PCI Express as a Killer of Software-based Real-Time Ethernet

Rostislav Lisový, Michal Sojka, Zdeněk Hanzálek
Czech Technical University in Prague,
Faculty of Electrical Engineering
Technická 2, 121 35 Prague 6, Czech Republic
Email: {lisovros,sojkam1,hanzalek}@fel.cvut.cz

***Abstract*—The time-triggered Ethernet gains in popularity in many different industrial applications. While several hardware implementations exist, software implementations are also very attractive for their price-to-performance ratio. The main parameter that influences the performance of time-triggered protocols is the transmission jitter, which is greater in software implementations.**

In this paper we evaluate one source of transmission jitter occurring in such software implementations – the PCI Express bus, which interconnects the CPU, memory and the network interface card in modern multi-core computers. We show that the contribution of the PCI Express to the transmission jitter of Ethernet frames is significant and is in the same order of magnitude as the scheduling jitter of modern real-time operating systems. PCI Express latency and jitter are evaluated under various loads produced by virtual machines running on dedicated CPU cores. We use the IEEE 1588 feature of the network card for precise timing measurements.

I. Introduction

Ethernet-based networks are becoming more and more popular in industrial communication. This is because it is a historically well proven technology, it offers high bandwidth and many real-time (RT) extensions that make the Ethernet communication deterministic exist. Unfortunately, there is no single universal real-time Ethernet extension. Several companies offer their proprietary solutions [1]. Together with the high price of those solutions, this might be the reason why software-based real-time Ethernet implementations are so popular [2–6].

We investigate the possibility of implementing a software-based real-time Ethernet protocol while utilizing the extensive virtualization capabilities of modern x86 hardware. Our focus is on the commercial-off-the-shelf (COTS) networking and computing hardware, which is gaining in popularity for industrial automation, not only because its favorable price and widespread availability but also because of the familiar environment when used with any of the real-time Linux derivatives.

One way of achieving deterministic medium access is to use *time division multiple access* method employed by the so called *time-triggered* (TT) protocols [7, 8]. TT protocols need to maintain a notion of global time in order to synchronize the transmission in all nodes. One way to achieve this is to use *Precision Time Protocol* (PTP), standardized in IEEE 1588 [9], that allows one to synchronize the time with sub-microsecond precision over the Ethernet. The advantages of TT networks are determinism and trivial evaluation of the worst-case behavior. A disadvantage is inefficient use of available bandwidth, because temporarily unused slots must be either retained in the schedule or complex rules for their skipping must be introduced. In addition, if the used technology exhibits transmission (TX) jitter[1], which is common with software-based solutions, it is necessary to insert large inter-frame gaps that decrease bandwidth utilization even more. Examples of TT protocols are TTEthernet [8], ProfiNet IRT [10] or FlexRay [11].

Some of the drawbacks of TT protocols are mitigated by event-triggered protocols. There, the medium access is controlled by the reception of specific messages from other nodes. For example *Ethernet Powerlink* [5] has a managing node that controls it when so called controlled nodes can access the medium. In *Node Ordered Protocol* [12], the medium access is determined by the predefined order of nodes. Another principle is used in *Avionics Full-Duplex Switched Ethernet* (AFDX) [4], which employs bandwidth limiting to ensure that the network is not overloaded and latencies remain low.

For today's industry, determinism of the network communication is necessary but not sufficient. The efficiency of resource usage is also important but it contradicts the demand for determinism. Therefore, there are attempts to integrate multiple subsystems of different criticality in a single platform to improve the efficiency. This contrasts to the federated principle applied so far, where every subsystem was implemented as a separate node. The examples of modern integrated architectures are IMA [13] in avionics and AUTOSAR [14] in the automotive domain. One of the means for efficient integration of subsystems is the use of *virtualization*. Here, *hypervisors* are used to provide strict separation of independent subsystems [15] allowing one to build a mixed-criticality system.

In the past, it was believed that the biggest source of TX jitter occurring in software implementations of the real-time Ethernet was the operating system's (OS) scheduler [3]. With appropriate hardware and modern RT operating systems, the worst-case scheduling latencies are below $30\,\mu s$ [16]. Nowadays, with the advent of multi-core CPUs, it is possible to dedicate one or more cores for network processing and completely eliminate the non-determinism of the OS scheduler. We have performed this by using a NOVA microhypervisor [17]. We isolate one CPU from all unrelated activities such as timer interrupts. This is not yet possible with standard Linux and their virtualization platforms such as KVM. Moreover, the

[1] *Transmission jitter* is the difference between maximum and minimum deviation from the intended transmission time.

minimalist design of NOVA, together with a small memory footprint of our network processing code, ensures that all the network processing code and related data can be fetched from the CPU's private level-2 cache memory without interference caused by memory traffic from other cores. Additionally, the isolation and fault-containment properties of the NOVA system make it suitable for use in safety-critical environments, which would be impossible for systems such as Linux or RTAI, where a huge amount of code (Linux kernel) runs in the most privileged CPU mode.

