University of Würzburg
Institute of Computer Science
Research Report Series

Performance Comparison of Common Server Hardware Virtualization Solutions Regarding the Network Throughput of Virtualized Systems

Daniel Schlosser, Michael Duelli,
and Sebastian Goll

Report No. 479 March 2011

University of Würzburg, Germany
Institute of Computer Science
Chair of Communication Networks
Am Hubland, D-97074 Würzburg, Germany
phone: (+49) 931-31-86642, fax: (+49) 931-31-86632
{schlosser|duelli|goll}@informatik.uni-wuerzburg.de

Performance Comparison of Common Server Hardware Virtualization Solutions Regarding the Network Throughput of Virtualized Systems

Daniel Schlosser, Michael Duelli, and Sebastian Goll
University of Würzburg, Germany
Institute of Computer Science
Chair of Communication Networks
Am Hubland, D-97074 Würzburg, Germany
phone: (+49) 931-31-86642, fax: (+49) 931-31-86632
{schlosser|duelli|goll}@informatik.uni-wuerzburg.de

Abstract

Hosting virtual servers on a shared physical hardware by means of hardware virtualization is common use at data centers, web hosters, and research facilities. All virtualization platforms include isolation techniques that restrict resource consumption of the virtual guest systems. Therefore, they setup quotas for the virtual guest systems on the use of processors, memory, and hard disk of the host. However, these isolation techniques have an impact on the performance of the guest systems. In this technical report, we study how popular hardware virtualization approaches, namely OpenVZ, KVM, Xen v4, VirtualBox, and VMware ESXi, affect the network throughput of a virtualized system. We compare their performance impact in a dedicated and a shared host scenario as well as to the bare, i.e. not virtualized host system. Our results provide an overview on the performance of popular hardware virtualization platforms on commodity hardware in terms of network throughput and should be considered when performing scientific experiments in a virtualized environment.

Keywords: hardware virtualization; analysis; commodity hardware; network throughput; isolation

1 Introduction

Hardware virtualization means abstracting functionality from physical components. This principle is used for sharing hardware by multiple logical instances – the *virtual guest systems* – and originated already in the late 1960s. Recently, the importance of virtualization has drastically increased due to its availability on commodity hardware, which allows multiple *virtual guests* to share a physical machine's resources. The resource access is scheduled and controlled by the *virtual machine monitor* (VMM), also called *hypervisor*. Different virtualization platforms have been implemented and are widely used in professional environments, e.g. data centers and research facilities like G-Lab [1], to increase efficiency and reliability of the offered resources. The resources that may be used by a virtual guest system as well as lower performance bounds regarding these resources are determined in *service level agreements* (SLAs) between providers and customers. The implementation of the VMM influences how well the resource allocation complies with these SLAs and to which extent different guests interfere with each other. Hence, a provider has to know the performance limitations, the impact factors, and the key performance indicators of the used VMM to ensure isolation and SLAs.

There are two basic scenarios in which hardware virtualization is used. In the *dedicated* scenario, only a single virtual guest is run on a physical host. By means of hardware virtualization, the guest system is made independent of the underlying physical hardware, e.g., to support migration in case of maintenance or hardware failures. Hardware virtualization is also used to consolidate resources. Therefore, the physical hosts are shared among multiple virtual guest systems in a *shared* scenario. While the performance of a virtualized system can be directly compared to the bare host system in the dedicated scenario, the performance of a virtual guest may also be affected by interference with other virtual guests on the same physical host in a shared scenario.

The performance cost of virtualization and *isolation*, i.e. the prevention of virtual guest interference, has been studied for CPU, memory, and hard disk usage. Nowadays, most commodity servers have built-in hardware support for virtualization enabling fast context switching in the CPU and resource restriction for the memory. However, the *input/output* (I/O) system, especially network throughput, is still a crucial factor.

Therefore, we focus on the network throughput of virtualized systems as the performance metric. We apply this metric on a non-virtualized Linux and on a virtualized version of this system using several popular hardware virtualization platforms, i.e. OpenVZ, KVM, Xen v4, VirtualBox, VMware ESXi. Furthermore, we install a second virtual guest and analyze the effects when the second virtual guest is idle, running some CPU and memory intensive tasks, or is transmitting packets.

The remainder of this report is structured as follows. In Section 2, we present the different virtualization concepts used by the considered VMMs and related work. We explain our measurement setup and methodology in Section 3. Results for the dedicated scenario considering only a single guest are described in Section 4. An analysis of two virtual guests in shared scenarios is discussed in Section 5. In Section 6, we draw conclusions and give an outlook on future work.

2 Background and Related Work

In this section, we describe the techniques used by the considered VMMs to virtualize the underlying hardware and give an overview on related work.

2.1 Virtualization Concepts

The recent popularity of virtualization on commodity hardware created a plethora of virtualization platforms with different complexity and environment-dependent applicability. An overview of virtualization platforms is given in [2].

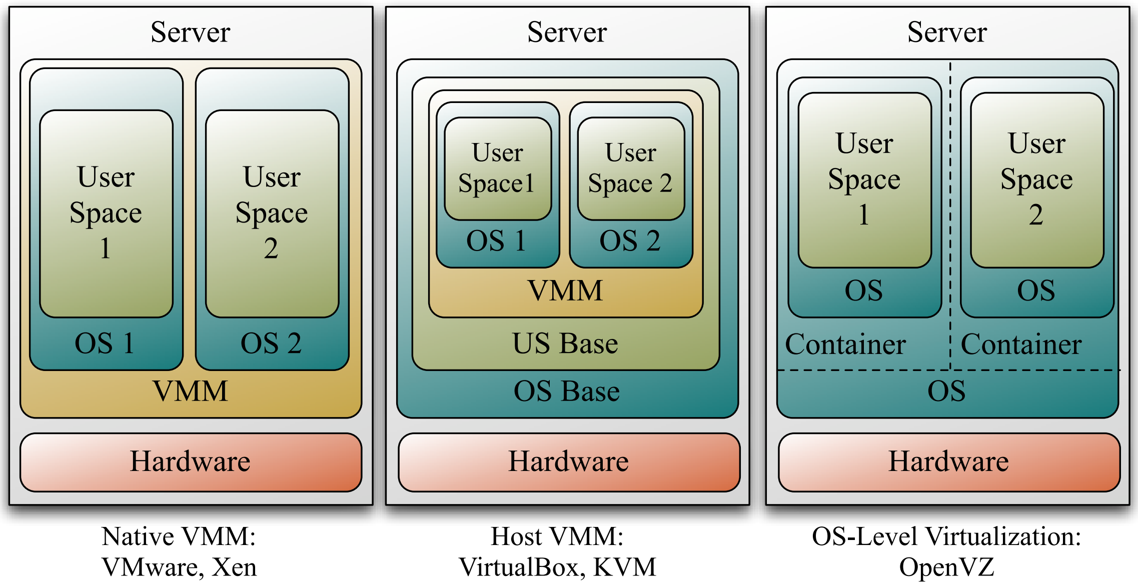


Figure 1: Virtualization concepts of the considered VMMs.

Several ways to realize hardware virtualization have evolved, which differ in the required adaptations of the guest system. For instance, a VMM may run on bare hardware (called *Type 1*, *native*, or *bare metal*) or on top of an underlying/existing *operating system* (OS) (called *Type 2* or *hosted*). Also *hybrids* between Type 1 and 2 are possible. Finally, OS-level virtualization is a special form, where the guests are run in a container of the OS operating directly on the hardware.

