A Hardware/Software Approach for Mitigating Performance Interference Effects in Virtualized Environments Using SR-IOV

Andre Richter, Christian Herber, Stefan Wallentowitz, Thomas Wild, Andreas Herkersdorf
Institute for Integrated Systems - Technische Universität München - Germany
firstname.lastname@tum.de

***Abstract*—Single Root I/O Virtualization (SR-IOV) is an extension to the PCI Express (PCIe) standard that allows virtual machines (VMs) to directly access shared I/O devices without host involvement. This enabled SR-IOV to become the best-performing solution for virtual I/O to date, which lead to its commercial adoption, e.g., in the Amazon EC2. On the downside, a malicious VM can exploit the direct access to an SR-IOV device by flooding it with PCIe packets. This results in a congestion on the PCIe interconnect, which leads to performance interference effects between the malicious VM, concurrent VMs and even the host.**

In this paper, we present a hardware/software approach that detects and mitigates such Denial-of-Service (DoS) attacks. On the hardware side, we propose monitoring extensions within SR-IOV devices that distinguish legal device use from malicious device use by observing the rate of incoming PCIe transactions at VM granularity. Malicious VMs are reported to the host via interrupts. On the software side, performance interference effects can then be mitigated by dynamically adjusting the host's scheduling of the malicious VM or even shutting it down.

We implement a prototype with a commercial off-the-shelf SR-IOV Ethernet controller and an FPGA board. On it, we demonstrate that appropriate scheduling of malicious VMs successfully mitigates interference effects for three cloud-relevant benchmarks. For example, Memcached is restored to 99.4% of baseline performance (compared to 61.8% without our extensions). In contrast to QoS features proposed in the PCIe 3.0 standard, our solution is more flexible. Additionally, it can be realized as an add-on to existing misuse detection hardware like the Intel Malicious Driver Detection (MDD).

***Keywords*-Virtualization, SR-IOV, Performance Interference**

I. Introduction

Virtualization technology for sharing an x86 computer's physical resources is a key enabler in today's cloud computing landscape. It enables fault isolation and allows for high utilization of costly hardware resources. This is achieved by encapsulating applications or servers into Virtual Machines (VMs) and consolidating these VMs onto a single physical machine. The resulting efficiency and reliability makes virtualized datacenters an economically worthwhile business.

In such consolidated environments, VMs often concurrently access a machines's shared resources. In certain cases, this leads to performance interference effects. For example, interference arises if two applications are deployed on different cores of a multi-core processor that share the same last level cache (LLC) [1]–[4]. If both applications run memory intensive workloads, they evict each other's data from the LLC when bringing in their own. If such interference effects can be mitigated, cloud service providers benefit, because they can offer better service level agreements.

Previous studies analyzed performance interference effects caused by different types of shared resources, like CPU cores, the CPU's memory subsystem, disk-I/O [5] and network-I/O [6]–[8]. Some studies also address multiple resources [9]–[11]. Proposed techniques for mitigating performance interference effects cover isolation of shared resources between concurrent VMs [1]–[3], [12], [13] or sharing-based resource allocations [5], [14].

However, previous work does not cover PCI Passthrough and Single Root I/O Virtualization (SR-IOV) [15], a recently introduced class of x86 virtualized I/O that is supported by commercial off-the-shelf devices. SR-IOV offers superior performance [16] relative to legacy I/O virtualization approaches like emulation [17] and paravirtualization [18], because it allows VMs to share and directly access a physical I/O device without host involvement. This boosts performance, because the overhead of multiplexing and forwarding VM-I/O via the host is offloaded to the SR-IOV device. This advantage in performance lead to SR-IOV's commercial adoption, e.g., in the Amazon EC2, where customers can purchase instances with SR-IOV network devices.

In a recent paper [19], we showed that the direct access to SR-IOV devices is exploitable. It enables a malicious VM to flood the SR-IOV device with PCIe packets, which causes the PCIe interconnect to congest. This adds delays to I/O operations, effectively degrading the performance of the attacked SR-IOV device as well as every other I/O device that utilizes congestion-affected PCIe lanes. This happens because CPUs always run at higher frequencies than I/O devices. Therefore, malicious VMs can generate PCIe packets faster than SR-IOV devices can consume them. Additionally, I/O devices have no means of preventing CPU packet generation. Eventually, such Denial-of-Service (DoS) attacks manifest in performance interference effects between the malicious VM, concurrent VMs and even the host.

