Securing Self-Virtualizing Ethernet Devices

Igor Smolyar Muli Ben-Yehuda Dan Tsafrir
Technion – Israel Institute of Technology
{igors,muli,dan}@cs.technion.ac.il

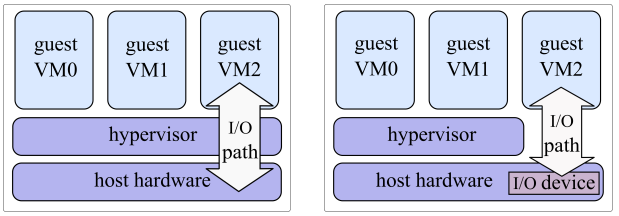
Abstract

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 near 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 Ethernet SRIOV NIC deployments that enables untrusted VMs to completely control the throughput and latency of other, unrelated VMs. The attack exploits Ethernet "pause" frames, which enable network flow control functionality. We experimentally launch the attack across several NIC models and find that it is effective and highly accurate, with substantial consequences if left unmitigated: (1) to be safe, NIC vendors will have to modify their NICs so as to filter pause frames originating from SRIOV instances; (2) in the meantime, administrators will have to either trust their VMs, or configure their switches to ignore pause frames, thus relinquishing flow control, which might severely degrade networking performance. We present the Virtualization-Aware Network Flow Controller (VANFC), a software-based SRIOV NIC prototype that overcomes the attack. VANFC filters pause frames from malicious virtual machines without any loss of performance, while keeping SRIOV and Ethernet flow control hardware/software interfaces intact.

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 [71]; (2) a *paravirtual driver* is installed in the guest [20, 69]; (3) the host assigns a real device to the guest, which then controls the device directly [22, 52, 64, 74, 76]. 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.



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

Figure 1: Types of I/O Virtualization

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 [38,53,65,76], 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.

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 [60]. 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 perfor-

mance even for the most demanding I/O-intensive workloads [38, 39]. We describe how SRIOV works and why it improves 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 NIC, as currently deployed, suffers from a major design flaw and cannot be used securely together with network flow control.

We make two contributions in this paper. The **first contribution** is to show how a malicious virtual machine with access to an SRIOV device can use the Ethernet flow control functionality to attack and 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 on its host's link to the edge switch. If Ethernet flow control is enabled, the switch will then shut down traffic on the link for a specified amount of time. Since the link is shared between multiple untrusted guests and the host, none of them will receive traffic. The details of this attack are discussed in Section 4. We highlight and experimentally evaluate the most notable ramifications of this attack in Section 5.

Our **second contribution** is to provide an understanding of the fundamental cause of the design flaw leading to this attack and to show how to overcome it. We present and evaluate (in Section 6 and Section 7) the Virtualization-Aware Network Flow Controller (VANFC), a software-based prototype of an SRIOV NIC that successfully overcomes the described attack without any loss in performance.

With SRIOV, a single physical 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 and, consequently, giving up on the performance and efficient resource utilization flow control provides.

With SRIOV NICs modeled after VANFC, cloud users could take full advantage of lossless Ethernet in SRIOV device assignment setups without compromising their security. By filtering pause frames generated by the malicious virtual machine, VANFC keeps these frames from reaching the edge switch. The traffic of virtual machines and host that share the same link remains unaffected; thus VANFC is 100% effective in eliminating the attack. VANFC has no impact on throughput or latency compared to the baseline system not under attack.

VANFC is fully backward compatible with the current hardware/software SRIOV interface and with the Ethernet flow control protocol, with all of its pros and cons. Controlling Ethernet flow by pausing physical links has its fundamental problems, such as link congestion propagation, also known as the "congestion spreading" phenomenon [13]. The attack might also be prevented by completely redesigning the Ethernet flow control mechanism, making it end-to-end credit-based, as in InfiniBand [18], for example. But such a pervasive approach is not practical to deploy and remains outside the scope of this work. Instead, VANFC specifically targets the design flaw in SRIOV NICs that enables the attack. VANFC prevents the attack without any loss of performance and without requiring any changes to either Ethernet flow control or to the SRIOV hardware/software interfaces.

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 flow control is required for high performance setups, such as those using Converged Enhanced Ethernet, in Section 8.

