Securing Self-Virtualizing I/O Devices

Igor Smolyar[1,2] Muli Ben-Yehuda[1] Dan Tsafrir[1]
[1]*Technion—Israel Institute of Technology*
[2]*The Open University of Israel*
{igors,muli,dan}@cs.technion.ac.il

Abstract

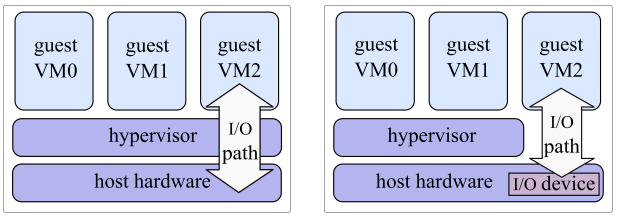
*Single root I/O virtualization (SRIOV) is a hardware/software interface that allows devices to "self-virtualize" and thereby remove the host from the critical I/O path. SRIOV thus brings bare-metal performance to untrusted guest virtual machines (VMs) in public clouds, enterprise data centers, and high-performance computing setups. We identify a design flaw in current SRIOV deployments that enables untrusted VMs to completely control the throughput and latency of other, unrelated VMs using network flow control functionality. Addressing this flaw with current network controllers (NICs) and switches requires either forgoing SRIOV or forgoing flow control, thereby trading off much of the performance benefit that SRIOV provides. We present and experimentally demonstrate the viability of the Virtualization-Aware Network Flow Controller (*VANFC*), a secure SRIOV setup that eliminates this flaw without requiring any changes to the software/hardware interface.*

1. Introduction

A key challenge when running untrusted virtual machines is providing them with efficient and secure I/O. Environments running potentially untrusted virtual machines include enterprise data centers, public cloud computing providers, and high-performance computing sites.

There are three common approaches to providing I/O services to guest virtual machines: (1) the hypervisor *emulates* a known device and the guest uses an unmodified driver to interact with it [63]; (2) a *paravirtual driver* is installed in the guest [18, 62]; (3) the host assigns a real device to the guest, which then controls the device directly [20, 48, 57, 66, 68]. When emulating a device or using a paravirtual driver, the hypervisor intercepts all interactions between the guest and the I/O device, as shown in Figure 1a, leading to increased overhead and significant performance penalty.

The hypervisor can reduce the overhead of device emulation or paravirtualization by assigning I/O devices directly to virtual machines, as shown in Figure 1b. Device assignment provides the best performance [49, 58, 68], since it minimizes the number of I/O-related world switches between the virtual machine and its hypervisor. However, assignment of standard devices is not scalable: a single host can generally run an order of magnitude more virtual machines than it has physical I/O device slots available.



(a). Traditional Virtualization (b). Direct I/O Device Assignment

Figure 1. Types of I/O Virtualization

One way to reduce I/O virtualization overhead further and improve virtual machine performance is to offload I/O processing to scalable self-virtualizing I/O devices. The PCI Special Interest Group (PCI-SIG) on I/O Virtualization proposed the Single Root I/O Virtualization (SRIOV) standard for scalable device assignment. PCI devices supporting the SRIOV standard present themselves to host software as multiple virtual interfaces. The host can assign each such partition directly to a different virtual machine. With SRIOV devices, virtual machines can achieve bare-metal performance even for the most demanding I/O-intensive workloads [34, 35]. We describe how SRIOV works and why it benefits performance in Section 2.

New technology such as SRIOV often provides new capabilities but also poses new security challenges. Because SRIOV provides untrusted virtual machines with unfettered access to the physical network, such machines can inject malicious or harmful traffic into the network. We analyze the security risks posed by using SRIOV in environments with untrusted virtual machines in Section 3. We find that SRIOV, as currently deployed, is flawed and cannot be used securely while also using network flow control functionality.

In Section 4, we show how a malicious virtual machine with access to an SRIOV device can use network flow control functionality to completely control the bandwidth and latency of other unrelated VMs using the same SRIOV device, without their knowledge or cooperation. The malicious virtual machine does this by transmitting a small number of Ethernet PAUSE or Priority Flow Control (PFC) frames every so often.

The aforementioned flaw can, however, be overcome once we understand its fundamental cause: Ethernet flow control functionality operates on the assumption that the edge switch can trust the network endpoint. With SRIOV, a single endpoint includes both the host (usually trusted) and multiple untrusted guests, all of which share the same link to the edge switch. The edge switch must either trust all the guests and the host or trust none of them. The former leads to the flow control attack we show; the latter means doing without flow control.

The attack we describe works by having a malicious guest send Ethernet PAUSE or PFC frames to the switch. If the switch honors them, it will shut down traffic (for a specified amount of time) on the link. Since the link is shared between multiple untrusted guests and the host, none of them will receive traffic.

In Section 5 we propose the Virtualization-Aware Network Flow Controller (VANFC) to overcome this flaw. By managing flows per virtual machine instead of per link, VANFC only stops traffic for the virtual machine that sent PAUSE or PFC frames. The traffic of other virtual machines and of the host that share the same link remains unaffected; thus VANFC eliminates the attack.

We evaluate a software-based prototype of VANFC in Section 6. VANFC is 100% effective in addressing the attack we describe.

VANFC has no impact on throughput compared to the baseline system not under attack but does increase latency by the latency of a single layer 2 (L2) device (~50 µs). We expect that an eventual hardware implementation will eliminate the additional latency.

One could argue that flow control at the Ethernet level is not necessary, since protocols at a higher level (e.g., TCP) have their own flow control. We show why Converged Enhanced Ethernet requires flow control in Section 7. We discuss several other problems in Section 8, followed by related work in Section 9, and our conclusions and future work in Section 10.

2. SRIOV Primer

Hardware emulation and paravirtualized devices impose a significant performance penalty on guest virtual machines [14, 15, 19, 20, 21]. Seeking to improve virtual I/O performance and scalability, PCI-SIG proposed a specification for PCIe devices with self-virtualization capabilities. Known as the SRIOV specification, it defines how host software can partition a single SRIOV PCIe device into multiple PCIe "virtual" devices.

Each SRIOV-capable physical device has at least one Physical Function (PF) and multiple virtual partitions called Virtual Functions (VFs). Each PF is a standard

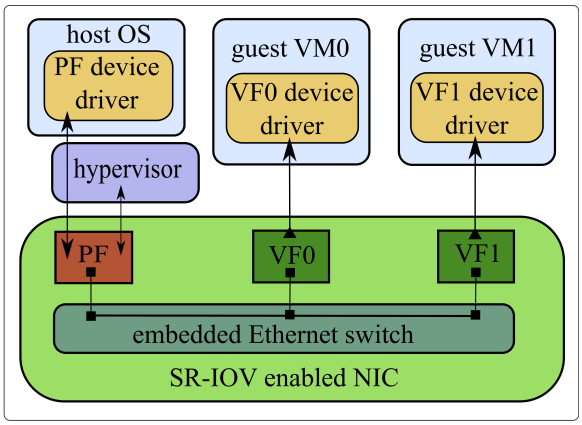


Figure 2. SRIOV NIC in a virtualized environment

PCIe function: host software can access it the same way it accesses any other PCIe device. A PF also has a full configuration space. Through the PF, host software can control the entire PCIe device as well as perform I/O operations. Each PCIe device can have up to eight independent PFs.

VFs, on the other hand, are "lightweight" (virtual) PCIe functions that implement a subset of standard PCIe device functionalities. Virtual machines driving VFs perform only I/O operations through them. For a virtual machine to use a VF, the host software must configure that VF and assign it to the virtual machine. Host software often configures a VF through its PF.VFs have a partial configuration space and are usually presented to virtual machines as PCIe devices with limited capabilities. In theory, each PF can have up to 64K VFs. Current Intel implementations of SRIOV enable up to 63 VFs per PF [39] and Mellanox ConnectX adapters usually have 126 VFs per PF [53].

