Performance Isolation Exposure in Virtualized Platforms with PCI Passthrough I/O Sharing

Andre Richter, Christian Herber, Holm Rauchfuss,
Thomas Wild, and Andreas Herkersdorf

Institute for Integrated Systems, Technische Universität München,
Arcisstr. 21, 80290 Munich, Germany
{andre.richter,christian.herber,holm.rauchfuss,
thomas.wild,herkersdorf}@tum.de
http://www.lis.ei.tum.de

Abstract. PCI Passthrough is an x86 virtualization technology that enables low overhead, high performance I/O virtualization. It is an established technology in server and cloud computing environments and a promising technology for sharing I/O devices in future Cyber Physical Systems that consolidate mixed-criticality applications on multi-core CPUs. In this paper, we show that current implementations of x86 PCI Passthrough are prone to Denial-of-Service attacks. We demonstrate that attacks can be launched from within Virtual Machine environments and affect the performance of *every* I/O device on the interconnect. This means that malicious or malfunctioning applications inside Virtual Machines can impair the I/O performance of co-residential Virtual Machines. For example, attacking an SR-IOV capable Gigabit Ethernet NIC causes its TCP throughput to drop by 326 Mbit/s; latencies for reading 32 bit words from the NIC increase by over 650%. We investigate which hardware parameters influence the impact of such attacks and introduce three protection approaches.

Keywords: Performance Isolation, Virtualization, Passthrough I/O

1 Introduction

A current research challenge in Cyber Physical Systems (CPS) is the enablement of multi-core processor platforms for the consolidation of multiple, independent software applications [8] [7]. Parallel execution on a multi-core processor introduces concerns regarding performance and risks regarding safety and security, because they have to share resources like memory, caches and I/O devices. In order to prevent consolidated applications running on a multi-core from interfering with each other, spatial and temporal isolation of the shared resources is mandatory. This is especially important for future mixed-criticality CPS systems, where applications with different real-time requirements are consolidated, e.g. safety-critical driver-assist and high data volume Infotainment applications. At the same time, the overhead of isolation mechanisms should impair the performance of individual applications only as little as possible.

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-04891-8_15

Virtualization of computing resources is a promising approach for solving these challenges [10]. Spatial isolation of memory is nowadays enforced via hardware virtualization extensions, which ensure that co-residential Virtual Machines (VMs) do not spy on or corrupt each other's memory. These hardware extensions have low overheads and are available in modern, commercial off-the-shelf (COTS) processors and chipsets. On the x86 platform, for example, processors provide MMUs with virtualization extensions for isolating CPU-to-memory transactions, while virtualization-enabled IOMMUs isolate I/O-to-memory transactions.

Besides isolation, IOMMUs also help to solve the problem of efficiently sharing I/O devices for virtualized applications running on a multi-core. They enable VMs to directly communicate with PCI(e)-connected I/O devices, without any involvement of the Hypervisor. This is called direct device assignment or, on x86 platforms, PCI Passthrough. Utilizing PCI Passthrough also allows to make use of the PCI Express Single Root I/O Virtualization (SR-IOV) technology. SR-IOV enables a single physical PCIe device to offer several hundred Virtual Functions (VFs), each of which can be directly assigned to a distinct VM. SR-IOV offloads routines for virtualizing I/O devices into the target device's hardware. This makes PCI Passthrough with SR-IOV the currently best performing I/O virtualization approach [2], because in contrast to emulation [12] and paravirtualization [1] approaches, the computation of isolation and I/O device virtualization routines are offloaded into hardware accelerators. This minimizes processor overhead and therefore boosts I/O virtualization performance. Additionally, PCI Passthrough and SR-IOV technologies are nowadays widely deployed in server and cloud computing environments. This ensures that virtualization extensions and (IO)MMU enforced spatial separation are quite mature and tested in currently available x86 COTS hardware.

However, besides superior performance and low processor overhead, PCI Passthrough has yet-to-be-solved problems regarding temporal isolation [6], because multiple VMs running on different cores share a single bus for accessing their assigned I/O devices or VFs. Current PCI Passthrough setups have no instance that monitors access to shared devices, because the datapath of PCI Passthrough bypasses the Hypervisor. Therefore, no control instance is present that may block exhaustive access to them.