In this paper, we present an approach that combines hardware and software extensions to detect and mitigate such DoS attacks on SR-IOV devices. As hardware extensions,

Conference Proceedings Article DOI 10.1109/CLOUD.2015.129 – http://dx.doi.org/10.1109/CLOUD.2015.129

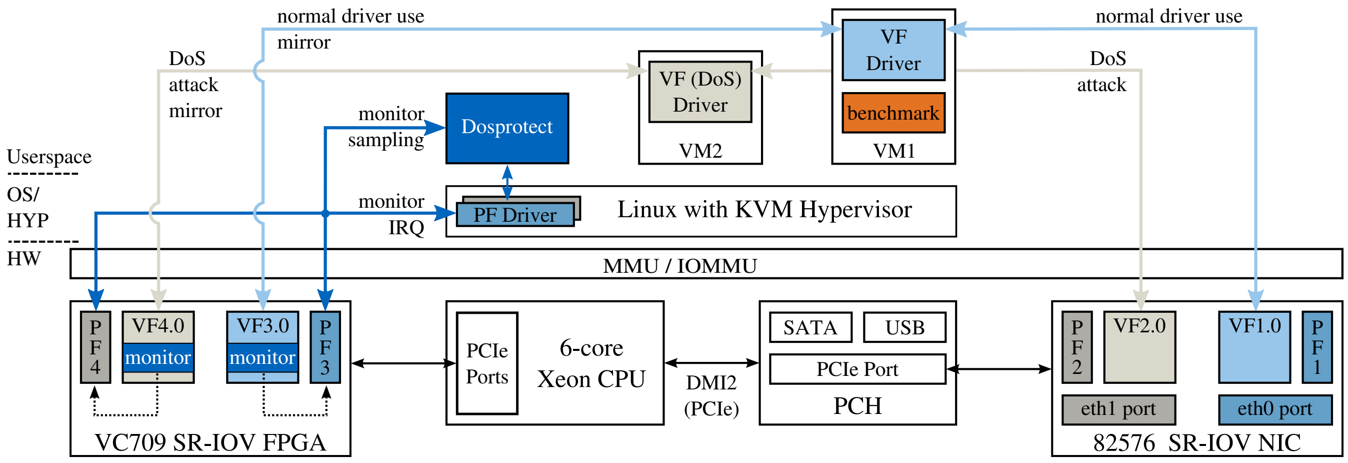

Figure 1. Block diagram of our proof-of-concept demonstrator. VM1 and VM2 each run on a dedicated core.

we propose monitoring facilities within SR-IOV devices that observe the rate of incoming PCIe transactions at VM granularity. The transaction rates are compared to a host-defined threshold value that distinguishes legal device use from malicious device use. Therefore, we contribute an approach to determine suitable threshold values. If the hardware extensions detect a malicious VM that floods the interconnect, the host is informed by an interrupt. Software extensions then mitigate performance interference effects by dynamically adjusting the host's scheduling of the malicious VM or even shutting it down.

We implement our approach on a virtualized x86 system with a commercial off-the-shelf SR-IOV Ethernet controller and an SR-IOV capable FPGA board. On this prototype, we verify our approach with help of three cloud-relevant micro- and macro-benchmarks, namely Memcached, Apache and netperf. For example, Memcached performance degrades to 61.8% of baseline performance during a DoS attack. Detecting the malicious VM and taking appropriate scheduling measures restores 99.4% of baseline performance.

The remainder of this paper is organized as follows: Section II describes design and implementation of our prototype. Section III evaluates our approach. Section IV discusses related work and Section V concludes this paper.

II. Design and Implementation

In this section, we first present our hardware prototype configuration. We go on by motivating our approach to implement PCIe transaction monitoring within an SR-IOV device and use interrupts to report malicious VMs. Subsequently, we describe how we implemented the proposed PCIe transaction monitoring and DoS detection on an FPGA. It follows a description of how we enable the FPGA to monitor transactions to a commercial off-the-shelf SR-IOV NIC. The section concludes by explaining how software extensions determine malicious VMs and how their DoS attacks can be mitigated.

A. *Hardware Configuration*

Figure 1 depicts the utilized hardware configuration: Our machine uses an Intel S2600COE dual-socket Motherboard housing a C602 chipset as its Platform-Controller-Hub (PCH). The first socket is equipped with a Xeon E5-2630 six-core CPU running at 2.30 GHz. For measurement, Hyperthreading is disabled so that VMs can be pinned to full physical cores and do not have to share cores with other threads/VMs. SpeedStep and TurboBoost are disabled as well in order to prevent non-deterministic frequency scaling. The second socket of the Motherboard is unpopulated so that our measurements are not influenced by non-uniform memory access (NUMA) effects and latencies due to socket crossings. Above described precautions minimize variances in the conducted measurements and therefore guarantee highly reproducible results without loss of generality.

The target device of our approach is an Intel 82576 PCIe card, which is a dual-port gigabit Ethernet SR-IOV NIC. SR-IOV devices offer multiple Virtual Functions (VFs), which can be directly assigned to VMs via PCI Passthrough. This enables VMs to directly access the physical PCIe device without involvement of the host, which enables state-of-the-art performance [16]. VFs are created and destroyed by Physical Functions (PFs) that are exclusively managed by the host. SR-IOV devices may have multiple PFs. This is the case for the 82576, where each gigabit Ethernet port has one PF, which can itself spawn up to eight VFs each. The NIC is configured to have one VF spawned for each PF (named VF1.0 and VF2.0). VF 1.0 is used by the uncompromised VM1 to conduct performance benchmarks like network throughput tests. VF2.0 is used as a DoS attack target by malicious VM2.