While PFs provide both control plane functionality and data plane functionality, VFs provide only data plane functionality. PFs are usually controlled by device drivers that are part of the trusted computing base (TCB), i.e., reside in the privileged host operating system or hypervisor. As shown in Figure 2, in virtualized environments each VF can be directly assigned to a VM using device assignment, which allows each VM to directly access its corresponding VF, without hypervisor involvement on the I/O path.

Studies show that direct assignment of VFs provides virtual machines with nearly the same performance as direct assignment of physical devices (without SRIOV) while allowing the same level of scalability as software-based virtualization solutions such as device emulation or paravirtualization [29, 34, 37, 69]. Furthermore, two VMs that share the same network device PF can communicate

efficiently since their VM-to-VM traffic can be switched in the network adapter. Generally, SRIOV devices include embedded Ethernet switch functionality capable of efficiently routing traffic between VFs, reducing the burden on the external switch. The embedded switch in SRIOV capable devices is known as a Virtual Ethernet Bridge (VEB) [47].

SRIOV provides virtual machines with I/O performance and scalability that is nearly the same as bare metal. Without SRIOV, many use cases in cloud computing, high-performance computing and enterprise data centers would be infeasible. With SRIOV it is possible to virtualize High Performance Computing (HPC) setups [22, 33]. In fact, SRIOV is considered the key enabling technology for fully virtualized HPC clusters [50]. Cloud service providers such as Amazon Elastic Compute Cloud (EC2) use SRIOV as the underlying technology in EC2 HPC services. Their Cluster Compute-optimized virtual machines with high performance enhanced networking rely on SRIOV [2]. SRIOV is important in traditional data centers as well. Oracle, for example, created the Oracle Exalogic Elastic Cloud, an integrated hardware and software system for data centers. Oracle Exalogic uses SRIOV technology to share the internal network [36].

3. Analyzing SRIOV Security

Until recently, organizations designed and deployed Local Area Networks (LANs) with the assumption that each end-station in the LAN is connected to a dedicated port of an access switch, also known as an edge switch.

The edge switch applies the organization's security policy to this dedicated port according to the level of trust of the end-station connected to the port: some machines and the ports they connect to are trusted and some are not. But given a port and the machine connected to it, the switch enforcing security policy must know how trusted that port is.

With the introduction of virtualization technology, this assumption of a single level of trust per port no longer holds. In virtualized environments, the host, which is often a trusted entity, shares the same physical link with untrusted guest VMs. When using hardware emulation or paravirtualized devices, the trusted host can intercept and control all guest I/O requests to enforce the relevant security policy. Thus, from the point of view of the network, the host makes the port trusted again.

Hardware vendors such as Intel or Mellanox implement strict VF management or configuration access to SRIOV devices. Often they allow VFs driven by untrusted entities to perform only a limited set of management or configuration operations. In some implementations, the VF performs no such operations; instead, it sends requests to perform them to the PF, which does so after first validating them.

On the data path, the situation is markedly different. SRIOV's raison d'être is to avoid host involvement on the data path. Untrusted guests with directly assigned VFs perform data path operations—sending and receiving network frames—directly against the device. Since the device usually has a single link to the edge switch, the device aggregates all traffic, both from the trusted host and from the untrusted guests, and sends it on the single shared link. As a result, untrusted guests can send any network frames to the edge switch.

Giving untrusted guests uncontrolled access to the edge switch has two implications. First, since the edge switch uses its physical resources (CAM tables, queues, processing power) to process untrusted guests' traffic, the switch becomes vulnerable to various denial of service attacks. Second, sharing the same physical link between trusted and untrusted entities exposes the network to many Ethernet data-link layer network attacks such as Address Resolution Protocol (ARP) poisoning, Media Access Control (MAC) flooding, ARP spoofing, MAC address spoofing, and Spanning Tree Protocol (STP) attacks [13, 16, 43, 52, 65, 67]. Therefore, the edge switch must never trust ports connected to virtualized hosts with SRIOV device.

Although the problem of uncontrolled access of untrusted end-points is general to Ethernet networks, using an SRIOV devices imposes additional limitations. As we will see in the next few subsections, not trusting the port sometimes means giving up the required functionality. Organizations deploying SRIOV today must choose between SRIOV and important functionality such as Ethernet flow control.

3.1. Traditional Lossy Ethernet

Traditional Ethernet is a lossy protocol; it does not guarantee that data injected into the network will reach its destination. Data frames can be dropped for different reasons: because a frame arrived with errors or because a received frame was addressed to a different end-station. But most data frame drops happen when the receiver's buffers are full and the receiving end-station has no memory available to store incoming data frames. In the original design of the IEEE 802.3 Ethernet standard, reliability was to be provided by upper-layer protocols, usually TCP [56], with traditional Ethernet networks providing best effort service and dropping frames whenever congestion occurs.

3.2. Flow Control in Traditional Ethernet

Ethernet Flow Control (FC) was proposed to control congestion and create a lossless data link medium. FC enables a receiving node to signal a sending node to temporarily stop data transmission. According to the IEEE 802.3x standard [6], this can be accomplished by sending a special Ethernet PAUSE frame. The IEEE 802.3x PAUSE frame is defined in Annex 31B of the IEEE 802.3 specification [9] and uses the MAC frame format to carry PAUSE commands.

When a sender transmits data faster than the receiver can process it and the receiver runs out of free buffers, the receiver generates a MAC control frame and sends a PAUSE request to the sender. Upon receiving the PAUSE frame, the sender stops transmitting data. The PAUSE frame includes information on how long to pause transmission.

The `pause_time` is a two byte MAC Control parameter in the PAUSE frame that is measured in units of `pause_quanta`. It can be between 0 to 65535 `pause_quanta`. The `pause_time` tells the sending node how long to pause. The receiver can also tell the sender to resume transmission by sending a special PAUSE frame with the `pause_time` value set to 0.

Each `pause_quanta` equals 512 "bit times," defined as the time required to eject one bit from the NIC. One bit time is 1 divided by the NIC speed. The maximal PAUSE frame `pause_time` value can be 65535 `pause_quanta`, which is $65535 \times 512 = 33553920$ bit times.

For 1 Gbps networks, one PAUSE frame with `pause_time` value of 65535 `pause_quanta` will tell the sender to stop transmitting for 33553920 bit times, i.e., 33.55392 ms. A sender operating at 10 Gbps speed will pause for 3.355392 ms. A sender operating at 40 Gbps speed will pause for 0.838848 ms.

As shown in Table 1, sending such a PAUSE frame at a rate of 30 frames/second will tell the sender to completely stop transmission on a 1Gbps link. For a sender operating at 10 Gbps speed to stop transmission requires sending 299 frames/second. For a sender operating at 40 Gbps speed to stop transmission requires sending 1193 frames/second.

3.3. Priority Flow Control in Converged Ethernet

To improve the performance and reliability of Ethernet and make it more suitable for data centers, the IEEE 802.1 working group proposed a new set of standards. These new Ethernet standards are known as Data Center Bridging (DCB) or Converged Enhanced Ethernet (CEE).

link speed, Gbps	single frame pause time, ms	frame rate required to stop transmission, frames/second
1	33.554	30
10	3.355	299
40	0.849	1193

Table 1. The rate at which a network device should receive PAUSE frames in order to stop transmission completely. The `pause_time` value of each frame is 0xFFFF.

In addition to IEEE 802.3x Ethernet PAUSE, the new standard proposed to make Ethernet truly "lossless" in data center environments by adding Priority-based Flow Control (PFC), standardized in IEEE standard 802.1Qbb [8].

Similar to the 802.3x FC, PFC is a link-level flow control mechanism, but it is implemented on a per-flow basis. While 802.3x FC pauses all traffic on the link, PFC allows you to pause specific flows of traffic using the same PAUSE frame structure. PFC operates on individual flows or traffic classes, as defined by Annex I of IEEE 802.1Q standard [7]. Up to 8 traffic classes can be defined for PFC per link.

3.4. Attacking VMs via Flow Control

Direct device assignment enables malicious guests to attack the Ethernet network via well-known Layer 2 attacks [13, 16, 43, 52, 65, 67]. Even when using virtualization-aware switching extensions such as the Virtual Edge Port Aggregator (VEPA) [26,27] (also discussed in Section 8), all guests with direct access to the VFs of the same PF still share the same physical link to the edge switch, and the edge switch still allocates processing resources per link.