In this paper, we demonstrate the severeness of this lack of temporal isolation. We show how it can be exploited in order to significantly degrade the performance of co-residential VMs and the host system: We present how Denial-of-Service attacks on an SR-IOV capable Gigabit Ethernet NIC cause its TCP throughputs to drop by 326 MBit/s and how latencies for reading 32 bit words from the NIC to increase by over 650%. We investigate why PCI Passthrough setups are prone to such attacks and propose three different approaches for tackling the presented problems.

The remainder of this paper is organized as follows: Section 2 explains the evaluation methodology, experimental setup and threat model. Section 3 presents results of our experiments. Section 4 proposes protection approaches. Section 5 reviews related work. Finally, Section 6 concludes this paper.

2 Experimental Setup and Threat Model

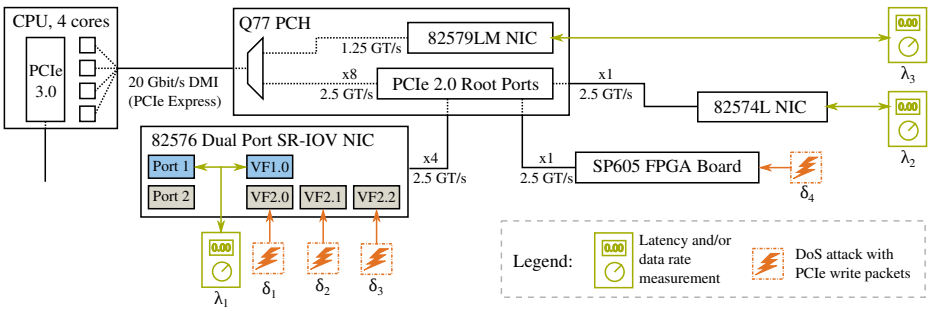

Fig. 1. Block diagram of virtualization platform.

For our experiments, we utilized an advanced x86 platform that was chosen for providing the latest generation of hardware virtualization accelerators. An overview is depicted in Figure 1. An Intel DQ77MK Motherboard is equipped with a Core i7-3770T CPU (2.5 GHz, 4 physical cores, 8 logical cores with HyperThreading enabled) and 32 GB of RAM. The proprietary 20 Gbit/s Direct Media Interface (DMI) 2.0 connects the CPU to the Q77 Platform-Controller-Hub (PCH), which provides access to the platform's PCIe 2.0 subsystem.

PCIe is a serial, point-to-point, packet-switched interconnect. A connection between two PCIe devices is called a link. A link's bandwidth is determined by its number of lanes, the encoding on the physical layer and the physical layer bit rate. The latter is always given in GigaTransfers per second (GT/s). For example, PCIe 2.0 uses 8b/10b encoding, which means a four lane connection with a speed of 2.5 GT/s (x4, 2.5GT/s) can move $2.5 \cdot 4 \cdot (8/10) = 8\,Gbit/s$ in one direction.

The Q77 hosts three PCIe 2.0 Root Ports, one featuring four lanes (x4), while the other two provide one lane (x1), respectively. Each lane has a rate of 2.5 GigaTransfers per second (GT/s). The Q77 also integrates an 82579LM Gigabit Ethernet NIC, which mimics a PCIe interface so it can be accessed easily with standard drivers. However, the connection is only capable of providing a speed of 1.25 GT/s [4]. Another Gigabit Ethernet NIC, the 82574L, is soldered onto the Motherboard and connected to one of the x1 Root Ports. The x4 Root Port is connected to a PCIe slot where we put in an SR-IOV capable, dual port gigabit Ethernet extension card with an Intel 82576 controller. Each of the controller's ports can provide up to seven VFs. We configured the card to use one VF (VF1.0) for Port 1, and three VFs (VF2.X) for Port 2. The remaining x1 Root Port is equipped with a Xilinx SP605, a PCIe board housing a Spartan 6 FPGA. We use this board as a target for standard read/write operations. Its purpose will be discussed in detail in Section 3.2. The PCIe 3.0 controller embedded onto the CPU will be discussed in Section 3.3.