We believed that our implementation outlined in the previous paragraph would provide very good performance, especially in terms of TX jitter figures. To our surprise, the real jitter was greater than $10\,\mu s$, which was comparable to other Linux-based solutions found in the literature. Therefore, we decided to investigate the cause of it.

The contributions of this paper are: We evaluate the properties of the PCI Express bus (PCIe), which interconnects the CPU, memory and the network interface card (NIC). We show that the contribution of the PCIe to the TX jitter of Ethernet frames is significant. PCIe latency and jitter are evaluated under various loads produced by virtual machines running on other CPU cores. We use the IEEE 1588 feature of the NIC for precise timing measurements. Our findings are useful for all SW-based real-time protocols implemented on modern x86 hardware. For time-triggered networks our results can be used to determine the proper size of inter-frame gaps. For event-triggered networks, the TX jitter influences the timing precision, which might be an important parameter for many applications.

The paper is structured as follows: After reviewing the related work in Section II, we describe the architecture of modern computers and of our hardware and software used for measurements in Section III. The results of our experiments are presented in Section IV and we conclude with Section V.

II. Related work

Many software implementations of real-time Ethernet exist. Probably, the most well known is RTnet [18]. It is a generic networking stack for RTAI and Xenomai – real-time extensions of Linux. As RTnet is implemented as a kernel module sharing an address space with the Linux kernel, it is not well suited for safety-critical applications.

Grillinger, Ademaj, Steinhammer, *et al.* [3] describe software implementation of the Time-Triggered Ethernet (TTE) implemented in RTAI. The authors evaluated the achieved latencies and jitters and found them in the order of tenths of microseconds. They claim that the main bottleneck of their implementation is the interrupt latency that influences the precision of software packet timestamping and that hardware time stamping would help. In this paper, we show that despite the fact that hardware timestamping is used, the PCI Express causes significant jitter.

Bartols, Steinbach, Korf, *et al.* [19] analyzed the latencies of the TTE hardware by using a Linux kernel with rt_preempt patches. They implemented software-based timestamping of the packets and report that the precision of their measurements is in units of microseconds. Since the system used for their measurement was based on a PCI Express bus, it is questionable whether the precision was really so low. We show that a PCI Express can introduce jitter over $10\,\mu s$.

Cena, Cereia, Bertolotti, *et al.* [20] describe a software implementation of the IEEE 1588 time synchronization protocol based on RTnet. The accuracy of their implementation is assessed by generating a signal on a parallel port of a PC and measuring the properties of that signal. Since the parallel port is connected over a slow LPC bus as detailed in Section III-A, the jitter of the parallel port's generated signal is also influenced by the PCI Express jitter, which can be quite high.

Pure software implementation of the OpenPOWERLINK, open source Industrial Ethernet solution, are described in [5]. Safety-certifiable software implementation of the AFDX stack and the achieved latencies are analyzed in [4]. None of those papers give sufficient details on the CPU-NIC interconnect.

III. Architecture

A. Today's PC architecture

The architecture of the modern PC and of many industrial computers is determined by the architecture of the PCI Express (PCIe) bus [21]. The central component called a *root complex* connects the CPU cores with memory controllers and other peripherals (Fig. 1). It is usually integrated on the same chip as the CPU. The other components of the PCIe topology are *endpoints*, which usually represent devices, and *switches*, which are responsible for interconnecting all of the endpoints with the root complex. All those components are interconnected via PCIe links that are formed by one or more lanes. The more lanes the higher bandwidth of the link. N-lane link is denoted as xN. All PCIe communication is packet-based and packets are routed from the root complex via switches to the destination endpoints and back. Since one link can transfer only one packet in one direction at a time, packets may be delayed by waiting for a free link.

Root complex typically has several PCIe links. In PCs, one is dedicated to a graphics adapter, another is connected to a so called *platform controller hub* [22] (or chipset in short). It contains PCIe switch(es) interconnecting different PCIe endpoints and conceivably a bridge to the legacy PCI bus. PCH also integrates other controllers such as USB, SATA, LPC (*low pin count interface* – used to connect legacy peripherals such as a parallel port or a PS/2 keyboard). Those additional controllers appear to the operating system as PCI devices. Besides PCI devices, PCH also contains non-PCI devices such as high-precision event timers (HPET).

Due to the packet-based character of the PCIe communication, sharing of PCI links between devices and several sources of latency in the PCIe communication protocol [23] (e.g. the need for acknowledging received packets), the total latency of PCIe communication can be relatively high compared to an older parallel PCI bus.

