future internet

MDPI

Article

Impact of Modern Virtualization Methods on Timing Precision and Performance of High-Speed Applications

Veronika Kirova [1,†,*], Kirill Karpov [1,†,*], Eduard Siemens [1,*], Irina Zander [1], Oksana Vasylenko [2], Dmitry Kachan [1] and Sergii Maksymov [1]

[1] Department of Electrical, Mechanical and Industrial Engineering, Anhalt University of Applied Sciences, Bernburger Str. 55, 06366 Köthen, Germany

[2] O. S. Popov Odessa National Academy of Telecommunications, Department of Higher Mathematics, Kovalska Str. 1, 65045 Odessa, Ukraine

* Correspondence: veronika.kirova@hs-anhalt.de (V.K.); kirill.karpov@hs-anhalt.de (K.K.); eduard.siemens@hs-anhalt.de (E.S.)

† These authors contributed equally to this work.

Received: 17 July 2019; Accepted: 14 August 2019; Published: 16 August 2019

Abstract: The presented work is a result of extended research and analysis on timing methods precision, their efficiency in different virtual environments and the impact of timing precision on the performance of high-speed networks applications. We investigated how timer hardware is shared among heavily CPU- and I/O-bound tasks on a virtualized OS as well as on bare OS. By replacing the invoked timing methods within a well-known application for estimation of available path bandwidth, we provide the analysis of their impact on estimation accuracy. We show that timer overhead and precision are crucial for high-performance network applications, and low-precision timing methods usage, e.g., the delays and overheads issued by virtualization result in the degradation of the virtual environment. Furthermore, in this paper, we provide confirmation that, by using the methods we intentionally developed for both precise timing operations and AvB estimation, it is possible to overcome the inefficiency of standard time-related operations and overhead that comes with the virtualization. The impacts of negative virtualization factors were investigated in five different environments to define the most optimal virtual environment for high-speed network applications.

Keywords: virtualization; cloud computing; timestamps precision; timekeeping; networking; QEMU; KVM; Virtual Box; Xen; VMware ESXi

1. Introduction

Deployed in the recent decade, Ethernet speeds have been increased from 10 Gbps to 40 Gbps and 100 Gbps and will go in the future towards Terabit per second, as stated in the Ethernet Alliance roadmap [1]. However, implementation and deployment of multi-gigabit applications, optimization of their performance and accuracy of the transmission speed remain significant challenges [2].

As a representative network application for which the accuracy of timing functions is the performance basis, we put forward the following requirements: the application shall have strict timing requirements, the application must be based on high-speed data transmission, and the application must also have practical applicability in modern networks. In this case, the precision of available bandwidth estimation applications is strictly tied to the precision of timing operations and meets mentioned requirements.

Recent network measurement projects, focusing on performance evaluation, capture such performance metrics as path capacity (maximum possible throughput) and available bandwidth

(maximum available throughput) of a network path. Estimation of these parameters requires creating explicit probe packets that consume CPU resource and possess enough timers precision to generate accurate timestamps of probe packets under bursty traffic conditions. Moreover, in this kind of application, it takes more than 200 time acquisition function calls per packet and about one sleep operation per issued data packet.

Because of virtualization, datacenter tenants no longer have direct access to the underlying physical network. The extra layer of indirection makes it hard to optimize application performance. AvB can measure the underlying network and provide performance metrics to the virtualized applications to optimize tasks such as VM placement and route selection [2].

Therefore, the general requirements of such time-sensitive networking applications are based on precise and rapid timing measurements, such as low latency, dealing with a limited performance of the end systems (e.g., frequency of the kernel interrupts), adoption of inter-packet interval, etc. In this paper, we address this challenge and aim to evaluate how the accuracy of modern timers can influence the result measurements of network performance under the virtualization conditions. For this, we have extended our previous research in timing operations with its performance influence on the real network application. As a matter of fact, keeping high time precision of control algorithms and meeting soft real-time requirements is a complex problem on modern multi-purpose operating systems such as Linux. This task is becoming even more challenging under virtual environments which require efficient access to the physical platform through the software middleware of the hypervisor or host operating system [3–5]. Thus, timing operations used by the applications are usually dependent on certain software layers in the operating system. As our previous research has shown [6,7], even operating systems running on bare hardware are often not efficient enough for high-performance applications. Running such operations in virtual environments potentially adds a significant overhead to these operations by the hypervisor OS, by means of passing data to bare hardware by it as well as by the means of the guest OS. However, the use of virtual machines as an intermediate layer for simplification of deployment and operations of high-performance applications is “tempting”. Virtualization is very widely used in data centers and in cloud environments. The possibility to run high-performance applications from out of a virtual environment is a very important factor for deployments of such applications, in which timer performance plays a significant role. In the present work, we consider the question of whether it is feasible and if so, on which systems and at which “costs”. After all, direct access to the *TSC* counter via *RDTSC* instruction can improve CPU overhead for time acquisition and computations on Linux by avoiding expensive system calls, but precise sleep operations remain a more challenging task. Sleep procedures, whose realization in the standard GNU C library has an insufficient precision due to the long wake-up time, which can be even more aggravated when sharing resources between host and virtual machines (VM). To overcome issues with standard system calls, an interface to available time counters as well as for sleep operations the *HighPerTimer* library [6] have been used besides the standard timing means of Linux OS. This library has been developed in terms of our previous research as an alternative to the Linux system timer functions, which operate timing functions by direct access to the hardware.

By applying different techniques for timekeeping on Linux machines, we intend to disclose problems that can arise for time-critical network applications running in a virtualized environment. Further, during the experiments, we investigated the impact of timing implementation within the workflow of two AvB tools—*Kite2* [8] and *Yaz* [9]—for estimation of network performance. In our previous work, the accuracy of these tools in high-speed networks has been compared [8,10]. By contrast, the target of this research is to explore which timing methods can be more suitable for implementing high-speed applications in a virtual environment. The first tool *Kite2* is our implementation of the modified AvB active probing algorithm and one of a very few tools which can deal with 10 Gbps and faster links. The second one, *Yaz*, was approved as an accurate tool, which scales well with the increasing cross traffic solution on dozens of Mbps links, and is available in open source [11,12]. The algorithms of *Kite2* have been developed in terms of our previous data

transport research [13–17] and use *HighPerTimer* library, whereas *Yaz* is based on system calls for timing acquisition and rough timing error correction. For 1 Gbps links, a specific adoption for the source code using *Yaz* was required. Further details about that experiment setup can be found in Section 2.2.3.

Figure 1 shows the active probing model used by both *Yaz* and *Kite2* AvB tools where S_n is a probe packet n, i_{sn} is the sending interpacket interval and i_{rn} is the interpacket interval on the receiver side. The main principle of this method is to find the optimal inter-packet interval, which is strictly dependent on timing operations precision. Therefore, the accurately chosen inter-packet interval on the sender side allows achieving the actual available bandwidth of the network path. On the other hand, inaccurate timestamping of the packets leads to under- or overestimation by the application, which is not able to meet an adequate sending data rate to the AvB. The goal of the AvB algorithm is to find the minimal value i_s after which the difference $i_r - i_s$ is minimal.

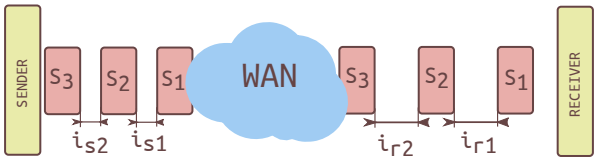


Figure 1. AvB active probing model.

Firstly, the study characterizes the timing behavior under stressful virtualization conditions, where the median wake up time of standard system sleep function is about 40–100 µs, which also implies the lower bound of inter-packet interval. Applying Equation (1), it is found that the corresponding data rate limit is 120–320 Mbps using standard 1500 bytes of MTU size.

$$R = \frac{8 \cdot MTU}{T_i} \tag{1}$$

where R is data rate in bps and T_i is the inter-packet time interval in seconds.

The replacement to the alternative timing methods can provide up to 56 ns wake up time and increasing the precision allows us to operate with 100 Gbps and higher, which better matches to the requirements of modern Ethernet networks.

To increase network performance and synchronize work of applications such as video streaming, multiplayer video games and content delivery services, it is necessary to use timing functions that are as precise as possible. Improved timing methods (direct call to *TSC* hardware timer and *HPTSleep* method from *HighPerTimer* library) that have been studied in this research show half the time cost of a standard Linux system call, and up to thousands of times lower miss time values than the sleep function from the standard C library. Accordingly, the main objectives of this paper are: (a) summarize empirical research of time acquisitions and sleep methods within virtual machines; (b) indicate the influence of timing methods on performance and accuracy of available bandwidth estimation in multi-megabit and multi-gigabit networks; and (c) provide recommendations of minimizing the possible errors during performance measurements in a real network.

Paper Contribution

In this work, we summarize the previous research to show the efficiency of proposed methods. The novelty of this research is in a comprehensive analysis of how different virtualization methods impact time and sleep measurements of two timing approaches—direct call to hardware timer and standard timing *System Call*. Furthermore, we investigate the impact of these approaches in a virtual environment on two real networking applications for available bandwidth estimation of middle- and high-speed networks.

The remainder of this paper is structured as follows. Section 2 reviews the materials and methods that were used for the experimental setup to identify which virtual environments are more or less feasible for deployment of high-performance networking applications in virtual environments. In this

section, software and hardware environments, load types, and scenarios between remote hosts and VMs are described in detail. Section 3 is devoted to the retrieved results of experiments with time acquisition, sleep function accuracy measurements and applying AvB tools performance in different virtual environments. Interpretation and discussion of the results can be found in Section 4.

2. Testbed Setup and Measurement Methodology

Experimental setup and measurement approaches description for both Timing and AvB measurements are described in this section. Experiments were performed on the 40 Gbps WAN lab of the Future Internet Lab Anhalt of Anhalt University of Applied Sciences.

2.1. Experimental Setup for Timing Measurements

System configurations and methodologies of the following experiments were deployed to perform initial tests on timing accuracy in a virtual environment as follows.

2.1.1. Used Timing Tools

The standard method of time acquisition in Linux are system calls such as *clock_gettime* and *gettimeofday* and (hereafter referred to as *System Call*). Until version 2.6 of Linux kernel, those functions were ordinary system calls, and afterwards they were reconstructed as virtual system calls to get rid of slow context switches and to give the possibility to get the timestamps through the memory mapping to the kernel space. Now, it is part of the VDSO library.

Besides that *clock_gettime* is implemented in VDSO system library call in Linux and is not performing a full context switched system call, there is a possibility to reduce its overhead even more. For this, a developed in term of our research HighPerTimer library uses direct access to TSC counter via RDTSCP instruction (hereafter referred to as *TSC*). All arithmetical operations were implemented as inline functions, thus the compiler maps it directly to assembler code without a function call overhead. The timers are internally kept as a single 64-bit counter and calculation of seconds/nanoseconds is performed only when the particular components are "exposed" to the calling functions component-wise.

The Linux waiting operation is presented as the uSleep function. It puts the process into the scheduler's waiting queue and returns its state after the target time. This leads to large overheads in the scale below microseconds. One possible solution to reduce the miss time value is to use busy-wait loop when some atomic operation will be executed repetitively during the target time. This method is very efficient from the precision perspective; however, it takes 100% of CPU resources during sleep.