Native VMMs and hosted VMMs might provide *full virtualization*, i.e. the guest system does not have to be modified. This means that the virtualized guest operating system is working as if it would run directly on the hardware. But whenever the guest wants to access a hardware resource, which is only a virtualized resource, the VMM has to interrupt the guest and 'emulate' this hardware access. This effects the performance of the guest system. To alleviate these problems, the guest system can be modified, e.g. with special drivers, to redirect resource accesses to function calls of the VMM. This method is called *para-virtualization* and may improve the performance of the guest. OS level virtualization solves this resource access problem by using the same modified kernel in the guest containers as in the hosting OS. This technique has the advantage that the overhead for virtualization is quite low compared to the other solutions. However, the OS and the kernel of the guest is predetermined. Figure 1 depicts how the considered virtualization platforms encapsulate virtual guests and indirect access to resources for applications, which run in user space. In this work, we consider five popular hardware virtualization platforms which can be categorized as

- *native virtualization*, represented by VMware ESXi [3] and Xen v4 [4],
- *host virtualization*, represented by VirtualBox [5] and KVM [6], and
- *operating system-level virtualization*, represented by OpenVZ [7].

Fair access to the physical hardware is important and known to work for resources that can be split into quotas, e.g. memory, disk space, or processor cycles per second. However, access to any of these separately restricted resources involves the *input and output* (I/O) of data which typically shares a single bottleneck, e.g. an internal bus. Especially, network I/O fairness is a crucial issue which involves another shared physical resource, the *network interface card* (NIC). Hence, we consider the packet forwarding throughput of the aforementioned virtualization platforms as a performance metric.

2.2 Related Work

An overview of virtualization techniques and mechanisms as well as their history and motivation is given in [8]. Several publications consider *resource access fairness* when multiple guest systems run on a single host. In [9], virtualization was done using Xen. The authors show that while the processor resources are fairly shared among guests, the scheduling of I/O resources is treated as a secondary concern and decreases with each additional guest system. The authors of [10] use a NetFPGA-based prototype to achieve network I/O fairness in virtual machines by applying flexible rate limiting mechanisms directly to virtual network interfaces. Fairness issues in software virtual routers using *Click* are considered in [11]. The impact of several guests with and without additional memory intensive tasks on packet forwarding is measured. More details and performance tuning for the Click router is provided in [12].

As mentioned before, the performance and fairness heavily depend on the used VMM. Hence, several comparisons of mostly two VMMs have been conducted. We focus on the references considering network I/O performance which altogether consider packet forwarding performance with different packet lengths from 64 to 1500 bytes. In [13], a comparison of VMware ESX Server 3.0.1 and open-source Xen 3.0.3 with up to two guests was conducted by VMware using the Netperf packet generator. They show that Xen heavily lags behind VMware ESX. This study was repeated with VMware ESX Server 3.0.1 and open-source Xen Enterprise 3.2.3 in [14] by Xen Inc. which showed that a specialized Xen Enterprise lags by a few percent. The authors of [15] consider multiplexing of two data flows with different scheduling mechanisms on a real Cisco router, a virtual machine host, in two VM guests using Xen, and a NetFPGA packet generator. They show the dependency of the software routers on the packet length. Measurements of OpenVZ and User Mode Linux (UML) were conducted in [16] on wireless nodes of the Orbit test bed. UDP throughput and FTP performance is measured, but it is also stated that the hardware is not the best choice for virtualization. In [17], the network virtualization platform *Trellis* is presented and packet-forwarding rates relative to other virtualization technologies and native kernel forwarding performance are compared. The considered virtualization technologies comprise Xen, Click, OpenVZ, NetNS, and the Trellis platform. The Linux kernel module *pktgen* [18] is used for packet generation. In their results, Xen has the lowest performance followed by OpenVZ.

To the best of our knowledge, this is the first paper to compare the packet forwarding performance of today's most popular hardware virtualization platforms on a common commodity server system without special hardware requirements, e.g. NetFPGA cards, or fine-tuned virtualization parameters, such that comparisons can be easily repeated.

3 Hardware Setup and Measurement Scenarios

In the following section, we give an overview on the hardware setup of our measurements and provide technical details on the considered scenarios.

3.1 Hardware Setup

For our measurements, we use seven Sun Fire X4150 x64 servers from the G-Lab testbed [1]. Each server is equipped with 2×Intel Xeon L5420 Quad Core CPU (2×6 MB L2 Cache, 2.5 GHz, 1333 MT/s FSB) and 8×2 GB RAM. The *unit under test* (UUT) is equipped with eight 1 Gigabit ports on two network interface cards (NICs), while all other servers are equipped with only four 1 Gigabit ports. The network controller chip sets identified themselves as Intel 80003ES2LAN and Intel 82571EB. The UUT is connected on each NIC to three other servers each with a single direct connection using Cat.5e TP cable or better. The setup and interconnection of the servers is illustrated in Figure 2.

Each machine is running Debian 5.0.4 ("lenny") with Linux kernel 2.6.26 compiled for the amd64 (x86_64) architecture. Only for the test of Xen we needed to adjust the base system with Debian 5.0.5 ("lenny") with Linux kernel 2.6.31 for amd64. Since VMware ESXi installs directly on the host machine, the choice of a custom operating system is not applicable in this case. We investigate the following versions of the different VMMs with a Debian "lenny" as a guest system and the given Kernel installed:

- KVM 0.9.1 (kvm-72) [6]: Guest running Linux 2.6.26 for amd64
- OpenVZ 3.0.22 [7]: Guest running same OS kernel as host
- VirtualBox 3.2.8 [5]: Guest running Linux 2.6.26 for amd64
- VMware ESXi 4.1.0 [3]: Guest running Linux 2.6.26 for amd64
- Xen development version of July 16th, 2010 [4]: Guest running Linux 2.6.26 for i386 (i686; x86) with *physical addressing extensions* (PAE) support.

For the KVM and VirtualBox machines, virtual NICs are using the "virtio" para-virtualized network driver. For OpenVZ and Xen, para-virtualization is achieved implicitly through the OS-level or para-virtualization approach taken by OpenVZ and Xen, respectively. For VMware ESXi, both the "E1000" full-virtualized network card emulation and the "VMXNET 3" para-virtualized network driver (using the VMware Tools on the guest) are considered.

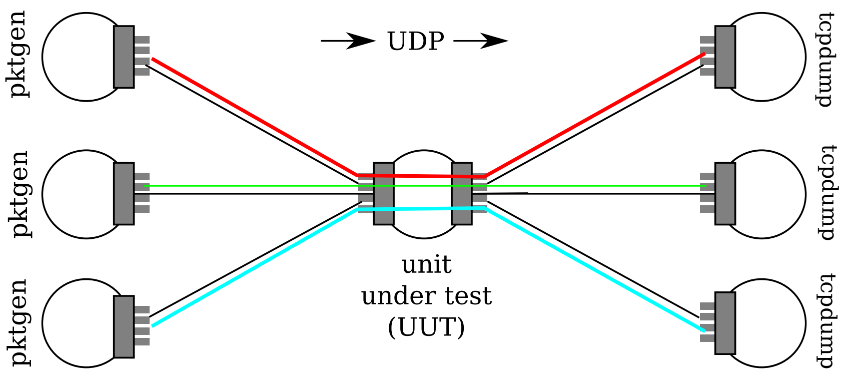


Figure 2: The hardware setup.