On the software side, the host system is running Ubuntu 12.10 with the KVM Hypervisor. Each guest VM uses the same Ubuntu version and kernel as the host and is assigned 4096 MB of RAM.

2.1 Threat Model

Our attack scenarios assume that the virtualization platform hosts several VMs concurrently. Each VM is pinned on a dedicated physical core with Hyper-Threading disabled. This is because we do not want two VMs running on the same core to influence each other's performance. Additionally, we needed this partitioning for accurate latency measurements, which will be explained in subsection 2.2.

We assume that an attacker overtakes VMs by gathering root privileges inside the VMs operating system, which enables him to install his own device drivers. An overtaken, malicious VM is either directly assigned to a Virtual Function of the SR-IOV NIC or to the SP605 via PCI Passthrough. In order to compromise performance of the host system and co-residential VMs, the attacker will abuse the directly assigned PCIe device by launching a Denial-of-Service attack on one of the device's Memory Mapped I/O (MMIO) resources. Accessing MMIO resources of a device means that the CPU reads from or writes to registers or other forms of memory that are located on the device.

Malicious VMs launch a MMIO DoS attack against its PCI Passthrough device by removing the device's standard driver and inserting a Linux kernel module that contains an attack method. This DoS attack method is realized with a for-loop that floods the PCIe device with 32 bit write packets. We chose write packets for flooding because a PCIe write is, in contrast to a PCIe read, a so called *posted* transaction. This means that a write transaction does not wait for an acknowledge or any other type of response. Therefore, write packets can be generated with a very high rate. The generation rate is only constrained by the CPU clock, which is 2.5 GHz in our case.

2.2 Evaluation Methodology

To quantify the impact of DoS attacks on the performance of co-residential VMs and the host system, we measured two indicators:

Latencies for PCIe 32 bit reads are a general metric of a DoS attack's impact on the interconnect level. Latency is measured by counting the time it takes from requesting a word from the PCIe device until the response with the respective data arrives at the requesting CPU core. To realize this, we extended our Linux kernel module for launching the MMIO DoS attacks with a latency measurement function. The function utilizes the Linux kernel's `readl()` instruction, which reads a 32 bit word from the target device. Time is counted using the CPU's Time Stamp Counter (TSC) register like described in [9]. There is one TSC per core available. Latency results presented in this paper are always the average of one million samples. HyperThreading has been disabled in order to prevent that an active measuring thread is pushed from its core, which would

cause erroneous results. To get consistent and reproducible results, we disabled SpeedStep and TurboBoost technologies. Otherwise, the frequency of the CPU cores would have varied depending on the current thermal conditions and the workload of other cores.

TCP and UDP throughputs of the multiple NICs of our hardware platform are the second set of indicators used in the following experiments. They directly depend on the read/write latencies of the PCIe devices, but give a better idea of a DoS attack's impact on standard I/O workloads in a computer system. Additionally, it is easier to put network throughput benchmarks in context than raw read/write latencies of interconnects. Network throughputs were measured with the `netperf` tool between our virtualization platform and a dedicated PC. We used standard TCP and UDP stream tests (1500 Byte Ethernet Frames) without any additional parameters.

3 Results

This Section presents three experiments we conducted in order to evaluate the impact of MMIO DoS attacks, including the respective results and conclusions.

3.1 Experiment 1: Attacking SR-IOV Virtual Functions

SR-IOV capable PCIe devices are built with the intention to be shared between multiple, potentially untrusted virtual machines. The 82576 NIC used in our hardware platform represents such a commercially available device. It features two disjunct gigabit Ethernet ports, each of which may be shared by up to eight Virtual Functions, respectively. The PCIe board on which the 82576 controller resides is designed to operate both Ethernet ports at full speed simultaneously (x4, 2.5 GT/s). Our `netperf` tests showed 941 Mbit/s for TCP stream tests and 961 Mbit/s for UDP stream tests for both ports in concurrent operation.