In our previous paper [19], we showed that passthrough I/O devices that are (i) connected to or (ii) integrated into the PCH are specifically prone to DoS attacks. This, and the fact that vendors (plan to) implement SR-IOV devices directly into an x86 system's PCH[1] motivated us to connect our target I/O device to the PCH's PCIe slot. As our target I/O device is an Intel 82576, we have a NIC and PCIe-slot combination that is also featured in our previous paper cited above. This renders results of this paper more comparable. Investigating other PCIe-slot (e.g., CPU-integrated slots) and NIC (e.g., 10 GBit) combinations is subject to future work.

B. Motivating Monitoring Extensions for SR-IOV Devices

The main objective of our approach is to monitor PCIe transactions rates to the 82576 NIC at VF granularity. First, we evaluated the use of CPU performance monitoring counters (PMC) for this task, but concluded that they are not ideally suited, because they lack granularity: PCIe transactions can be monitored by configuring PMCs to track Memory Mapped I/O (MMIO) transactions. However, this configuration tracks non-PCIe MMIO as well as transactions to multiple PCIe devices. This disqualifies PMCs for scenarios where a VM is assigned to multiple PCIe devices. Additionally, using interrupts to notify the host of suspected DoS attacks, like our approach does, is non-trivial with PMCs if an offending VM is deployed on more than one core. Although PMCs can be configured to interrupt the host if their counter value exceeds a threshold, it must be done separately for each core. Therefore, configuring PMC thresholds becomes difficult if the malicious VM distributes its DoS attack on multiple cores.

These reasons motivate an implementation of PCIe transaction monitors *within* the SR-IOV device. It enables monitoring at VF granularity and works with VMs that distribute their DoS attacks between multiple CPU cores, because monitoring is done at the target, where incoming PCIe transactions join. Additionally, it eases detection of malicious VMs, because VF to VM assignment is static and therefore known by the host.

More importantly, our approach of using interrupts to alert the host about VF misuse is already employed in commercial off-the-shelf hardware. For example, the Intel I350 Ethernet controller offers Malicious Driver Detection (MDD) [20]. If enabled, MDD checks if a VM tries to supply wrong source MAC addresses (spoofing) or invalid transmit descriptors (e.g., wrong IP header sizes). However, MDD looks for security violations concerning Ethernet and some of its higher-level protocols and does not detect PCIe DoS attacks. Our approach aims to extend these misuse report facilities by adding PCIe transaction monitors and DoS attack detection. By leveraging and sharing existing interrupt report facilities, our approach can be realized with only few extensions to existing hardware.

[1]This is, for example, considered/realized for storage controllers. In fact, the PCH of our test system features an SR-IOV capable SATA controller. Unfortunately, at the time of our evaluations, no SR-IOV capable drivers were available; therefore we exclusively evaluate networking performance.

C. Hardware Extensions (Monitoring / DoS Detection)

Unfortunately, it is not possible to extend the 82576 NIC itself with hardware monitors. Therefore, we built a proof-of-concept prototype by using a Xilinx VC709 FPGA board (see Figure 1). The VC709 is a development board with a Virtex7 FPGA that is capable of integrating an SR-IOV enabled PCIe endpoint with two PFs and six VFs. The board is connected to a slot of the CPU-integrated PCIe controller. We configured the VC709 to provide one VF per PF, so that the board acts like a clone of the 82576 NIC. VF3.0 is intended to mimic VF1.0 of the 82576 NIC, VF4.0 is the counterpart to VF2.0.

For each VF, we implemented **monitoring** via *two* counters for each type of PCIe transaction; two read counters and two write counters. The two counter-sets work differently and are intended for different purposes. One set is accessible via the PF to which the VF belongs. It monotonically increases the respective counter for each observed transaction, and is implemented as Clear-on-Read for low overhead use by monitoring/scheduling software.

The second counter-set is internal to the VC709 and is used for **DoS detection**. Hardware periodically compares the counters to a host-configurable *threshold value* and clears them afterwards. If a counter exceeds the threshold value, a bit in the VF DoS Detection Register (VFDDR) is set and an interrupt from the PF is issued to the host. The VFDDR is accessible via the PF, and each bit of the word stored in the register represents an overflow condition for a specific VF. On reception of the interrupt, software in the host system can then read out the VFDDR and determine the offending VF. The *period* at which these counters are checked is also host-configurable. In conclusion, threshold value und counting/checking period determine the rate of PCIe transactions that, if surpassed, indicates a DoS attack.

D. Emulation of 82576 Monitoring (Transaction Mirroring)

With the VC709 monitoring implementation mentioned above, we emulate a system where the 82576 NIC appears to have these counters built-in. We employ a host system with Ubuntu 14.04 running Linux 3.13 and the KVM Hypervisor. We spawn two fully virtualized Linux guest VMs that run the same OS and kernel as the host. Each VM is pinned to its own core and assigned 4 GB of RAM (total 32 GB). The guest VMs are directly assigned to VF1.0 and VF2.0 of the 82576 NIC and their counterpart VFs from the VC709 board, respectively.