In Section 9 we provide some notes on the VANFC implementation and on several aspects of VM-to-VM traffic security. We present related work in Section 10. We offer concluding remarks on SRIOV security as well as remaining future work in Section 11.

2 SRIOV Primer

Hardware emulation and paravirtualized devices impose a significant performance penalty on guest virtual machines [15, 16, 21, 22, 23]. Seeking to improve virtual I/O performance and scalability, PCI-SIG proposed the SRIOV specification for PCIe devices with self-virtualization capabilities. The SRIOV spec 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 PCIe function: host software can access it as it would 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)

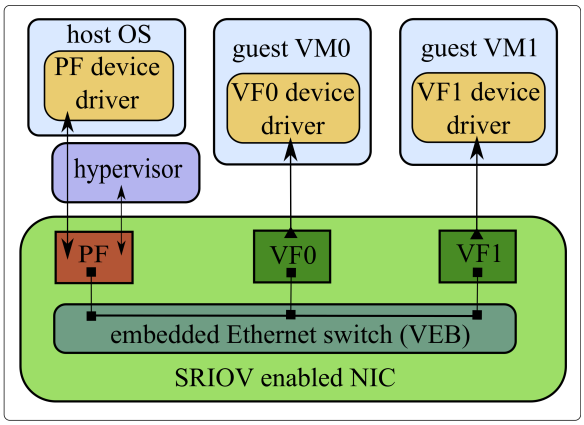


Figure 2: SRIOV NIC in a virtualized environment

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 [42] and Mellanox ConnectX adapters usually have 126 VFs per PF [57].

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 reside in the trusted, 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 [33, 38, 41, 77]. 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) [51].

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 (HPC) and enterprise data centers would be infeasible. With SRIOV it is possible to virtualize HPC setups [24, 37]. In fact, SRIOV is considered the key enabling technology for fully virtualized HPC clusters [54]. 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 [40].

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 [14, 17, 47, 56, 73, 75]. Therefore, the edge switch must never trust ports connected to virtualized hosts with an SRIOV device.

Although the problem of uncontrolled access of untrusted end-points is general to Ethernet networks, using 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.

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 [63], 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 spec [9] and uses the MAC frame format to carry pause commands.

link speed, Gbps	single frame pause time, ms	frame rate required to stop transmission, frames/second
1	33.6	30
10	3.36	299
40	0.85	1193

Table 1: Pause frame rate for stopping traffic completely

When a sender transmits data faster than the receiver can process it and the receiver runs out of space, the receiver sends the sender a MAC control frame with a pause request. Upon receiving the pause frame, the sender stops transmitting data.

The pause frame includes information on how long to halt the 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 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 (i.e., 1 divided by the NIC speed). The maximal pause frame `pause_time` value can be 65535 `pause_quanta`, which is $65535 \times 512 = 33.6$ million bit times.

For 1Gbps networks, one pause frame with `pause_time` value of 65535 `pause_quanta` will tell the sender to stop transmitting for 33.6 million bit times, i.e., 33.6 ms. A sender operating at 10 Gbps speed will pause for 3.36 ms. A sender operating at 40 Gbps speed will pause for 0.85 ms.

Table 1 shows the rate at which a network device should receive pause frames to stop transmission completely. The `pause_time` value of each frame is 0xFFFF. Sending 30 pause frames per 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 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, known as Data Center Bridging (DCB) or Converged En-

hanced Ethernet (CEE).

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

Similar to the 802.3x FC, PFC is a link-level flow control mechanism, but it is implemented on a per traffic-class basis. While 802.3x FC pauses all traffic on the link, PFC makes it possible to pause a specific class of traffic using the same pause frame structure. PFC operates on individual traffic classes, as defined by Annex I of the 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 [14, 17, 47, 56, 73, 75]. Even when using virtualization-aware switching extensions such as the Virtual Edge Port Aggregator (VEPA) [30,31], 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.

Since both 802.3x and 802.1Qbb perform flow control on a link-level basis, and the link is shared between VMs, any flow control manipulation 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 a specific traffic class 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. The server includes 16GBs of memory and an SRIOV-capable Intel NIC (10GbE 82599 or 1GbE I350) installed in PCIe generation 3 slots with two VFs enabled.