If the same Ethernet port is used by two VMs at the same time, e.g. one VM assigned to VF2.0 and one VM assigned to VF2.1, the resulting throughput for each VM halves. For 3 VMs it divides by three, and so on. Given the premise that all VMs attached to a VF are non-malicious and use the standard driver for the VF, throughput of Port 1 is never influenced by usage of Port 2, no matter how many VMs are accessing Port 2 concurrently.

In the following experiment, we want to show that malicious guests can have a huge impact on the throughput of co-residential VMs. Therefore, we first measured latencies and TCP throughput for VF1.0 (compare Figure 1, λ_1) when the rest of the system is idle. The results, 941 Mbit/s TCP throughput and a latency of 1.58 µs, shall serve as the baseline measurements and are depicted in the "none" column of Figure 2. The remaining columns show how latency and throughput of VF1.0 are impaired by MMIO DoS attacks on VFs2.X of disjunct Ethernet Port 2. Results are depicted for a single DoS attack $A = \{\delta_1\}$ and concurrent attacks $A = \{\delta_1, \delta_2\}$ and $A = \{\delta_1, \delta_2, \delta_3\}$. Each DoS attack δ_x was launched from within a VM running on a dedicated core and targeted its respective VF2.X (compare Figure 1).

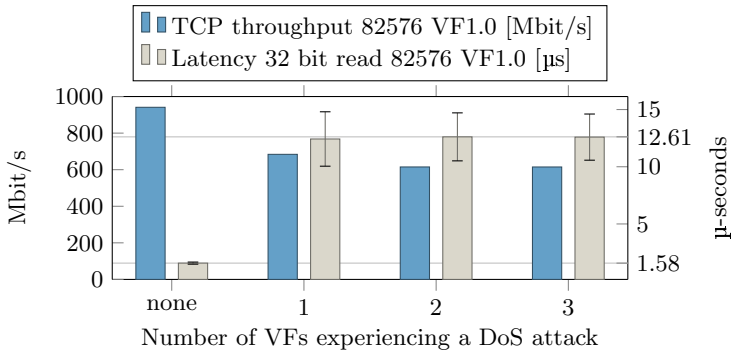

Fig. 2. Latency for 32 bit reads and TCP throughput for VF1.0 of the 82576 NIC while one or more VFs of the 82576 NIC experience MMIO DoS attacks.

The results show that a single DoS attack (δ_1 on VF2.0) from one malicious VM suffices to force a TCP throughput drop from 941 to 684 Mbit/s for VF1.0, which is allocated to a different physical Ethernet port than the DoS target. A second DoS attack (δ_2 on VF2.1) which is launched concurrently from an additional VM causes the TCP throughput to drop further down to 615 Mbit/s. A third attack does not result in an additional performance drop.

The reason for these performance drops lies within the architecture of PCIe interconnects. The CPU cores of the malicious VMs are able to produce write packets at a rate much higher than the 82576 NIC can consume them. Eventually, this leads to a congestion in the ingress buffers of the 82576 NIC. PCIe flow control mechanisms then pass on the backpressure to the next device in the PCIe hierarchy, and so on. This congestions builds up through the Q77 PCH until it reaches the CPU itself. This is also visible from the latencies shown in Figure 2, which increase from 1.58 µs for an idle system to 12.61 µs. This increase of 689% is the actual reason for the TCP throughput drop. With latencies of 12.61 µs , it is no longer possible to transfer data at near 1 Gbit/s volumes to and from the PCIe device.

That the congestion during DoS attacks spreads over the whole PCIe interconnect can be shown by exchanging the NIC for which the TCP throughputs are measured. Therefore, the 82574L and 82579LM NICs (compare λ_2 and λ_3 in Figure 1) were used for measuring TCP throughputs, one at a time. Both NICs are connected at different locations of the PCIe tree, compared to the DoS target (82576 NIC). Figure 3 compares the results of the new measurements to the previously measured throughputs of the 82576's VF1.0

The results show that the congestion spreads over the whole PCIe interconnect, because all three NICs suffer from throughput degradation. In comparison to VF1.0 of the 82576 NIC, the other NICs perform worse. Throughputs drop to 506 Mbit/s for the 82574L and 325 Mbit/s for the 82579LM. The observed gaps result from the different link speeds at which the NICs are connected to their PCIe ports. These link speeds define the time it takes to transfer an Ethernet frame to the NIC. As mentioned in Section 2, a PCIe link's data bit rate R_{data}