For example, both 802.3x and 802.1Qbb perform flow control on a link-level basis, the same link that is shared between VMs. That is, any flow control manipulation performed by a single VM will affect the PF and all VFs associated with this PF. This means that a malicious VM is capable of controlling the bandwidth and latency of all VMs that share the same adapter.

The malicious VM can pause all traffic on the link by sending 802.3x PAUSE frames and can stop specific flows by sending 802.1Qbb PAUSE frames. To stop all traffic on a 10 Gbps Ethernet link, an attacker needs to transmit PAUSE frames at a rate of 300 frames/second, which is about 155 Kbps of bandwidth. The attacker can fully control the bandwidth and latency of all tenant VMs with minimal required resources and without any cooperation from the host or from other guest VMs.

4. Attack Evaluation

4.1. Experimental Setup

We constructed a lab setup in which we perform and evaluate the flow-control attack described in the previous section. We use a Dell PowerEdge R420 server, which is a dual socket with six cores per socket, with Intel Xeon E5-2420 CPUs running at 1.90GHz. The chipset is the Intel C600 series, which supports Intel virtualization technology for directed I/O (VT-d) [38]. The server includes 16GBs of memory and an SRIOV-capable Intel NIC installed in PCIe generation 3 slots with two VFs enabled.

We use the KVM Hypervisor [46] and Ubuntu server 13.10 with 3.11.0 x86_64 kernel for the host, guest VMs, and the client. Each guest is created with 2GBs of memory, two virtual CPUs, and one VF directly assigned to it. Client and host machines are identical servers connected to the same dedicated switch, as shown in Figure 3.

To achieve consistent results, the server's BIOS profile is performance optimized, all power optimizations are tuned off, and Non-Uniform Memory Access (NUMA) is enabled. The guest virtual CPUs are pinned to the cores on the same NUMA node to which the Intel PF is connected. The host allocates to the guest memory from the same NUMA node as well.

For our 1GbE environment, we use an Intel Ethernet I350-T2 network interface connected to a Dell PowerConnect 6224P 1Gb Ethernet switch. For our 10GbE environment, we use an Intel 82599 10 Gigabit TN network interface connected to an HP 5900AF 10Gb Ethernet switch.

Host and client use their distribution's default drivers with default configuration settings. Guest VMs use version 2.14.2 of the `ixgbevf` driver for the Intel 10G 82599 Ethernet controller virtual function and the default `igbvf` version 2.0.2-k for the Intel 1G I350 Ethernet controller virtual function. Ethernet flow control IEEE 802.3x is enabled on switch ports. We set the Ethernet Maximal Transfer Unit (MTU) to 1500 bytes on all Ethernet switches and network interfaces in our tests.

4.2. Benchmark Methodology

We conduct a performance evaluation according to the methodology in RFC 2544 [23]. For throughput tests, we use an Ethernet frame size of 1518 bytes and measure maximal throughput without packet loss. Each throughput test runs for at least 60 seconds and we take the average of 5 test cycles. To measure latency, we use 64 and 1024 byte messages. Each latency test runs at least 120 seconds and we measure the average of at least 15 test cycles.

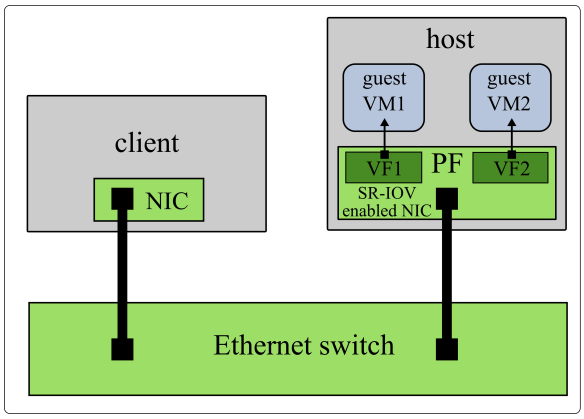


Figure 3. Setup scheme

Benchmark Tools: We measure throughput and latency with two well-known network benchmark utilities: `iperf` [3] and `netperf` [41]. The `iperf` and `netperf` clients are run on the client machine and `iperf` server and `netperf` servers are run on VM1. We measure on the client the bandwidth and latency from the client to VM1.

Traffic Generators: In addition to the traffic generated by the benchmark tools, we use `tcpdump` [40] to capture traffic and `tcpreplay` [5] to send previously captured and modified frames at the desired rate.

Testbed Scheme: The testbed scheme is shown in Figure 3. Our testbed consists of two identical servers. One server is the client and the other server is the host with SRIOV capable NIC. We configure two VFs on the host's SRIOV PF. We assign VF1 to the guest VM1 and VF2 to the guest VM2. Client and host machines are connected to the same Ethernet switch. We generate traffic between VM1 and the client using `iperf` and `netperf`. VM2 is the attacking VM.

4.3. Flow-Control Attack Implementation

We use the tcpreplay [5] utility to send specially crafted 802.3x PAUSE frames at the desired rate from the malicious VM2. We use 802.3x PAUSE frames for the sake of simplicity, but we could have used PFC frames instead. PFC uses exactly the same flow control mechanism and has the same MAC control frame format. The only difference between PFC frames and PAUSE frames is the addition of seven `pause_time` fields in PFC that are padded in 802.3x frame format.

When the switch receives a PAUSE frame from VM2, it inhibits transmission of any traffic on the link between the switch and the PF, including the traffic between the client and VM1, for a certain number of `pause_time` quanta. Sending PAUSE frames from VM2, we can ma-

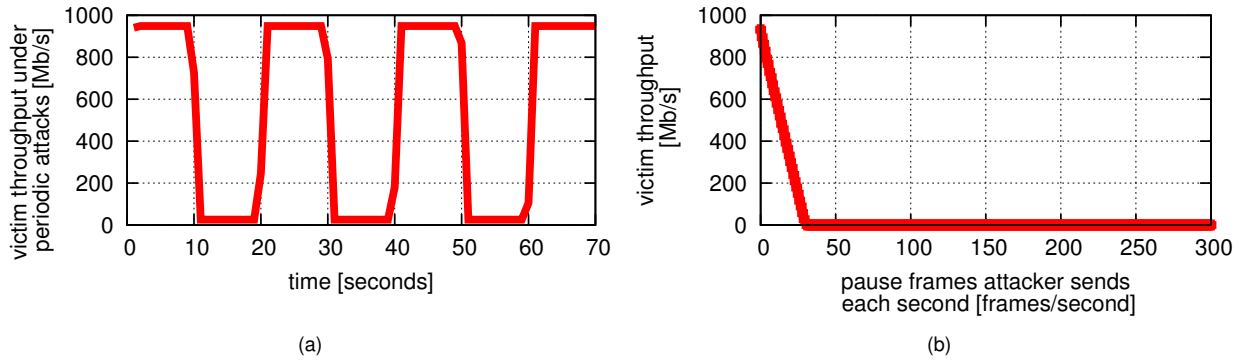


Figure 4. PAUSE frame attack: victim throughput in 1GbE environment

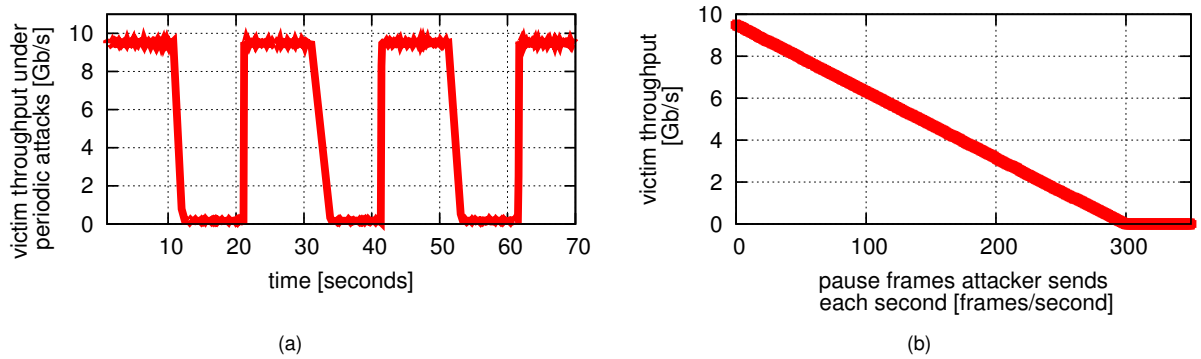


Figure 5. PAUSE frame attack: victim throughput in 10GbE environment

nipulate the bandwidth and latency of the traffic between VM1 and the client. The value of `pause_time` of each PAUSE frame is 0xFFFF `pause_quanta` units. Knowing link speed, we can calculate PAUSE frame rate, as described in Section 3, and impose precise bandwidth limits and latency delays on VM1. The results of the attack in both 1GbE and 10GbE environments are presented in Section 4.4.