The sleep from HighPerTimer library (*HPTSleep*) provides the hybrid solution to get the precision of busy-wait and low CPU utilization of System Calls. HPTSleep separates the waiting process into two phases to reduce the miss time value. Simplified scheme of this process is shown in Figure 2.

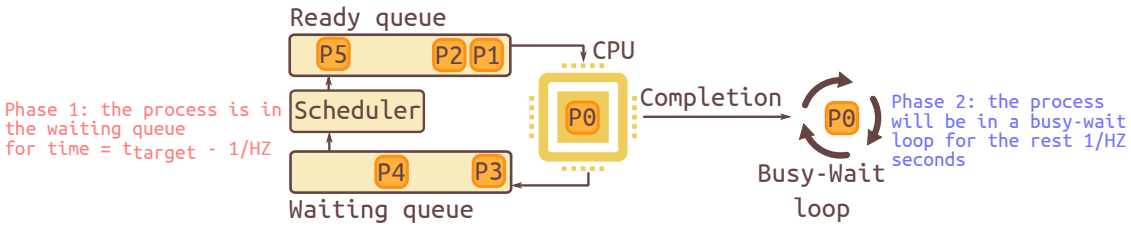


Figure 2. Simplified scheme of executing processes with HPTSleep.

In Phase 1, the process P0 will be set by uSleep function into the waiting queue for most of the target sleep time. The time of the first phase calculated as $t_{target} - \frac{1}{HZ}$, where HZ is the frequency of interrupts in Linux, limited by 1000 Hz. This phase will be skipped if the target time is lower than $\frac{1}{HZ}$. In Phase 2, the process will be executed in a busy-wait loop for the rest of the target time.

This leads to the high precision which is comparable with pure busy wait loop. At the same time, this hybrid method takes only 1.5% of CPU resources [6].

2.1.2. System Configuration

All tests were performed on the same hardware platform, a usual hardware server-like set up with the main requirement—CPU capability to include Intel-VT/AMD-V Virtualization Technology (VT) support [18]. For the test, Intel(R) Core(TM) i7-860 @ 2.80 GHz (Lynnfield) with 4 physical cores along with 8 GB of RAM and Broadcom BCM5751 Netxtreme Gigabit NIC were used. All measurements on both host and guest operating systems were performed using Ubuntu 18.04.2 LTS 4.15.0-46-generic kernel with standard scheduling configuration. Measurements were performed on the system with *invariant TSC counter* capability with a constant rate which guarantees a constant clock frequency even in the presence of power saving means and also a consistency of clock tick among different CPUs. Hyper-threading mode was disabled to provide a strictly serialized execution of CPU instructions. Virtual machines were configured with enabled VT technology to replicate physical CPU and were provided with two CPUs and 1 GB of RAM each. The following virtualization platforms were used due to being the most popular: Citrix XenServer Host 7.0.0-125380c, Virtual Box 4.3.36, QEMU/KVM, and VMware ESXi.

2.1.3. Load Types

In the context of a multiprogrammed computing systems, several kinds of tasks can be distinguished [19,20]. According to this classification, we distinguish the following load scenarios and therefore create corresponding load types for the experiments in the testbed:

- *Idle system*: In this load type, the operating system works in normal mode, without any additional working applications. An idle system represents the best-case scenario.
- *CPU-bound load*: This type creates a very high CPU load using up to 100% of available CPU resources. For this, a utility creates 100 threads with square root calculations. This type of load is often created by using video/audio conversion/compression algorithms or by applications for graphics processing.
- *IO/network-bound load*: This load type simulates typical VMs load, which can exist on hosting in a large data center. For this, we create an NFS server on the VM and start sending and receiving data calls as standard read/write calls. This workload results in a large number of context switches and disk interrupts. As an example, this scenario can be used for solving such problems as distributed processing of big data.

2.2. Experimental Setup for Available Bandwidth Estimation

Experiments described in this section were gained to test previously obtained results of timing accuracy in a virtual environment on the application example—AvB measurement tools.

2.2.1. Used AvB Tools

To investigate the timing operations' influence on networking application performance, the available bandwidth estimation tools were chosen. It was expected that a highly accurate estimation of bandwidth in fast networks challenges the timing operations the most and, to prove or reject this statement, it was decided to investigate two AvB tools designed for medium- and high-speed networks. AvB estimation approaches are generally divided into Probe Rate Model (PRM), Probe Gap Model (PGM) and probability-based models [21,22]. Previous research [21] showed that PGM model cannot estimate accurately the end-to-end available bandwidth of a multi-hop path, which is essential for wide area networks and most private networks. Thus, PGM approach was not considered in this paper. Performance of a probability-based models is not well-investigated and the results of experiments are provided for the low-speed networks only [22]. Investigations on a comparison of AvB tools

accuracy [11] in a medium-speed networks shows applicability and one of the highest performances of the *Yaz* tool [9] despite the relative network intrusiveness with the packet probes. The research on AvB estimation applications in the 10 Gbps networks [10,13] present additional challenges, which are caused by the increased accuracy sensitivity of the inter-packet interval. In these conditions, the tool named *Kite2* shows the most accurate results with low estimation latency. Both *Kite2* and *Yaz* are based on the Probe Rate Model (PRM) [21,22] of the active probing measurement, with the concept of self-induced congestion. This concept relies on sending probe trains or pairs with a specific data rate, which is further compared to the receiving one on the opposite side. If the send data rate appears to be equal to the receiver data rate, then it is lower than AvB in a link on path. An indicator of exceeded AvB by the sender is the lower receiver data rate than sending data rate. Reaching maximum available data rate can be also noticed, e.g., by the growth of probe RTT time or packet losses on the server. The data rate adjustment continues until the send and receive rates would approach each other. *Yaz* or so-called *calibrated pathload* tool is based upon and enhances the *Pathload* algorithm [23] of self-load periodic streams but uses only one probe packets train. It evaluates inter-packet interval (and corresponding constant bit rate, CBR) in the fleets of trains for each iteration of measurement as an increased value of the previous inter-packet interval on the sender side by its difference with the inter-packet interval on the receiver side. *Yaz* terminates the process of measurements as soon as the difference of inter-packet intervals meets predefined thresholds. The *Kite2* algorithm uses iterative AvB measurements to get "instant" bandwidths and further analyzes them to provide one final value of AvB. It has control of packet probing size to achieve the most accurate result with less intrusiveness. To reach a high-speed data rate with the high-precision bandwidth measurement results, HighPerTimer [6] timing library is used. It enables reading the value of TSC register before the packet is passed to socket for packets timestamping. A high-precision sleep function in HighPerTimer is used for keeping an accurate interval between packets, thus decreasing the send data rate deviation. For incoming packets, timestamps in *Kite2* are also provided directly by the kernel and are used to ensure an accurate time of a packet appeared on receiving host.

2.2.2. Network Topology for AvB Estimations

Experiments on AvB estimation with different types of timing acquisition and with different load profiles are performed on a testbed shown in Figure 3.

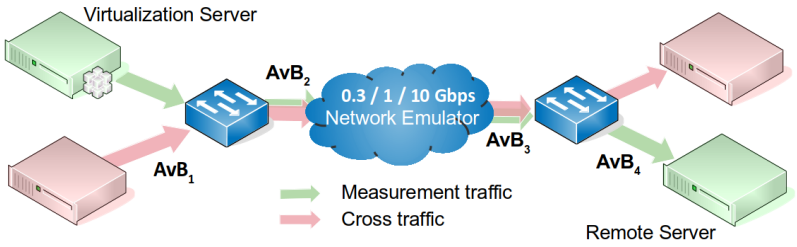


Figure 3. AvB Network topology.

The laboratory environment of a 10 Gbps network consists of:

- a Netropy 10G WAN Emulator, for emulating wide area network using different inferences such as packet loss ratio up to 100%, delays of up to 100 s, and various types of packet corruption and duplication;
- two instances of servers used for the cross traffic for an additional impact on AvB estimation equipped with 12× Intel Xeon®CPUX5690 3.47 GHz, 32 GB RAM, and 10G Chelsio NIC;
- one instance of server used for measurement traffic in the 10G network, equipped with 6× Intel® Xeon®;
- one instance of a physical server with VMs described in Section 2.1.2, periodically hosted VMs with XEN, QEMU, VirtualBox and ESXI equipped with a 1GE Intel I-219 NIC; and

- 10GE network switches Summit x650 from Extreme Networks.

2.2.3. Test Scenarios

To analyze the accuracy of AvB estimation results in different network configurations, experiments were conducted on three different scenarios: 300 Mbps, 1 Gbps, and 10 Gbps. The first scenario is the point of interest due to the limitation of a Linux sleep accuracy to 40 µs, which also corresponds to a minimal achieved inter-packet interval and, therefore, to 300 Mbps data rate. Data rates of 1 and 10 Gbps were chosen as upper limits for the corresponding Ethernet standards. In each of the experimental scenarios, the accuracy and performance of used AvB tool measurement were tested in a consecution with *HighPerTimer* and standard timing methods. Substitution of different timing methods for each AvB estimation tools was done without source code re-compilation. For this, the *LD_PRELOAD* environment variable which preloaded specified timing libraries before others were used. The scenarios are the following:

1. Between VMs and a remote server with an end-to-end path limited to 300 Mbps capacity, using *Yaz* for AvB estimation. Cross traffic data rate in this scenario was varied from 0 to 250 Mbps with 50-Mbps increments and was generated by *iperf* [24] tool.
2. Between VMs and a remote server connected over 1 Gbps path. For the experiment, a patched version of *Yaz* was used. To achieve specified data rate, this version uses a smaller inter-packet spacing than the original one. The reduced inter-packet spacing resulted in the AvB tool using inaccurate sleep time for inter-packet interval setting. As mentioned above, the concept of the algorithm is based on the iterative adjustments of inter-packet time interval until the maximal data rate is achieved, which also corresponds to the available bandwidth of the path. The accuracy of AvB tool was predicted to suffer from the algorithm using cross traffic set as 500 Mbps in this case.
3. Experiments were performed between a VM and host; and between two hosts to compare the virtualization impact on estimation accuracy. Servers were connected over 10 Gbps path with injected 20 ms RTT and 1 Gbps cross traffic. Measurements were performed using *Kite2* tool.

2.3. Measurement Methodology

In the first step, the experiment evaluated the cost of setting a timer: the time required to obtain a time value from the timing hardware. We use the term timer cost as a measuring parameter here. Secondly, the measurements of sleeping operations were performed and the difference between the targeted wake-up time and actual times of wake-ups (referred to a *miss time*) was calculated. For measuring both the timer cost and miss time, using the standard Linux *System Call* was compared with gathering the counter cycles from timer hardware. As an interface to available time counters, the *HighPerTimer* library was applied. It was used to obtain access to the *TSC* timer on the current processor from user-space and provides a mechanism of precise sleep operations. The sleep function from *HighPerTimer* library (*HPTSleep*) allows significantly decreasing the overhead of miss time without starving the CPU. In the present research, the median and MAD (Median Absolute Deviation) were chosen as the main statistical parameters. Since the data follow a non-parametrical distribution, the median value carries more information about distribution than the mean value. The MAD is calculated as half the interquartile dataset range and can be treated as non-parametric analogous to standard deviation. In the next sections, two types of plots are shown: plots with raw data with bold lines that are the medians, and plots with the representation of complementary cumulative distribution function (CCDF) with dashed lines that are the means. CCDF is a distribution function that shows how often the random variable is measured above a particular level. Additionally, the mean values are illustrated within CCDF plots, which allow estimating how mean values deviate from medians. Every experimental result was analyzed with non-parametrical variance analysis Kruskal–Wallis ranking test [25] (or ANOVA on ranks) and posthoc test using ranking analogous of Tukey-HSD [26],

with a significance level 0.05, to determine the influence of factors, such as timing operations (*TSC*, *clock_gettime()*, *HPTSleep*, and *uSleep()*), virtual platform and load type.