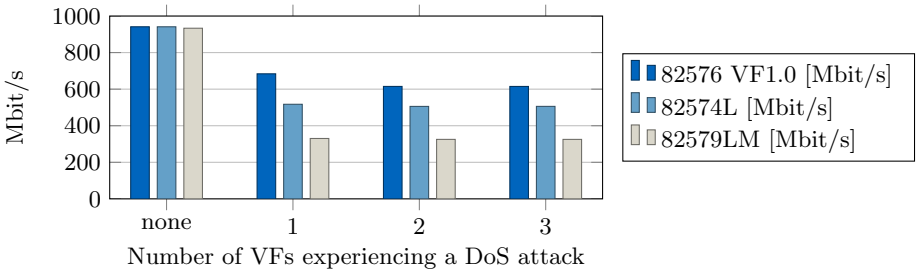

Fig. 3. TCP throughputs of the three NICs of the test system while one or more VFs of the 82576 NIC experience a MMIO DoS attack.

Table 1. Transfer time for a 1500 Byte Ethernet Frame

NIC	n_{lanes}	R_{phys}	enc	R_{data}	T_{trans}
82576	x4	2.5 GT/s	8b/10b	8 Gbit/s	1.5 µs
82574L	x1	2.5 GT/s	8b/10b	2 Gbit/s	6.0 µs
82579LM	x1	1.5 GT/s	8b/10b	1 Gbit/s	12 µs

is calculated from the number of lanes n_{lanes}, the physical layer bit rate R_{phys} and the encoding on the physical layer: $R_{data} = n_{lanes} \times R_{phys} \times encoding$.

Table 1 shows these parameters for the three NICs (also depicted in Figure 1), together with the calculated time T_{trans} it takes to transfer a 1500 Byte Ethernet Frame. As one can see, the 82576 NIC has 4.5 µs and 10.5 µs faster transfer times than the 82574L and 82579LM, respectively. This means that the overall delay for the NICs is the sum of the DoS-caused latency and the transfer time T_{trans} of the NIC's PCIe link. Consequently, the overall latencies for the three NICs differ, which results in the diverse throughput drops.

3.2 Experiment 2: Influence of the DoS Target's Processing Speed

In our second experiment, we measured how the processing speed of the device under attack influences the system's performance drop. To do so, we used a SP605 FPGA board to implement a standard PCIe endpoint. We designed the logic on the FPGA to be able to dynamically adjust the time it takes to process an incoming PCIe write packet. The minimum time the SP605 achieves is 112 ns. Processing time is modified by artificially keeping the write busy signal of the ingress port on `high`. This triggers the PCIe flow control mechanism to stop sending more packets to the SP605, which eventually leads to the congestion build-up on the PCIe interconnect. To quantify the impact of the different processing times, the following measurement series has been conducted:

1. The SP605 was attached to a VM via PCI Passthrough.
2. Packet processing times varied from 112 ns to 7.15 µs.
3. For each processing time, the VM executed a MMIO DoS attack on the SP605 device (see δ_4, Figure 1).

4. During the DoS attack, the latency for reading from the 82576 NIC was measured.

We chose the 82576 NIC for measuring, because out of the three NICs in our test system, it is the most advanced and has the fastest link speed. The results are depicted in Figure 4.

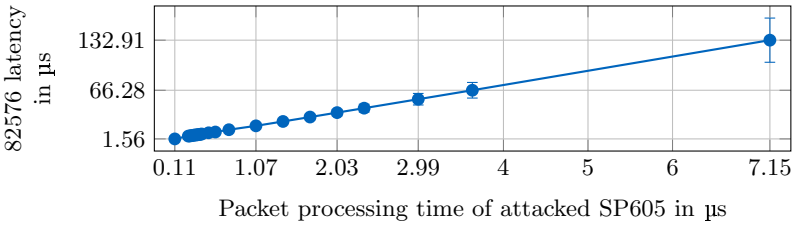

Fig. 4. Latency degradation for reading 32 bit words from the 82576 NIC while the SP605 experiences a MMIO DoS attack.

