

## Validation of virtualization platforms for I-IoT purposes

Jordi Mongay Batalla<sup>1</sup> · Konrad Sienkiewicz<sup>2</sup> ·  
Waldemar Latoszek<sup>2</sup> · Piotr Krawiec<sup>1</sup> ·  
Constandinos X. Mavromoustakis<sup>3</sup> ·  
George Mastorakis<sup>4</sup>

Published online: 13 August 2016

© The Author(s) 2016. This article is published with open access at Springerlink.com

**Abstract** Virtualization deployment in I-IoT domain is associated with many potential benefits. However, to achieve benefits from virtualization, which is nowadays a well-known technology, it is necessary to verify usability of virtualization platforms in context of I-IoT. In this article, we present a quantitative comparison of two leading open-source hypervisors, XEN and KVM, focusing on throughput and latency, which are key factors from I-IoT point of view. This paper analyzes the methodology for calculating throughput and latency in an computing device that hosts multiple I-IoT subsystems.

---

✉ Konrad Sienkiewicz  
k.sienkiewicz@itl.waw.pl

Jordi Mongay Batalla  
jordim@interfree.it

Waldemar Latoszek  
w.latoszek@itl.waw.pl

Piotr Krawiec  
p.krawiec@tele.pw.edu.pl

Constandinos X. Mavromoustakis  
mavromoustakis.c@unic.ac.cy

George Mastorakis  
gmastorakis@staff.teicrete.gr

<sup>1</sup> Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland

<sup>2</sup> National Institute of Telecommunications, Szachowa St. 1, 04-894 Warsaw, Poland

<sup>3</sup> Department of Computer Science, University of Nicosia, Nicosia, Cyprus

<sup>4</sup> Department of Informatics Engineering, Technological Educational Institute of Crete, Heraklion, Crete, Greece

**Keywords** Internet of things · Virtualization · Latency · Throughput

## 1 Introduction

Internet of things (IoT) and cloud computing have exhibited the greatest growth in the IT market in the recent past and this trend is expected to continue. Nowadays, we can also observe a growing number of various companies that are increasingly interested in using IoT technologies. This trend leads to a new concept of industrial-IoT (I-IoT), which addresses many existing industrial processes and systems such as manufacturing, logistics, facility diagnostics, and product inspection. To offer I-IoT services in a cost-efficient way, many companies decided to reduce costs and enhance the efficiency of their servers by adopting a private cloud computing. Private cloud computing involves the use of virtualization technology of servers where resources such as CPU, RAM, and storage are shared. Virtualization provides the capability to combine different functionalities on a single piece of hardware. It means that control units for different I-IoT subsystem (e.g., smart surveillance systems [1] and data storage service for IoT [2]) can be placed on one physical server, where full isolation of the subsystems is available. The consolidation of the subsystems into one server removes a lot of complexity from the end system, simplifies the maintenance and replacement process. Another benefit of the virtualization is that developers do not need to rewrite applications code to match the new hardware. Instead, they can replicate the old system in the virtual machine (VM) and simply run their application in a virtual environment. Based on this model and above-mentioned solution, the results are not just CAPEX and OPEX savings, but also significant savings in development time for end devices.

In this context, usability of hypervisors for I-IoT purpose becomes a very important factor. Therefore, in this paper, we present results from quantitative analysis of two leading open-source hypervisors, XEN and KVM. This study focuses on the overall performance and packet forwarding latency, which are important from the I-IoT point of view. While there are a lot of articles related to a virtualization performance [3] and [4], we do not find many articles, which focused on latency of packet forwarding (especially taking into account traffic distribution between VMs). However, it is worth noting that in case of many IoT services (as well as I-IoT) low latency is more important than performance [5, 6]. This is due to the fact that we often require to interact with the “real world” in “real time”. I-IoT applications are distributed and require to correlate information from many different places to understand what happened even within a single transaction. Obviously this process takes time [7]; therefore, role of the networks that connect devices and systems together cannot be ignored. For this reason, managing and coordinating real-time performance in the I-IoT becomes a new challenge and wherever possible, latency should be reduced in the network. Confirmation of the correctness of this thesis can be found in [8–10], where latency is mentioned as a key factor in the context of IoT.

The remainder of the paper is organized as follows. In Sect. 2, we present main virtualization platforms and briefly introduce popular virtualization solutions, such as full virtualizations and para-virtualization. In Sect. 3, we propose a test methodology, tests configuration as well as others test approaches. In Sect. 4, we present tests results together with conclusions. Section 5 concludes.

## 2 The main virtualization platforms

Virtualization is a hardware and/or software technique to run multiple operating systems and applications on virtual machines. In this case, one single physical platform can host several virtual machines (VMs). Guest operating systems or applications are isolated from one to another and, therefore, they look as they are running on different physical platforms. The use of virtual machines (VM) gives an opportunity for resource control and isolation. However, this impacts on overall performance (including packets forwarding). There have been many studies showing how VM execution compares to native execution and such studies have been used in generally improving the quality of VM technology [4].

There are various ways to handle I/O in virtualization, and depending on the complexity of the virtualized device, as well as the constraints of the selected use case, different approaches and compromises are needed. The most prominent options are full virtualization (emulation) or para-virtualization.

Full virtualization is the full system emulation which allows the unmodified guest operating system (OS) to run directly on the virtual machine. In this case, operating systems (such as Windows, Linux, and Mac Os.) of the VM are not aware that it is running in a virtualized environment. The guest operating system can run any application that has been designed for it. The layer responsible for emulating the machine is called virtual machine monitor (VMM) and it converts the instructions of the guest operating system into instruction of the host system. In addition, the VMM must emulate all hardware resources of the platform.