3. Results Analysis

In the following section, the experimental results of time measurements, sleep measurements, and AvB estimation accuracy with different timing operations are presented.

3.1. Experimental Results for Time Measurements

Time measurements were performed with *TSC* timers and *clock_gettime()* System Call in three scenarios: Host OS, where the operating system is placed on bare hardware; VMware ESXi and Xen for hypervisor-based virtualization; and Virtual Box and QEMU for virtual monitor based virtualization.

3.1.1. Idle System Test Results

Initially, all measurements were performed in idle state to compare the results with heavy loads on the next stages. As shown in Figure 4, median values on Host OS, Xen, QEMU, and ESXi are almost the same in the *TSC* timers case (bold red lines). In all cases, *TSC* timers show better performance than the *System Call*. As expected, the Host OS has the lowest time overhead (Figure 4a), with a median about 17.2 ns when using the *TCS* timer respective 27 ns in the *System Call* case. The next hypervisor by time overhead is Xen, shown in Figure 4b, which has 18.6 ns median values with *TSC* timer; however, the *System Call* shows quite high costs with a median at 119 ns, which is the second highest value, below Virtual Box. VMware ESXi (Figure 4e) has slightly higher median values of 20.1 ns for *TSC* and a fairly low median value of 34 ns with the *System Call*. QEMU, shown in Figure 4d, has an insignificantly higher median value for *TSC* with 21.5 ns and also 57 ns median values for the *System Call*. Thus, even virtual monitor based virtualization can pass timing requests very efficiently to the underlying hardware. However, another representative of virtual monitors, Virtual Box (Figure 4c), has a much greater median overhead in comparison to Xen, ESXi, QEMU virtualized OS and Host OS: 2.12 µs for *TSC* timer and almost the same 2.18 µs for the *System Call*. Additionally, Virtual Box has the largest spread of values, up to 1 ms for the max value and 1 ns for the minimum value. Considering the CCDFs of given results presented in Figure 5, it can be assumed that data from all sources, except for Virtual Box (blue and red lines), have quite similar distributions and their *TSC* timer without load can guarantee overhead not higher than 27 ns with a probability of 0.999. Comparing the results of forwarding the *RDTSC* instruction on Xen, VMware ESXi and QEMU, Xen shows less overhead on the translation. Considering Virtual Box, it has the longest tails, up to 1 ms for *System Call* and up to 2 ms for *TSC* timer.

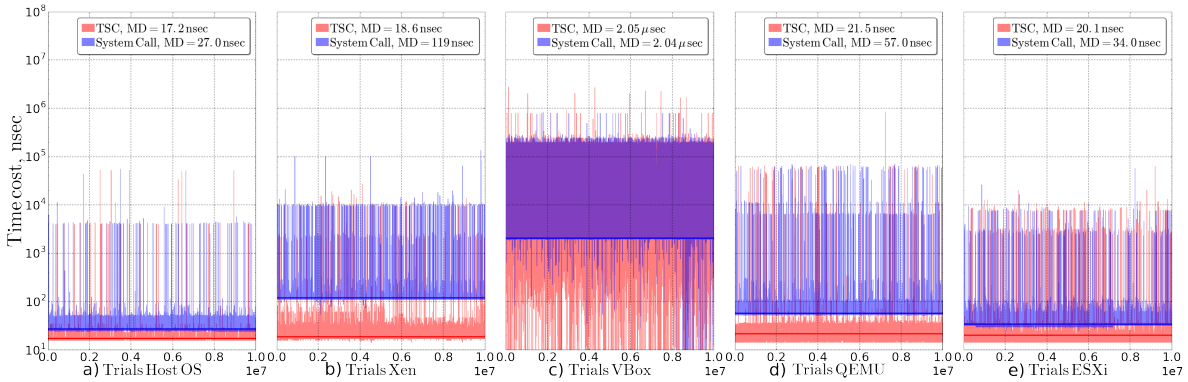


Figure 4. Measurements of timer cost for the Idle testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

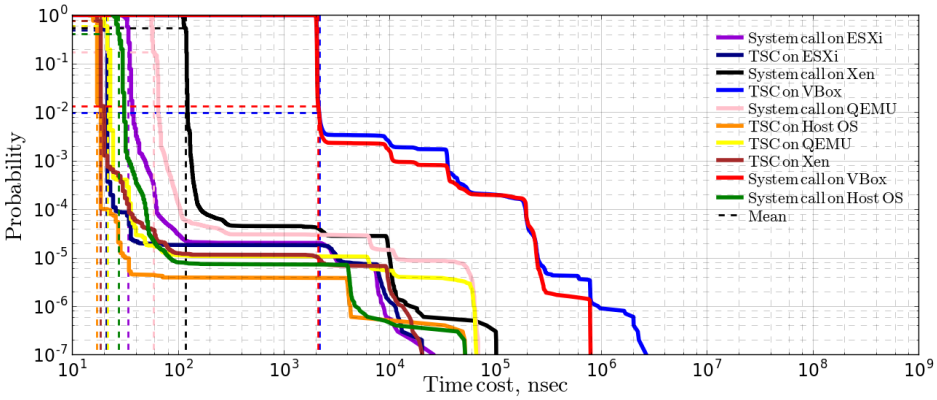


Figure 5. CCDF representation of measurements of time cost for the idle testbed.

3.1.2. CPU-Bound Load Test Results

CPU-bound background workload can be expected to result in worsened latencies due to limited hardware resources such as memory bandwidth and shared caches [20]. The results of the experiment with CPU-bound load are shown in Figures 6 and 7. They show that there is no big effect on median values in comparison to the Idle system testbed. However, it causes significantly longer tails in the CCDF plot (Figure 7), which imply significant influence on the mean of the distributions (dashed lines). The means are shifted to the right, to the region of 1 µs. However, the 99.9% percentile for the *TSC* based timer overhead is, as in the idle system case, about 27 ns for all virtualization platforms, besides the Virtual Box case, for which the value is approximately 2 µs. The shape of CCDF for this experiment is quite similar to the idle case, however this load affects mostly on rare and heavy tails.

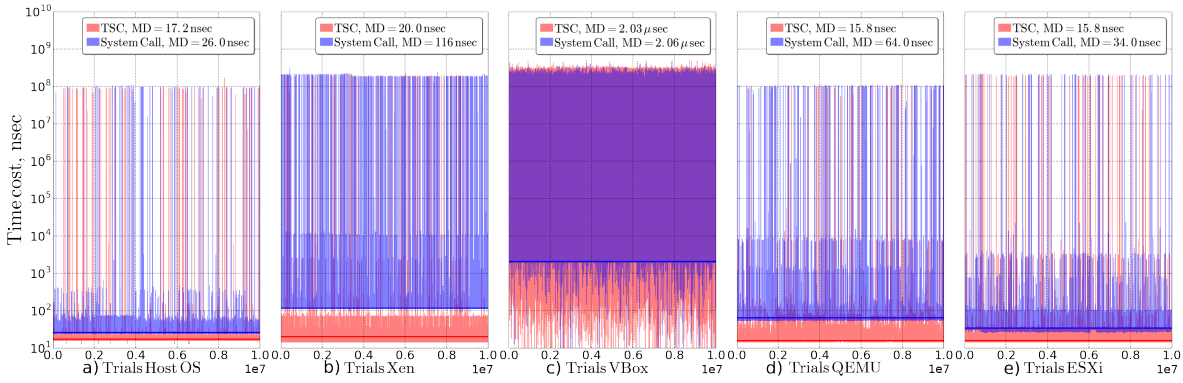


Figure 6. Measurements of timer cost for the CPU testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

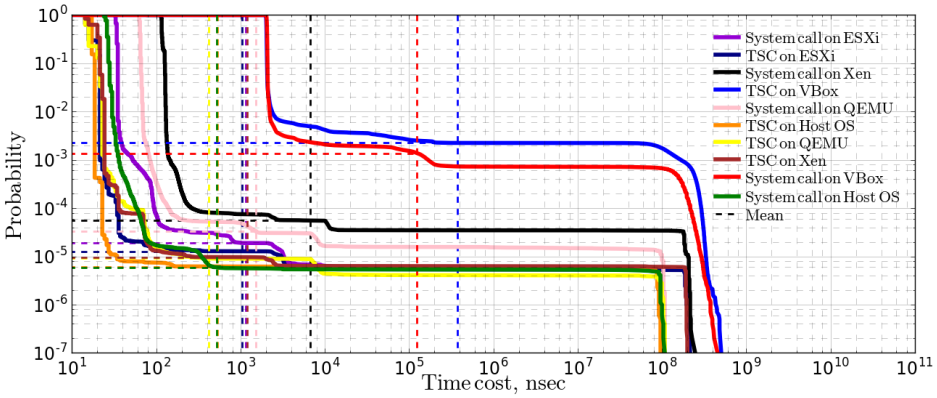


Figure 7. CCDF representation of measurements of time cost for the CPU testbed.

3.1.3. IO/Network-Bound Load Test Results

Further, the impact of interrupt handling caused by IO-bound tasks, such as network load, etc., is illustrated in Figures 8 and 9. Time values from *TSC*-based time operations show almost native overhead on all platforms, except Virtual Box. Although VMware ESXi has median values (Figure 8c) of data acquired with *TSC* timer higher than on Host OS, Xen, and QEMU, it has significantly lower spread than on QEMU, Host OS and Xen. This also can be seen in Figure 9, where the tails of distributions from VMware ESXi datasets indicate lower time cost values than in other cases. Nevertheless, the virtualized *TSC* timer on QEMU shows an unexpected advantage: its median overhead value is even lower than the median value obtained on the non-virtualized hardware on which QEMU is placed. This can be seen in the CCDF plot (Figure 9 yellow line), where 90% of outcomes are concentrated below 17 ns and past this value the distribution line falls steeply and then follows the behavior of other *TSC* distributions. For Xen and Virtual Box, tails are much longer. At the same time, the probability of having time cost values greater than median value is greater for all cases; however, median value occurs with a probability around 0.99. The impact of IO/network load can be characterized by a long tail in the middle area of the CCDF plot, which means higher variance of measurements. This effect appears mostly on Virtual Box and Xen.