B. Intel 82576 Gigabit Ethernet Controller

In our experiments, presented in Section IV, we used a modern network interface card (NIC) based on Intel's 82576 Gigabit Ethernet controller. The main reason we chose this

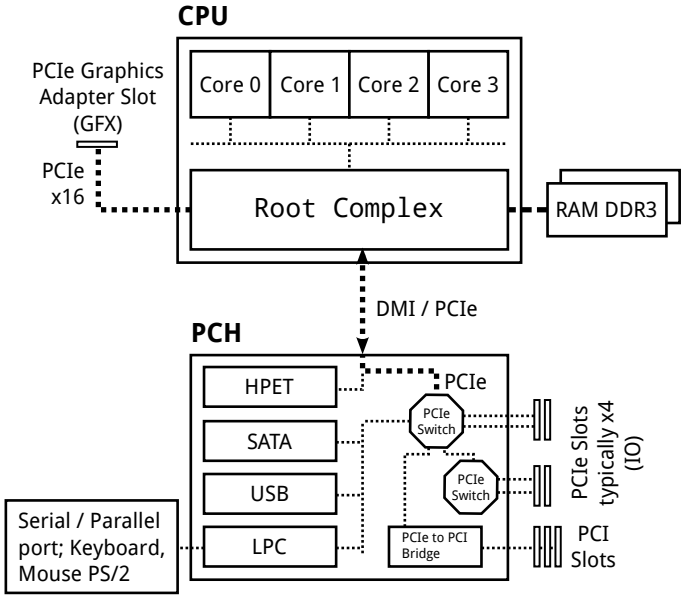

Figure 1. Typical architecture of a modern PC

COTS NIC was the built-in hardware support for the IEEE 1588 standard. This support was used for precise measurements of the PCIe latencies in this paper. The NIC contains two Ethernet controllers but in our experiments we use only one of them.

The key features supporting the implementation of PTP on this device are an adjustable clock and hardware timestamping.

The adjustable clock is implemented as a 64-bit up counter. The readout is possible through two 32-bit registers (the higher half of the value is latched when the lower half is read). The clock is periodically incremented. Both, the period and the increment are configurable. The increment period can be set as a multiple of 16 ns.

The hardware timestamping feature allows one to capture timestamps (i.e. the value of clock described above) of the received PTP packets and of arbitrary transmitted packets. Only one RX and TX timestamp may be stored at the same time in dedicated pairs of 32-bit wide registers. The hardware responsible for timestamping is as close as possible to the PHY circuit, which performs the conversion between logical signals and the physical layer. This ensures a high precision of the captured timestamps, which are taken immediately after transmitting/receiving the Ethernet *Start of frame delimiter*.

C. Software architecture

Our longer-term goal is to build a software-based time-triggered Ethernet stack on COTS x86 computers with a NOVA microhypervisor. While this stack is not yet implemented, we outline its planned software architecture in this section, because it is the same as in our experimental setup for this paper.

The software architecture is depicted in Figure 2. The lowest level consists of a NOVA microhypervisor [17]. It is responsible for hardware resource management and scheduling and it is the only component that runs in privileged processor

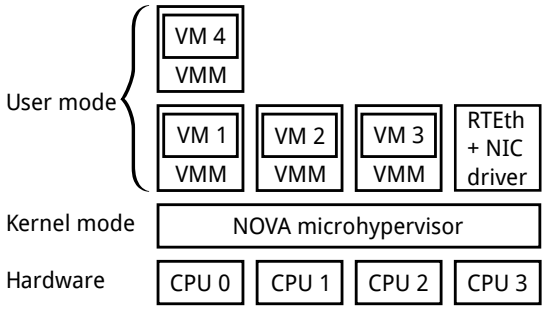

Figure 2. Software architecture of our implementation

mode (kernel mode). Its very small trusted computing base (9 kLoC) together with the virtualization support makes it a very interesting solution for safety-critical applications. Note that in NOVA, device drivers are not the part of the kernel.

NOVA can execute applications in two modes: native mode and virtual machines (VM). *Virtual machine monitor* (VMM) is a NOVA application running in the user mode comprising the native part that emulates the PC platform and the VM part that executes the VM code.

The code implementing the real-time Ethernet functionality is placed in RTEth application running in the user mode. It is a native NOVA application and besides other things, it contains the device driver for the NIC. The application is responsible for managing the transmission and reception of the Ethernet frames according to the predefined schedule. It is important to note, that the application does not touch (i.e. copy) the data to be transmitted or received. The data to be transmitted is stored in the main system memory directly by the application that produces it (e.g. a virtual machine). This application only notifies the RTEth application that the data is ready in the memory and RTEth instructs the NIC to transmit it. A similar principle is applied for packet reception. This is possible because of the use of the shared memory between the RTEth application and its clients. The implementation is simplified by the use of IOMMU.