The idea of this setup is to use the VC709 counters for counting each PCIe transaction that a VM issues to its 82576 VF. Therefore, we modified the original Linux VF driver for the 82576 VFs such that each PCIe transaction is also issued to the counterpart VC709 VF. The same mirroring is implemented for the attack code of the malicious VM that issues DoS attacks to VF2.0. Figure 1 depicts the mirroring.

Naturally, transaction mirroring imposes a small overhead on the conducted benchmarks (single-digit percentage, which is suitable for a proof-of-concept). Detailed numbers will be presented in the evaluation section. However, keep in mind that there is no overhead at all if our proposed hardware extensions are not emulated via FPGA and transaction mirroring, but instead built into the SR-IOV device itself, like we propose.

E. Software Extensions (DoS Mitigation Scheduling)

Finally, above described prototype can be used to mitigate DoS attacks in a running virtualized environment. A simple approach is to shut down VMs that are reported as malicious by an IRQ, which restores baseline performance for concurrent VMs. A less harsh solution is to enforce a schedule on malicious VMs that prevents them from producing more PCIe transactions than allowed. This can be realized by taking a suitable amount of CPU time away from an malicious VM, so that it cannot cause a harmful congestion on the PCIe interconnect. In the following, the technical details of our implementation are explained.

1) Malicious VM Detection: If a VC709 counter exceeds the host-configured threshold that indicates a DoS attack (Section II-C), an interrupt is issued to the host. A VC709 PF kernel driver receives the interrupt and triggers the **Dosprotect** process (compare Figure 1). Dosprotect then reads the contents of the *VF DoS Detection Register* from the VC709 and determines the VF under attack. Subsequently, the QEMU process (aka the malicious VM) to which the attacked VF is assigned is determined.

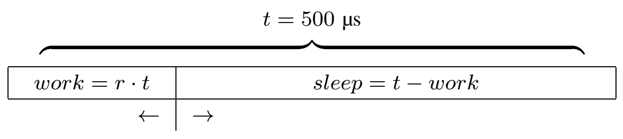

Figure 2. Scheduling malicious VMs. Parameter r is adjusted on-the-fly with help of counter feedback.

2) Mitigation Scheduling: Knowing the malicious process, Dosprotect enforces a schedule on it that interleaves sleep and wake times. This is done by explicitly telling the Linux kernel to sleep or wake the process. Figure 2 depicts the approach: A scheduling time slice of $t = 500$ µs, which is an average value for common operating systems, is divided into work and sleep quantums. To split the time slice accordingly, Dosprotect first reads out the PCIe transaction count $tr_{counted}$ that caused the interrupt. Together with the threshold value for allowed transactions $tr_{allowed}$, a first ratio $r = tr_{allowed}/tr_{counted}$ that determines work and sleep quantums is calculated. In the following, after each passed time slice t, the PCIe transaction counter is read out again and used to fine-tune r by increasing or decreasing it in small steps, but it is always true that $r \in [0, 1]$. In case the VM no longer overcommits for a certain amount of time, Dosprotect backs off and normal OS scheduling continues.

While actively scheduling, Dosprotect's CPU overhead was 5%. In systems where VMs are pinned to certain CPU cores, Dosprotect can be pinned to the same core as the malicious VM. This prevents stealing CPU time from non-malicious VMs. If no DoS attacks are going on, Dosprotect does not produce overhead.

III. Evaluation

The objective of this section is to (i) determine an appropriate, real-world threshold value for PCIe transaction rates that indicate DoS attacks in a reliable way and (ii) demonstrate that our approach successfully detects DoS attacks and mitigates performance interference effects. Both objectives are realized by employing three established network micro- and macro-benchmarks that are introduced beforehand.

Due to the nature of network benchmarks, an additional remote machine is needed that acts as a client or server. As remote machine we used a Core i7-3770 four-core CPU with 16 GB of RAM and on-board gigabit Ethernet. It runs the same Ubuntu and kernel version as our prototype. We do not use jumbo frames and leave the Maximum Transmission Unit (MTU) at its default value of 1500. All benchmarks run for a period of 10 seconds and the presented results are the average of 5 runs.

Memcached is a distributed memory object-caching server. It is used to accelerate web applications by caching frequently used database requests or small static page elements in memory. We installed Memcached in VM1; performance is measured by loading the server with the memaslap benchmark, which runs on the i7 remote machine. Memaslap is configured with 4 threads and a concurrency of 64. Other parameters are left default, which means that generated requests are randomly distributed with probabilities of 90% for `get` and 10% for `set` requests. Memaslap counts the number of completed transactions per second.