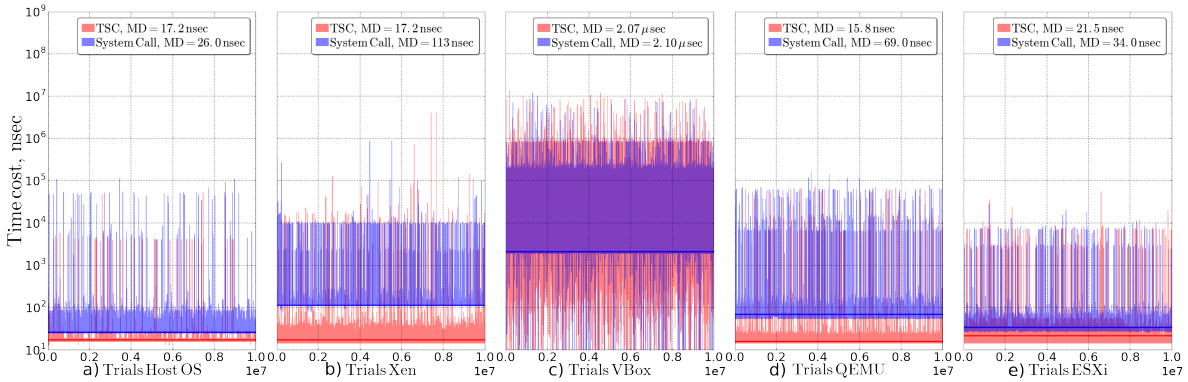


Figure 8. Measurements of timer cost for the IO/network testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

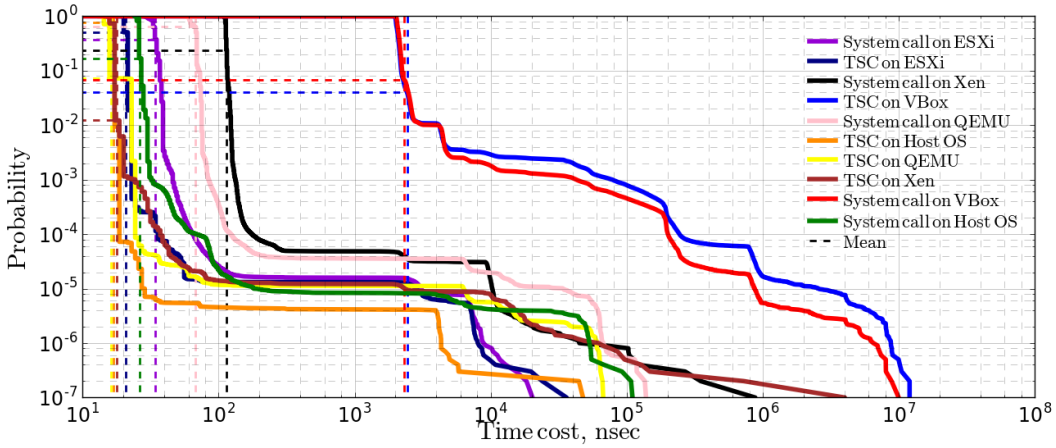


Figure 9. CCDF representation of measurements of time cost for the IO/network testbed.

3.1.4. Summary of Time Acquisition Experiment Results

The results present above are collected in Table 1. It consists of main statistical parameters of experiment datasets: median value, MAD, minimal and maximal values of the obtained results. Statistical parameters are provided for the different conditions of the experiments: investigated function (*TSC* or *System Call*), platform or experimental environment (Host OS, QEMU, ESXi, Xen, and

Vbox) and the load type under which the experiment was performed (idle System, CPU-bound, and IO/network-bound).

Table 1. Results for all experiments with timing functions.

Function	Platform	Load	MAD, ns	Max, ns	Median, ns	Min, ns
TSC	ESXi	CPU-bound	2.9	216,009,000.0	15.8	14.3
TSC	ESXi	IO/network-bound	0.7	55,265.8	21.5	14.3
TSC	ESXi	Idle System	0.7	62,208.5	20.1	14.3
TSC	HostOS	CPU-bound	0.0	168,004,000.0	17.2	12.9
TSC	HostOS	IO/network-bound	0.0	51,820.0	17.2	12.9
TSC	HostOS	Idle System	0.0	51,732.5	17.2	15.8
TSC	QEMU	CPU-bound	0.7	116,009,000.0	15.8	14.3
TSC	QEMU	IO/network-bound	0.7	73,509.8	15.8	14.3
TSC	QEMU	Idle System	0.7	818,343.0	21.5	14.3
TSC	VBox	CPU-bound	12.9	507,099,000.0	2033.0	0.7
TSC	VBox	IO/network-bound	57.6	13,733,500.0	2070.0	1.1
TSC	VBox	Idle System	10.9	2,744,610.0	2053.9	1.1
TSC	Xen	CPU-bound	2.9	200,008,000.0	20.0	14.3
TSC	Xen	IO/network-bound	0.7	4,046,770.0	17.2	14.3
TSC	Xen	Idle System	0.0	27,250.5	18.6	14.3
System Call	ESXi	CPU-bound	1.0	212,006,000.0	34.0	25.0
System Call	ESXi	IO/network-bound	1.0	28,608.0	34.0	25.0
System Call	ESXi	Idle System	1.0	56,966.0	34.0	25.0
System Call	HostOS	CPU-bound	0.5	108,005,000.0	26.0	24.0
System Call	HostOS	IO/network-bound	0.0	112,208.0	26.0	24.0
System Call	HostOS	Idle System	1.5	55,249.0	27.0	24.0
System Call	QEMU	CPU-bound	1.0	108,022,000.0	64.0	51.0
System Call	QEMU	IO/network-bound	3.5	167,598.0	69.0	52.0
System Call	QEMU	Idle System	1.0	69,456.0	57.0	51.0
System Call	VBox	CPU-bound	14.5	464,957,000.0	2060.0	1.0
System Call	VBox	IO/network-bound	57.5	12,064,200.0	2104.0	1.0
System Call	VBox	Idle System	13.0	826,773.0	2042.0	2.0
System Call	Xen	CPU-bound	1.5	296,001,000.0	116.0	80.0
System Call	Xen	IO/network-bound	0.5	885,662.0	113.0	78.0
System Call	Xen	Idle System	3.5	133,775.0	119.0	54.0

To identify hidden factors and determine whether a given set contains any additional properties, an analysis of variance (ANOVA on ranks) was performed. This test was targeted to determine which factors significantly affect the four parameters of the dataset: median, MAD, max, and min. Considering function, load and platform columns of the table as the different factors, Kruskal–Wallis analysis on ranks and post hoc tests using Dunn's test of timer cost datasets were used. The mentioned methods allow the investigation of the effects of multiple factors on the statistical parameters: the median value of the dataset, MAD, and its minimum and maximum values.

Variance analysis of medians, MADs, minimum and maximum values shows that time acquisition function is a significant factor for all virtualization types. However, virtualization itself is a significant factor only on the Virtual Box as a virtualization platform. Load type has a significant effect only on the MAD parameter, however MAD changes insignificantly switching from Idle to CPU load. Interaction between factors function and platform affects significantly median, max, and min values, but not MAD. Interaction between factors platform and load affects only MAD.

There is no significant difference for median, MAD, max, and min values when switching between Host OS and ESXi platforms, ignoring other factors. Instead, much more effect was evident in switching from *TSC* to *System Call*. It significantly affects the median, max values on Xen and QEMU platforms as well as in the case of idle system, CPU-bound and IO/network-bound load.

Analysis of interaction effects between one platform-load and other platform-load factor shows that a significant effect is present in the following cases: idle system switching from Xen to Host OS or

to ESXi has an effect on median values, while switching from Virtual Box to any other platform has effects on median, MAD, min and max.

Under CPU-bound or IO/network-bound load, switching from Virtual Box to any other platforms significantly affects all statistical parameters of the datasets: median, MAD, min, and max. Switching from Host to Xen affects the median, min, and max, but does not affect the statistical parameters of timers on ESXi platform. At the same time, switching from Xen to ESXi affects the median, min, and max.

Based on this and previous analysis, it can be said that, firstly, there is a statistically significant difference between time cost values gathered using *System Call* and *TSC* timer. Secondly, in all cases, Virtual Box affects negatively and statistically significant on timer cost values. This information can be considered by choosing the virtualization environment, depending on which time parameters are the more critical for the certain application.

3.2. Experimental Results for Sleep Measurements

In a next series of experiments, sleep precision measurements were performed by comparing the wakeup precision of the HighPerTimer library sleep function and the wakeup precision of *uSleep()* call of the standard C library in three scenarios: Host OS, VMware ESXi and QEMU. Experiments with the sleep operation from HighPerTimer library are called *HPTSleep* in this paper. Experiments with *uSleep()* form standard C library are referred to as *uSleep()*.

3.2.1. Idle System Test Results

As in the previous section, measurements were performed on an idle system to establish a baseline when comparing the different load scenarios. As shown in Figure 10, median values of miss times during *HPTSleep* function on all platforms, except Virtual Box, are in the same range between 80 and 116 ns. The invocation of *uSleep()* function shows miss times that are almost three orders of magnitude greater than *HPTSleep* function: 62 µs compared to 82 ns on Host OS (Figure 10a); 109 µs compared with 94.5 ns on Xen (Figure 10b); 131 µs compared with 5.12 µs on Virtual Box (Figure 10c); 97 µs compared with 116 ns on QEMU (Figure 10d); and 63.1 µs compared with 95 ns on VMware (Figure 10e). Measurements with *HPTSleep* operations are distributed near to log-normal distribution for Host OS, Xen, QEMU and ESXi, which is shown below in a CCDF plot. Figure 11 shows that the respective distributions of the data are grouped together by the chosen sleep method. *HPTSleep* for Xen, Host OS, ESXi, and QEMU are situated in the left part of the plot near median values, which is an indication of log-normality.

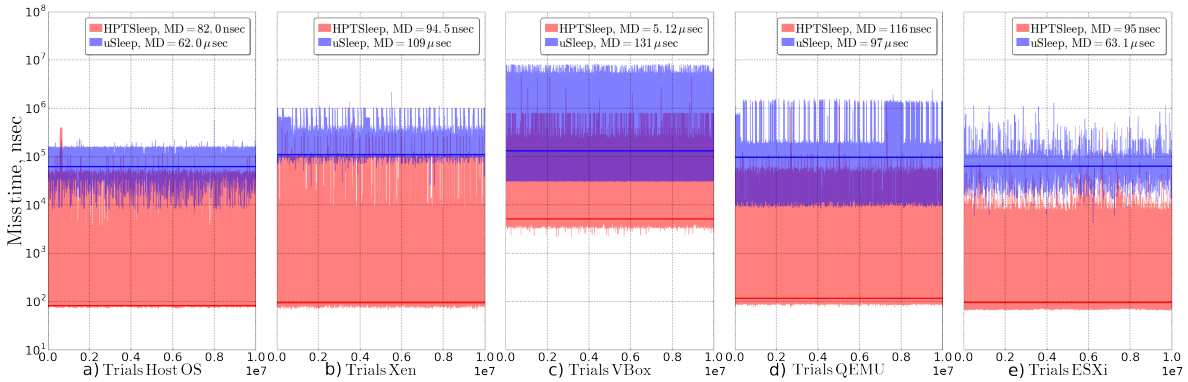


Figure 10. Measurements of miss time for the Idle testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

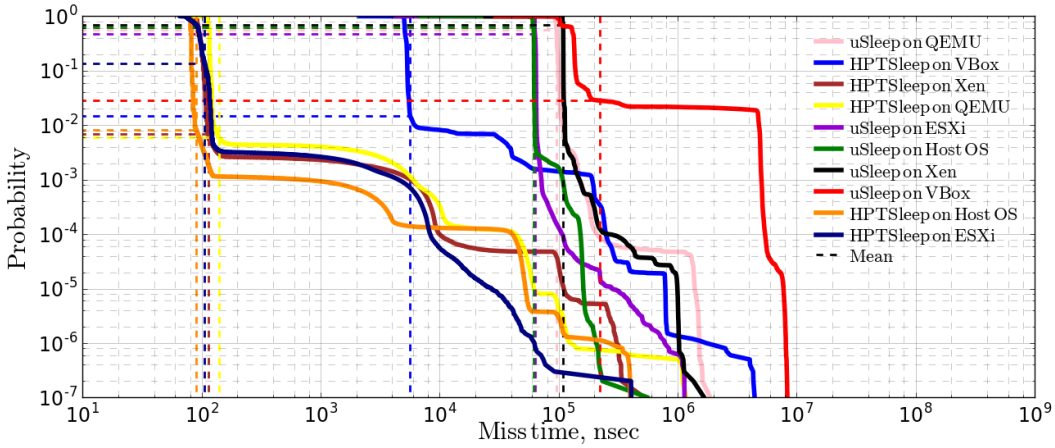


Figure 11. CCDF representation of measurements of miss time for the idle testbed.