The RTEth application itself is pinned to one CPU core, which is reserved solely for it. The size of application's code and data is 40 KiB, which means it fits into the CPU's 256 KiB of L2 cache. Note that the kernel code used for inter-process communication and scheduling is less than 2 KiB in size and it fits into the cache together with the application. This means that the application does not suffer from interference caused by memory traffic from other cores in the system. This reduces the execution time jitter of the application and makes its execution more predictable.

D. Testbed setup

The computer used for the evaluation was a common PC computer equipped with an Intel i5-3550 CPU (IvyBridge, 4 cores, no hyper-threading), 4 GiB of RAM and a network add-on card with an Intel 82576 GbE controller (NIC in the following). The NIC is equipped with an x4 PCIe connector.

The computer comprises two PCIe slots. One of them is an x16 slot connected directly to the root complex inside the CPU, the other, an x4 slot, is connected to the chipset (PCH).

```
+-00.0  Intel Corp. Ivy Bridge DRAM Controller
+-01.0-[01]--+-00.0  Intel Corp. 82576 Gigabit Network Connection
|            \-00.1  Intel Corp. 82576 Gigabit Network Connection
+-02.0  Intel Corp. Ivy Bridge Graphics Controller
+-14.0  Intel Corp. Panther Point USB xHCI Host Controller
+-16.0  Intel Corp. Panther Point MEI Controller #1
+-19.0  Intel Corp. 82579LM Gigabit Network Connection
+-1a.0  Intel Corp. Panther Point USB Enhanced Host Controller #2
+-1b.0  Intel Corp. Panther Point High Definition Audio Controller
+-1d.0  Intel Corp. Panther Point USB Enhanced Host Controller #1
+-1e.0-[02]--
+-1f.0  Intel Corp. Panther Point LPC Controller
+-1f.2  Intel Corp. Panther Point 6 port SATA AHCI Controller
\-1f.3  Intel Corp. Panther Point SMBus Controller
```

Figure 3. Logical PCIe topology as shown by `lspci -tv` command

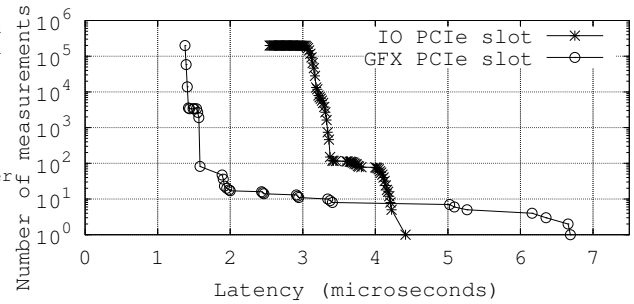

Figure 4. Latency profile of the NIC clock register readout (for NIC in two different PCIe slots). System with no load.

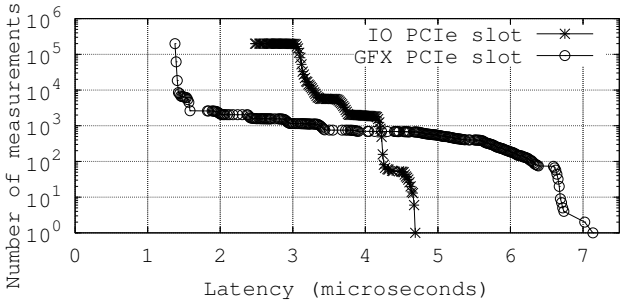

Figure 5. Latency profile of the NIC clock register readout (for NIC in two different PCIe slots). System with CPU load.

For the purpose of this paper we call the former the GFX slot and the latter the IO slot.

Besides experimenting with the NIC, we also utilized the SATA controller to generate interfering PCIe traffic. We connected a common rotating hard drive with a SATA 3.0 interface to one of the on-board SATA ports.

We tried to extract the physical PCIe topology of our system, but it does not provide the relevant PCIe capabilities to do that. The logical PCI topology presented to the operating system is flattened and does not correspond to the physical topology. Nevertheless, Figure 3 shows the output of the `lspci` tool. In this case, the NIC was connected to the GFX slot, which is denoted as PCI bus number 1 (`[01]` in the figure). When the NIC was connected to the IO slot, the corresponding entries appeared on bus 2 (`[02]` in the figure).

In our measurements, we did not identify any anomalies that could be caused by System Management Interrupts, so we did not attempt to eliminate or mitigate them.

IV. Evaluation

This section presents the results of our measurements of the PCI Express latencies. We measured two types of latencies in our experiments: the latency of the NIC clock register readout and the hardware TX latency. The experiments are described in more details in the following subsections.