3.2 Measurement Metrics and Methodology

In all scenarios, we investigate the packet forwarding throughput performance of the different virtualization solutions on the UUT. In each experiment, network traffic is generated using the packet generator integrated into the Linux kernel (`pktgen` [18]). It sends out UDP packets which are forwarded by the UUT to another destination server where the traffic is recorded by `tcpdump` [19] and discarded afterwards. After stopping packet generation, the packet generator reports how many packets it was able to send. We calculate the "offered bandwidth" from this value and the time in which the traffic has been sent as well as the "measured throughput" from the data of the tcpdump file.

For our performance evaluation, we set up two different scenarios, a dedicated host scenario and a shared host scenarios, which we describe in the next sections.

3.3 Dedicated Host Scenario – Impact of Virtualization

In this scenario, we measure the packet forwarding performance of the Linux system without any virtualization (*raw*) and compare it with a setup where the same system is installed inside the tested VMM. We consider up to three traffic sources. It has to be noted that each traffic stream through the UUT does not interfere with any other stream outside the server, i.e. each traffic stream enters the server over a different network interface and also leaves the system via another distinct network interface. Each network interface is directly bridged to a corresponding virtual interface of the guest system.We consider packets with an Ethernet frame size of 64, 500, 1000, and 1500 byte for each data stream. The complete logical size "on the wire", i.e. bits used on layer 1, is 20 byte larger, due to the layer 1 preamble of 7 bytes, the start of frame delimiter, and the inter frame gap of 12 octets, as defined in the 802.3 standard[20].

For each packet size, a target bandwidth of 10, 100, 200, 400, 600, 800, 900, and 1000 Mbit/s was run nine times for statistical evidence. Since `pktgen` does not allow for setting a target bandwidth, but instead manages different loads of traffic by waiting a certain amount of time between sending two consecutive packets, this delay had to be calculated. For target bandwidth b [Mbit/s] and target packet size s [byte], the resulting delay [ns] would be $8000 \times (s + 24)/b$. This formula works well for large packet sizes, but the kernel is not able to generate small packets fast enough. Hence, we additionally adopted the inter packet delay to 1 and 0 ns for each nine tests, in order to maximize the generated traffic and send as many packets as possible.

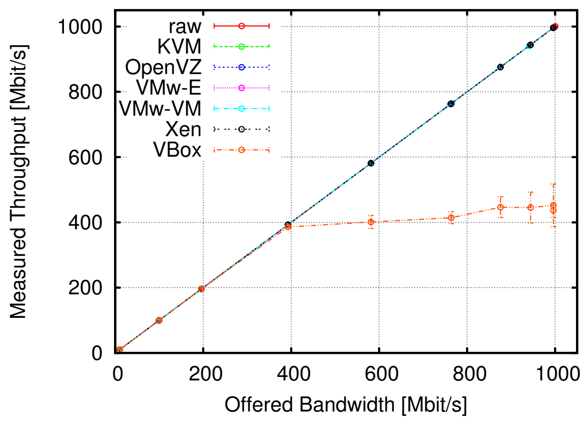
3.4 Shared Host Scenario – Impact of other Guest Systems

The consolidation of multiple virtual systems on a single physical host may have an impact on the network performance of the virtualized guests. Thus, we add a second virtual guest to the VMM, which runs another Debian "lenny" and repeat all measurements. Initially, the second guest does not run any processes generating additional load (*no load*). In order to investigate if CPU intensive processes in another guest systems has an influence of the performance of our UUT, we generate variable amounts of CPU and virtual memory load with the `stress` tool (version 0.18.9) [21] and considere two more cases. In the first case, called *light load*, `stress` was running with parameter `--cpu 1`, corresponding to a single CPU worker thread. In the second case, called *heavy load*, `stress` was running with parameters `--cpu 8 --vm 4`, corresponding to 8 CPU worker threads, and 4 virtual memory threads which allocate, write to, and release memory. In case of full virtualization (KVM, VirtualBox, VMware, and Xen), each virtual guest was assigned a single (virtual) CPU out of the 8 physical CPU cores installed on the server.

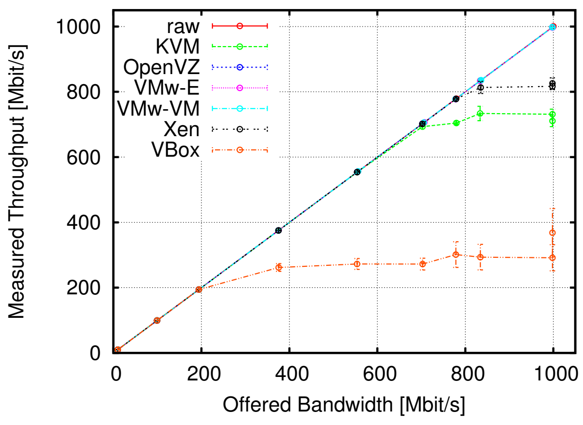
Besides CPU and memory load, we targeted the influence of a second guest systems network load, even if the guests do not share physical network interfaces. We focus in this scenario on Ethernet frame sizes of 64 byte, as the previous scenarios revealed this packet size to be most critical. Hence, we reconfigured the UUT to have only two virtual interfaces, which are bridged to different physical interfaces. We adjust the second guest system to have also two virtual interfaces, which are bridged to two distinct physical interfaces. We run the same tests as before, but this time we send traffic through the second guest. We adjust the traffic in such a way that it is 50% and 95% of the maximum rate, which we measured in the test with the second guest being idle. For this test, we also measure the throughput of the second guest to investigate if the second guest system can still forward the offered data rate.

4 The Performance Cost of Virtualization

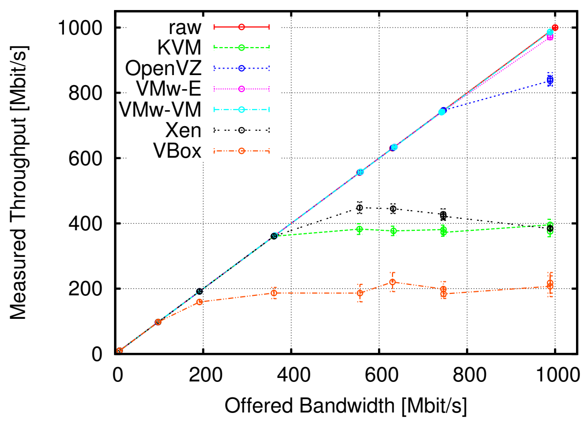
Depending on the used VMM, the network throughput performance of the virtual guests differs significantly. In Figure 3, we plot the bandwidth the traffic generator produced ("offered bandwidth") against the throughput the central test system was able to forward ("measured throughput") averaged over all nine repetitions of a test. The error bars, which in many cases can hardly be seen, present the 95% confidence intervals for the mean values. We use the following abbreviations to shorten the labels in the figures: raw for the system without virtualization, VMw-E for VMware with the full-virtualized network card E1000, VMw-VM for VMware with the para-virtualized network VMXNET3, and VBox for VirtualBox. The graphs for the 1500 byte packets reveal no difference between the system without virtualization and most of the VMMs. Only the VirtualBox