The main issue in full virtualization is its complicated implementation and noticeably lower performance compared to native hardware performance. Therefore, full virtualization can be used only in the most simple cases.

Para-virtualization is similar to the concept of full virtualization. It allows to run a guest operating system in a virtual machine, but unlike the full virtualization, selective modifications are necessary to run the guest operating system. Therefore, changes to system call interfaces, memory management and interrupt handling are necessary to make the guest operating system recognize that it is running in a virtual machine. In case of para-virtualization, drivers of the host operating system are usually modified to enable direct communication between OS of VM and real device driver without the emulation layer of abstraction. This approach allows for a direct exchange of data with the real device drivers.

The main advantage of the para-virtualization approach is its high performance and reactivity since the guest OS can implement the necessary optimizations that allow to by-pass several software abstraction interfaces. On the other hand, to support this model it is necessary to modify the real device driver accordingly. Implementation details vary from one hypervisor to another. One clear drawback of para-virtualization is requirement for access to the guest OS source code as well as the rights to modify it.

It is worth noting that some hardware are able to provide virtualization extensions (for example: virtualization technology—VT in case of Intel processors). Hardware extended support for virtualization (also called hardware-assisted virtualization) removes the bottlenecks in emulating processor instructions, which are difficult to virtualize. It also adds the necessary support to reduce the typical operations performed

in a VMM, such as address translation, and it implies that many of the instructions of the guest OS are executed by the processor without intervention of the VMM. It has to be mentioned that such a process results with higher overall performance.

In this study, we compare two open-source hypervisors: the XEN Citrix hypervisor and the kernel-based virtual machine (KVM).

XENServer is a server virtualization platform for Windows and Linux. XENServer supports full virtualization and runs directly on the server hardware, without any underlying operating system. XENServer works by abstracting elements from the physical machine (such as hard drives, resources and ports) and allocating them to the virtual machines (VMs) that run on it. Moreover, the XEN hypervisor natively supports different virtualization modes such as full virtualization, para-virtualization and hardware-assisted mode. The XEN software can be run using the standard tool-stack, with libvirt [11] and with XAPI. Citrix XENServer and the various open-source XEN vendors are using the same hypervisor. Furthermore, there are some new features added by Citrix Systems Inc., which perform mainly administrative functions.

Kernel virtual machine (KVM) [12] is a common Linux operating system feature that allows to run an unmodified guest operating system (OS) in a Linux process. KVM is one of the most popular open-source hypervisor. It is important that KVM is supported by renowned management tools such as libvirt [13], making it relatively easy to maintain. KVM supports both full virtualization of I/O devices by QEMU [14] and para-virtualization of I/O devices with Virtio [15]. KVM uses hardware support to manage processor states and Memory Virtualization, what assures near to native performance. KVM also takes advantage of hardware virtualization features in the latest processors (i.e., Intel VT). The combination of hardware acceleration and para-virtual I/O is designed to reduce virtualization overhead to very low levels [16]. KVM supports live migration, allowing physical servers to be evacuated for maintenance without interrupting the guest operating system [17].

### 3 The concept of the tests

#### 3.1 Related works

The virtualized network device could be modeled as a quite complex queuing system, where we consider requests of the VMs to access particular resources, i.e., CPU, memory, Network Interface Controller, etc. Proposing a good model seems to be unfeasible, since the interrelation of all the above-mentioned effects introduces a high complexity in the system. Anyway, as performed in many queuing theory studies, we may see the system from the point of view of one single client (in our case the client is the VM). For a single VM, the access to the system resources depends on both its own offered load and the offered load of all other VMs. Then, to assess the performance of the whole system, we should measure the relation of the offered load between the different VMs.

The common use benchmarking test methodology for packet devices is described in [18] and considers several measurements of packet forwarding performance of IP routers including throughput, latency, loss rate, back-to-back frames, system recovery

and reset. According to above-mentioned RFC, the throughput is defined as the maximum offered traffic load that can be forwarded by the device with no packet loss and latency is defined as average of the packet latency values (min 20) at the determined throughput rate. However, above methodology focuses on measurement of parameters for devices and does not take into account virtualization. It should be noted that in virtualization platforms the total offered traffic load is the sum of the loads offered to each VM. Moreover, the total offered traffic load is partitioned across particular VM on different ways. The model of the presented problem considers the data set of loads offered to each of the  $N$  virtual machines  $\mathbf{O} = \{O_i : i = 1, \dots, N\}$ , which represents the work point of the virtualized device. The total load offered to the network device  $O_{\text{tot}}$  is the following

$$O_{\text{tot}} = \sum_{i=1}^N O_i \quad (1)$$

The number  $N$  of virtual machines is 12 for the both virtualization platforms.