All measurements were performed under several different loads of the system:

- *No load*: No intentional load was placed on the PCIe or CPU.
- *CPU load*: Three Linux 3.6 VMs running on dedicated CPUs (VM1-3 in Fig. 2) run a Sysbench CPU benchmark[2].
- *Disk load*: One Linux 3.6 VM with direct access to the SATA controller (connected to the PCIe bus) was run on a dedicated CPU. This VM was executing a `dd if=/dev/sda of=/dev/null bs=8M count=1 iflag=direct` command in an infinite loop. The size of the block (8 MB) was chosen on purpose to fit into the on-disk cache. This way, the disk traffic consumes the maximum possible PCIe bandwidth.
- *Combined load*: A combination of disk and CPU load – one VM with disk load and two VMs with CPU load.
- *Disk + serial load*: The same as disk load but the output of the `dd` command (about 100 characters) was sent to the serial line.

Besides changing the load, we also changed the PCIe slot where the NIC was plugged in during the experiment (the GFX and IO PCIe slot).

We present the results of some experiments in the form of a *latency profile*. This is a cumulative histogram of the measured values with a reversed vertical axis in log scale. The advantage over a plain cumulative histogram is that the worst-case latencies are magnified by the log scale (see for instance lower right hand corner of Fig. 4).

A. Latency of NIC clock register readout

In this experiment, we measured the time spent by reading the clock register located in the NIC. The resulting latency was calculated as the difference between the values obtained from two consecutive reads of the whole 64-bit clock register ($t_{clk2} - t_{clk1}$ in Fig. 6). Despite the fact we do not exactly know how big a fraction of the total time was spent in the NIC's internal logic and what was caused by the PCIe latencies, we believe that the measured time represents the lower bound of the communication latencies between the CPU and the NIC.

Figure 4 shows the values measured for the *no load* scenario whereas Figure 5 contains values measured with *CPU*

[2]Available from http://sysbench.sourceforge.net/; the command was `sysbench --test=cpu --cpu-max-prime=999999999 run --num-threads=1`

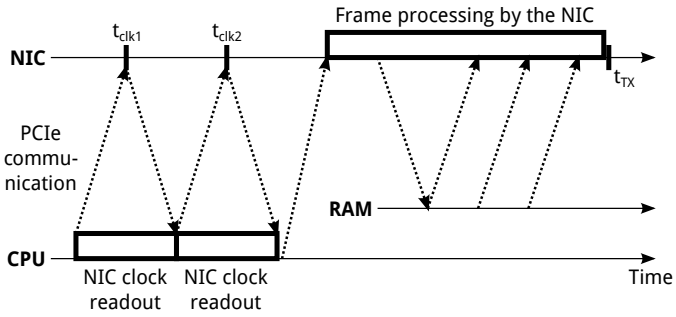

Figure 6. Explanation of the measured latencies

load. It can be seen that there are significant differences in the latency and jitter figures between the PCIe slots. We summarize the measured values in the table below:

	Avg. latency		Jitter	
Slot	*No load*	*CPU load*	*No load*	*CPU load*
GFX	$1.38\,\mu s$	$1.41\,\mu s$	$5.31\,\mu s$	$5.76\,\mu s$
IO	$3.11\,\mu s$	$3.12\,\mu s$	$1.87\,\mu s$	$2.21\,\mu s$

Figure 7 shows the results of the *disk + serial load* scenario for the NIC in the IO slot. There are periodic spikes of increased latency. Although we first thought that the spikes are caused by disk transfers, it turned out that they are brought about by communication over the serial port that we used as a console for the VM. In NOVA, the serial driver uses polling to wait for an empty TX buffer register and this results in a high PCIe bus load. In production systems, polling is avoided whenever possible but sometimes device drivers have to use polling to work around hardware bugs [24].

A careful reader can also identify a small increase in latencies (cca. $0.5\,\mu s$) with a 40 ms period in Figure 7. This was caused by updating the text screen in the VGA video RAM of the integrated graphics adapter. If the VGA is configured in NOVA, the screen of the foreground application gets copied to the VGA memory 25 times per second. A similar increase of latencies can also be seen in Fig. 4. If the external graphics adapter and/or fully graphical mode was used, the latencies could be much worse [25].

B. Hardware NIC TX latency

In this experiment, we measured the time needed by the NIC to start the transmission of a frame. More precisely, we measured the time between the moment when the NIC got the information about a new frame to send (setting the NIC TX descriptor register to point to the ready packet descriptor) and the timestamp captured by the NIC while the frame was being transmitted. During the transmission the NIC autonomously fetches the frame payload from the RAM (via PCIe).

The results presented in this section are valid for 166 byte long frames. When we increased the frame length to 1 KiB, the latency increased by $1.5\,\mu s$ in both the GFX and IO slots.