We use the KVM Hypervisor [50] 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 [25]. 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. (While RFC 2544 dictates running 20 cycles, we obtained plausible results after 15 cycles; thus, we decided to reduce test runtime by running each test only 15 cycles.)

Benchmark Tools: We measure throughput and latency with two well-known network benchmark utilities: `iperf` [3] and `netperf` [45]. We use the `iperf` TCP stream test to measure throughput and the `netperf` TCP_RR test to measure latency. The `iperf` and `netperf` clients are run on the client machine, while

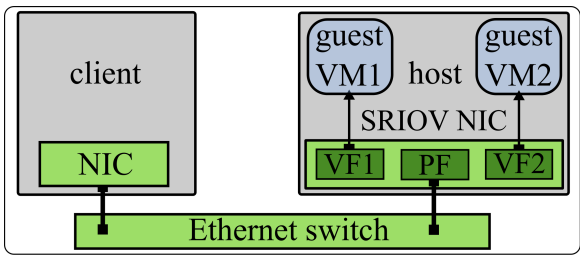


Figure 3: Setup scheme

the `iperf` 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` [44] 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 acting as client and the other as the host with SRIOV capable NIC. We configure two VFs on the host's SRIOV PF. We assign VF1 to guest VM1 and VF2 to guest VM2. Client and host 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 tcpreplay [5] to send specially crafted 802.3x pause frames at the desired rate from the malicious VM2.[1] 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 manipulate 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 the link speed, we can calculate the 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.

[1] 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 frames.

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. The figures clearly show 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.

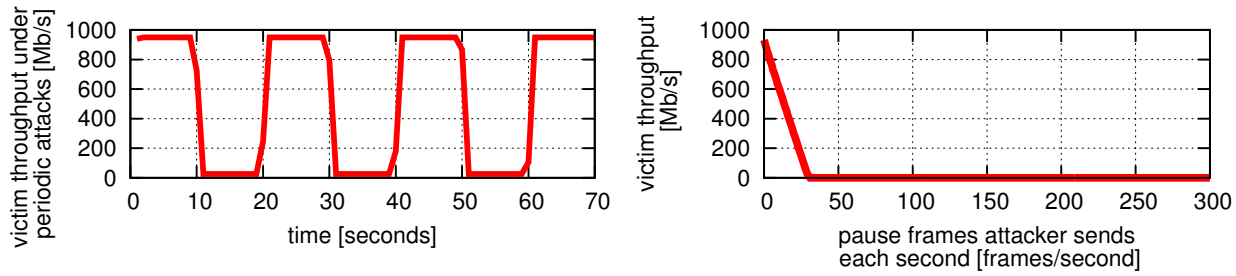
Figure 6 shows the results of the pause frame attack on victim latency in the 10GbE environment. Figure 6a shows victim latency under the same periodic attack described above. In this test we use 64B and 1024B messages. For better result visualization, we lowered the attack rate to 150 pause frames/second. 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 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.[2] 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.

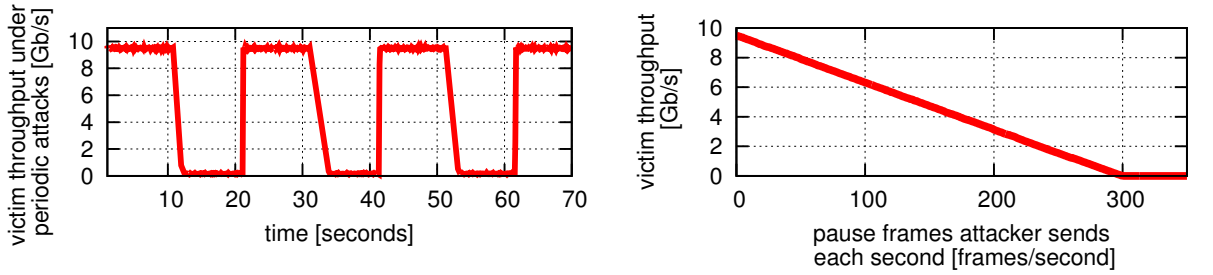
Victim latency Figure 6b shows the victim's latency as a function of the attacker's pause frame rate. In this test we measure the latency of 64 byte messages and 1024 byte messages. We see that the figures for both 64B