Many research papers compare virtualization platforms using RFC 2544 methodology with considering equal traffic offered to all the VM. However, according to [19], this approach is not sufficient because performance of the virtualized network device depends on the distribution of the traffic load among VMs and the platforms should be always compared in the worst work point. Moreover, authors of [19] proposed a methodology that extends RFC 2544 methodology of throughput measures by considering the impact of heterogeneity of the offered load at the level of virtual routers. For measuring the distribution of the traffic load among different VMs, they propose to use the Jain's fairness index [20] which is defined as

$$J = \frac{O_{\text{tot}}^2}{N \times \sum_{i=1}^N O_i^2} \quad (2)$$

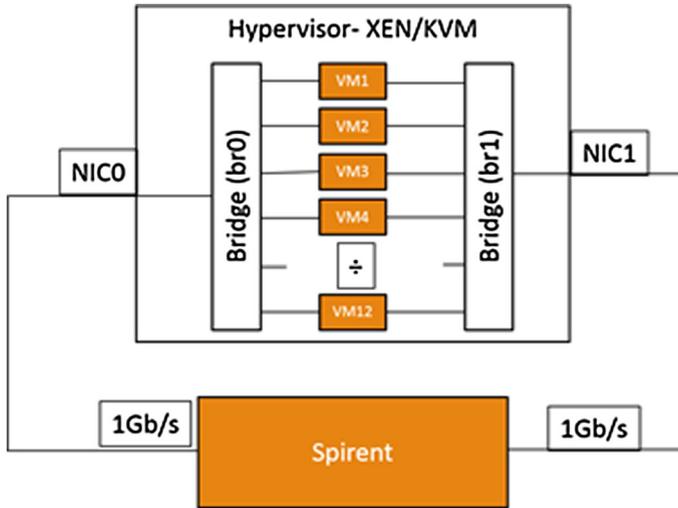
The parameter of Jain's fairness index determines the second moment of the data set that, distinct the standard deviation, takes values from a limited range  $J \in [0, 1]$ . In the case of tests, this parameter can range from  $1/N$  (where the whole traffic is generated to the single VM) to 1 (where all VM are loaded with the same volume of traffic).

In our research we decide to adopt and expand (for latency measurements) the above methodology, which is more suitable for virtual devices.

### 3.2 Test configuration

Figure 1 presents the test configuration. Generally, there are two hardware devices: server HP ProLiant DL360G6 deployed as a virtualization platform and Spirent Test-Center as a traffic generator/analyzer. Main parameters of the HP server are listed in Table 1.

In our test, we consider two virtualization platform: XEN Citrix and KVM. The most significant parameters of software with regard to both platforms are listed in Table 2.



**Fig. 1** The general test configuration

**Table 1** Server technical parameters

Parameter	Value
CPU	2 × Intel Xeon X5660 @ 2.80 GHz
RAM	6 × 4 GB DIMM Synchronous 1333 Mhz
NIC	Intel 82571 EB Gigabit Ethernet Controller

**Table 2** Software specification

	XEN citrix	KVM
Hypervisor version	6.50	2.1.3
OS of DOM0	Red Hat 4.1.2–51 (64 bit)	Fedora 21 (64 bit)
RAM allocated to VM	1024 MB	1024 MB
NIC emulation	Broadcom BCM5709	Virtio

We assume that there are 12 VMs launched by the hypervisor (XEN/KVM) deployed on an HP ProLiant DL360G6. Two bridges br0 and br1 (based on OVS switch) operate in the host domain (dom0) of hypervisor. The bridges, respectively, connect ingress physical NIC (NIC0) of server with virtual ingress NIC of each VM and egress physical NIC (NIC1) of server with virtual egress NIC of each VM. Each VM performs IP software routing between two network interfaces that are connected to both bridges.

The first bridge-br0 is responsible for sending incoming IP packets to the appropriate virtual machine. To ensure isolation, an adequate OpenFlow rule was configured. Individual IP flows are directed to virtual machines based on the destination MAC

addresses of virtual machine. This rule eliminates the possibility of packet appearance in two different virtual machines.

The second bridge br1 captures flows from each virtual machine and sends them to the output interface NIC1. In this case, the OpenFlow rule eliminates sending packets back to the other virtual machines.

Both Spirent TestCenter equipped with CM-1G-D4 card and the Device Under Test(hypervisor) were connected by two 1 Gbps Ethernet links.

### 3.3 Test methodology

As we mentioned above, our test methodology takes into account the distribution of the traffic load among different VMs and is generally based on [1].

The assumptions of the methodology are:

1. DUT operates 12 VMs.
2. 12 different IP flows are generated to the appropriate virtual machines.
3. We assume 2 groups of VMs: (1)  $k$  VMs are strongly loaded and (2)  $N-k$  VMs are slightly loaded. Therefore, it is necessary to perform  $N$  configurations in this scheme,  $k \in \{1, 2, \dots, N\}$ .
4. For each configuration, we seek maximum offered load  $O_{tot}$  taking into account that in all VMs there is no validated packet loss. Small part of this load, i.e.,  $qO_{tot}$ , is divided equally among slightly loaded VMs for  $k \neq N$ . Remaining load is distributed equally among strongly loaded VMs. We assume  $q = 0.01$  since this value ensures that the virtualization platform may handle slightly loaded VMs and simultaneously, forwarding of this part of traffic has negligible impact on the strongly loaded VMs.
5. The durations of one test scenario should last at least 60 seconds. We propose the following set of the Ethernet frame sizes {64, 128, 512, 1024, 1518}. Offered load for each strongly loaded VM is given by

$$O_h = \begin{cases} \frac{(1-q)O_{tot}}{k} & k \in 1, \dots, N - 1 \\ \frac{O_{tot}}{N} & k = N \end{cases} \tag{3}$$

6. Offered load for each slightly loaded VM is the following

$$O_l = \begin{cases} \frac{qO_{tot}}{N-k} & k \in 1, \dots, N - 1 \\ 0 & k = N \end{cases} \tag{4}$$