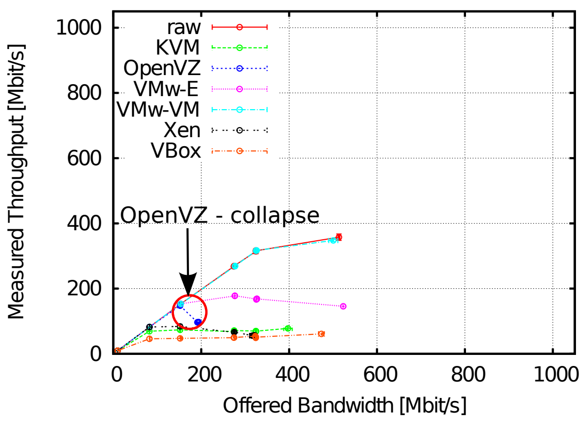
(a) 1500 byte packets



(b) 1000 byte packets



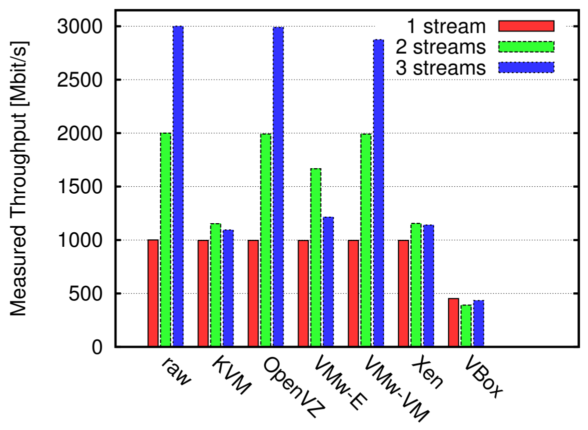
(c) 500 byte packets



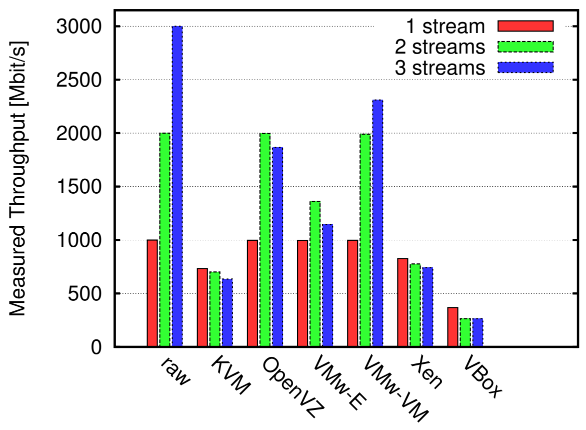
(d) 64 byte packets

Figure 3: Offered bandwidth vs throughput of VMMs for different packet sizes.

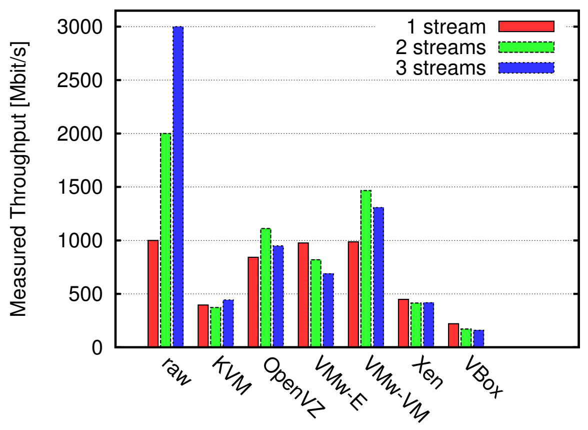
system is not able to forward all packets. For smaller packet sizes, i.e. 1000 bytes and 500 bytes, the difference between the different VMMs is clearly visible. The fact that the offered bandwidth for all packet sizes above 500 bytes increases up to the maximum of the link, i.e. 1 Gbit/s, shows that the VMMs are able to fetch the packets from the network interface. However, in all cases in which the measured bandwidth is lower than the offered bandwidth, a significant number of packets get lost in the *unit under test* (UUT). Looking at Figure 3d, two additional effects can be noticed for the smallest used packet sizes, i.e. 64 bytes. Not all graphs extend to the same length on the x-axis. This means that the traffic generator was not able to offload the same amount of traffic to the UUT in all cases. This behavior is caused by *layer-2 flow control*. Whenever the frame buffer of the network card is fully occupied, the network card sends a notification to the sending interfaces to throttle down the packet rate. The maximum throughput of the UUT is never affected by this behavior. Hence, we keep it activated, as it provides another option to the test system to react on traffic rates it cannot handle besides dropping packets. Please note that in all cases in which the layer-2 flow control is actively reducing the number of sent packets, the test system already drops packets. This behavior, therefore, never influences the maximum throughput of the system. The second observ-



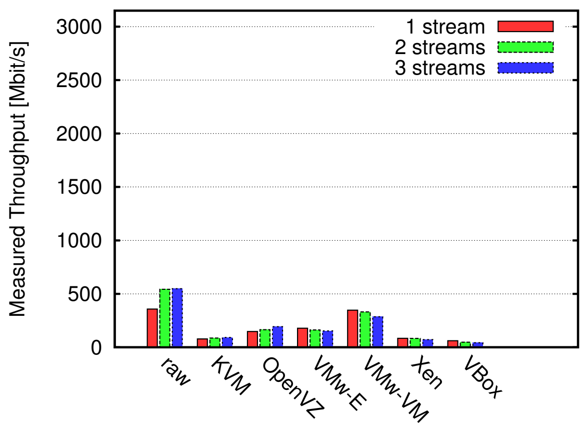
(a) 1500 byte packets



(b) 1000 byte packets



(c) 500 byte packets

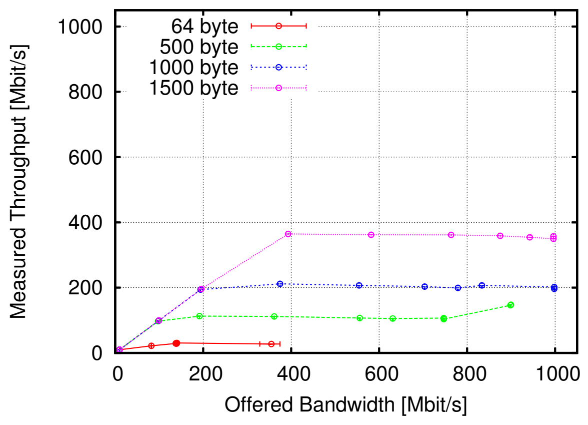


(d) 64 byte packets

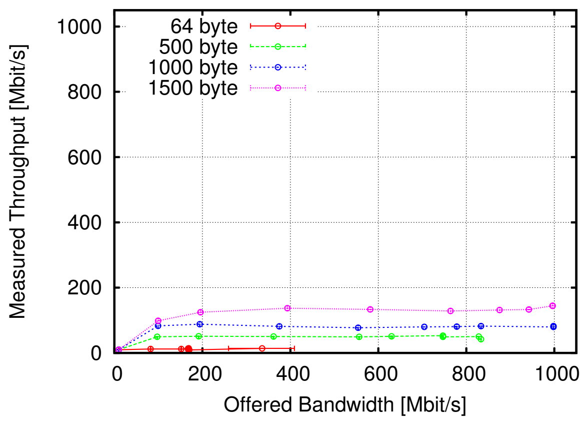
Figure 4: Maximum accumulated throughput of VMMs.

able effect is the behavior of OpenVZ when forwarding 64 byte packets. The OpenVZ system achieves a maximum throughput at a given input rate and collapses afterwards. We also notice this effect in other scenarios with 64 byte and 500 byte packet sizes. But in these cases, the throughput rate does not only collapse but is heavily varying.