They show a linear dependency between the processing time of the DoS target and the latency for reading from the 82576 NIC. This is because the PCIe packets which request a data word from the 82576 NIC must traverse many buffers and their respective slots which are also traversed by the DoS packets flowing to the SP605 device. This follows that, in the congested case, a packet can only advance one buffer slot further on its way to the endpoint after the DoS target has processed a packet. Therefore, the additional flight time of a packet, in the case of a fully congested interconnect, depends on two characteristics:

- The number of buffer slots shared with the DoS packets.
- The DoS target's processing time of a PCIe packet.

How the different processing speeds of the DoS target translate to network throughput degradation is shown in Figure 5.

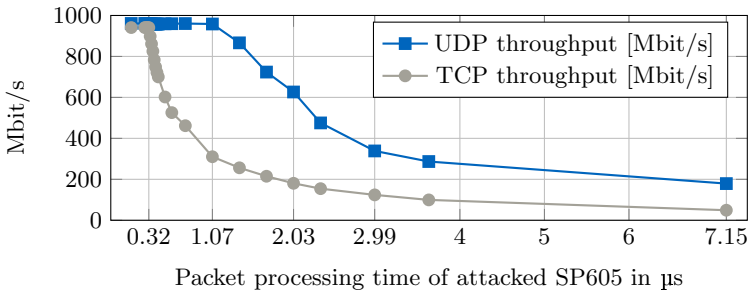

Fig. 5. TCP and UDP throughput degradation of the 82576 NIC while the SP605 experiences a MMIO DoS attack.

The results show UDP and TCP throughputs of the 82576 NIC during a DoS attack on the SP605 as a function of the packet processing time of the attacked SP605 device. TCP throughput is experiencing a drop for packet processing times greater than 320 ns, while UDP throughput sees a degradation for times greater than 1.07 µs. TCP is more affected by DoS attacks because of its protocol nature. Besides packets carrying the actual payload, additional SYN and ACK packets need to be transferred, which are also affected by the congestion on the interconnect. UDP, on the other hand, is a fire and forget protocol where every packet contains payload.

3.3 Experiment 3: Influence of the Path to the DoS Target

Our third experiment shows the influence of buffers and switching circuitry on the path from the CPU to the DoS target. Therefore, we put the 82576 SR-IOV NIC into the CPU slot, which connects to the PCIe 3.0 controller that is integrated into the CPU (compare Figure 1). As PCIe 3.0 is backwards compatible, the NIC runs with the same speed (x4, 2.5 GT/s) as in the Q77 slot.

Similar to Experiment 1 in Section 3.1, latencies for VF1.0 were measured for the idle case, as well as for one, two and three VMs attacking their respective VFs2.X. The results are shown in Figure 6, where they are compared to the results of the 82576 NIC residing in the Q77 slot (aka results of Experiment 1).