7. Consequently, the Jain's fairness index for given  $k$  is equal to

$$J(k) = \begin{cases} \frac{k(N-k)}{N[N(1-q)^2 - k(1-2q)]} & k \in 1, \dots, N - 1 \\ 1 & k = N \end{cases} \tag{5}$$

$$\lim_{q \rightarrow 0^+} J(k) = \frac{k}{N} \tag{6}$$

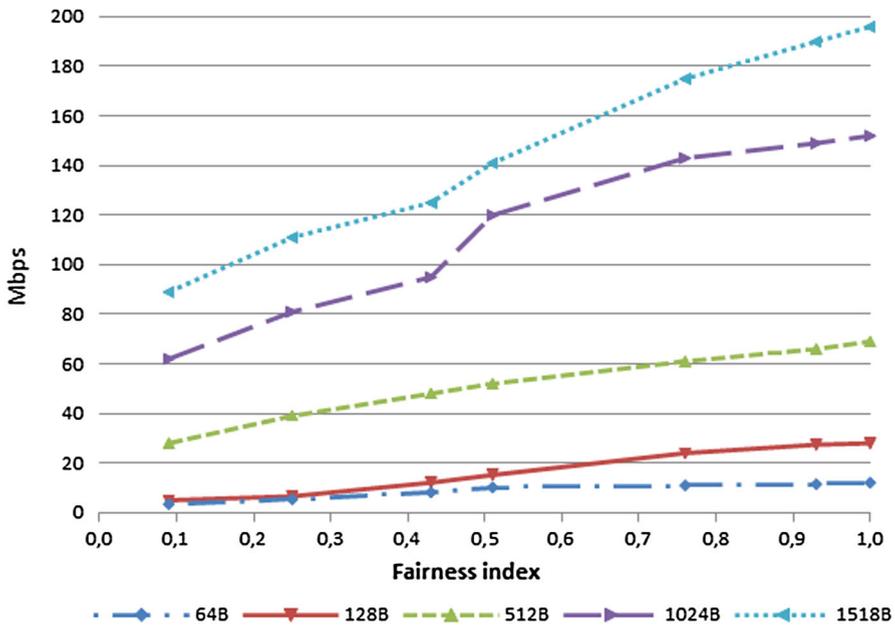


Fig. 2 The throughput in function of the J-fairness index for XEN

- 8. Conditional throughput  $T|_{J=J(k)}$  is equal to achieved  $O_{tot}$ .
- 9. The latency in each VM is measured according to the RFC2544 methodology. It is performed for previously calculated throughput  $T|_{J=J(k)}$ . The final result of the latency is equal to:

$$L_{tot} = \frac{q}{N - k} \sum_{i=1}^{N-k} L_i + \frac{(1 - q)}{k} \sum_{i=1}^k L_i \tag{7}$$

- 10. The tests were repeated several times to calculate the 95 % confidence intervals.

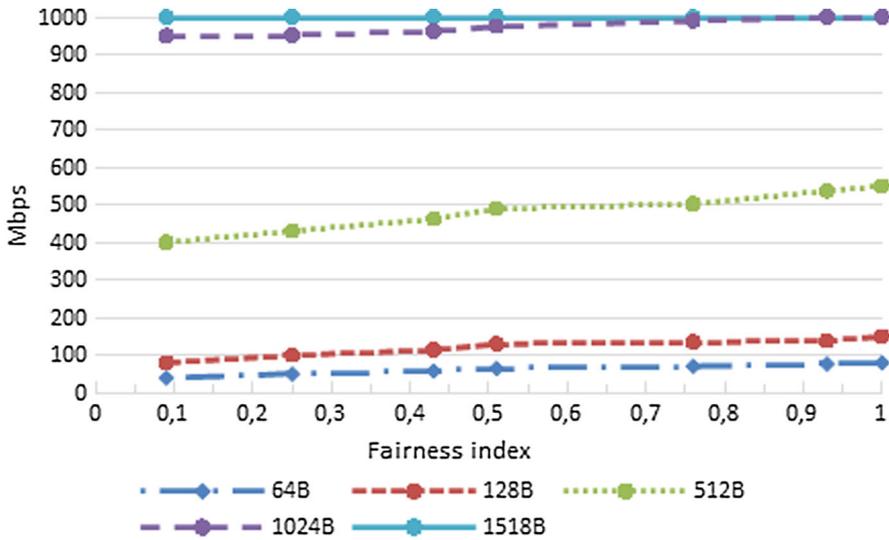
## 4 Test results

In following subsections, we present test results including throughput and latency for both virtualization platforms as well as a brief summary in context of IoT.

### 4.1 Results of throughput tests

Figure 2 shows the results of XEN throughput parameter properly depending on the J-fairness index and the packet sizes.

We can conclude that, irrespective of the packets size, it is observed that the throughput increases with increasing J-index. This increase is approximately linear for all packets sizes, where average twofold increase of the throughput value between the