4.4. Attack Results

Figures 4 and 5 show the results of the PAUSE frame attack on victim throughput in the 1GbE and 10GbE environments respectively. Figures 4a and 5a show victim (VM1) throughput under periodic attack of VM2. Every 10 seconds, VM2 transmits PAUSE frames for 10 seconds at 30 frames/second (as shown in Figure 4a) and at 300 frames/second (as shown in Figure 5a). In this test we measure the throughput of the victim system VM1. We can clearly see from the figures that VM2 can gain complete control over VM1 throughput: starting from the tenth second, the attacker completely stops traffic on the link for ten seconds.

Figure 6 shows the results of the PAUSE frame attack on victim latency in the 10GbE environment. Figure 6a shows victim latency under the same periodic attack described above. In this test we use 64B and 1024B messages. For better result visualization, we lowered the attack rate to 150 PAUSE frames/second. Figure 6a shows that the attacker can increase victim latency to 250% by running the attack at a rate of only 150 frames/second.

Victim throughput Figures 4b and 5b display throughput of VM1 as a function of the rate of PAUSE frames VM2 sends. From Figure 4b we can see that VM2 can pause all traffic on the 1GbE link with almost no effort, by sending PAUSE frames at a rate of 30 frames/second. For the 10GbE link, VM2 needs to work a little bit harder and raise its rate to 300 frames/second. This test's results confirm the calculations shown in Table 1. Figures 7a and 7b confirm that the measured victim throughput is exactly as predicted. In other words, it is easily and completely controlled by the attacker.

These tests show that a malicious VM can use the PAUSE frame attack to control the throughput of other VMs with precision. Furthermore, we see that the PAUSE frame attack requires minimal effort from the attacker and will be hard to detect amid all the other network traffic. To

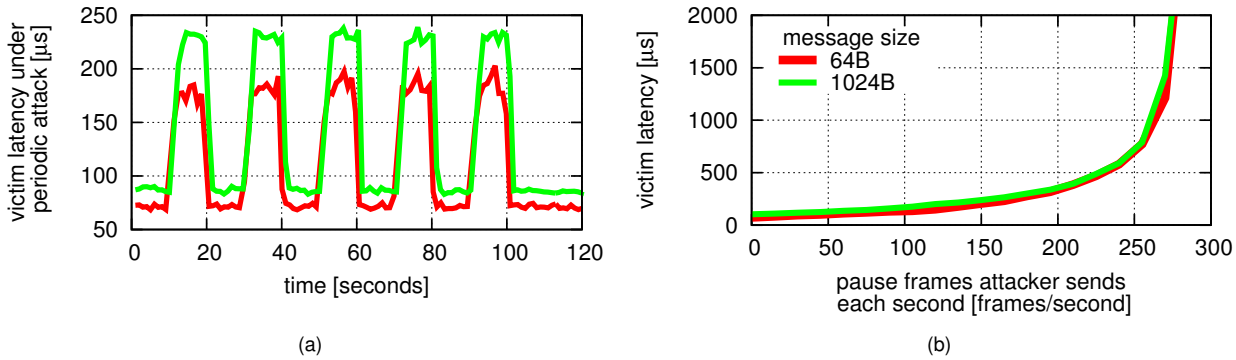


Figure 6. PAUSE frame attack: victim latency in 10GbE environment

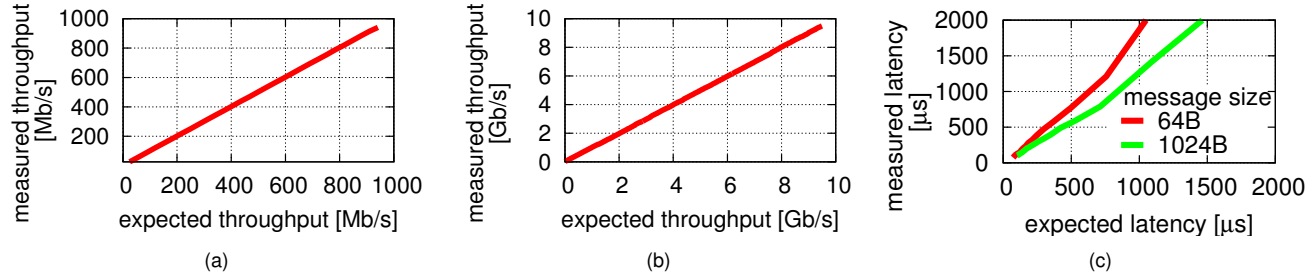


Figure 7. PAUSE frame attack: expected vs. measured throughput and latency

halt all transmissions on the 10GbE link, the attacker only needs to send 64B PAUSE frames at 300 frames/second. 300 frames/second is approximately 0.002% of the 14.88 million frames/second maximum frame rate for 10GbE. [1] Discovering such an attack can be quite challenging, due to the low frame rate involved, especially on a busy high-speed link such as 10GbE or 40GbE.

Victim latency Figure 6b shows the victim's latency as a function of the attacker's PAUSE frame rate. In this test we measure the latency of 64 byte messages and 1024 byte messages. We see that the figures for both 64B and 1024B are barely distinguishable and almost converge; the latency is the same for small and large size messages under attack.

In Figure 7c we see that measured latency and expected latency differ somewhat. We are currently investigating these results to understand why. In practice, this difference means that an attacker can control the victim's latency with slightly less precision than it can control its throughput, but it can still control both with high precision and relatively little effort.

Experiments with Non-Intel Devices We also tried the attack described above on another vendor's 40GbE SRIOV adapter. Whenever the attacking VM transmitted MAC control frames (PAUSE frames) through its VF, the adapter completely locked up and became unresponsive. It stopped generating both transmit and receive interrupts, and required manual intervention to reset it, by reloading the PF driver on the host. This lockup appears to be a firmware issue and has been communicated to the adapter vendor.

Clearly, with this adapter and this firmware issue, a malicious VM could trivially perform a straightforward denial of service attack against its peer VMs that use this adapter's VFs and against the host. But since this attack is trivial to discover, we focus instead on the stealthier PAUSE frame attack, which is much harder to discover and protect against.

5. Securing SRIOV

The attack described in the previous section is the result of a fundamental limitation of SRIOV: from the network point of view, VFs and their associated untrusted VMs are all lumped together into a single end-station. Therefore, to secure SRIOV and eliminate the attack while keeping flow control functionality, we propose to enhance Ethernet NICs and/or switch awareness of VFs of connected hosts. We propose a system in which Ethernet flows are managed per VF of the SRIOV device and not per physical link.

In such a system, either the VEB in the NIC or the

[1] The maximum frame rate equals the link speed divided by the sum of sizes of the preamble, frame length and inter-frame gap.

edge switch become aware of different VFs and implement flow control (and all related functionality) for each VF. This can be done in the NIC itself, in which case the rest of the network can remain unaware of it, or it can be done in the edge switch. If done in the edge switch, the switch needs to become aware of which VF a given Ethernet frame is coming from (using its MAC address). The switch can discover the NIC's virtualization capabilities and each of its VF's MAC addresses and network states through the virtual system interface (VSI) discovery and configuration protocol (VDP) defined in the IEEE 802.1Qbg standard [10].

