ethernet NIC using the e1000e kernel driver configured to *InterruptThrottleMode=0, RxIntDelay=0* and *RxAbsIntDelay=0*. It should be noted that this mode does not necessary mean single frame per interrupt operation mode, since the driver throttles on high intensity of traffic by all means to avoid critical system overload.

On Fig. 3, short-term capture results of series of 144-Byte packets with 12-Byte IFG are shown. The constant line represents the inter-arrival time of packets at the MAC layer, which is exactly 1248 ns because of the following calculation: 144-Byte packet + 12-Byte IFG = 1152 bits + 96 bits on wire. The packets were sent out with a constant rate.

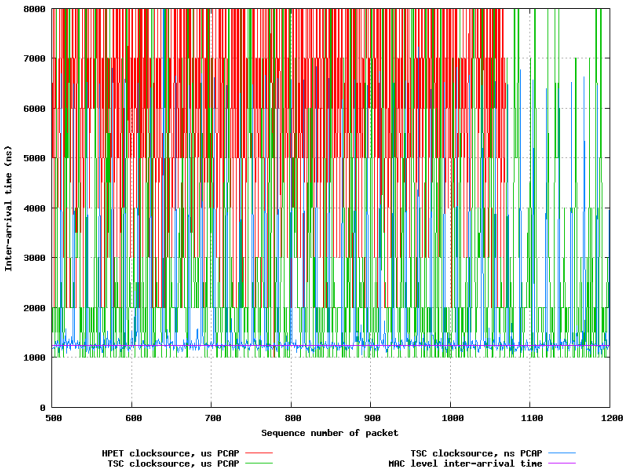

Figure 3. Short-term capture results of different clock sources and different resolution libpcap variants

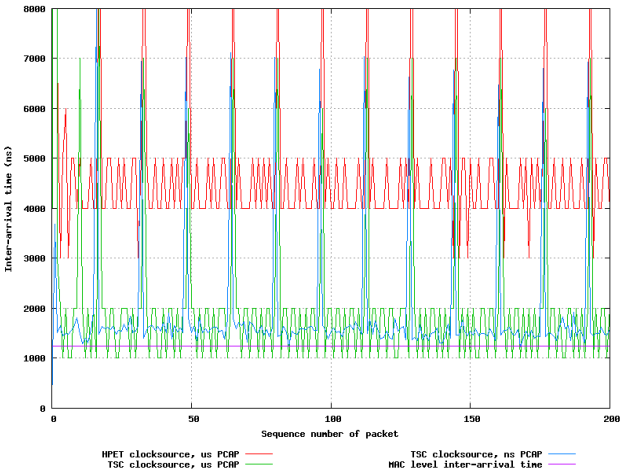

Figure 4. Detailed view of different capture sessions

Microsecond PCAP files are timestamped using HPET and TSC clock sources, while nanosecond ones are based on TSC clock source and our high resolution

timestamping method. Additionally, the calculated inter-arrival times are demonstrated with the constant line (as frames appear at the MAC layer).

It is apparent on Fig. 4 that measured inter-arrival times represented by nanosecond resolution software timestamps can come close to the MAC layer inter-arrival times. Although microsecond resolution TSC measurement follows the same tendency, low resolution values can make traffic analysis more difficult. HPET-based timestamps show more variation and their greater calculation time also can be shown. Another fact is that only nanosecond session could keep up with the high intensity flow of small packets, while microsecond captures get into packet losses. Of course this scenario of small packets is an artificially produced of extremity, but represents performance differences of the capture setups expressively.

Fig. 4 gives a detailed view of a short-term capture session similar to the previous one. Frequent spikes of inter-arrival time values are introduced as a signature of the e1000e driver operation under high traffic intensity.

Table III shows statistical information of the measurement. Although its variance is relatively low, HPET-based measurement presented large packet loss. In contrast, TSC-based measurement done by the original µs resolution libpcap did not lose any packets and shows less variance. The TSC-based measurement captured by the nsec resolution libpcap shows lower variance and reduced processing overhead.