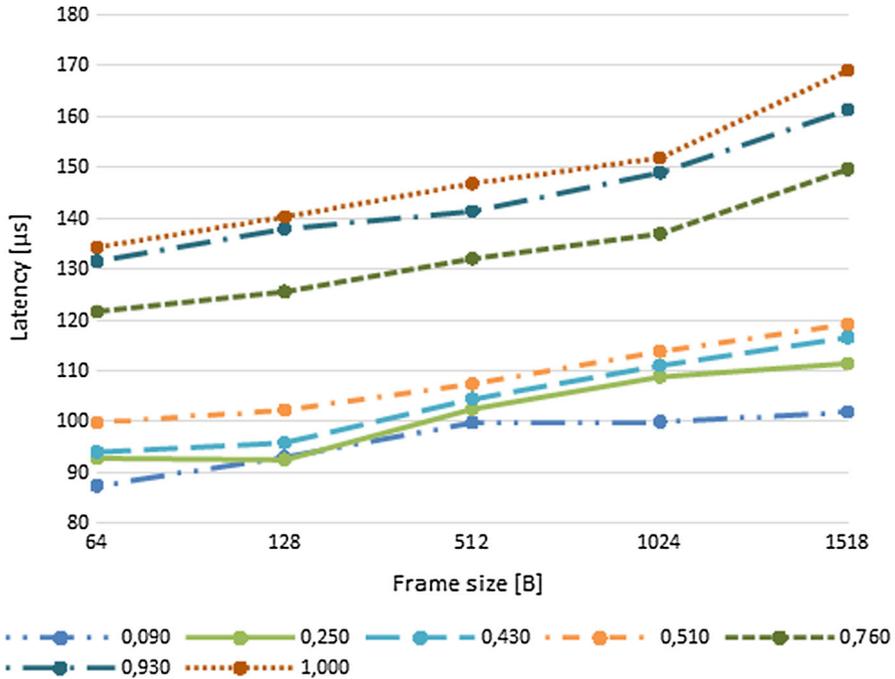


**Fig. 3** The throughput in function of the J-fairness index for KVM

lowest and the highest J-fairness index is noticed. Above dependency can be explained by level of resource utilization. In hypervisor, every single VM has allocated physical resources of host (i.e., CPU, RAM, storage) and in consequence several VMs use more physical resources. Moreover, a particular VM cannot use resources assigned to others VMs (even if they are not using it at the time). This leads to limited computing performance of single VM, because overall host performance is divided to all VMs. The performance of single VM is lower if hypervisor serves more VMs. In case of higher J-index values, the traffic is better distributed and strongly loaded VMs forward less traffic. These result in higher overall performance.

In parallel, the throughput increases together with packet size. This relationship results from the hypervisor limitations in terms of maximum number of packets possible to handle in time unit (there are more packets to forward in the same traffic load level in case of short packets).

The following figure presents analogical results for KVM hypervisor. Significantly higher carried load of KVM hypervisor is the most visible difference. It means that Virtio NIC emulation in the KVM hypervisor offers much better performance of packet forwarding, comparing to XEN, which uses full emulation. Moreover, Fig. 3 shows that in case of KVM hypervisor J-index slightly affects throughput results. The lines which describe throughput in function of the J-fairness index are nearly flat. However, it can be observed that throughput slightly grows with increasing of J-index. For largest packet size, the throughput does not depend on J-index at all (it is equal to 1 Gbit/s). In this case, hypervisor forwarding performance is higher than limitations resulting from the medium (Gigabit Ethernet). Moreover, the throughput strongly depends on a packet size of forwarding traffic. This is due to the limited efficiency in handling a large number of packets, which is common to XEN hypervisor.



**Fig. 4** Latency results of XEN hypervisor in function of packet size

## 4.2 Results of the latency tests

Figure 4 shows the results of the XEN latency parameter as a function of packet size for the selected values of the J-fairness index. We can generally conclude that the increase of the latency is approximately linear for each frame sizes and dispersion of the latency values is quite small. The maximum value of the latency obtained for the packet size 1518B does not exceed  $170 \mu\text{s}$  and the minimum is more than  $80 \mu\text{s}$  for the 64B packet size.

Figure 5 shows results of latency parameter of XEN hypervisor in function of the J-fairness index. It could be seen that with the increase of the J-fairness index the increase of the latency is observed. Moreover, the latency grows faster for J-index values above 0.51 regardless of frame size. The reason for this fact is probably the packet processing algorithm in XEN hypervisor. If we have to deal with the cycle of transmission to multiple VM (in the extreme case to the 12 VMs) there is an additional delay time, which affects the growth of latencies of individual packets. This delay time is reduced with decreasing of the J-index parameter, due to packets transmitted to a smaller number of VM.

Figure 6 shows the KVM hypervisor results of latency in function of packet size. The chart shows that there is a significant increase of latency when the packet size is equal to 1024 B or higher. The values of latencies are quite constant and do not exceed  $100 \mu\text{s}$  when the packet size is less than 512 B, while in case of larger packets the

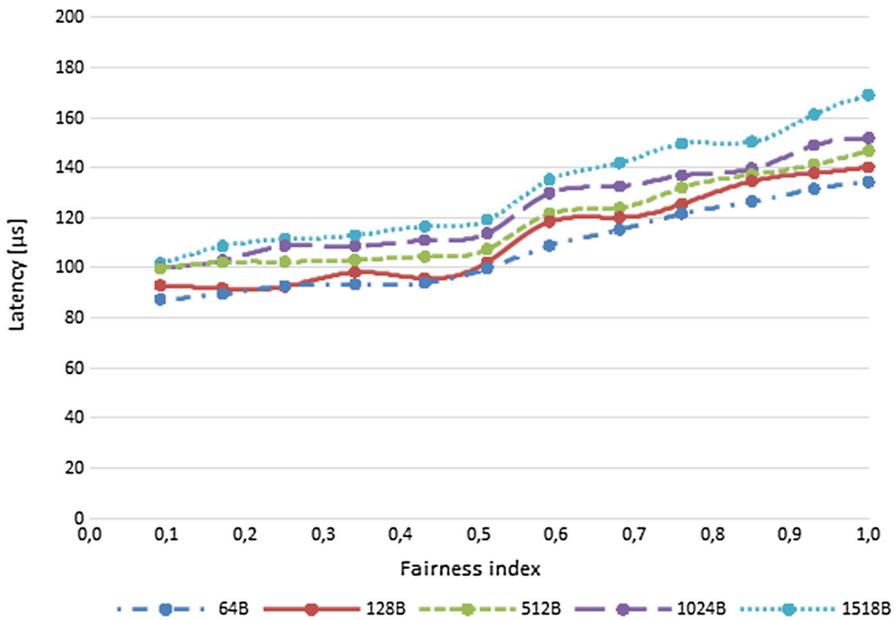


Fig. 5 The latency parameter of XEN hypervisor in function of the J-fairness index