We built a prototype of such a system, where Ethernet flows are managed per VF. The architecture of our Virtualization-Aware Network Flow Controller (VANFC) is shown in Figure 8. Our prototype VANFC does not extend or change the functionality of either the Ethernet switch or the SRIOV device, as would be required for a hardware-based VANFC system. Instead, we approximate such a hardware-based implementation by putting a machine running the Linux Ethernet bridge [4] between the host's unmodified SRIOV adapter and the unmodified Ethernet switch. This machine is a "bump on the wire," transparent to the host and to the switch. Using the Ethernet bridge to direct each VF's traffic to a different switch port, we approximate a hardware-based VANFC system where every VF's flows are tracked and handled separately.

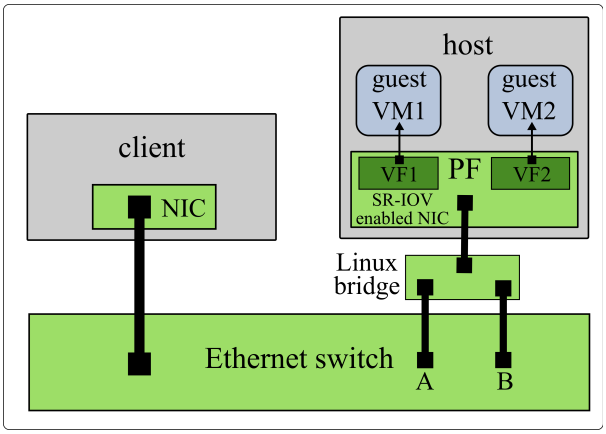


Figure 8. Virtualization-Aware Network Flow Controller

We emphasize that the eventual hardware-based VANFC system will be implemented in more optimal way; it is clear that assigning a dedicated port on a switch to each VF of an SRIOV device is neither practical nor scalable. We use this setup to demonstrate the viability of the proposed system without building a new adapter or a new switch.

The Linux bridge is configured on an x86-based commodity server running Ubuntu server 13.10 with kernel 3.11.0. We use Dell PowerEdge R610, which is a four-core single-socket server with Intel Xeon E5620 CPU running at 2.40GHz. The server includes 16GB of memory and two Intel 82599 10 Gigabit TN Network controllers installed in PCIe gen 2 slots.

Linux Bridge Configuration We configure the Linux bridge to use three 10GbE interfaces. One is connected back-to-back to the host PF and the other two are connected to ports A and B of the switch, as shown in Figure 8. We use `ebtables` [28] to configure the bridge to route VM1's traffic to port A and VM2's traffic to port B of the Ethernet switch.

The standard Ethernet bridge should not forward MAC control frames that are used to carry PAUSE commands since MAC control frames are designed to be processed by Ethernet devices. Since we want the bridge to deliver all of the traffic from VM1 and VM2, including the PAUSE frames sent by malicious VM2, we modify the Linux bridging code to forward MAC control frames and use `ebtables` to route frames to the relevant outgoing interface. We also enable flow control functionality on the switch. Our experiments use static configuration for `ebtables` and for the Linux bridge, but we could have automated the process using the VDP protocol [10].

Device Driver Modification We use a modified `ixgbe` driver version 3.21.2 for Intel 10G 82599 network controllers on the bridge machine. According to the Intel 82599 controller data-sheet [39], the flow control mechanism of the device receives PAUSE frames when flow control is enabled; when flow control is disabled the device silently drops PAUSE frames.

In our setup, we disable the flow control feature of Intel NICs of the bridge machine (using `ethtool -A ethX autoneg off rx off tx off`) and we configure the device to forward PAUSE frames up to the OS, where they should be processed by the bridge and `ebtables`. We do this by enabling the Pass MAC Control Frames (PMCF) bit of the MAC Flow Control (MFLCN) register, as described in section 3.7.7.2 of the Intel 82599 data-sheet [39].

Putting It All Together Our prototype system routes all traffic between VM1 and the client through port A on the switch. When malicious VM2 issues the attack and sends PAUSE frames, the Linux bridge forwards these frames to port B of the switch. When the switch receives a PAUSE frame on port B, it pauses the traffic transmission on that port for the requested amount of time and does

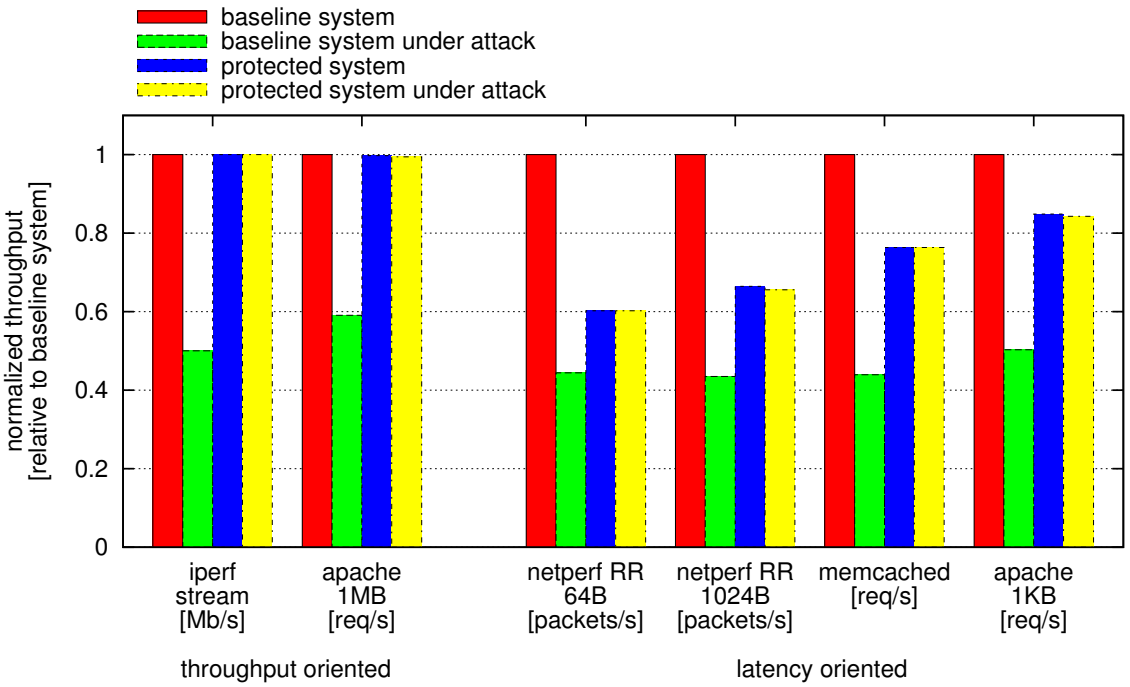


Figure 9. VANFC performance evaluation results

not pause traffic on port A. While port B inhibits transmissions, traffic between VM1 and the client continues flowing through port A, without any interruption from malicious VM2. This way, flow control in the system is handled on a per VF and not a per link basis. Each VM, through its assigned VF, has dedicated and independent flow control resources on the switch.

6. Evaluating VANFC

We evaluate VANFC in several scenarios. The **baseline** scenario includes an unprotected system, as shown in Figure 3, and no attack is performed during the test. In this scenario we measure the system's baseline throughput and latency. The **baseline system under attack** includes the same unprotected system but here VM2 runs the attack during the test, sending PAUSE frames at constant rate of 150 frames/second. In this scenario we measure the effectiveness of the attack on an unprotected system.

In the **protected system** scenario, VANFC, shown in Figure 8, replaces the unprotected system. In this scenario VM2 does not perform any attack during the test. We use this scenario to measure the performance overhead introduced by VANFC compared to the baseline. In the **protected system under attack** scenario, we also use VANFC, but here the attacker VM2 sends PAUSE frames at a constant rate of 150 frames/second. In this scenario we verify that VANFC indeed eliminates the attack.

We perform all tests on the 10GbE network with the same environment, equipment, and methodology as described in Section 4.1.