Apache is a widely used HTTP server and represents a classic application that runs in a virtualized server. Apache is running inside VM1 and provides three static HTML pages of the sizes 4 KiB, 16 KiB, 64 KiB and 256 KiB. Our i7 remote machine runs Apachebench with four concurrent threads (one per core) and measures the number of completed requests per second for a given page.

Netperf measures the data transfer performance of UDP and TCP streams. VM1 is acting as the initiator for both protocols. In all our netperf tests, we vary the message size between 16 byte and 4 KiB. We found this window to contain the relevant insights about DoS attacks during streaming benchmarks.

As a first result, we noticed that the VF driver barely issues PCIe reads. The monitored rate ranged from five to ten reads per second throughout all of the conducted benchmarks. Because they are of insignificant volume, we only consider PCIe writes in the following sections.

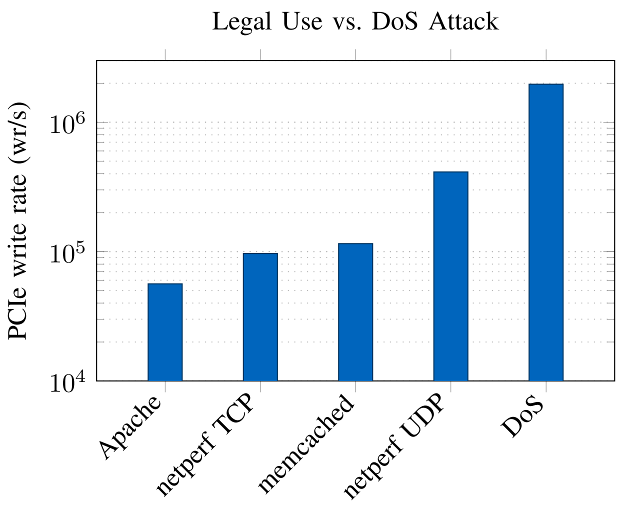

Figure 3. Maximum PCIe write rates to a Virtual Function for different benchmarks and a DoS attack. Depicted netperf TCP used 128 byte message size, netperf UDP 16 byte. Apache used 4 KiB page size.

A. *Configuring the DoS Detection*

In order to appropriately configure the DoS detection, we need to find a threshold value for PCIe transaction rates that reliably indicates DoS attacks. Therefore, we first recorded the PCIe transaction rates of the benchmarks mentioned above, running in VM1. Subsequently, we recorded the PCIe transaction rate of VM2 while executing a DoS attack on its assigned VF. In contrast to our previous paper [19], the DoS attack targets the Receive/Transmit Descriptor Base Address registers of the 82576, as we found them to cause more degradation. Additionally, 64 bit PCIe writes were used instead of 32 bit writes, because they also increase degradation.

Each benchmark was run for the parameter range mentioned in the previous subsection. Results are depicted in Figure 3. The PCIe write rate of a DoS attack is 5x larger than the maximum rate witnessed in any of the benchmarks. This gap allows a clear distinction between legal and malicious device use, which we can use to define threshold values. Possibilities include checking for rates just beneath the DoS rate, just above the highest legal rate or anything in between.

For this paper, we chose a detection threshold of 420k writes per second, which is about 1% above the highest value for legal use (netperf UDP) and did not trigger any false positives in our experiments. The VC709 DoS detection is configured to check at this rate. In the following, this configuration is used to compare (i) baseline results of the benchmarks with (ii) results during a DoS attack and (iii) results during an attack with active DoS detection and mitigation scheduling. The approach to shut down malicious VMs equals restoring baseline performance.

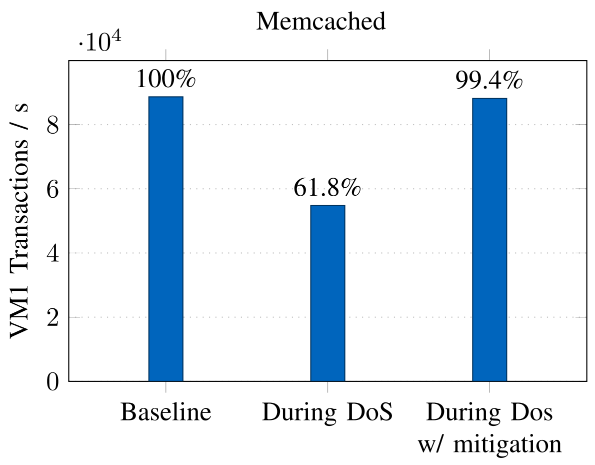

Figure 4. Memcached results.