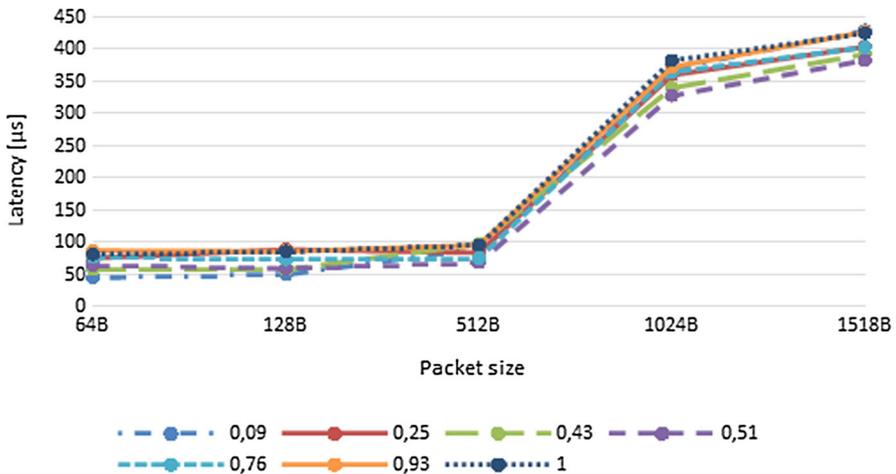
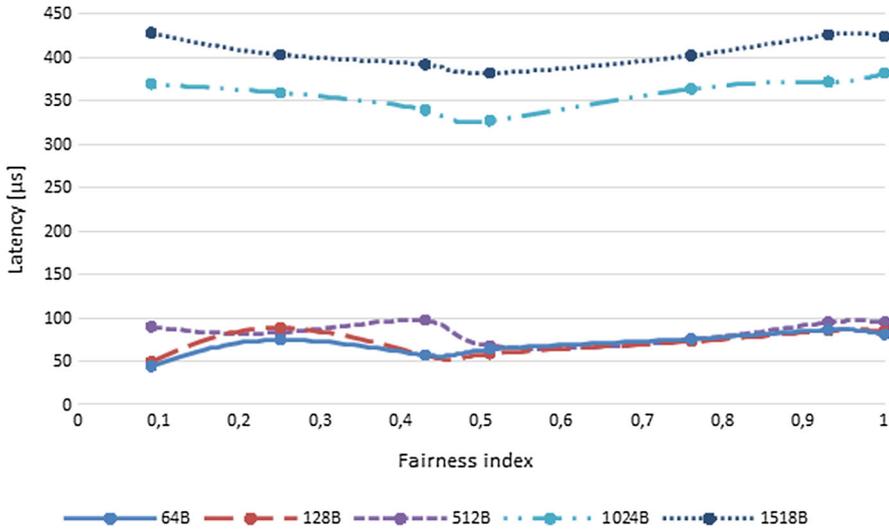


Fig. 6 The latency parameter for KVM hypervisor in function of packet size

latency exceeds 300  $\mu$ s. Moreover, in contrast to XEN hypervisor, there is no clear dependence between the latency value and the J-fairness index (see Fig. 7). On this figure, it is also visible that the latency for long packets (1024 and 1518 B) is much higher than the others. It is worth to note that this higher latency is not related with the packet duration, because it is higher. We performed additional research and test to find explanation for these results. Additional tests proved that the latency in KVM



**Fig. 7** The latency parameter of KVM hypervisor in function of the J-fairness index

hypervisor also depends on the current traffic load. Well this can be seen in the next two charts. These charts include latency results in function of served load for different J-index values and packet size, respectively.

Figure 8 shows the KVM hypervisor latency values in dependence of traffic load for packet size equal to 1518 B (for different J-fairness index values). We can observe that in low traffic conditions the increase of the load is associated with latencies decrease. Further, between 200 and 750 Mbit/s, the values of latencies are quite stable, then they begin to increase rapidly since the throughput overcome 750 Mbit/s and for the load equal to 950 Mbit/s the latency is about 400  $\mu$ s. The latency depends very slightly on J-fairness index. Figure 9 shows the KVM hypervisor latency values in dependence of a load for J-fairness index values equal to 0,09 and different packet sizes. We can observe that for all packet sizes latency values are higher when the load is low (below approx. 50–80 Mbit/s depends on a packet size) as well as for high load (in case of long packets). In case of high traffic load, long latency is a result of heavy utilization of buffers. We suggest that high latency values in low traffic condition is a matter of processes optimization in VM. Probably, in low traffic condition, the KVM hypervisor reduces resources allocated for packets handling. In return, it allocates more resources to other processes of VM (e.g., video handling). This way hypervisor assure high overall performance with reasonable latency of packets forwarding. In higher traffic condition, packets processing needs more physical resources. Therefore, latency is lower due to more resources allocated to NIC emulation process. Summarized, in case of KVM hypervisor, the latency depends on traffic load much more, than on J-fairness index.