Figures 8 and 9 show the latency profiles of the TX latencies under different loads in the GFX and IO slots, respectively. The latencies were calculated as a difference between the TX timestamp and the value read from the NIC clock register just before setting the NIC TX descriptor register

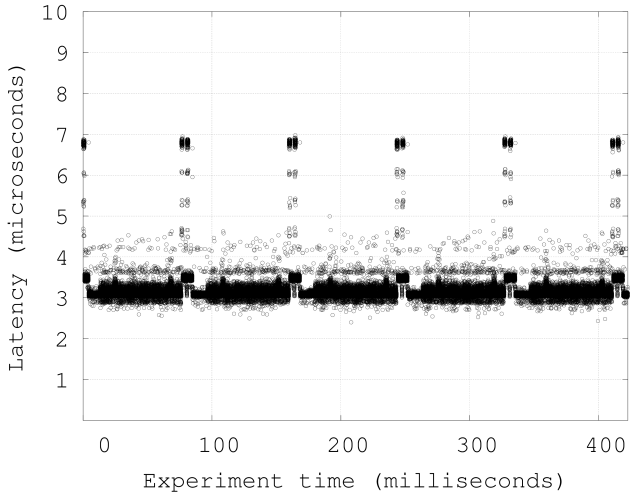

Figure 7. NIC clock readout latency in disk + serial load scenario (NIC was in the IO slot)

($t_{TX} - t_{clk2}$ in Fig. 6). The latencies for the GFX slot and IO slot ranged from $5\,\mu s$ to $14\,\mu s$ and from $8.5\,\mu s$ to $19.5\,\mu s$, respectively.

The distribution of latencies in time is shown in Figure 10. In the depicted experiment, the periods of no load and disk load scenarios were interleaved with a period approximately equal to 60 seconds. It can be seen that the distribution of the increased latencies in time is uniform.

The precision of our measurement method is influenced by the following factors: The end of the measured interval is captured with very high precision (hardware timestamp), but the start of the interval (NIC clock readout) suffers from an error in the range of several microseconds as shown in Section IV-A. If we wanted to decrease the error, we would need another clock synchronised with an NIC clock having a negligible readout time.

It is interesting to see that even a sole CPU load on unrelated CPUs caused big increases in latencies. The reason is that the CPU load makes the Linux scheduler to be invoked frequently. This resulted in about 1500 timer interrupts per second per VM. As NOVA uses HPET timers as a backend for all virtual timers, the communication between the CPU and HPET, located in the chipset, has an influence on the PCIe bus and, therefore, also on the NIC latencies.

The worst latencies were achieved for the *disk + serial load* although the sole disk load exhibits a low average latency. As mentioned above, this is caused by polling in the serial port driver. In summary, the jitter of the PCIe latency is similar for both slots and is approximately $10\,\mu s$.

V. Conclusion

With hardware support for IEEE 1588, it is possible to synchronize NIC clocks with sub-microsecond precision. However, if one wants to schedule the Ethernet traffic in the software, as many popular real-time Ethernet software stacks do, the achieved frame transmission precision is much worse. It is believed that the main cause of the transmission jitter is the

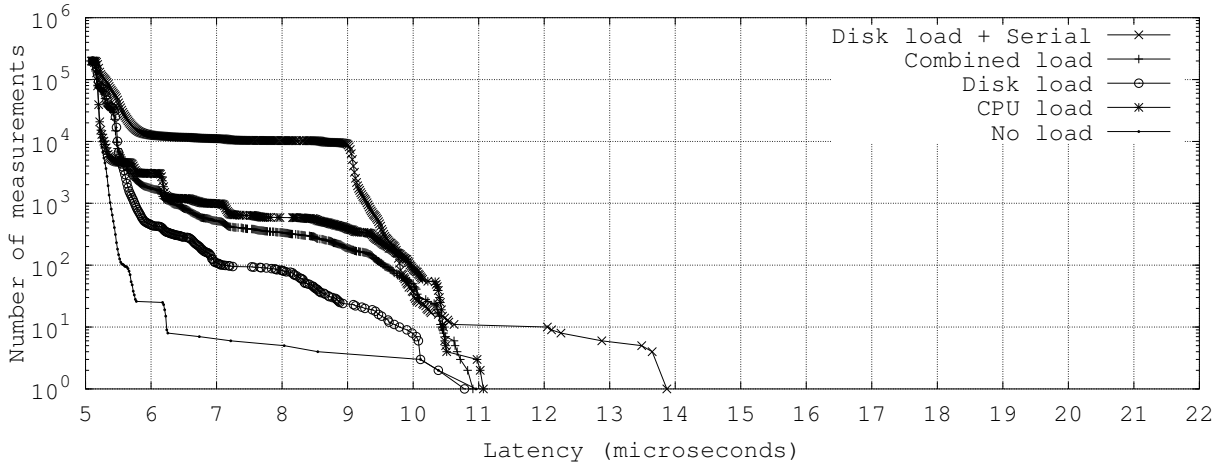

Figure 8. Latency profiles of the frame transmission on an Intel 82576 GbE controller for different PCIe and system loads (an NIC plugged into the GFX PCIe slot)