3.2.2. CPU-Bound Load Test Results

The distributions for the CPU-bound load are very similar to their respective distributions in the idle system scenario. The distributions (Figure 12) of *HPTSleep* miss, as well as their medians, are in the same range on all five platforms: Host OS has a 102 ns median value of miss time (Figure 12a); Xen virtualized OS has 94.5 ns (Figure 12b); Virtual Box has 5.18 µs (Figure 12c); QEMU-based OS has 137 ns (Figure 12d); and VMware ESXi has 114 ns (Figure 12e). Distributions of data with *uSleep()* function are also close to each other: 54.7 µs on Host OS, 99 µs on Xen, 61.3 µs on QEMU and 56 µs on VMware ESXi hypervisor-based OS, except Virtual Box with 7.6 ms. Considering the CCDF in Figure 13, *HPTSleep* produces much longer tails than *uSleep()*. Mean values are grouped together in the range between 0.1 and 1 ms, except the Virtual Box *uSleep()* case, while median values are around 100 ns in the case of *HPTSleep* (7.6 ms for Virtual Box) and 50 µs in case of *uSleep()*. However, median values of *HPTSleep* differ by 15% max, while medians of *uSleep()* function differ by up to 45% (ignoring Virtual Box). In addition, the CCDF shows that outcomes of the *HPTSleep* function have a miss time below 120 ns with a probability 0.999 for HostOS, Xen, QEMU, and ESXi. *HPTSleep* for Virtual Box keeps to the right, because of the higher order of values in comparison to Host OS and Xen. QEMU and VMware ESXi behave almost the same as Host OS and Xen.

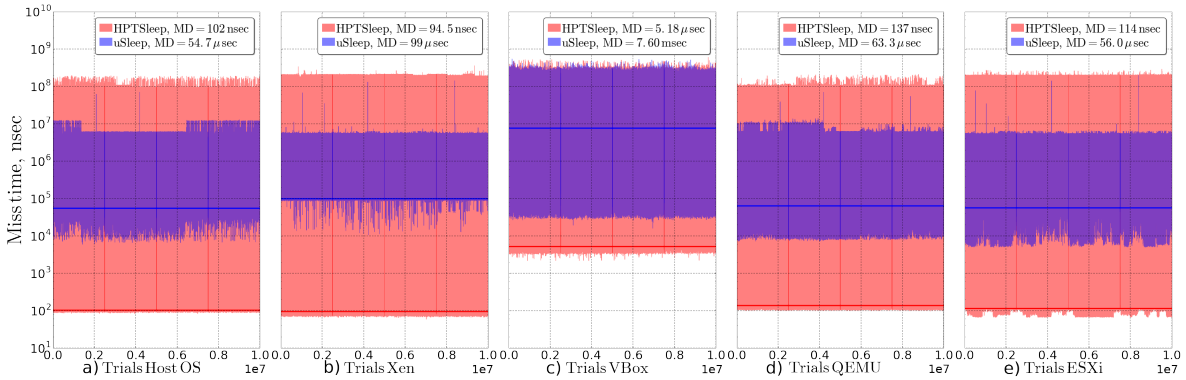


Figure 12. Measurements of miss time for the CPU testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

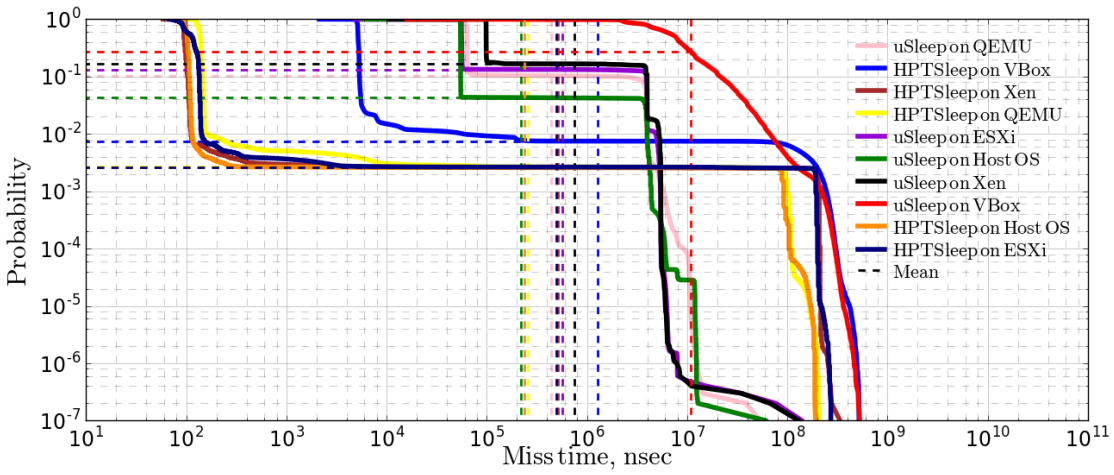


Figure 13. CCDF representation of measurements of miss time for the CPU testbed.

3.2.3. IO/Network-Bound Load Test Results

The results of this experiment show that medians of sleep misses datasets obtained with *HPTSleep* are significantly lower than using *uSleep()* function (tens of nanoseconds with *HPTSleep* versus tens of microseconds with *uSleep()*). All examined platforms exhibit this behavior. VMware ESXi shows a lower spread than other platforms in the case of *HPTSleep* function (Figure 14e). Xen has the highest spread with *HPTSleep* function among all the platforms (Figure 14b). Virtual Box has a relatively small spread with *HPTSleep* function, however max values are near to Xen's maximums (Figure 14c). Xen has the smallest spread for *uSleep()*. Median for *HPTSleep* values on all platforms stay the same as in previous experiments, including the idle system tests. However, median values for *uSleep()* have the same order of median values among all platforms: 57.8 µs for Host OS, 108 µs for Xen, 131 µs for Virtual Box, 74.5 µs for QEMU, and 62.2 µs for ESXi.

The CCDF plot in Figure 15 shows that IO/network-bound load has the greatest effect on the tails of distributions, while the shapes of those tails do not have stable patterns as in previous cases. This might mean that different virtualization platforms handle IO interrupts differently.

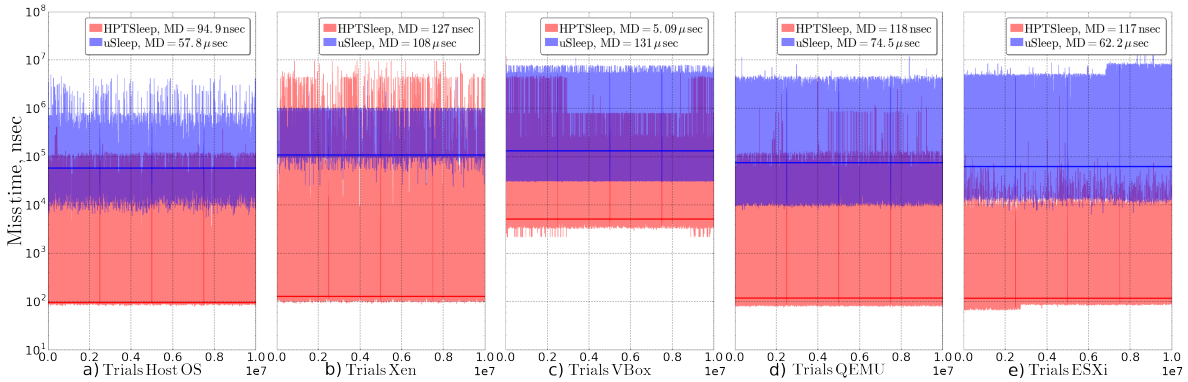


Figure 14. Measurements of miss time for the IO/network testbed on: (**a**) Host OS; (**b**) Xen; (**c**) Virtual Box; (**d**) QEMU; and (**e**) VMware ESXi.

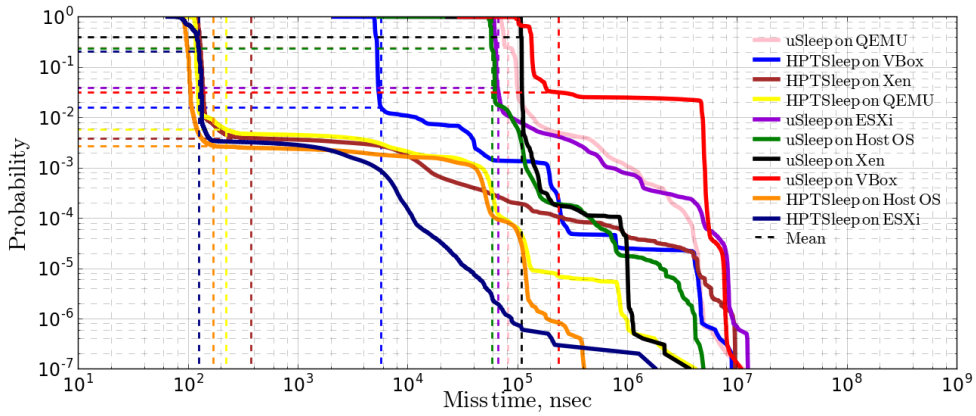


Figure 15. CCDF representation of measurements of miss time for the IO/network testbed.

3.2.4. Summary of Sleep Functions Experiment Results

The results presented above are collected in Table 2. It consists of the main statistical parameters of experiment datasets: median, MAD, minimum and maximum values. Statistical parameters are provided for the different conditions of the experiments: investigated function (*HPTSleep* or *uSleep()*), platform or experimental environment (Host OS, QEMU, ESXi, Xen, and Vbox) and the load type under which the experiment was performed (idle System, CPU-bound, and IO/network-bound).

Table 2. Results for all experiments with sleep functions.