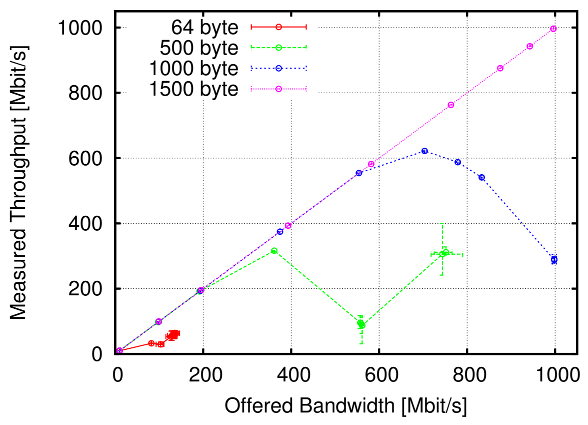
Next, we consider how the throughput of the UUT scales if we increase the number of traffic generators. Figure 4 depicts the maximum achieved throughput with one, two, and three traffic sources. Note that as we have seen, e.g. in Figure 3c, the maximum achieved throughput is not necessarily recorded at the highest offered bandwidth. The bar plot for the UUT without any VMM reveals that in all cases, except the one sending the 64 byte packets, the performance of the raw system scales linearly, i.e. we measure a throughput of 1 Gbit/s, 2 Gbit/s, and 3 Gbit/s. In the case of 64 byte packets, Figure 4d reveals a limitation of the raw system, as the accumulated throughput for two and three traffic generators is the same. In this case the raw system uses layer-2 flow control to throttle down the traffic generators. But the raw system is able to forward all received packets, in contrast to all VMMs which accept up to 600 Mbit/s input rates but drop packets down to the depicted maximum accumulated throughput.



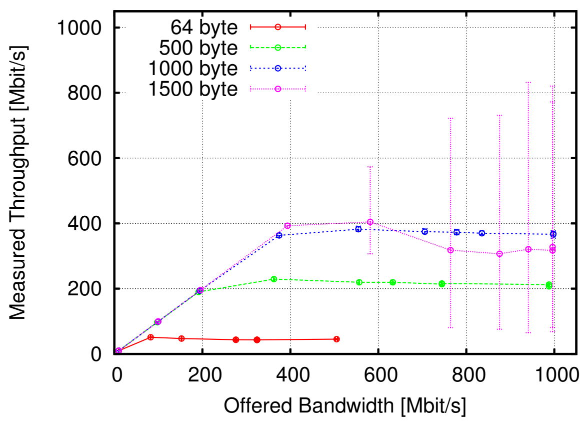
(a) KVM



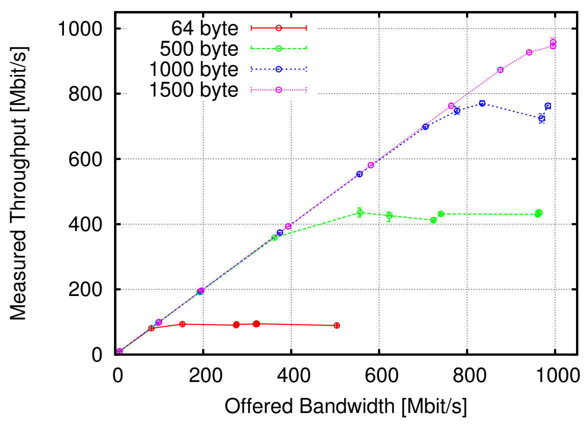
(b) Virtualbox



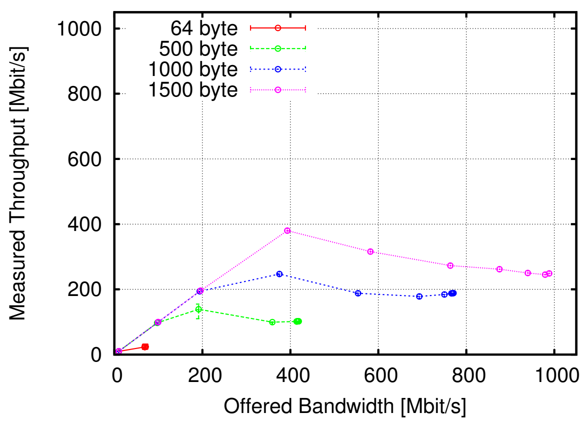
(c) OpenVz



(d) VMware-E1000



(e) VMware-VMXNET3



(f) Xen

Figure 5: Differences in throughput of three streams for different VMMs

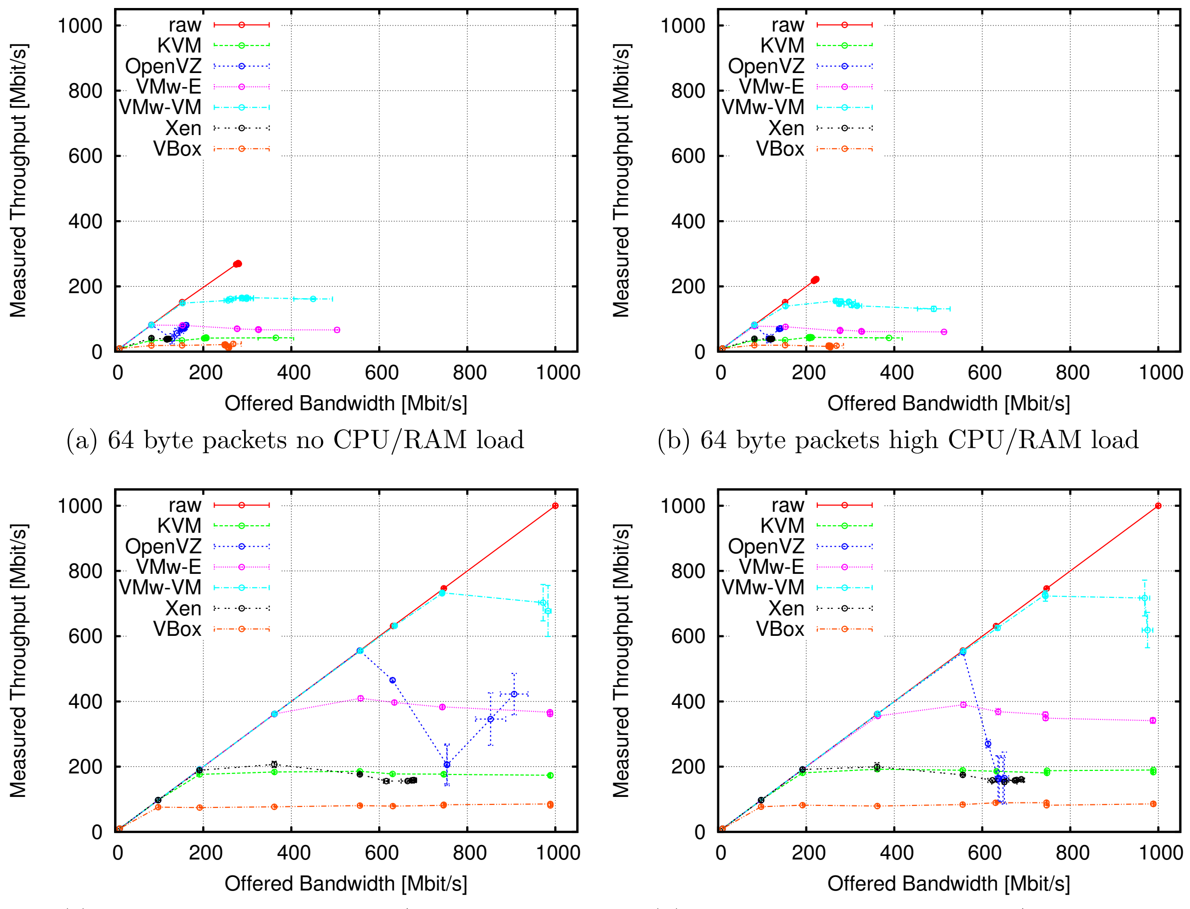
More details of the throughput performance are presented in Figure 5. In its subfigures we plot the mean measured throughput as well as the minimum and maximum measured throughput of all tests for the different VMMs. This means, that whenever there is a noticeable error bar plotted, the three parallel streams have a different throughput measured at the receiving devices. The measured throughput for KVM, cf. Figure 5a and Virtualbox, cf. Figure 5b, shows that both systems have a stable maximum throughput rate. This means that the measured throughput reaches a certain threshold but does not increase any further. But the measured throughput does also not decrease if more traffic is offered to the UUT. The VMMs just drop packets which exceed the maximum achievable throughput. However, the maximum, mean, and minimum bandwidth forwarded through the VMMs is not diverging, i.e. the measured throughput of all parallel streams is equal. In case of OpenVZ, cf. Figure 5c, we observe a different behavior. For packet sizes of 1000 byte, the maximum throughput is reached for an offered bandwidth of about 700 Mbit/s. If more 1000 byte packets are offered, the throughput dramatically decreases down to about 300 Mbit/s. For 500 byte packets, the behavior is even worse. In this case, the maximum throughput is achieved at an offered rate of about 380 Mbit/s. For higher offered packet rates, the throughput of the system is changing between minimum rates about 50 Mbit/s and rates up to 400 Mbit/s for individual flows. Furthermore, the throughput of the parallel flows is no longer the same, but varies reasonably. Using VMware without the para-virtualized driver, cf. Figure 5d, we measure a high variation in the throughput of the parallel flows while offering 1500 byte packet streams. For offered bandwidths higher than 500 Mbit/s, the difference between the parallel streams is increasing. In the worst case, we measured about 100 Mbit/s for the lowest forwarding bandwidth of the three streams and about 800 Mbit/s for the highest forwarding bandwidth. In Figure 5e, the measured bandwidth for VMware with the para-virtualized network driver VMXNet3 is depicted. This configuration provides high throughput rates and does not have the problem of different forwarding rates for different parallels streams. However, even in this configuration, the VMM takes more packets of the network card than it can handle, which results in high packet loss rates. The results for the Xen VMM are shown in Figure 5f. In the considered scenarios, Xen achieves a forwarding bandwidth that does not vary much between the parallel streams. However, the maximum forwarding bandwidth is not achieved for the maximum offered traffic. Instead, the maximum forwarding bandwidth is reached for a lower offered traffic rate and decreases for higher rates.