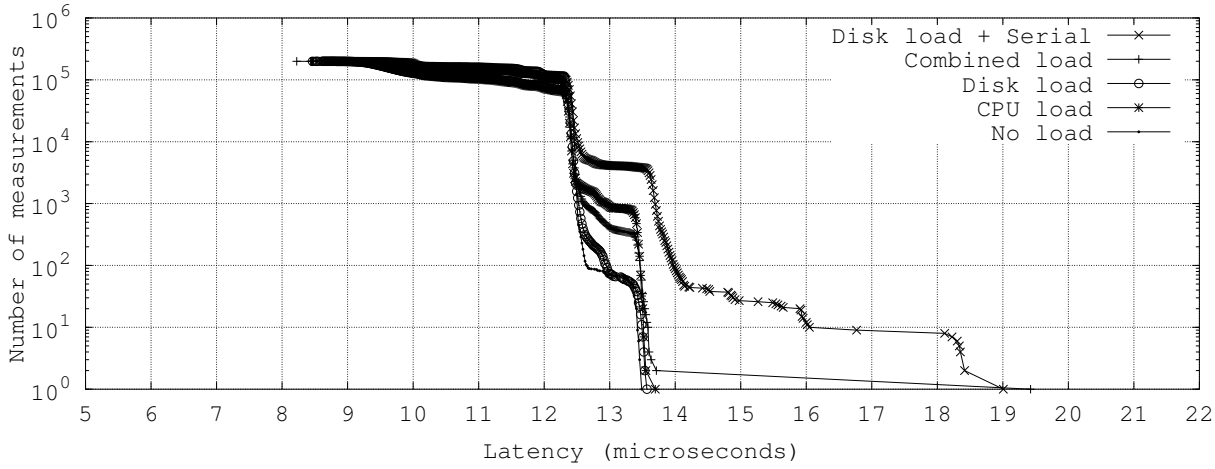

Figure 9. Latency profiles of the frame transmission on an Intel 82576 GbE controller for different PCIe and system loads (an NIC plugged into the IO PCIe slot)

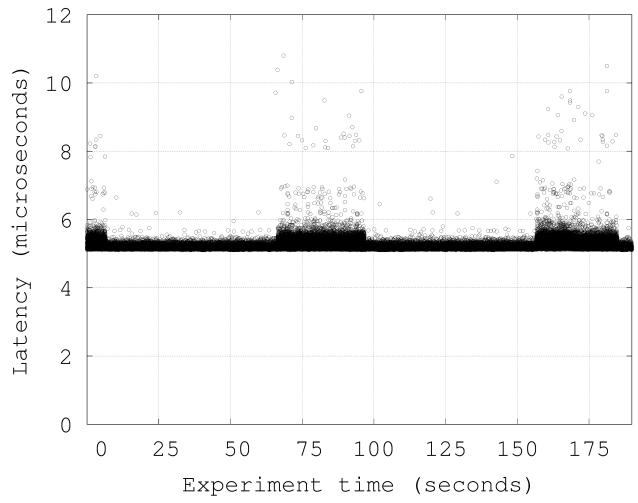

Figure 10. No load and disk load scenarios. Latencies calculated as $t_{TX} - t_{clk2}$.

scheduler of the operating system. In this paper, we identified another often neglected source of the transmission jitter, which is the PCI Express bus. We measured its contribution to the overall transmission jitter and found it to be around $10\,\mu s$. This value is in the same order of magnitude as the scheduling jitter of modern real-time operating systems.

As for our future work, we will look at improving the PCI Express induced jitter by using PCI Express QoS features such as isochronous virtual channels mentioned in [21]. We could not use them in this work, because our NIC does not support them. We plan to use the NetFPGA [26] platform to experiment with those.

ACKNOWLEDGMENT

The authors would like to thank Zoltan Fodor from Intel for providing the network cards for experiments.

The research leading to these results received funding from the ARTEMIS Joint Undertaking under the grant agreement n^{o} 295354 and from the Grant Agency of the Czech Republic under the Project GACR P103/12/1994.

References

[1] M. Felser, "Real time ethernet: standardization and implementations", in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, 2010, pp. 3766–3771. DOI: 10.1109/ISIE.2010.5637988.

[2] J. Kiszka, B. Wagner, Y. Zhang, and J. Broenink, "RTnet – a flexible hard real-time networking framework", in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, (Catania, Italy), Sep. 12–22, 2005. DOI: 10.1109/ETFA.2005.1612559.