Function	Platform	Load	MAD, ns	Max, ns	Median, ns	Min, ns
HPTSleep	ESXi	CPU-bound	18.6	287,997,000.0	113.9	66.6
HPTSleep	ESXi	IO/network-bound	14.3	2,024,870.0	116.7	65.2
HPTSleep	ESXi	Idle System	6.4	432,825.0	95.2	65.2
HPTSleep	Host OS	CPU-bound	3.6	196,002,000.0	102.4	82.3
HPTSleep	Host OS	IO/network-bound	2.9	407,121.0	94.9	79.1
HPTSleep	Host OS	Idle System	0.7	402,269.0	82.0	72.0
HPTSleep	QEMU	CPU-bound	2.9	208,008,000.0	136.8	96.7
HPTSleep	QEMU	IO/network-bound	0.7	4,351,030.0	117.8	74.8
HPTSleep	QEMU	Idle System	1.4	1,187,960.0	116.4	74.8
HPTSleep	VBox	CPU-bound	76.1	596,501,000.0	5183.1	2161.1
HPTSleep	VBox	IO/network-bound	125.0	12,048,600.0	5087.4	2134.6
HPTSleep	VBox	Idle System	121.1	4,697,970.0	5117.3	2142.5
HPTSleep	Xen	CPU-bound	2.9	355,964,000.0	94.5	58.7
HPTSleep	Xen	IO/network-bound	3.6	9,644,930.0	127.5	87.4
HPTSleep	Xen	Idle System	4.3	573,593.0	94.5	67.3
uSleep	ESXi	CPU-bound	294.3	196,105,000.0	56,044.5	4883.0
uSleep	ESXi	IO/network-bound	1123.6	12,689,600.0	62,199.1	6365.4
uSleep	ESXi	Idle System	851.4	1,272,060.0	63,085.2	4300.1
uSleep	Host OS	CPU-bound	55.2	69,705,100.0	54,674.9	5918.4
uSleep	Host OS	IO/network-bound	792.7	4,971,040.0	57,756.1	5478.3
uSleep	Host OS	Idle System	680.0	689,113.0	61,996.3	6848.2
uSleep	QEMU	CPU-bound	290.0	69,372,000.0	63,264.2	7132.5
uSleep	QEMU	IO/network-bound	3738.6	11,674,900.0	74,540.7	7807.0
uSleep	QEMU	Idle System	1820.0	4,150,180.0	97,351.3	8167.9
uSleep	VBox	CPU-bound	3,611,915.2	539,619,000.0	7,601,230.0	15,913.7
uSleep	VBox	IO/network-bound	16,410.0	11,469,400.0	131,192.0	29,421.1
uSleep	VBox	Idle System	16,395.5	8,670,880.0	131,252.0	29,471.2
uSleep	Xen	CPU-bound	229.8	141,160,000.0	98,544.0	10,624.7
uSleep	Xen	IO/network-bound	935.0	3,924,120.0	107,852.0	23,060.8
uSleep	Xen	Idle System	260.0	2,176,090.0	109,411.0	24,399.8

Further, similar to the time acquisition function overhead, ANOVA on ranks analysis was performed to determine which factors affect significantly the same parameters of sleep miss values. Function, load and platform columns of the table are considered as the different factors. Kruskal–Wallis analysis on ranks and post hoc tests using Dunn's test of timer cost datasets were also used.

Kruskal–Wallis rank analysis of medians, MADs, minimum and maximum variables show that type of the sleep function is a significant factor for all virtualization types. As in the case of time acquisition experiments discussion, virtualization is a significant factor only on the Virtual Box as a virtualization platform. The load type only significantly affects the max values.

Interaction between factors function and platform significantly affects the median, MAD and min values, but not the max. Interaction between factors platform and load only affects max values. Interaction between factors function and load affects all parameters, unlike the case with time acquisition experiments.

There is no significant effect for median, MAD, max, and min values in the case of interaction between function and load when the function is the same; in all other cases, there is a significant effect.

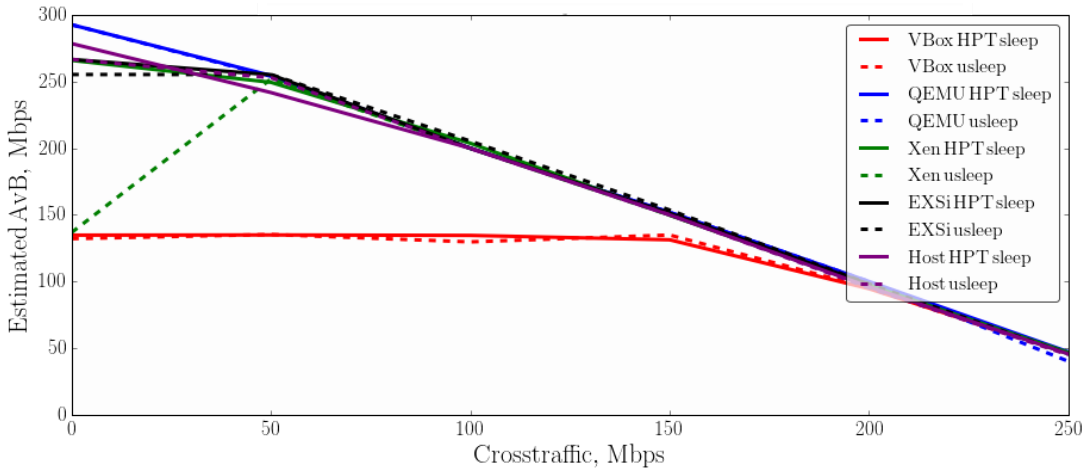
Therefore, the above analysis has proven that changing sleep function from *uSleep()* to *HPTSleep* has a significant effect on miss time values. The effect from changing sleep function was revealed on platform and load factors as well as on the interactions of them. Similar to the time acquisition function overhead, Virtual Box affects negatively and statistically significantly miss time values in all cases. This information can be considered when choosing the virtualization environment, depending on which time parameter is the more critical for the certain applications using sleep function.

3.3. AvB Estimation Results

The investigation of AvB estimation results with different types of timing operations was performed to assess the impact of the timer in the virtual environment on challenging tasks: the precision of the packet timestamping and therefore measurement accuracy, which has a direct impact of the data transport performance of data transport protocols.

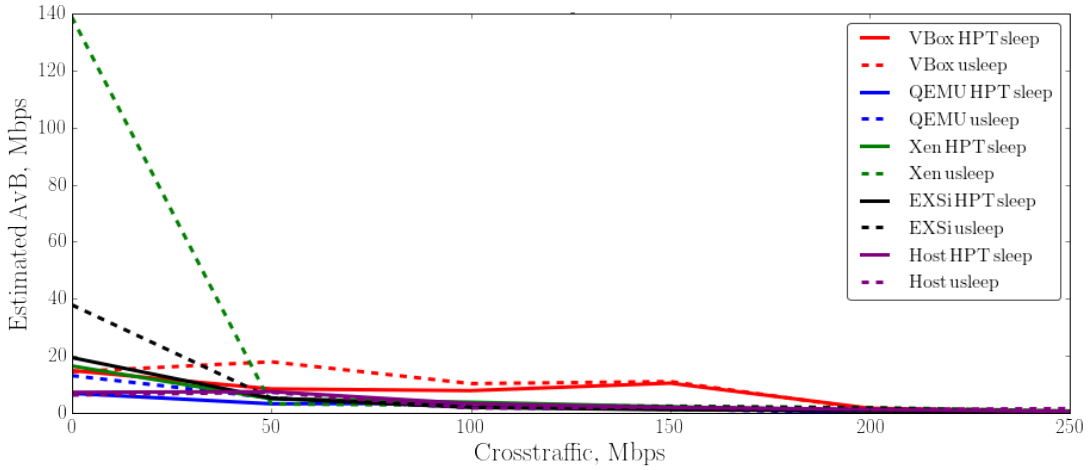
3.3.1. Experimental Results in 300 Mbps Network

The results of the experiments of AvB estimation in different virtual environments are depicted in Figure 16, where OS indicates estimated results obtained with system timers and *HPT* indicates those with *HighPerTimer*.



(**a**)

Figure 16. *Cont.*



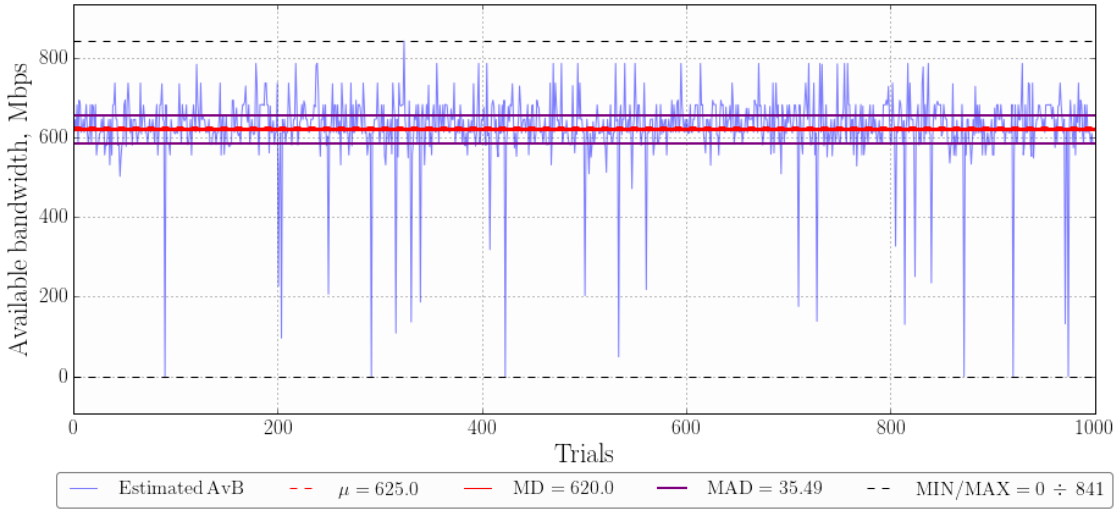
(**b**)

Figure 16. AvB estimation results by *Yaz* in Virtual Environment: (**a**) medians; and (**b**) variances.

As can be seen, MAD and median were calculated for six types of cross traffic on a 300 Mbps link for 250 estimation trials of the tool for each type. They show a strong decreasing tendency of the variance, and accordingly the median, of the estimated AvB with the growth of cross traffic presence in a path; however, in the case with standard timer acquisition, it always has higher variance in comparison to *HPT*. In the case of spare link, when AvB is equal to the path capacity, all types of virtualization have the highest impact on AvB estimation accuracy. Results with VM running on Xen virtualization have the highest variance. Such behavior was repeated several times when the tests were reloaded.

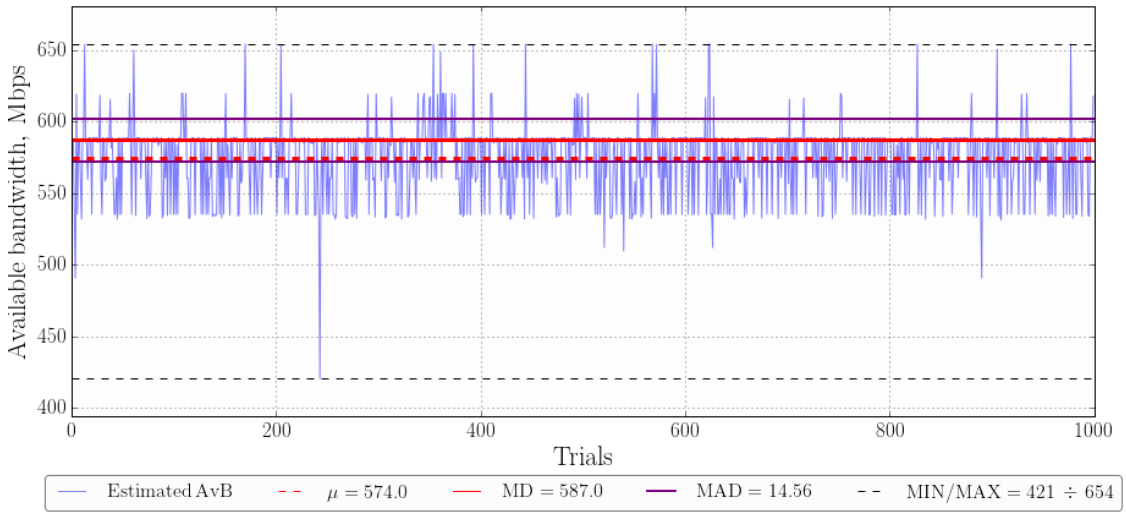
3.3.2. Experimental Results in 1 Gbps Network

From the previous experiment, estimations in the QEMU environment showed the most precise results. Test results of AvB estimation by *Yaz* on a QEMU VM with 1 Gbps link with approximately 500 Mbps cross traffic are shown in Figure 17.