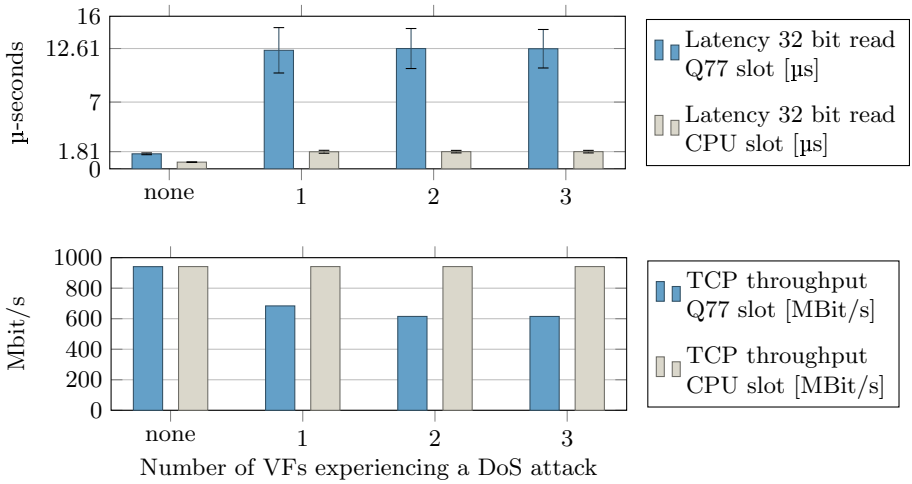

Fig. 6. Latency for 32 bit reads and TCP throughput for VF1.0 of the 82576 NIC while one or more VFs of the 82576 NIC experience MMIO DoS attacks. Results distinguished by the utilized slot (Q77 or CPU).

The results show, that for the idle case, the CPU slot is already faster than the Q77 slot (0.7 µs vs. 1.58 µs). Both latencies suffice to saturate a Gigabit link. However, when experiencing MMIO DoS attacks, the CPU slot is less impaired

than the Q77 slot. Latencies for the CPU slot increase up to 1.81 µs. This is still fast enough for Gigabit Ethernet, but a 10 Gigabit connection would most likely experience a drop in throughput. This experiment shows that a shorter path to the I/O device (less buffers and switching circuitry) lowers the impact of DoS attacks. However, we still observe an increase in latencies by 157% (compare to the 698% increase from Experiment 1).

4 Protection Approaches

The three experiments conducted in Section 3 demonstrated that the impact of MMIO DoS attacks mainly depends on three key factors:

- The production rate of PCIe packets of the attacking CPU/core, aka frequency of the core.
- The consumption rate of the PCIe device targeted by a MMIO DoS attack.
- Switching circuitry and buffers on the path to the DoS target.

In the following, we present three approaches for tackling these weak points of current PCI Passthrough implementations. Each approach aims at a different component involved in the CPU to I/O device path.

Modern x86 CPUs support performance monitoring through built-in performance counters. They are available per core and can be programmed to count different events, like cache-misses or memory requests. The counters could be utilized by the Hypervisor to enforce **I/O-access policing** for VMs running on different cores. Similar approaches have been successfully implemented in the past, e.g. for mitigating performance degradation due to shared caches or memory [5][14]. The policing algorithms could be designed to implicitly prevent or detect DoS attacks. It must be evaluated if there are suitable counters available for detecting abusive access patterns on PCI(e) devices.

The PCIe standard already defines a set of **Quality-of-Service** features. PCIe packets can be assigned different Traffic Classes (TCs). These TCs can be assigned to different Virtual Channels (VC) via various arbitration policies, like fixed priority or time based weighted round robin. VCs must be implemented as dedicated physical buffers. With the help of these QoS features and by adapting Hypervisors and hardware such that it would be possible to assign distinct TCs for each Virtual Machine, it should be possible to enforce strong temporal isolation. Unfortunately, the PCIe standard only defines the use of one Virtual Channel (VC0) as mandatory. Therefore, the major part of today's commercially available hardware only employs a single buffer for ingress and egress ports. It should be evaluated how multiple VCs and available arbitration schemes could be leveraged to enforce temporal isolation.

Experiment 2 showed that the impact of a DoS attack depends on the DoS targets processing speed, given that the interconnect is faster than the packet production rate of the CPU. This has been shown in Figure 5, where performance drops for gigabit Ethernet emerged only for processing times that are slower than a certain threshold. Therefore, it is possible to employ countermeasures into the endpoint device. Like mentioned in [3], a **DoS detection** inside

the I/O device could identify possibly harmful access patterns. If such a pattern is detected, the endpoint can discard these packets as fast as possible, which lowers the impact of the DoS attack. At the same time, the device reports the DoS attack's source via an interrupt to the Hypervisor. The latter can then take appropriate countermeasures, e.g. by shutting down the attacking VM.

5 Related Work

We are not aware of previous work that quantifies the impact of MMIO DoS attacks on the PCIe interconnect. However, other attacks relevant to PCI Passthrough have been demonstrated.

In [13], software attacks against Intel's VT-d IOMMU are demonstrated. They allow to break out of a Xen driver domain by generating specially crafted Message Signaled Interrupts with a PCI Passthrough device. Interrupt Remapping technology, which is available on recent hardware, can be used to prevent the demonstrated attacks if set up properly.