Evaluation Tests To evaluate the performance of the described scenarios, we test throughput and latency using `iperf` and `netperf`, as previously described.

In addition, we configure the `apache2` [30] web server on VM1 to serve two files, one sized 1KB and one sized 1MB. We use `apache2` version 2.4.6 installed from the Ubuntu repository with the default configuration. We run the `ab` [1] benchmark tool from the client to test the performance of the web server on VM1.

VM1 also runs `memcached` [31] server version 1.4.14, installed from the Ubuntu repository with the default configuration file. On the client we run the `memslap` [70] benchmark tool, part of the `libmemcached` client library, to measure the performance of the `memcached` server on VM1.

Figure 9 displays normalized results of the performed tests. We group test results into two categories: throughput oriented and latency oriented. Throughput oriented tests are `iperf` running pure TCP stream and `apache2` serving a 1MB file. These tests are limited by the 10GbE link bandwidth. During the tests client and server CPUs are almost idle.

In the throughput oriented tests we see that VANFC completely blocks VM2's attack and introduces no per-

formance penalty.

In the latency oriented tests we see that VANFC blocks the attack effectively as well. However, in our current implementation, VANFC is actually an additional L2 device (Linux bridge) and any latency test must include some additional constant latency due to the Linux bridge. This constant latency is approximately 50µs in our setup. An eventual implementation of VANFC in hardware, at either the NIC or the edge switch, will eliminate this overhead.

7. Necessity of Flow Control

One can argue that flow control is not required for proper functionality of high level protocols such as TCP. It then follows from this argument that SRIOV can be made "secure" simply by disabling flow control.

The TCP protocol does provide its own flow control mechanism. However, many studies have shown that TCP's main disadvantage is high CPU utilization [24, 32, 42, 51, 59]. Relying on TCP alone for flow control leads to increased resource utilization.

In public cloud environments, users pay for computational resources. Higher CPU utilization results in higher charges. In enterprise data centers and high-performance computing setups, resource consumption matters as well. Ultimately, someone pays for it. In clouds, especially, effective resource utilization will become increasingly more important [12].

As part of a recent effort to converge current network infrastructures, many existing protocols were implemented over Ethernet, e.g., Remote DMA over Converged Ethernet (RoCE) [17]. RoCE significantly reduces CPU utilization when compared with TCP.

A few recent studies about performance evaluation of different data transfer protocols over high speed links have been published [44, 45, 60, 64]. Kissel et al. [45] compare TCP and RoCE transfers over 40GbE links using the same application they developed for benchmarking. Using TCP, they managed to reach a speed of 22Gbps while the sender's CPU load was 100% and the receiver's CPU load was 91%. With OS-level optimizations, they managed to reach a speed of 39.5 Gbps and slightly reduce the sender's CPU load to 43%. Using the RoCE protocol, they managed to reach 39.2 Gbps while the CPU load of the receiver and sender was less than 2%! These results clearly show that RoCE significantly reduces CPU utilization and thus the overall cost of carrying out computations. It is especially important when a large amount of data is being moved between computational nodes in HPC or data center environments, where virtualization is becoming prevalent and increasing in popularity [22, 33, 50].

Studies show that RoCE cannot function properly without flow control [44, 45, 60, 64]. Figure 10, taken from Kissel et al. [45], with the authors' explicit permission, shows the performance effect of flow control on two competing data transfers using the RoCE protocol. Figure 10a shows the performance of a single RoCE data transfer while another RoCE data transfer is competing with it for bandwidth and flow control is enabled. Both transfers effectively share link bandwidth. Figure 10b shows the performance of the same RoCE data transfer when flow control is disabled. As can be seen in the figure, without flow control the RoCE data transfer suffers, achieving a fraction of the performance shown in Figure 10a.

Kissel et al. also [45] show that the same problem is relevant not only to RoCE but can be generalized to TCP as well. Thus we conclude that disabling flow control would cause less effective resource utilization and lead to higher cost for cloud customers and for any organization deploying SRIOV. Conversely, securing SRIOV against flow control attacks would make it possible for SRIOV and flow control to coexist, providing the performance benefits of both without relinquishing security.

8. Discussion

Notes on Implementation VANFC can be implemented as part of an SRIOV device already equipped with an embedded Ethernet switch or it can be implemented in the edge switch. Adding VANFC functionality to the NIC requires less manufacturing effort; it is also more convenient and cheaper to replace a single NIC on a host than to replace an edge switch. Nevertheless, in large-scale virtualization deployments, such as those of cloud providers or corporate virtual server farms, a single 10GbE Ethernet switch with high port density (for example, the 48 port HP 5900AF 10Gb Ethernet switch in our testbed) serves many host servers with SRIOV capable devices. In such scenarios, extending the Ethernet capabilities of each SRIOV device will greatly increase management complexity and introduce compatibility issues. Implementing VANFC in the edge switch will keep network infrastructure converged and device management simple. In addition, upgrade of 48 SRIOV devices connected to the 48 port switch requires considerably more resources than single switch upgrade.

VEB and VEPA Another important security aspect of SRIOV is VM-to-VM traffic. In SRIOV devices with an embedded VEB switch, VM-to-VM traffic does not leave the host network device and is not visible to the external edge switch, which enforces the security policy on the edge of the network. To make all VM traffic visible to the external switch, the VEB switch should act as a VEPA and send all VM traffic to the adjacent switch.

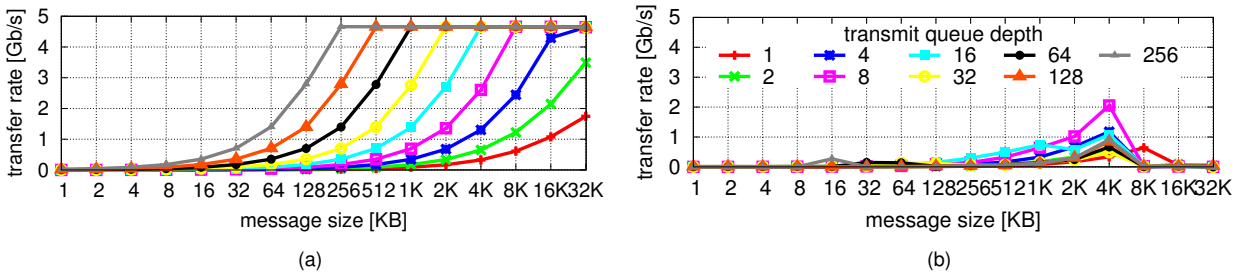


Figure 10. Performance of a single RoCE flow in the system with two competing RoCE flows. Graph (a) shows performance with enabled flow control; graph (b) shows performance with disabled flow control.

A properly configured Ethernet switch and the use of a VEPA device can enforce a security policy (ACL, port security) on malicious VM traffic and prevent most L2 attacks. However, while VEPA solves many manageability and security issues that pertain to switching in virtualized environments [25], it does not address the flow control attack we presented earlier. This is because VEPA still shares the same single link between multiple untrusted guests and the host and does not manage flow control per VF.

9. Related Work

Several recent works discussed the security of self-virtualizing devices. Pék et al. [55] described a wide range of attacks on host and tenant VMs using directly assigned devices. They performed successful attacks on PCI/PCIe configuration space, on memory mapped I/O, and by injecting interrupts. They also described an NMI injection attack. Most of the attacks they discussed can be blocked by a fix in the hypervisor or by proper hardware configuration.

Richter et al. [61] showed how a malicious VM with a directly attached VF can perform DoS attacks on other VMs that share the same PCIe link by overloading its own Memory Mapped I/O (MMIO) resources and flooding the PCIe link with write request packets. As the authors mention, this attack can be mitigated by using the QoS mechanisms defined by the PCIe standard [54].

All of the attacks discussed in the aforementioned papers are based on weak security implementations of software (e.g., a hypervisor) or hardware (a chipset system error reporting mechanism) that are internal to the host. Our attack exploits different design aspects of SRIOV devices: it targets the interoperability of SRIOV devices with software and hardware external to the host.