The results presented in Figure 4 and Figure 5 show that most VMMs have a throughput limitation which depends on the packet size of the offered stream. All VMMs except OpenVZ reveal this limitation even for 1500 byte packet sizes. Only OpenVZ is able to forward packets of this size with the same performance as the raw system. However, if we look at the results for the other packet sizes, we clearly see the impact of virtualization even for OpenVZ. The varying throughput rates for smaller packet sizes makes it hard to foresee the forwarding behavior of OpenVZ. In the same range, i.e. 500 byte packets and below, VMware with the para-virtualized VMXNET3 driver outperforms all other solutions.

In conclusion, we see that the performance costs of virtualization increases, the more small packets are sent and the higher the bandwidth on the UUT is. Even OpenVZ, which only provides OS-level virtualization, has a noticeable influence on the systems' performance in these cases. The biggest problem identified is that the VMMs try to take as many packets of the network card as possible and therefore prevent layer-2 flow control. Therefore, the data link layer cannot limit incoming data rate and a high number of packets are lost in the VMM.

5 Isolation of Guest Systems

In the following, we study how two virtual guests are interfering with each other. Therefore, we first conduct tests with a second virtual guest Linux that is only idle. The influence of a second idle guest compared to only one guest running is mostly negligible. Thus, we focus on the more interesting results.



(a) 64 byte packets no CPU/RAM load (b) 64 byte packets high CPU/RAM load

(c) 500 byte packets no CPU/RAM load (d) 500 byte packets high CPU/RAM load

Figure 6: Maximum accumulated throughput of VMMs.

The impact of a second virtual guest, which is performing CPU and memory intensive tasks, is also small in most cases. In Figure 6, we compare the impact of a second virtual guest system running CPU and RAM intensive tasks, as described in Section 3.4, while forwarding two packet streams over the first guest system. Figure 6a shows the mean forwarding capacity per stream while the second guest is idle. The plots for the measured throughput develop as expected from the previous section. All VMMs receive more packets from the network interfaces than they can forward and a high number of packets are lost. Comparing these results to the scenario in which the second guest is running CPU and memory intensive tasks, cf. Figure 6b, there is hardly any difference visible. If we consider forwarding two streams of 500 byte packets with an idle second guest, cf. Figure 6c, against a second guest running CPU and memory intensive tasks, cf. Figure 6d, some small changes can be seen. These differences apply to VMware with the para-virtualized network driver and OpenVZ. For VMware, the plot might indicate a lower throughput performance for high offered loads. However, the confidence intervals overlap. Thus, this result is not statistical significant and need further investigation. In case of OpenVZ, Figure 6d shows that the variation of the forwarded bandwidth is smaller while the measured mean throughput is decreasing rapidly.

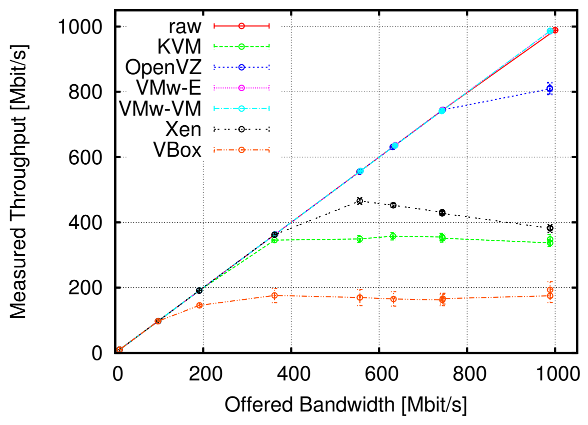
We provide the difference in maximum achieved throughput in Table 2 for CPU intensive (*light*) and CPU and memory intensive (*heavy*) loaded second guest. It has to be noted that positive values in Table 2 correspond to an increase of throughput under these conditions. For instance, Xen achieves in any considered scenario a higher maximum forwarded bandwidth if a second virtual guest is running CPU or CPU and memory intensive tasks. Only OpenVZ is negatively affected by more than 10% relative forwarding performance considering small packets. This effect might be caused by the fact that OpenVZ, in the original implementation, does not foresee the option to restrict the execution of processes to predefined CPUs. Thus, the `stress` processes are run on all CPUs compared to the other scenarios, where the `stress` processes are strictly bound to the CPU of the second guest. It has to be noted that both OS containers in OpenVZ are started with the "CPUUNIT" parameter, which means equal sharing of CPU resources. This result is surprising and we intend to study it closer in future work.

Table 1: Averaged offered bandwidth (in) and throughput measured(out) for 2nd guest in background at 50% relative load, c.f. Figure 4d.