(**a**)

Figure 17. *Cont.*



(**b**)

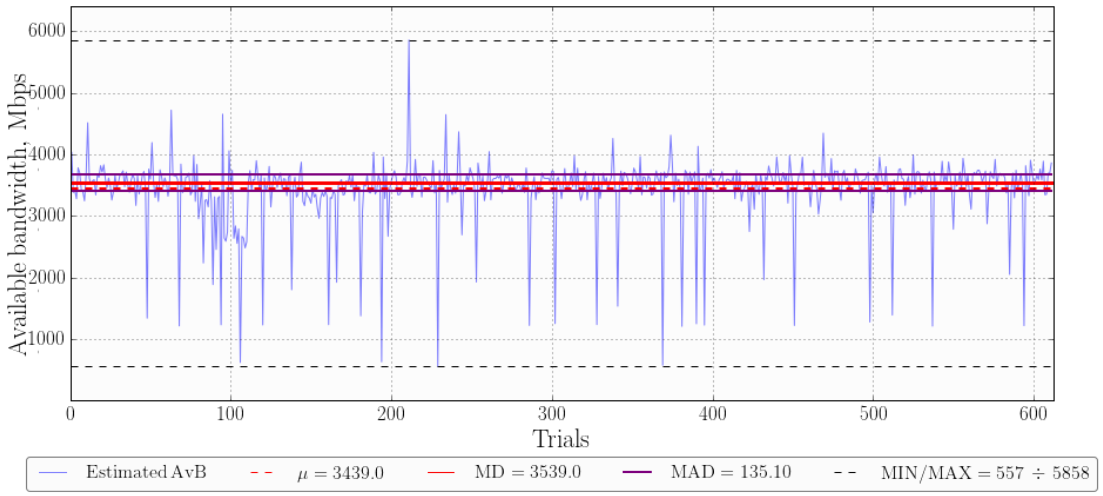
Figure 17. Yaz AvB estimation of 1 Gbps Network with 500 Mbps cross traffic: (**a**) standard system operations; and (**b**) with HighPerTimer.

It should be noted that such data rate cannot be achieved with the initial algorithm of *Yaz* due to the limitations based on the precision of the standard time acquisition and needed additional tuning of inter-packet spacing in the source code. Experiments show that MADs of estimations with *HPT* are less than half those with system timing operations. The rate of outliers is also noticeably higher for the standard way of time acquisition. The median in the case of *HPT* is equal to 587 Mbps and closer to the actual AvB in comparison to 620 Mbps with system timers.

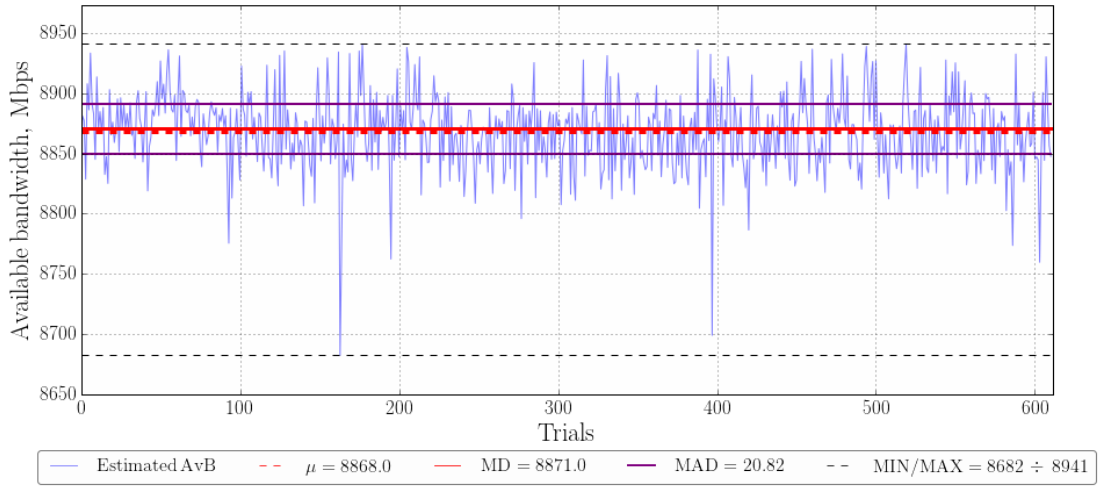
3.3.3. Experimental Results in 10 Gbps Network

The results of the third test scenario are presented in Figure 18. In this case, the 10 Gbps network with the presence of 1 Gbps cross traffic was analyzed only by *Kite2* tool since no other tools are able to produce that probe packet rate.

Hereby, another crucial limitation was revealed during the test using *Kite2* with the system timers. Using the system timing operations, *Kite2* was not able to exceed a packet sending data rate of 3.6 Gbps. However, with more efficient *HighPerTimer* (as shown in previous experiments), up to 8.9 Gbps could be estimated for the given link. On higher data rates (more than 1 Gbps), overhead of *System Call* becomes much more notable. Moreover, through the higher deviation, we can see the impact of an inaccurate *uSleep()* call. In both cases (Figure 18a,b), *Kite2* tended to underestimate the AvB, which is caused by its algorithm logic—to avoid overestimation and consequently overloads and packet loses. However, in the case of *HighPerTimer* estimation, mean and median are close to one another, which is an indicator of normal distribution. It is worth noting that estimation outliers without *HighPerTimer* are much more frequent and greater. The variance of 135.1 Mbps with standard timer in a contradiction to 20.82 Mbps of *HighPerTimer* points to the inconsistency of AvB measurements in high-speed networks using the first approach.



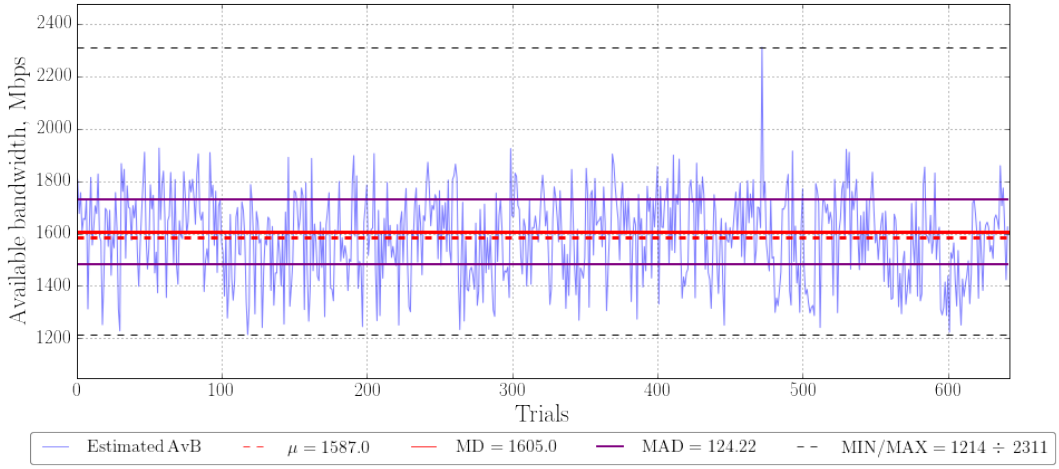
(**a**)



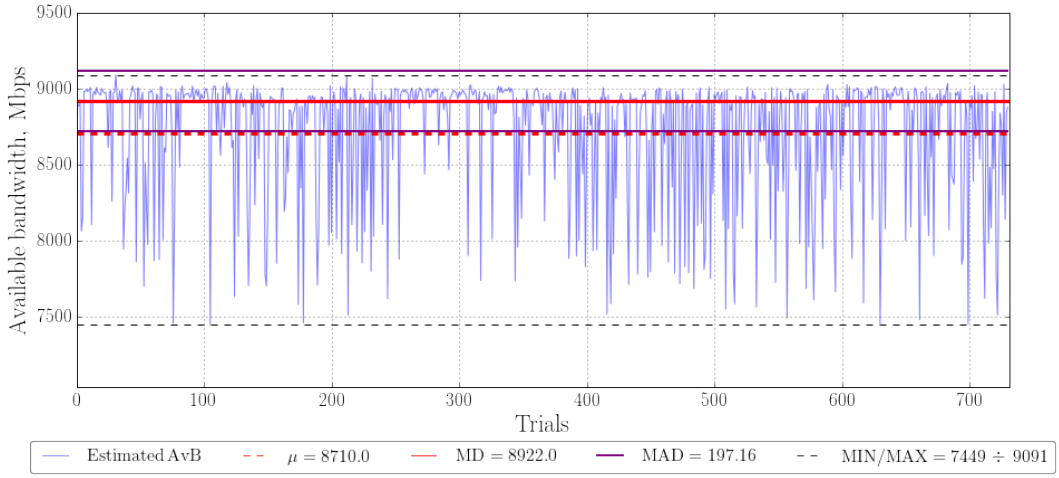
(**b**)

Figure 18. Kite2 AvB estimation of 10 Gbps Network on a host with 1 Gbps cross traffic: (**a**) standard system operations; and (**b**) with HighPerTimer.

Previous results show that the most precise timing operations have been achieved in Qemu environment. In Figure 19, the results of AvB estimations on a VM with Qemu virtualization are presented. They show significantly higher underestimation outliers in a contradiction to the estimations on a host, but the median values for the case with HighPerTimer timer are close to one another—8871 Mbps and 8922 Mbps, respectively. For the case of standard timing operations, the send data rate of a tool could not exceed 1.9 Gbps, which led to 82% of estimation error. The MAD value for the case with HighPerTimer increased to 197.16 Mbps in a Qemu environment and is almost 10 times higher than in the case when the experiments were performed on a host. AvB estimation results of both experiments—on a host and in a virtual environment—confirm that virtualization has a significant negative impact on the network application performance. The experiments with AvB tools with standard timing operations have shown their absolute inconsistency for the 10 Gbps networks. Hence, we can conclude that a low overhead of time operations and therefore timestamping on incoming and outgoing packets are crucial points in available bandwidth estimations algorithms in a virtual environment and increases its significance with increasing data rates.



(**a**)



(**b**)

Figure 19. Kite2 AvB estimation of 10 Gbps Network in Qemu environment with 1 Gbps cross traffic: (**a**) standard system operations; and (**b**) with HighPerTimer.