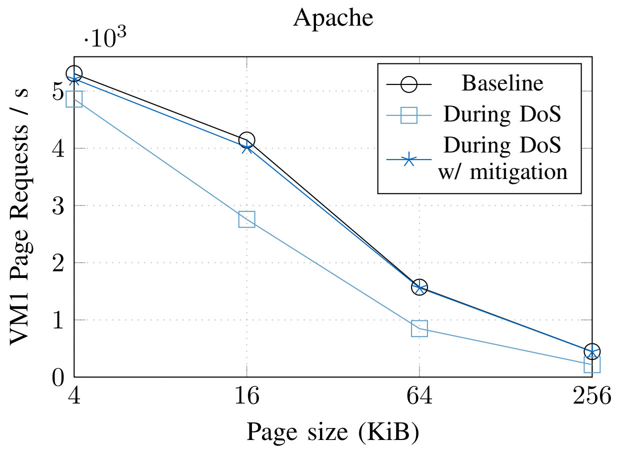

Figure 5. Apache results for four page sizes.

B. *Memcached and Apache Results*

Figure 4 shows results for Memcached. During a DoS attack without detection and mitigation, transactions/s decline by 38,2% (33887 trans/s). Identifying and properly scheduling the malicious VM restores 99.4% of baseline performance. Baseline performance results in this and subsequent benchmarks were gathered on a configuration without any of our hardware or software extensions.

Figure 5 depicts the results obtained for the Apache benchmarks. For page sizes of 4 KiB and 16 KiB, we observe 8.4% (446 req/s) and 33,5% (1386 req/s) less page requests/s during a DoS attack, respectively. The decrease gets worse with growing page sizes. At 64 KiB, we have a 46,1% (725 req/s) lower request rate, at 256 KiB it is 51,3% (229 req/s).

With DoS detection and mitigation scheduling, performance almost returns to baseline. For 16 KiB page sizes, the request rate returns to 95.6% of baseline performance. For all other page sizes, performance is restored to ~98%.

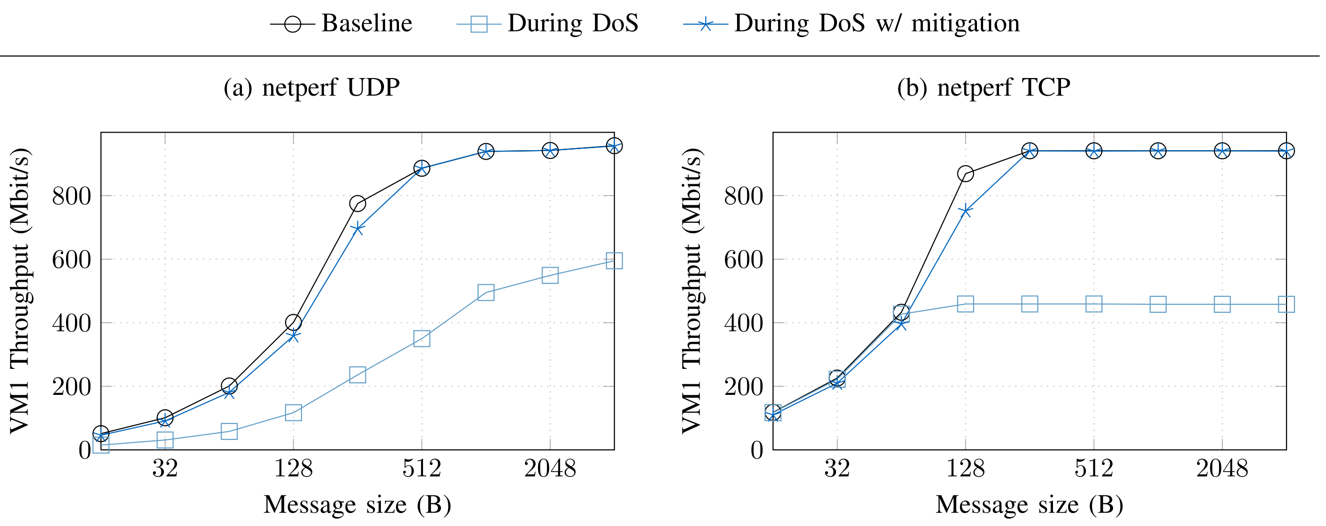

Figure 6. Netperf results for UDP and TCP throughput benchmarks for a range of message sizes.

C. Netperf Results

Figure 6 depicts the results for netperf UDP and TCP streaming benchmarks, conducted for different message sizes. During a DoS attack without mitigation, both protocols see degradation for all message sizes. UDP streams degrade between 37,8% (4096 byte message size) and ~70% (256 - 16 byte). With mitigation scheduling, baseline performance is restored for message sizes between 4096 and 512 byte. In the range of 512 down to 16 byte, our approach restores 90% of baseline performance.

TCP streams show similar results. During a DoS attack, performance degrades by 51% for messages sizes between 4096 and 256 byte. Here, DoS detection and mitigation scheduling is able to completely restore baseline performance. Starting with 128 byte (47% degradation), performance degradation during a DoS attack becomes smaller for smaller message sizes (down to 1% for 16 byte). In this range, mitigation scheduling restores between 86.5% (128 byte) and 93.7% (16 byte) of baseline performance.

In conclusion, we found that mitigation scheduling restores baseline performance for message sizes where netperf is not CPU-bound. This is true for sizes of 512 byte and greater for UDP and sizes of 256 byte and greater for TCP.