TABLE III.
STATISTICS FOR SHORT-TERM MEASUREMENT BASED ON SERIES OF 144-BYTE PACKETS WITH 12-BYTE IFG

	Packet loss (%)	Variance (ns)
HPET/µs PCAP	76.083	5033
TSC/µs PCAP	0.000	2744
TSC/ns PCAP	0.000	1323

A long-term measurement of a sequence of 768-Byte packets with 96-Byte IFG has also been performed (Fig. 5). Importance of tuned kernel buffers and increased capture buffer gets more unavoidable. Nevertheless, all measurement sessions ran with extended buffer sizes, microsecond targeted traffic captures still could not keep up without packet loss.

TABLE IV.
STATISTICS FOR LONG-TERM MEASUREMENT BASED ON SERIES OF 768-BYTE PACKETS WITH 96-BYTE IFG

	Packet loss (%)	Variance (ns)
HPET/µs PCAP	31.244	28593
TSC/µs PCAP	7.639	109098
TSC/ns PCAP	0.000	11292

Table IV sums statistics of the measurement. Although traffic intensity in this case was much lower than in the previous one, HPET-based measurement still shows large loss of packets. TSC-based measurement performed by the standard µs resolution libpcap also has a small but non-negligible amount of packet loss and greater variance than the HPET-based one. The measurement based on our solution – with TSC clock source and ns resolution PCAP as trace file – did not lose any packets and shows the lowest variance.

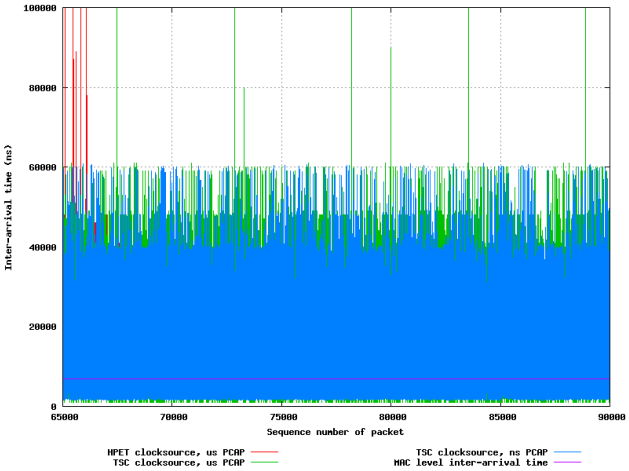

Figure 5. Long-term capture session

V. CONCLUSION

Although libpcap 1.1.x supports 10^{-6} second TSR only, the PCAP file format supports 2x32 bits for TSS. On 1 Gbps or higher bandwidth links, timestamps of 10^{-6} second resolution cannot adequately describe the time domain relation of the consecutive packets. Statistical analysis based on these measurements can lead to a false result.

We modified libpcap and related applications to support 10^{-9} second resolution timestamps, extended capture buffer size and important kernel parameters correlated with high performance packet processing to ensure lossless capturing. We showed up how timestamp accuracy depends on the typical TST of the system, and selected the most adequate clock source for generating software timestamps on a generic Linux system.

Nevertheless, for precise measurement of the inter-arrival times at the MAC layer, a network monitoring solution that features hardware timestamping should be applied, which is not affected by the Linux kernel internal scheduling and does not rely on shared resources, i.e., CPU and system memory.

Overall benefit of the modifications is to get a high resolution representation of the kernel level packet enqueueing times.

We also managed to reduce timestamping overhead of the packet processing, which optimizes the system to handle higher traffic rate. The presented modifications resulted on a generic Linux-based traffic measurement system capable of high resolution real-time monitoring of high bandwidth connections.

ACKNOWLEDGEMENT

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] 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

[4] 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

[5] The DAGproject, http://dag.cs.waikato.ac.nz, http://www.endace.com, 03/09/2011

[6] Cace TurboCap network interface card, http://www.cacetech.com/products/turbocap.html, 03/09/2011

[7] Libpcap, a common open source packet capture library for Unix/Linux systems, http://www.tcpdump.org/, 03/09/2011

[8] Wireshark, http://www.wireshark.org/, 03/09/2011

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

[10] Interrupt Moderation Using Intel® GbE Controllers, http://download.intel.com/design/network/applnots/ap450.pdf, 03/09/2011

[11] TSC, Intel 64 and IA-32 Architectures Software Developer's Manual, http://developer.intel.com/Assets/PDF/manual/253667.pdf, 03/09/2011