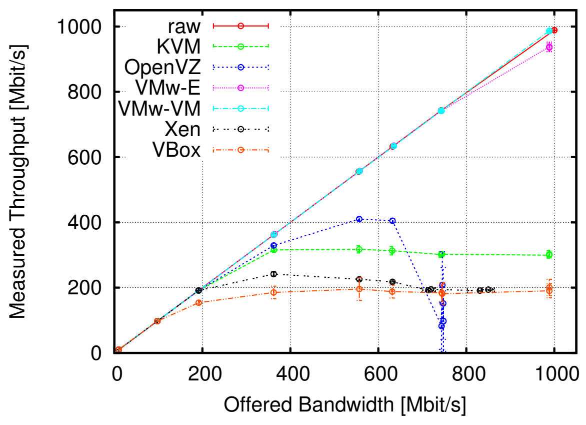
packet size	raw		KVM		OpenVZ		VM-E1000		VM-VMXNET3		Xen		VirtualBox	
	in	out	in	out	in	out	in	out	in	out	in	out	in	out
1500	153.1	153.1	37.1	37.1	72.1	72.1	71.8	71.8	152.2	151.9	37.1	36.8	37.1	36.7
1000	153.0	153.0	37.1	37.1	72.1	72.1	71.8	71.8	152.0	151.8	37.1	36.6	37.1	36.0
500	153.0	153.0	37.1	37.1	72.1	68.1	71.8	71.8	152.0	151.7	37.1	36.1	37.1	36.2
64	153.0	153.0	37.1	37.0	72.1	62.5	71.8	71.8	152.1	151.8	37.1	35.4	37.1	36.8

Table 2: Averaged relative throughput changes considering a single traffic source and a second guest running CPU/Memory intensive processes,cf. Section 3.4.

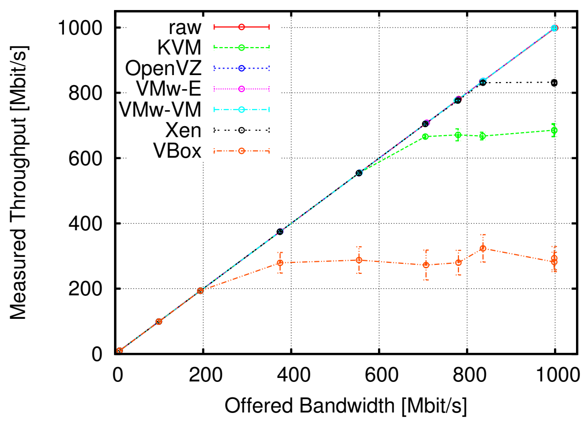
packet size	KVM		OpenVZ		VM-E1000		VM-VMXNET3		Xen		VirtualBox	
	light	heavy	light	heavy	light	heavy	light	heavy	light	heavy	light	heavy
64	4.2%	-0.4%	1.0%	-18.2%	-2.7%	-4.5%	-0.5%	-0.6%	6.8%	7.9%	2.1%	2.0%
500	5.5%	4.0%	1.5%	-11.4%	-4.4%	-5.3%	0.0%	-0.1%	3.7%	3.4%	-2.0%	-4.0%
1000	5.1%	5.6%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.8%	1.7%	3.6%	-4.8%
1500	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	3.4%	2%



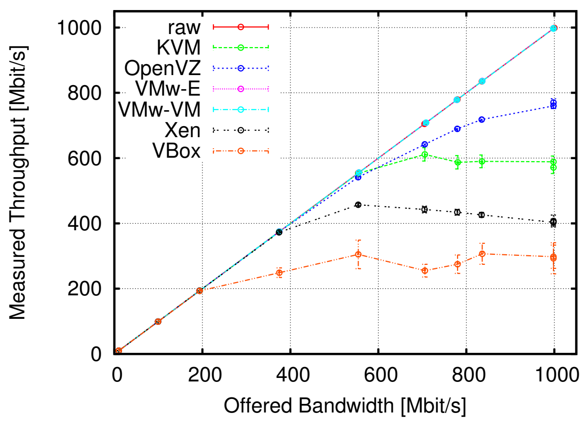
(a) 500 byte packets without network load



(b) 500 byte packets with high network load



(c) 1000 byte packets without network load

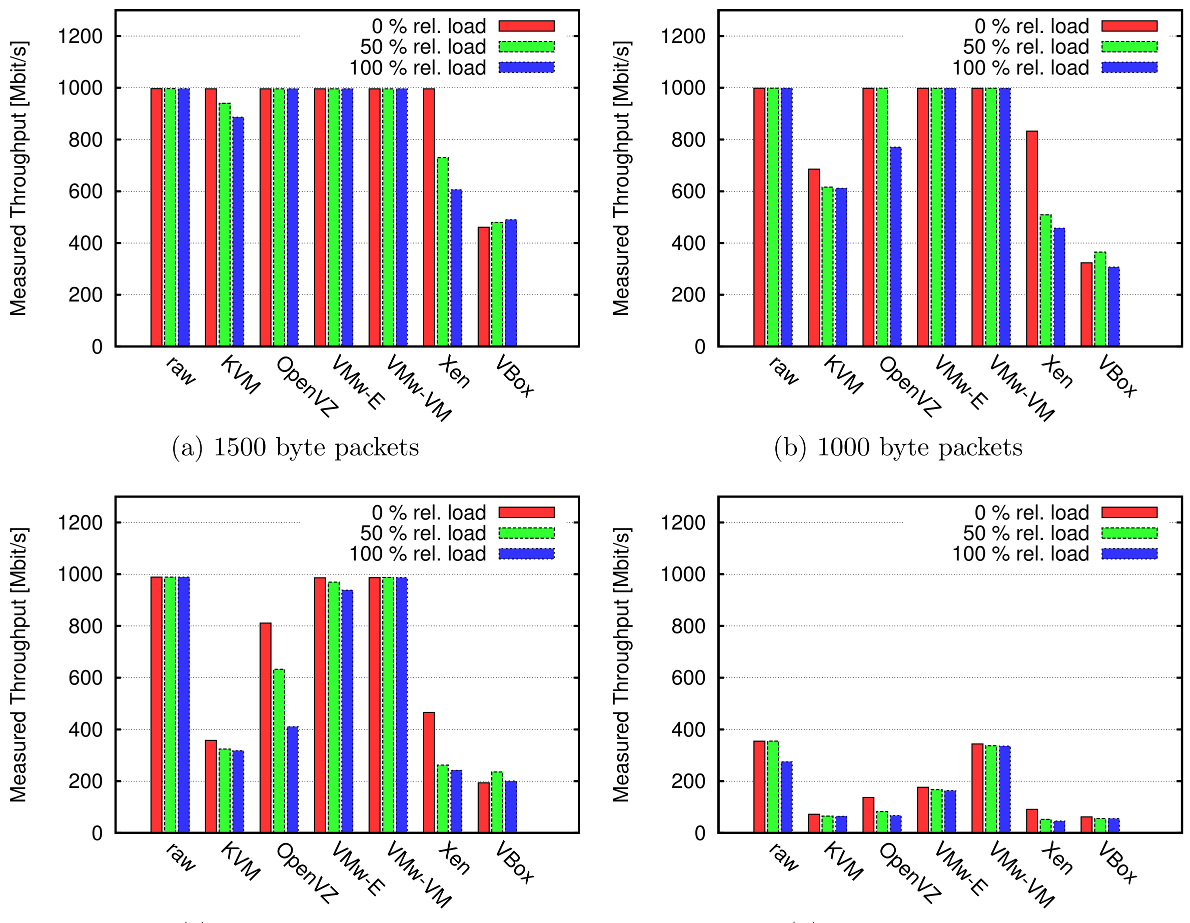


(d) 1000 byte packets with high network load

Figure 7: Throughput while transmitting 64 byte packets over a second VM.