[3] P. Grillinger, A. Ademaj, K. Steinhammer, and H. Kopetz, "Software implementation of a time-triggered ethernet controller", in *Factory Communication Systems, 2006 IEEE International Workshop on*, IEEE, pp. 145–150. DOI: 10.1109/WFCS.2006.1704143.

[4] I. Khazali, M. Boulais, and P. Cole, "AFDX software network stack implementation—practical lessons learned", in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, IEEE, 2009, 1–B. DOI: 10.1109/DASC.2009.5347574.

[5] J. Baumgartner and S. Schoenegger, "POWERLINK and real-time Linux: A perfect match for highest performance in real applications", in *Twelfth Real-Time Linux Workshop, Nairobi, Kenya*, 2010.

[6] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched Ethernet", in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, 2004, pp. 13–22. DOI: 10.1109/EMRTS.2004.1310992.

[7] A. Ademaj and H. Kopetz, "Time-triggered ethernet and IEEE 1588 clock synchronization", in *Precision Clock Synchronization for Measurement, Control and Communication, 2007. ISPCS 2007. IEEE International Symposium on*, IEEE, 2007, pp. 41–43. DOI: 10.1109/ISPCS.2007.4383771.

[8] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design", in *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, IEEE, 2005, pp. 22–33. DOI: 10.1109/ISORC.2005.56.

[9] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588*, 1st. Springer Publishing Company, Incorporated, 2010, ISBN: 184996565X, 9781849965651.

[10] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet io irt message scheduling with temporal constraints", *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 3, pp. 369–380, 2010, ISSN: 1551-3203. DOI: 10.1109/TII.2010.2052819.

[11] FlexRay Consortium *et al.*, "Flexray communications system", *Protocol Specification Version*, vol. 2, 2005.

[12] L. Chanjuan, N. McGuire, and Z. Qingguo, "A new real-time network protocol-node order protocol", in *Proceedings of eleventh real-time Linux workshop*, Open-Source Automation Development Labs, 2009, pp. 105 –109.

[13] C. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics", in *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, 2007, DOI: 10.1109/DASC.2007.4391842.

[14] AUTOSAR, *Specification of operating system*, R4.0 rev 3, Nov. 2011. [Online]. Available: http://www.autosar.org/download/R4.0/AUTOSAR_SWS_OS.pdf.

[15] C. Baumann, T. Bormer, H. Blasum, and S. Tverdyshev, "Proving memory separation in a microkernel by code level verification", in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, 2011, pp. 25–32. DOI: 10.1109/ISORCW.2011.14.

[16] C. Emde. (Oct. 2010). Long-term monitoring of apparent latency in preempt_rt Linux real-time systems, OSADL, [Online]. Available: https://www.osadl.org/fileadmin/dam/articles/Long-term-latency-monitoring.pdf (visited on 04/2013).

[17] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture", in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10, Paris, France: ACM, 2010, pp. 209–222, ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755935.

[18] J. Kiszka. (Apr. 3, 2013). RTnet, [Online]. Available: http://www.rtnet.org/.

[19] F. Bartols, T. Steinbach, F. Korf, and T. C. Schmidt, "Performance analysis of time-triggered ether-networks using off-the-shelf-components", in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, IEEE, 2011, pp. 49–56. DOI: 10.1109/ISORCW.2011.16.

[20] G. Cena, M. Cereia, I. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "A software implementation of IEEE 1588 on RTAI/RTnet platforms", in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, 2010, pp. 1–8. DOI: 10.1109/ETFA.2010.5640955.

[21] PCI Special Interest Group, *PCI Express Base Specification, Revision 2.1*. PCI-SIG, 2009.

[22] Intel, *Intel® 7 series / C216 chipset family platform controller hub (PCH) datasheet*, 326776-003, Jun. 2012. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/7-series-chipset-pch-datasheet.pdf (visited on 04/2013).

[23] K. Yogendhar, V. Thyagarajan, and S. Swaminathan. (May 21, 2007). Realizing the performance potential of a PCI-Express IP, [Online]. Available: http://www.design-reuse.com/articles/15900/realizing-the-performance-potential-of-a-pci-express-ip.html.

[24] Freescale Semiconductor, *MPC5200 (L25R) errata*, Rev. 5, ATA interrupt is not affected by FIFO errors, Dec. 2011, ch. 2.1. [Online]. Available: http://www.freescale.com/files/32bit/doc/errata/MPC5200E.pdf (visited on 04/2013).

[25] M. Cereia, I. Bertolotti, and S. Scanzio, "Performance of a real-time ethercat master under linux", *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 679–687, 2011, ISSN: 1551-3203. DOI: 10.1109/TII.2011.2166777.

[26] NetFPGA. (2013). NetFPGA, [Online]. Available: http://netfpga.org/ (visited on 04/2013).