In [11], two possibilities for circumventing VT-d protection are presented. I/O devices with reconfigurable hardware could be used to spoof another device's source-id. These IDs are used by the VT-d IOMMU to identify a device's access rights to the system's memory regions. Supplying wrong device IDs enables a VM to compromise memory of co-residential VMs or the host. A second attack scenario describes the exploitation of PCI-to-PCI Express bridges with modified peripheral hardware. This attack requires physical access to the system.

6 Conclusion And Outlook

In this paper, we demonstrated that current, commercially available x86 virtualization solutions utilizing PCI Passthrough are vulnerable to MMIO Denial of Service attacks from malicious VMs. Three experiments with our test system showed how these attacks exploit the lack of temporal isolation in modern x86 setups. As a result, the performance of co-residential VMs and the host is impaired, because performance of every I/O device connected to the interconnect degrades significantly. Our findings are relevant to server and cloud computing environments, where PCI Passthrough and SR-IOV are established and deployed technologies, as well as for future CPS systems that aim to consolidate mixed-criticality applications on multi-core CPUs.

We are currently investigating and evaluating the presented protection approaches. First results showed that performance counters in modern Intel CPUs may be capable of detecting abusive access to PCIe I/O devices. Simulations with QoS enabled PCIe switches promise a strong temporal isolation between different VMs. Furthermore, we are investigating additional attack vectors like programming DMA engines of PCI Passthrough devices to execute DoS attacks or exploiting hardware bugs of PCI Passthrough devices. Such attacks might also require new/different approaches for countermeasures.

Acknowledgments. 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. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: ACM SIGOPS Operating Systems Review. vol. 37, pp. 164–177. ACM (2003)
2. Dong, Y., Yang, X., Li, J., Liao, G., Tian, K., Guan, H.: High performance network virtualization with sr-iov. Journal of Parallel and Distributed Computing (2012)
3. Dong, Y., Yu, Z., Rose, G.: Sr-iov networking in xen: architecture, design and implementation. In: Proceedings of the First conference on I/O virtualization. pp. 10–10. USENIX Association (2008)
4. Intel: Intel 7 series / c216 chipset family platform controller hub (pch) datasheet (2012)
5. Jing, W.: Performance Isolation for Mixed Criticality Real-time System on Multicore with Xen Hypervisor. Master's thesis, Uppsala University, Department of Information Technology (2013)
6. Kotaba, O., Nowotsch, J., Paulitsch, M., Petters, S.M., Theiling, H.: Multicore in real-time systems–temporal isolation challenges due to shared resources. In: Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems (WICERT) (2013)
7. Navet, N., Monot, A., Bavoux, B., Simonot-Lion, F.: Multi-source and multicore automotive ecus-os protection mechanisms and scheduling. In: International Symposium on Industrial Electronics-ISIE 2010 (2010)
8. Nowotsch, J., Paulitsch, M.: Leveraging multi-core computing architectures in avionics. In: Dependable Computing Conference (EDCC), 2012 Ninth European. pp. 132–143 (2012)
9. Paoloni, G.: How to benchmark code execution times on intel ia-32 and ia-64 instruction set architectures. White paper, Intel Corporation (2010)
10. Reinhardt, D., Kaule, D., Kucera, M.: Achieving a scalable e/e-architecture using autosar and virtualization. In: SAE World Congress (2013)
11. Sang, F.L., Lacombe, E., Nicomette, V., Deswarte, Y.: Exploiting an i/ommu vulnerability. In: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. pp. 7–14. IEEE (2010)
12. Sugerman, J., Venkitachalam, G., Lim, B.H.: Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In: Proceedings of the General Track: 2002 USENIX Annual Technical Conference. pp. 1–14 (2001)
13. Wojtczuk, R., Rutkowska, J.: Following the white rabbit: Software attacks against intel vt-d technology (2011)
14. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing shared resource contention in multicore processors via scheduling. In: ACM SIGARCH Computer Architecture News. vol. 38, pp. 129–142. ACM (2010)