There are ongoing efforts of the Data Center Bridging Task Group, which is a part of the IEEE 802.1 Working Group, to standardize configuration, management and communication of virtual stations connected to the adjacent bridge. The working group proposed the 802.1Qbg Edge Virtual Bridging [10] and 802.1BR Bridge Port Extension [11] standards. Both standards concentrate on configuration and management of the bridge services for virtual stations, leaving the flow control of virtual stations out of their scope. To the best of our knowledge, our work is the first to present the problem of self-virtualizing devices in converged enhanced Ethernet environments with flow control, and the first to suggest a solution for it.

10. Conclusions and Future Work

Self-virtualizing devices with SRIOV lie at the foundation of modern enterprise data centers, cloud computing, and high-performance computing setups. We have shown that SRIOV, as currently deployed on current Ethernet networks, is incompatible with required functionality such as flow control. This is because flow control relies on the assumption that each endpoint is trusted, whereas with SRIOV, each network endpoint is comprised of multiple, possibly untrusted, virtual machines. We show how to overcome this flaw by teaching the network edge—either the NIC or the edge switch—about virtual functions. We present the prototype of such a system, VANFC, and its evaluation. Our prototype is 100% effective in securing SRIOV against this flaw while imposing no overhead on throughput-oriented workloads and the latency of a single L2 device (~50µs) on latency-oriented workloads.

Future work includes continuing to investigate the security of SRIOV devices; extending our work from Ethernet to other networking technologies such as Infiniband and Fiber Channel; and looking at the security of direct-assigned self-virtualizing devices other than NICs, such as high-end GPGPUs. On VANFC specifically, we plan to continue our evaluation and to explore what an eventual hardware-based implementation would look like, both at the NIC level and at the edge switch level.

References

[1] Apache HTTP server benchmarking tool. https://httpd.apache.org/docs/2.2/programs/ab.html. [Accessed Jul, 2014].

[2] High Performance Computing (HPC) on Amazon Elastic Compute Cloud (EC2) . Online : https://aws.amazon.com/hpc/. [Accessed Jun, 2014].

[3] Iperf - The TCP/UDP Bandwidth Measurement Tool. http://iperf.sourceforge.net. [Accessed Jul, 2014].

[4] Linux Ethernet Bridge. http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge. [Accessed Jul, 2014].

[5] Tcpreplay: Pcap editing and replay tools for Unix systems. http://tcpreplay.synfin.net/. [Accessed Jul, 2014].

[6] IEEE Standards for Local and Metropolitan Area Networks: Supplements to Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Specification for 802.3 Full Duplex Operation and Physical Layer Specification for 100 Mb/s Operation on Two Pairs of Category 3 Or Better Balanced Twisted Pair Cable (100BASE-T2). *IEEE Std 802.3x-1997 and IEEE Std 802.3y-1997 (Supplement to ISO/IEC 8802-3: 1996; ANSI/IEEE Std 802.3, 1996 Edition)*, pages 1–324, 1997.

[7] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pages 1–1365, Aug 2011.

[8] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 17: Priority-based Flow Control. *IEEE Std 802.1Qbb-2011 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011 and IEEE Std 802.1Qbc-2011)*, pages 1–40, Sept 2011.

[9] IEEE Standard for Ethernet - Section 2. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pages 752–762, Dec 2012.

[10] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 21: Edge Virtual Bridging. *IEEE Std 802.1Qbg-2012 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, IEEE Std 802.1Qbf-2011, and IEEE Std 802.aq-2012)*, pages 1–191, July 2012.

[11] IEEE Standard for Local and metropolitan area networks–Virtual Bridged Local Area Networks–Bridge Port Extension. *IEEE Std 802.1BR-2012*, pages 1–135, July 2012.

[12] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The Rise of RaaS: The Resource-as-a-service Cloud. *Commun. ACM*, 57(7):76–84, July 2014.

[13] Hayriye Altunbasak, Sven Krasser, HenryL. Owen, Jochen Grimminger, Hans-Peter Huth, and Joachim Sokol. Securing Layer 2 in Local Area Networks. In *Networking - ICN 2005*, volume 3421 of *Lecture Notes in Computer Science*, page 699–706. Springer Berlin Heidelberg, 2005.

[14] Nadav Amit, Muli Ben-Yehuda, Dan Tsafrir, and Assaf Schuster. vIOMMU: efficient IOMMU emulation. In *USENIX Annual Technical Conference (ATC)*, 2011.

[15] Nadav Amit, Muli Ben-Yehuda, and Ben-Ami Yassour. IOMMU: Strategies for Mitigating the IOTLB Bottleneck. In *Workshop on Interaction between Operating Systems & Computer Architecture (WIOSCA)*, 2010.

[16] Oleg K Artemjev and Vladislav V Myasnyankin. Fun with the Spanning Tree Protocol. *Phrack*, 11:61, 2003.

[17] InfiniBand Trade Association. InfiniBand Architecture Specification Release 1.2. 1 Annex A16: RoCE. *InfiniBand Trade Association*, 2010.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. volume 37, pages 164–177. ACM, 2003.

[19] Muli Ben-Yehuda, Eran Borovik, Michael Factor, Eran Rom, Avishay Traeger, and Ben-Ami Yassour. Adding Advanced Storage Controller Functionality via Low-Overhead Virtualization. In *USENIX Conference on File & Storage Technologies (FAST)*, 2012.

[20] Muli Ben-Yehuda, Jon Mason, Orran Krieger, Jimi Xenidis, Leendert Van Doorn, Asit Mallick, Jun Nakajima, and Elsie Wahlig. Utilizing IOMMUs for Virtualization in Linux and Xen. In *Ottawa Linux Symposium (OLS)*, pages 71–86, 2006.

[21] Muli Ben-Yehuda, Jimi Xenidis, Michal Ostrowski, Karl Rister, Alexis Bruemmer, and Leendert van Doorn. The Price of Safety: Evaluating IOMMU Performance. In *Ottawa Linux Symposium (OLS)*, pages 9–20, 2007.

[22] Georg Birkenheuer, André Brinkmann, Jürgen Kaiser, Axel Keller, Matthias Keller, Christoph Kleineweber, Christoph Konersmann, Oliver Niehörster, Thorsten Schäfer, Jens Simon, and Maximilian Wilhelm. Virtualized HPC: a contradiction in terms? *Software: Practice and Experience*, 42(4):485–500, 2012.

[23] S. Bradner and J. McQuaid. Benchmarking methodology for network interconnect devices. RFC 2544, Internet Engineering Task Force, March 1999.

[24] David D Clark, Van Jacobson, John Romkey, and Howard Salwen. An analysis of TCP processing overhead. *Communications Magazine, IEEE*, 27(6):23–29, June 1989.

[25] Paul Congdon. Enabling Truly Converged Infrastructure. http://sysrun.haifa.il.ibm.com/hrl/wiov2010/talks/100313-WIOV-Congdon-dist.pdf, 2010.

[26] Paul Congdon, Anna Fischer, and Prasant Mohapatra. A Case for VEPA: Virtual Ethernet Port Aggregator. In *Proc. 2nd Workshop on Data Center—Converged and Virtual Ethernet Switching (DC CAVES 2010), Amsterdam*, 2010.

[27] Paul Congdon and Chuck Hudson. Modularization of Edge Virtual Bridging–proposal to move forward. http://www.ieee802.org/1/files/public/docs2009/new-evb-congdon-vepa-modular-0709-v01.pdf, 2009.

[28] Bart de Schuymer and Nick Fedchik. Ebtables/Iptables Interaction On A Linux-Based Bridge. http://ebtables.sourceforge.net, 2003. [Accessed Jul, 2014].

[29] Yaozu Dong, Xiaowei Yang, Xiaoyong Li, Jianhui Li, Kun Tian, and Haibing Guan. High performance network virtualization with SR-IOV. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2010.

[30] Roy T. Fielding and Gail Kaiser. The Apache HTTP Server Project. *IEEE Internet Computing*, 1(4):88–90, 1997.

[31] Brad Fitzpatrick. Distributed Caching with Memcached. *Linux Journal*, (124), 2004.