The impact of a second guest is clearly recognizable, whenever it is handling small packets, as depicted by Figure 7. In the subfigures, we contrast the throughput performance of the VMMs forwarding 500 byte packets with an idle second guest, cf. Figure 7a, and a second guest forwarding the maximal 64 byte packet rate we measured in the dedicated host scenario, cf. Figure 7b. It is clear to see that the measured throughputs using Xen and OpenVZ as VMMs are noticeably decreasing. In the scenario with a high network load on the second guest, OpenVZ is no longer able to fetch all the packets from the network interface. We can see this problem in Figure 7b by looking at the offered bandwidth. In our tests, it was not possible to offload more than about 750 Mbit/s to the UUT. Although the measured throughput for OpenVZ is not reaching the maximum rate, it is again heavily varying. These effects are also visible for larger packet sizes, cf. Figure 7c and Figure 7d. Again Xen and OpenVZ are affected most, but with this larger packet sizes, OpenVZ does not show the fluctuation of the measured throughput.

In Figure 8 we present the maximum measured throughput for all scenarios in which the second guest forwards 64 byte packets. Even if the UUT is only handling 1500 byte packets KVM and Xen are not able to achieve their maximum throughput. But if we consider the results for streams of 1000 byte and 500 byte packets, also OpenVZ



(a) 1500 byte packets

(b) 1000 byte packets

(c) 500 byte packets

(d) 64 byte packets

Figure 8: Maximum throughput considering a second guest transmitting small packets relative to the maximum throughput in the dedicated scenario, c.f. Figure 4d.

and VMware without the para-virtualized network driver are affected, whereas the raw system and the VMware guest with VMXNET3 driver still handle all packets.

For 64 byte packets it seems as if the VMware system with the para-virtualized VMXNET3 driver outperforms the unvirtualized system. But this is not the case. Due to the fact that we send packets through the second guest relative to the performance we measured before, the raw system handles 270 Mbit/s of 64 byte packets in the background and forwards all packets. The VMware system is only forwarding about 200 Mbit/s in the second guest and is buying the higher throughput of the test system by a loss rate of about 20%.

The offered bandwidth and the measured throughput for the second guest in this experiment are provided for both load scenarios in Table 1 and Table 3, respectively. These measurements reveal that the second guest is also negatively affected. If both guests are forwarding 64 byte packets, loss rates up to 50% can be experienced. The most severe impact can again be seen for OpenVZ. It seems as if the well known problem of the Linux kernel to handle small packets, cf. [22], has an even larger impact when

Table 3: Averaged offered bandwidth (in) and throughput measured(out) for 2nd guest in background at 95% relative load, c.f. Figure 4d.

packet	raw		KVM		OpenVZ		VM-E1000		VM-VMXNET3		Xen		VirtualBox	
size	in	out	in	out	in	out	in	out	in	out	in	out	in	out
1500	275.8	270.4	72.1	63.2	127.0	127.0	126.6	126.6	274.4	268.3	72.0	56.2	72.1	42.1
1000	275.6	270.4	72.1	61.3	127.0	102.7	126.7	126.6	274.3	267.6	72.1	50.9	72.1	41.5
500	275.9	271.4	72.1	60.8	125.9	80.1	126.6	126.6	274.3	267.5	72.1	48.9	72.1	40.2
64	274.9	269.3	72.1	64.6	124.4	62.7	126.6	126.6	274.4	268.4	72.0	47.2	72.1	42.4

using OS-level virtualization. This effect has to be considered when planning experiments on experimentation facilities using this kind of virtualization, as it might cause a high variance in the results of experiments, depending on the processes in other guest systems.

6 Conclusion

In this paper, we analyzed the impact of virtualization for a single system in terms of packet forwarding throughput. We compared KVM, OpenVZ, VMware, Xen, and VirtualBox to a raw host system and the impact of another virtual guest system performing CPU and memory intensive tasks as well as forwarding of small packets. Our results revealed that the impact of virtualization is noticeable for most virtualization platforms even when only considering the throughput on a single network interface. When increasing the number of interfaces and traffic sources, each VMM revealed a throughput bottleneck, which depends on the size of the packets being transmitted but is significantly lower than the forwarding capacity of the raw system. The impact of a second guest, which is idle or running only CPU and memory intensive tasks, is rather small. However, a second guest that forwards small packets even at comparably low rates is able to influence the performance of a virtualized system severely.[1] Our findings show that virtualized systems and the traffic they handle need to be monitored. In professional system, network monitoring and exact measurements of the used systems enable the diagnosis of performance bottlenecks and definition of relocation strategies. But also for research facilities, in which many scientists share the resources of a test bed, it is crucial to record all influences which might be introduced by another test running on the same test bed, in order to provide credible scientific results.

In future work, we will integrate the results from this paper into performance models for virtualized systems and virtual software routers to analyze and optimize throughput and isolation. Additionally, we want to extend the monitoring in the G-Lab experimental facility to effectively monitor tests, which possibly influence other experiments.

[1]All measurement results are publicly available: `http://www3.informatik.uni-wuerzburg.de/staff/schlosser/Downloads/Complete.Data.VM.Measurement.zip`

References

[1] "German-Lab (G-Lab)." [Online]. Available: http://www.german-lab.de

[2] Wikipedia, "Comparison of platform virtual machines — Wikipedia, The Free Encyclopedia," 2010. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Comparison_of_platform_virtual_machines

[3] VMware ESXi. [Online]. Available: http://www.vmware.com/products/vi/esx/

[4] Xen. [Online]. Available: http://www.xen.org/

[5] Oracle, "Virtualbox." [Online]. Available: http://www.virtualbox.org/

[6] KVM, "Kernel-based virtual machine." [Online]. Available: http://www.linux-kvm.org/

[7] OpenVZ, "Open virtualization." [Online]. Available: http://openvz.org/

[8] L. van Doorn, "Hardware virtualization trends," in *Proceedings of the 2nd international conference on Virtual execution environments (VEE)*, New York, USA, 2006, pp. 45–45.

[9] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE)*, New York, USA, 2008, pp. 1–10.

[10] M. B. Anwer, A. Nayak, N. Feamster, and L. Liu, "Network I/O fairness in virtual machines," in *Proceedings of the 2nd ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures (VISA)*, New York, USA, 2010, pp. 73–80.

[11] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow (PRESTO)*, New York, USA, 2008, pp. 33–38.

[12] ——, "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT Conference*, New York, USA, 2008, pp. 20:1–20:12.

[13] VMware, "A Performance Comparison of Hypervisors," 2007. [Online]. Available: http://www.vmware.com/pdf/hypervisor_performance.pdf

[14] XenSource Inc, "A Performance Comparison of Commercial Hypervisors," 2007. [Online]. Available: http://www.vmware.com/pdf/hypervisor_performance.pdf

[15] M. Bredel, Z. Bozakov, and Y. Jiang, "Analyzing router performance using network calculus with external measurements," in *18th International Workshop on Quality of Service (IWQoS)*, Jun. 2010, pp. 1–9.

[16] G. Bhanage, I. Seskar, Y. Zhang, D. Raychaudhuri, and S. Jain, “Analyzing router performance using network calculus with external measurements,” in *TridentCom 2010*, May 2010.

[17] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, “Trellis: a platform for building flexible, fast virtual networks on commodity hardware,” in *Proceedings of the 2008 ACM CoNEXT Conference*, New York, USA, 2008, pp. 72:1–72:6.

[18] R. Olsson, “pktgen the Linux packet generator,” in *Proceedings of Linux symposium*, 2005.

[19] “tcpdump.” [Online]. Available: http://www.tcpdump.org

[20] IEEE Computer Society, “802.3 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications,” 2002. [Online]. Available: http://standards.ieee.org/getieee802/download/802.3-2002.pdf

[21] “Stress tool.” [Online]. Available: http://weather.ou.edu/~apw/projects/stress/

[22] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: a GPU-accelerated software router,” *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 195–206, 2010.