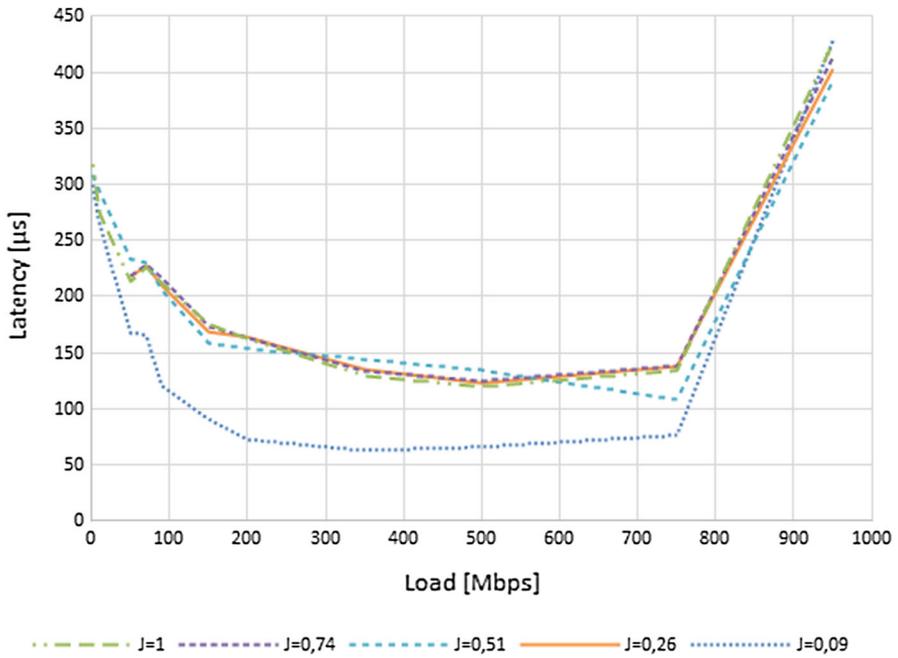


Fig. 8 KVM latency characteristic in function of load (packet size equal 1518 B)

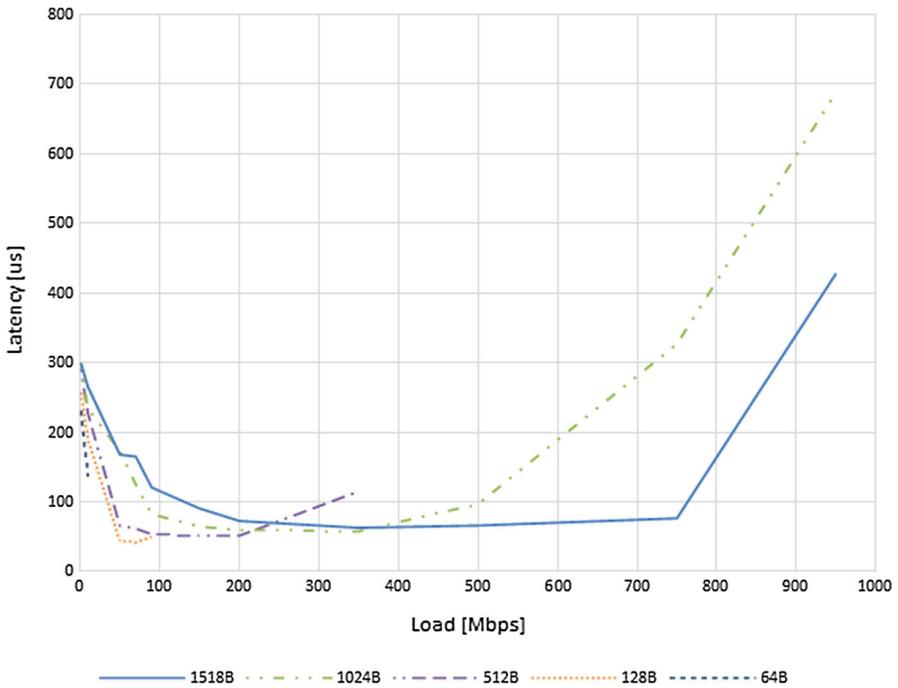


Fig. 9 KVM latency characteristic in function of load for J-index = 0.09

### 4.3 Tests results summary

Summarized, KVM hypervisor offers a significantly higher performance (throughput). It is the most visible difference. Throughput value in KVM is about five times higher than in XEN for every packet size measured. The most probable reason for this disparity is different types of network interfaces emulation inside hypervisors. KVM uses virtio NIC with para-virtualization while XEN uses full hardware emulation.

Evaluating the impact of traffic distribution between VMs, it could be concluded that in case of both hypervisors, the throughput increases together with increasing of packet size as well as J-index. Packet size has significant influence on achieved throughput in both hypervisors. This is due to the limited efficiency of hypervisors regarding handling a large number of packets. The scale of J-index influence is different for hypervisors. In case of XEN, the J-index influence is noticeable (throughput for  $J = 1$  is approx. twice than for  $J = 0.09$ ), while in case of KVM the J-index it slightly affects throughput results.

In case of latency, the test results indicate that the both hypervisors are comparable (about 100  $\mu$ s). However, there are two main differences. In case of XEN hypervisor, there are clear dependency on J-index and no dependency on carried load. While in case of KVM hypervisor, there are minor dependency on J-index and strong dependency on carried load.