[32] Annie P Foong, Thomas R Huff, Herbert H Hum, Jaidev P Patwardhan, and Greg J Regnier. TCP performance Re-visited. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 70–79, March 2003.

[33] Ada Gavrilovska, Sanjay Kumar, Himanshu Raj, Karsten Schwan, Vishakha Gupta, Ripal Nathuji, Radhika Niranjan, Adit Ranadive, and Purav Saraiya. High-Performance Hypervisor Architectures: Virtualization in HPC Systems. In *Workshop on System-level Virtualization for HPC (HPCVirt)*, 2007.

[34] Abel Gordon, Nadav Amit, Nadav Har'El, Muli Ben-Yehuda, Alex Landau, Assaf Schuster, and Dan Tsafrir. ELI: bare-metal performance for I/O virtualization. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12. ACM, 2012.

[35] Nadav Har'El, Abel Gordon, Alex Landau, Muli Ben-Yehuda, Avishay Traeger, and Razya Ladelsky. Efficient and scalable paravirtual i/o system. In *USENIX Annual Technical Conference (ATC)*, 2013.

[36] Adam Hawley and Yoav Eilat. Oracle Exalogic Elastic Cloud: Advanced I/O Virtualization Architecture for Consolidating High-Performance Workloads. *An Oracle White Paper*, 2012.

[37] Shu Huang and Ilia Baldine. Performance Evaluation of 10GE NICs with SR-IOV Support: I/O Virtualization and Network Stack Optimizations. In *Proceedings of the 16th International GI/ITG Conference on Measurement, Modelling, and Evaluation*

of Computing Systems and Dependability and Fault Tolerance, MMB'12/DFT'12, pages 197–205, Berlin, Heidelberg, 2012. Springer-Verlag.

[38] Intel Corporation. *Intel Virtualization Technology for Directed I/O, Architecture Specification*, 2013. Revision 2.2. Intel Corporation. [Accessed Sep, 2013].

[39] Intel Corporation. *Intel 82599 10 GbE Controller Datasheet*, 2014. Revision 2.9. [Accessed August 2014].

[40] Van Jacobson, Craig Leres, and Steven McCanne. Tcpdump: a powerful command-line packet analyzer. http://www.tcpdump.org. [Accessed Jul, 2014].

[41] Rick Jones. The Netperf Benchmark. http://www.netperf.org. [Accessed Jul, 2014].

[42] Jonathan Kay and Joseph Pasquale. The importance of non-data touching processing overheads in TCP/IP. *ACM SIGCOMM Computer Communication Review*, 23(4):259–268, 1993.

[43] Timo Kiravuo, Mikko Sarela, and Jukka Manner. A Survey of Ethernet LAN Security. *Communications Surveys Tutorials, IEEE*, 15(3):1477–1491, Third 2013.

[44] Ezra Kissel and Martin Swany. Evaluating High Performance Data Transfer with RDMA-based Protocols in Wide-Area Networks. In *IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, pages 802–811. IEEE, 2012.

[45] Ezra Kissel, Martin Swany, Brian Tierney, and Eric Pouyoul. Efficient Wide Area Data Transfer Protocols for 100 Gbps Networks and Beyond. In *Proceedings of the Third International Workshop on Network-Aware Data Management*, NDM '13, pages 3:1–3:10, New York, NY, USA, 2013. ACM.

[46] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: the Linux Virtual Machine Monitor. In *Ottawa Linux Symposium (OLS)*, 2007. http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf. [Accessed Apr, 2011].

[47] Mike Ko and Renato Recio. Virtual Ethernet Bridging. http://www.ieee802.org/1/files/public/docs2008/new-dcb-ko-VEB-0708.pdf, 2008.

[48] Joshua Levasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines. In *OSDI '04: 6th conference on Symposium on Operating Systems Design & Implementation*, page 2, 2004.

[49] Jiuxing Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In *IPDPS '10: IEEE International Parallel and Distributed Processing Symposium*, 2010.

[50] Glenn Lockwood. SR-IOV: The Key to Fully Virtualized HPC Clusters. Online : http://insidehpc.com/2013/12/30/sr-iov-key-enabling-technology-fully-virtualized-hpc-clusters/. Presented on SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. [Accessed Jun, 2014].

[51] Evangelos P Markatos. Speeding up TCP/IP: faster processors are not enough. In *21st IEEE International Conference on Performance, Computing, and Communications*, pages 341–345, 2002.

[52] Guillermo Mario Marro. Attacks at the Data Link Layer. Master's thesis, University of California, Davis, 2003.

[53] Mellanox Technologies. *Mellanox OFED for Linux User Manual*, 2014. Revision 2.2-1.0.1. [Accessed July 2014].

[54] PCI SIG. PCI Express Base Specification, Revision 3.0, 2010.

[55] Gábor Pék, Andrea Lanzi, Abhinav Srivastava, Davide Balzarotti, Aurélien Francillon, and Christoph Neumann. On the Feasibility of Software Attacks on Commodity Virtual Machine Monitors via Direct Device Assignment. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, pages 305–316. ACM, 2014.

[56] J. B. Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[57] Himanshu Raj and Karsten Schwan. High performance and scalable I/O virtualization via self-virtualized devices. In *HPDC '07: Proceedings of the 16th International Symposium on High Performance Distributed Computing*, pages 179–188, 2007.

[58] Kaushik K. Ram, Jose R. Santos, Yoshio Turner, Alan L. Cox, and Scott Rixner. Achieving 10Gbps using Safe and Transparent Network Interface Virtualization. In *ACM/USENIX International Conference on Virtual Execution Environments (VEE)*, 2009.

[59] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A Foong. TCP onloading for data center servers. *Computer*, 37(11):48–58, Nov 2004.

[60] Yufei Ren, Tan Li, Dantong Yu, Shudong Jin, T. Robertazzi, B.L. Tierney, and E. Pouyoul. Protocols for Wide-Area Data-Intensive Applications: Design and Performance Issues. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, Nov 2012.

[61] Andre Richter, Christian Herber, Holm Rauchfuss, Thomas Wild, and Andreas Herkersdorf. Performance Isolation Exposure in Virtualized Platforms with PCI Passthrough I/O Sharing. In *Architecture of Computing Systems (ARCS)*, pages 171–182. Springer International Publishing, 2014.

[62] Rusty Russell. virtio: towards a de-facto standard for virtual I/O devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, 2008.

[63] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing I/O Devices on Vmware Workstation's Hosted Virtual Machine Monitor. In *USENIX Annual Technical Conference (ATC)*, pages 1–14, 2001.

[64] Brian Tierney, Ezra Kissel, Martin Swany, and Eric Pouyoul. Efficient Data Transfer Protocols for Big Data. In *IEEE 8th International Conference on E-Science*, volume 0, pages 1–9, Los Alamitos, CA, USA, 2012. IEEE Computer Society.

[65] Luis A. Trejo, Raúl Monroy, and Rafael López Monsalvo. Spanning Tree Protocol and Ethernet PAUSE Frames DDoS Attacks: Their Efficient Mitigation. Technical report, Instituto Tecnológico de Estudios Superiores de Monterrey, ITESM-CEM, 2006.

[66] Paul Willmann, Jeffrey Shafer, David Carr, Aravind Menon, Scott Rixner, Alan L. Cox, and Willy Zwaenepoel. Concurrent Direct Network Access for Virtual Machine Monitors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2007.

[67] Angus Wong and Alan Yeung. Network Infrastructure Security. In *Network Infrastructure Security*, page 19–58. Springer US, 2009.

[68] Ben-Ami Yassour, Muli Ben-Yehuda, and Orit Wasserman. Direct device assignment for untrusted fully-virtualized virtual machines. Technical report, IBM Research Report H-0263, 2008.

[69] Ben-Ami Yassour, Muli Ben-Yehuda, and Orit Wasserman. On the DMA mapping problem in direct device assignment. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 18:1–18:12. ACM, 2010.

[70] Mingqiang Zhuang and Brian Aker. Memslap: Load Testing and Benchmarking Tool for memcached. http://docs.libmemcached.org/bin/bin/memslap.html. [Accessed Jul, 2014].