For message sizes where netperf fully loads the CPU, because it must prepare a large number of tiny packets, a malicious VM is still able to cause some degradation. This happens because the malicious VM is, despite being scheduled only for very small time slices, still able to cause a short lasting congestion on the PCIe interconnect, which still suffices to degrade some performance. In theory, this effect could be reduced by picking a smaller scheduling time slice t. This, however, has implications regarding the sampling rate of the SR-IOV device's PCIe transaction counters and the fact that the Dosprotect process, which schedules the malicious VM, is itself scheduled by the host kernel. As our choice of parameters performed fine for the Apache and Memcached macro-benchmarks, which represent real-world applications, we did not further investigate fine-tuning t.

D. Transaction Mirroring Overhead

In the following, we want to discuss the overhead that our PCIe transaction mirroring had on the conducted benchmarks. These overheads need only be considered for our proof-of-concept demonstrator. If transaction monitoring is built into SR-IOV I/O devices beforehand, like we propose, no mirroring is needed.

With active mirroring, the CPU duplicates each PCIe transaction. This results in a tiny CPU overhead (one assembler instr.) and doubles the number of PCIe packets. However, this is neglectable due to the fact that (i) the VC709 and the 82576 don't share PCIe lanes where interferences might occur and (ii) the CPU internal ring, which the original and mirrored packets traverse, is capable of much larger transaction volumes. Mirroring impact on benchmarks and DoS attacks shall be covered shortly in the following:

1) Benchmark Overhead: Mirroring for VM1, which executes the benchmarks, was only active for determining the maximum PCIe write rates of the respective benchmarks in Section III-A. We quantified mirroring impact and found variations of at most 7% on the benchmark performances. Considering the fact that the recorded PCIe write rate of the DoS attack is more than 4 times as big as the most demanding benchmark (netperf UDP), this overhead is neglectable. For evaluating our mitigation scheduling approach, mirroring for VM1 was disabled. Only VM2, which executes the DoS attacks, had activated mirroring. Therefore, performance results for the network benchmarks were not influenced. The impact of mirroring on the DoS attack code shall be discussed in the following.

2) No DoS Overhead: The DoS attack code that floods VF2.0 with PCIe write transactions is not affected by mirroring the packets to VF4.0. This is because our VC709 design does not share PCIe lanes with the 82576 NIC and completes PCIe writes faster than the NIC (compare Table I).

Table I
TIME FOR EXECUTING A FOR-LOOP (DOS ATTACK) WITH 10^8 CONSECUTIVE PCIE WRITES, DEPENDING ON DEVICE.

Device	**82576 NIC**	**VC709 FPGA**
time/write	361.38 ns	88.12 ns

Therefore, congestion on the interconnect during DoS attacks will first emerge on the PCIe lanes that connect to the 82576 NIC. Congestion in this case means that the CPU has to busy-wait for the NIC to process pending PCIe packets, so that there will be a free egress buffer slot on the port to which the 82576 is connected. In conclusion, the CPU can send a mirror packet to the VC709 while it has to wait for the 82576 to process a PCIe packet. Due to these circumstances, mirroring packets to the VC709 causes zero overhead regarding DoS attacks.

IV. Related Work

The PCIe 3.0 standard specifies Virtual Channels (VCs), which enable isolated PCIe communication by means of per-VC buffering and configurable VC scheduling. While we are not aware of CPU/Motherboard combinations that offer multi-VC support, this approach has the potential to provide isolated PCIe communication for concurrent VMs. However, an application scenario with *N* VMs requires the same amount of VCs to be implemented to provide complete isolation.

To provide a more flexible solution than the use of VCs, we pursued a monitoring based approach. Here, PCIe transactions are monitored for each virtual interface by hardware extensions embedded into the SR-IOV device itself. While this approach is more flexible, it is limited to reactive counter-measures following a detection of malicious behavior. However, we do not view the concepts of monitoring and VCs as mutually exclusive. Rather, a combination of both could be used to overcome the limitations of each approach. For example, sharing of VCs by multiple VMs can be enabled through monitoring.

We are first in tackling performance interference in SR-IOV enabled setups. Other research focuses on performance interference effects in paravirtualized environments. This includes research on the impact of schedulers on network performance [6], [7], Amazon EC2 performance interference [8], and improving CPU time sharing by accounting for time consumed in the driver domain [12]. Because SR-IOV setups are specifically designed to allow I/O requests bypassing the hypervisor, no privileged component like the hypervisor or driver domain can be used for monitoring.

Approaches that prevent performance interference effects that are caused by contention in memory subsystem components of the CPU, e.g., LLC or memory controller contention, are close to our work. Memory subsystem contention is prominently investigated in [1]–[3]. The work is conducted for OS-level workloads. However, it applies to virtualized environments if similar workloads run inside virtual machines. They leverage CPU performance monitoring counters (PMC) to monitor memory subsystem contention effects, e.g., LLC misses. Feedback from the PMCs is used for scheduling decisions that mitigate performance interference by enforcing CPU resource isolation. In [13], a similar PMC-based approach is used for VMs on Xen.