## 5 Summary

In the article, we have presented a quantitative comparison of XEN and KVM focusing on throughput, and latency which are key factors from I-IoT's point of view. Proposition of appropriate test methodology suitable for virtual environment and carry out of the tests were interim targets.

The most significant conclusion is that KVM hypervisor offers significantly higher performance. Throughput value in KVM is about five times higher than in XEN. The reason for this disparity is different types of network interfaces emulation inside the hypervisors. KVM uses virtio NIC with para-virtualization while XEN uses full hardware emulation. By utilizing virtio, KVM achieves high performance and keeps I/O-related bottlenecks to a minimum. It means that full emulation (XEN) is connected with a considerable overhead in virtualization and may be used only in the simplest cases with low carried traffic. The KVM with para-virtualization can be applied in more demanding cases.

In case of latency, two hypervisors are comparable. However, in contrast to the XEN, the KVM hypervisor has strong dependency between latency and carried load. For example, in low traffic conditions, the increase of traffic load is associated with latencies decrease. In result, XEN offers significantly lower latencies than KVM, if total traffic load is lower than 50 Mbit/s.

General conclusion is that the KVM hypervisor is a better choice for IoT virtualizations purposes in most of use cases. It is because the KVM offers significantly higher packet forwarding performance. The XEN hypervisor can be considered as a better solution for implementation in I-IoT environment with low traffic condition due to lower latency.

**Acknowledgements** This work was undertaken under the Pollux IDSECOM project supported by the National Research Fund Luxembourg (FNR) and the National Centre for Research and Development (NCBiR) in Poland.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Kokkonis G, Psannis KE, Roukemiots M, Schonfeld, Dan (2016) Real-time wireless multisensory smart surveillance with 3D-HEVC streams for internet-of-things (IoT). *J Supercomput.* doi:[10.1007/s11227-016-1769-9](https://doi.org/10.1007/s11227-016-1769-9)
2. Karolewicz K, Beben A, Batalla JM, Mastorakis G, Mavromoustakis CX (2016) On efficient data storage service for IoT. *Int J Netw Mgmt.* doi:[10.1002/nem.1932](https://doi.org/10.1002/nem.1932)
3. Felter W, Ferreira A, Rajamony R, Rubio J (2014) An updated performance comparison of virtual machines and linux containers. IBM Research, Austin, TX
4. McDougall R, Anderson J (2010) Virtualization performance: perspectives and challenges ahead. *SIGOPS Oper Syst Rev* 44:40–56
5. Kryftis Y et al (2016) Efficient entertainment services provision over a novel network architecture. *IEEE Wirel Commun* 23(1):14–21. doi:[10.1109/MWC.2016.7422401](https://doi.org/10.1109/MWC.2016.7422401)
6. Kokkonis G, Psannis KE, Roukemiots M, Ishibashi Y (2015) Efficient algorithm for transferring a real-time HEVC stream with haptic data through the internet. *J Real-Time Image Proc* 12(2):343–355. doi:[10.1007/s11554-015-0505-7](https://doi.org/10.1007/s11554-015-0505-7)
7. Bourdena A et al (2016) Using socio-spatial context in mobile cloud process offloading for energy conservation in wireless devices, *IEEE Trans Cloud Comput.* doi:[10.1109/TCC.2015.2511736](https://doi.org/10.1109/TCC.2015.2511736)
8. Mongay Batalla J, Krawiec P (2014) Conception of ID layer performance at the network level for internet of things. *Springer J Pers Ubiquitous Comput* 18(2):465–480
9. Mongay Batalla J, Gajewski M, Latoszek W, Krawiec P, Mavromoustakis C, Mastorakis G (2016) ID-based service-oriented communications for unified access in IoT. *Elsevier Comput Electr Eng J* 52:98–113
10. Stergiou C, Psannis KE (2016) Recent advances delivered by mobile cloud computing and Internet of things for Big data applications: a survey. *Int. J. Network Manage.* doi:[10.1002/nem.1930](https://doi.org/10.1002/nem.1930)
11. Bolte M, Sievers M, Birkenheuer G, Niehorster O, Brinkmann A (2010) Non-intrusive virtualization management using libvirt. In: *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '10*, pp 574–579
12. Avi K, Yaniv K, Dor L, Uri L, Anthony L (2007) KVM: the Linux virtual machine monitor. *Proc Linux Symp* 1:225–230
13. Libvirt. <http://libvirt.org>
14. Bellard F (2005) QEMU, a fast and portable dynamic translator, In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05*, pp 41–41
15. Russell R (2008) Virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Oper Syst Rev* 42:95–103
16. Padala P, Zhu X, Wang Z, Singhal S, Shin KG et al (2007) Performance evaluation of virtualization technologies for server consolidation. HP Labs Technical Report
17. Matthews JN, Hu W, Hapuarachchi M, Deshane T, Dimatos D, Hamilton G, McCabe M, Owens J (2007) Quantifying the performance isolation properties of virtualization systems. In: *Proceedings of the 2007 Workshop on Experimental Computer Science*
18. Bradner S, McQuaid J (1999) Benchmarking methodology for network interconnect devices, Request For Comments 2544
19. Batalla J Mongay, Kantor M, Mavromoustakis CX, Skourletopoulos G, Mastorakis G (2015) A Novel Methodology for Efficient Throughput Evaluation in Virtualized Routers. *IEEE International Conference on Communications ICC.* London, UK
20. Jain R, Durresti A, Babic G (1999) Throughput fairness index: An explanation, *ATM Forum/99-0045*