4. Discussion

The performed research and achieved results led us to conclusions regarding such challenges as selection of the timing approach for the high-performance network application, of a suitable virtualization method and available bandwidth estimation for networks with different path capacities. Various timers-related measurement results for the case where one virtual instance is located on hardware allow the following conclusions: (i) While Xen and ESXi, which are both bare-metal hypervisor, have almost no basis overhead in the majority of measurements, Virtual Box's access to the hardware adds the highest overhead among all virtualization platforms (QEMU, Xen, and VMware ESXi). (ii) In the case of sleep accuracy, the medians of sleeps methods prevail over the virtualization environment effects, except Virtual Box, when miss time median values are 100 times higher than other virtualizations in the case of *HPTSleep*. (iii) The hypervisor-based approach brings clearly measurable benefits compared with the OS-based one. (iv) Different loads affect mostly the variance of measurements and maximum values, while median and minimum values do not change dramatically. (v) Virtual Box has the highest median time overheads and the highest deviation. In addition, based on minimal overhead values, it makes sense to assume that this platform does not provide direct access to hardware. Thus, this platform is not recommended for time-critical applications. (vi) The AvB estimation measurement on Virtual Box reveals that, independently from timing operations, such

type of virtualizations is inapplicable for high-speed applications implementations, due to more than 50% of underestimation error. Nevertheless, even in the case of Virtual Box, *TSC*-based timers and *HPTSleep* are able to decrease estimation variance. (vii) The most accurate AvB estimation results were achieved on QEMU OS-based virtualization—in any tested condition, precision and variance of AvB evaluation prove themselves as the most faithful. (viii) Experiments show that the MAD of AvB estimations with *HPT* are less than half those of system timing operations. The rates of outliers are also noticeably higher for the standard timing functions in 1 Gbps networks. (ix) Due to the revealed limitations cause by *uSleep()* miss time of 62 µs, the maximal achieved send data rate in 10 Gbps networks did not exceed 3.6 Gbps, which caused more than 60% AvB estimation error and points to the inapplicability of standard timing operations in term of modern high-speed networking. Thus, the presented approaches of timing and AvB application can achieve the maximum performance in the researched virtual environments. Moreover, they are both software solutions which do not require significant hardware or software modifications of the Linux-based system structure. In the case of HighPerTimer library implementation, costs are limited to the presence of *RDTSCP* support in the CPU. In addition, it should be taken into account that the precision of sleep function achieved by busy-wait is in the range of microseconds. The pitfalls of *Kite2* are in its network intrusiveness by estimation samples. Despite this, it uses an adaptive train size to reduce traffic generated by probes. Thereby, the low-resolution timing problem in a virtual environment caused by the precision being lower than that required by high-performance network application can be overcome by using direct access to TSC and manipulation with busy-wait while sleep operations implemented in HighPerTimer library. Therefore, it allows availing precise packet timestamping to analyze high-speed networks with capacities up to 120 Gbps, which is not possible with system timers.

Author Contributions: Conceptualization, E.S., K.K. and V.K.; methodology, K.K. and V.K.; software, K.K., V.K., I.Z., D.K. and S.M.; validation, D.K.; formal analysis, K.K. and O.V.; investigation, K.K. and V.K.; resources, E.S. and S.M.; data curation, K.K. and V.K.; writing—original draft preparation, K.K., V.K. and I.Z.; writing—review and editing, I.Z., E.S., D.K.; visualization, K.K.; supervision, E.S.; project administration, E.S.; and funding acquisition, E.S.

Funding: This research was funded by Volkswagen Foundation grant number 90338 for trilateral partnership between scholars and scientists from Ukraine, Russia and Germany within the project CloudBDT: Algorithms and Methods for Big Data Transport in Cloud Environments.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CCDF	Complementary Cumulative Distribution Function
VM	Virtual Machine
Vbox	Virtual Box
HPT	High-Per-Timer
OS	Operating System
MD	Median
MAD	Median absolute deviation
RTT	Round Trip-Time Delay
AvB	Available bandwidth
GE	Gigabit Ethernet
PRM	Probe Rate Model
NIC	Network Interface Card
TSC	Time Stamp Counter
VDSO	Virtual Dynamic Shared Object
RDTSCP	Read Time-Stamp Counter and Processor ID

References

1. Kipp, S. The 2019 Ethernet Roadmap. Available online: https://ethernetalliance.org/technology/2019-roadmap/ (accessed on 16 August 2019).
2. Wang, H.; Lee, K.S.; Li, E.; Lim, C.L.; Tang, A.; Weatherspoon, H. Timing is Everything: Accurate, Minimum Overhead, Available Bandwidth Estimation in High-speed Wired Networks. In Proceedings of the 2014 Conference on Internet Measurement Conference—IMC'14, Vancouver, BC, Canada, 5–7 November 2014; pp. 407–420. [CrossRef]
3. Karpov, K.; Fedotova, I.; Siemens, E. Impact of Machine Virtualization on Timing Precision for Performance-critical Tasks. *J. Phys. Conf. Ser.* **2017**, *870*, 012007. [CrossRef]
4. Karpov, K.; Fedotova, I.; Kachan, D.; Kirova, V.; Siemens, E. Impact of virtualization on timing precision under stressful network conditions. In Proceedings of the IEEE EUROCON 2017 17th International Conference on Smart Technologies, Ohrid, Macedonia, 6–8 July 2017; pp. 157–163.
5. Korniichuk, M.; Karpov, K.; Fedotova, I.; Kirova, V.; Mareev, N.; Syzov, D.; Siemens, E. Impact of Xen and Virtual Box Virtualization Environments on Timing Precision under Stressful Conditions. *MATEC Web Conf.* **2017**, *208*, 02006. [CrossRef]
6. Fedotova, I.; Siemens, E.; Hu, H. A high-precision time handling library. *J. Commun. Comput.* **2013**, *10*, 1076–1086.
7. Fedotova, I.; Krause, B.; Siemens, E. Upper Bounds Prediction of the Execution Time of Programs Running on ARM Cortex—A Systems. In *Technological Innovation for Smart Systems*; IFIP Advances in Information and Communication Technology Series; Camarinha-Matos, L.M., Parreira-Rocha, M., Ramezani, J., Eds.; Springer International Publishing: Cham, Switzerland, 2017; pp. 220–229.
8. Kirova, V.; Siemens, E.; Kachan, D.; Vasylenko, O.; Karpov, K. Optimization of Probe Train Size for Available Bandwidth Estimation in High-speed Networks. *MATEC Web Conf.* **2018**, *208*, 02001. [CrossRef]
9. Sommers, J.; Barford, P.; Willinger, W. A Proposed Framework for Calibration of Available Bandwidth Estimation Tools. In Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06), Cagliari, Italy, 26–29 June 2006; pp. 709–718. [CrossRef]
10. Kachan, D.; Kirova, V.; Korniichuk, M. *Performance Evaluation of PRM-Based Available Bandwidth Using Estimation Tools in a High-Speed Network Environment*; Scientific Works of ONAT Popova, Odessa, Ukraine, 2017; pp. 160–168. Available online: http://irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/Nponaz_2017_2_24.pdf (accessed on 16 August 2019).
11. Goldoni, E.; Schivi, M. End-to-End Available Bandwidth Estimation Tools, An Experimental Comparison. In Proceedings of the International Workshop on Traffic Monitoring and Analysis, Zurich, Switzerland, 7 April 2010; Volume 6003, pp. 171–182. [CrossRef]
12. Aceto, G.; Palumbo, F.; Persico, V.; Pescapé, A. An experimental evaluation of the impact of heterogeneous scenarios and virtualization on the available bandwidth estimation tools. In Proceedings of the 2017 IEEE International Workshop on Measurement and Networking (M&N), Naples, Italy, 27–29 September 2017; pp. 1–6. [CrossRef]
13. Kachan, D.; Siemens, E.; Shuvalov, V. Available bandwidth measurement for 10 Gbps networks. In Proceedings of the 2015 International Siberian Conference on Control and Communications (SIBCON), Omsk, Russia, 21–23 May 2015; pp. 1–10. [CrossRef]
14. Maksymov, S.; Kachan, D.; Siemens, E. Connection Establishment Algorithm for Multi-destination Protocol. In Proceedings of the 4th International Conference on Applied Innovations in IT, Koethen, Germany, 10 March 2016; pp. 57–60.
15. Bakharev, A.V.; Siemens, E.; Shuvalov, V.P. Analysis of performance issues in point-to-multipoint data transport for big data. In Proceedings of the 2014 12th International Conference on Actual Problems of Electronics Instrument Engineering (APEIE), Novosibirsk, Russia, 2–4 October 2014; pp. 431–441.
16. Syzov, D.; Kachan, D.; Karpov, K.; Mareev, N.; Siemens, E. Custom UDP-Based Transport Protocol Implementation over DPDK. In Proceedings of the 7th International Conference on Applied Innovations in IT, Koethen, Germany, 6 March 2019.

17. Karpov, K.; Kachan, D.; Mareev, N.; Kirova, V.; Syzow, D.; Siemens, E.; Shuvalov, V. Adopting Minimum Spanning Tree Algorithm for Application-Layer Reliable Mutlicast in Global Mutli-Gigabit Networks. In Proceedings of the 7th International Conference on Applied Innovations in IT, Koethen, Germany, 6 March 2019.
18. Uhlig, R.; Neiger, G.; Rodgers, D.; Santoni, A.L.; Martins, F.C.; Anderson, A.V.; Bennett, S.M.; Kagi, A.; Leung, F.H.; Smith, L. Intel Virtualization Technology. *Computer* **2005**, *38*, 48–56. [CrossRef]
19. Litayem, N.; Saoud, S.B. Impact of the Linux real-time enhancements on the system performances for multi-core intel architectures. *Int. J. Comput. Appl.* **2011**, *17*, 17–23. [CrossRef]
20. Cerqueira, F.; Brandenburg, B. A Comparison of Scheduling Latency in Linux, PREEMPT-RT, and LITMUS RT. In Proceedings of the 9th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Paris, France, 9 July 2013; pp. 19–29.
21. Lao, L.; Dovrolis, C.; Sanadidi, M. The Probe Gap Model can underestimate the available bandwidth of multihop paths. *ACM SIGCOMM Comput. Commun. Rev.* **2006**, *36*, 29–34. [CrossRef]
22. Low, P.J.; Alias, M.Y. Enhanced bandwidth estimation design based on probe-rate model for multimedia network. In Proceedings of the 2014 IEEE 2nd International Symposium on Telecommunication Technologies (ISTT), Langkawi, Malaysia, 24–26 November 2014; pp. 198–203.
23. Jain, M.; Dovrolis, C. End-to-end available bandwidth.: Measurement methodology, dynamics, and relation with TCP throughput. In Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'02), Pittsburgh, PA, USA, 19–23 August 2002.
24. Dugan, J. Iperf Tutorial. Available online: https://slideplayer.com/slide/4463879/ (accessed on 16 August 2019).
25. Katz, B.M.; McSweeney, M. A multivariate Kruskal-Wallis test with post hoc procedures. *Multivar. Behav. Res.* **1980**, *15*, 281–297. [CrossRef] [PubMed]
26. Abdi, H.; Williams, L.J. Tukey's honestly significant difference (HSD) test. In *Encyclopedia of Research Design*; Sage: Thousand Oaks, CA, USA, 2010; pp. 1–5.