In a contrary approach, Kanemasa et al. [14] demonstrated that shortcomings of strict isolation can outweigh its benefits. They show that a 50-50 split of CPU resources between two concurrent VMs can yield lower performance than a fully-shared allocation (100% CPU for both). A similar sharing approach applied to disk-I/O is presented in [5].

In contrast to work dealing with LLC or memory controller contention, we did not use PMCs of the CPU as they provide insufficient granularity to monitor PCIe transactions (c.f. II-B). Due to these limitations, we embedded monitors into the SR-IOV device itself. Nevertheless, if current PMC limitations are overcome in future hardware, monitoring could also be done within the CPU or root complex.

In summary, we extend existing work by contributing an approach that mitigates a new class of performance interference, which has been discovered recently in virtualized environments using SR-IOV. Previous work mainly focuses on paravirtualization or uses CPU performance monitoring counters to mitigate performance interference in the memory subsystem. While we cannot use privileged components like the hypervisor or driver domain for monitoring in an SR-IOV scenario, we transfer the concept of performance monitoring counters to the virtualized I/O device.

V. Conclusions

In this paper, we demonstrated an approach for mitigating performance interference effects which manifest when SR-IOV capable I/O devices are exploited via Denial-of-Service attacks on the PCIe interconnect. To tackle the problem, we propose hardware monitoring extensions within SR-IOV devices that distinguish legal device use from malicious device use and report malicious VMs to the host. Knowing the offender, performance interference effects can then be mitigated by dynamically adapting the host's scheduling of the malicious VM or shutting the offender VM down. For a proof-of-concept, we implemented our approach on a system with a commercial off-the-shelf SR-IOV NIC and an FPGA prototyping board and evaluated it with three micro- and macro-benchmarks. Results showed that our concept successfully mitigates performance interference effects and restores close to baseline performance.

The core idea of our presented concept, implementing PCIe transaction monitoring at Virtual Function granularity within the I/O device, can be realized as an extension to existing misuse detection hardware inside commercial-off-the-shelf SR-IOV devices. Because the presented attacks can be easily executed, we recommend to consider PCIe transaction monitoring for implementation in future commercial SR-IOV devices.

Acknowledgment

This work was funded within the project ARAMiS by the German Federal Ministry for Education and Research with the funding IDs 01|S11035. The responsibility for the content remains with the authors.

References

[1] A. Fedorova, M. Seltzer, and M. D. Smith, "Improving performance isolation on chip multiprocessors via an operating system scheduler," in *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, ser. PACT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 25–38.

[2] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1. ACM, 2010, pp. 129–142.

[3] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 283–294.

[4] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 22.

[5] C. A. Lai, Q. Wang, J. Kimball, J. LI, J. Park, and C. Pu, "Io performance interference among consolidated n-tier applications: Sharing is better than isolation for disks," in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, June 2014, pp. 24–31.

[6] 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*, ser. VEE '08. New York, NY, USA: ACM, 2008, pp. 1–10.

[7] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao, "Who is your neighbor: Net i/o performance interference in virtualized clouds," *Services Computing, IEEE Transactions on*, vol. 6, no. 3, pp. 314–329, 2013.

[8] R. Chiang, S. Rajasekaran, N. Zhang, and H. Huang, "Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for i/o resources," 2014.

[9] Y. Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments." in *ISPASS*, 2007, pp. 200–209.

[10] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*. ACM, 2007, p. 6.

[11] G. Somani and S. Chaudhary, "Application performance isolation in virtualization," in *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*. IEEE, 2009, pp. 41–48.

[12] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Middleware 2006*. Springer, 2006, pp. 342–362.

[13] W. Jing, N. Guan, and W. Yi, "Performance isolation for real-time systems with xen hypervisor on multi-cores," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, Aug 2014, pp. 1–7.

[14] Y. Kanemasa, Q. Wang, J. LI, M. Matsubara, and C. Pu, "Revisiting performance interference among consolidated n-tier applications: Sharing is better than isolation," in *Services Computing (SCC), 2013 IEEE International Conference on*, June 2013, pp. 136–143.

[15] Y. Dong, Z. Yu, and G. Rose, "Sr-iov networking in xen: architecture, design and implementation," in *Proceedings of the First conference on I/O virtualization*. USENIX Association, 2008, pp. 10–10.

[16] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with sr-iov," *Journal of Parallel and Distributed Computing*, 2012.

[17] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, 2001, pp. 1–14.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.

[19] A. Richter, C. Herber, H. Rauchfuss, T. Wild, and A. Herkersdorf, "Performance isolation exposure in virtualized platforms with pci passthrough i/o sharing," in *Architecture of Computing Systems–ARCS 2014*. Springer, 2014, pp. 171–182.

[20] Intel, *Intel Ethernet Controller I350 Datasheet, Revision 2.2*, January 2014.