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



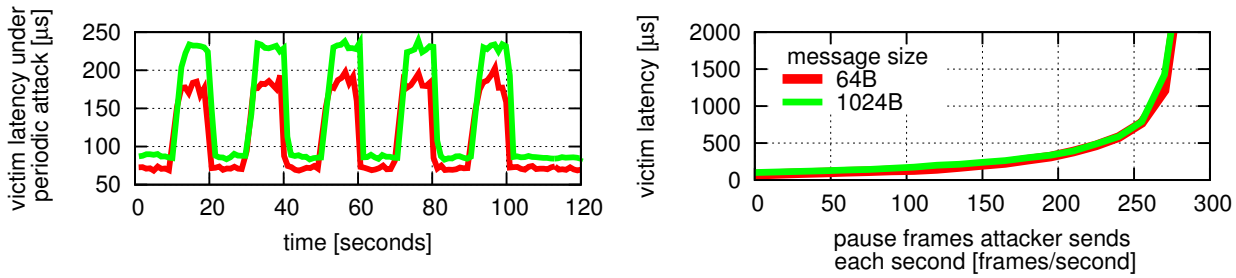
(a) (b)

Figure 4: Pause frame attack: victim throughput in 1GbE environment



(a) (b)

Figure 5: Pause frame attack: victim throughput in 10GbE environment



(a) (b)

Figure 6: Pause frame attack: victim latency in 10GbE environment

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. 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.

In back-to-back configuration, without a switch, latency behaves as expected. We believe this difference is caused by the switch's internal buffering methods—in addition to storing frames internally, the Ethernet switch prevents the possible side effects of such buffering e.g., head-of-line blocking [70] and congestion spreading [13]. To accurately explain this phenomenon, we need access to the switch internals; unfortunately, the Ethernet switch uses proprietary closed software and hardware.

Experiments with Non-Intel Devices We performed an identical experiment on same setup with an SRIOV Broadcom NetXtreme II BCM57810 10GbE NIC [26] and got the same results. Our attack is valid for this NIC as well.

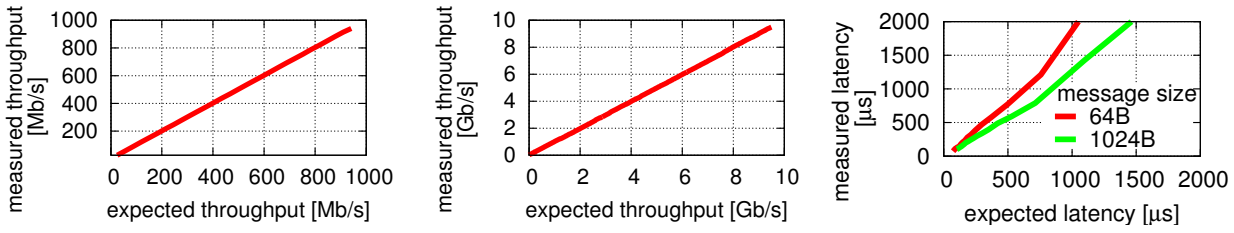


Figure 7: Pause frame attack: expected vs. measured throughput and latency

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 Attack Ramifications

The consequences of the attack are substantial. If Ethernet flow control is enabled on the SRIOV device, the host's VMs' security is compromised and the VM's are susceptible to the attack.

The attack cannot be prevented using the filtering capabilities of currently available SRIOV Ethernet devices due to their minimal filtering capability. At best, modern SRIOV NICs are capable of enforcing anti-spoofing checks based on the source MAC address or VLAN tag of the VM, to prevent one VM from pretending to be another. In the attack we describe, the adversary generates flow control frames with the malicious VM's source MAC and VLAN tag, so anti-spoofing features cannot block the attack.

Since the attack cannot be prevented with current NICs and switches, cloud providers must either be content with flawed security and fully trust the guest VMs or disable the Ethernet flow control in their networks. Neither option is palatable. The former is unrealistic for the public cloud and unlikely be acceptable to private cloud providers. The latter means giving up the performance benefits of lossless Ethernet, increasing overall resource utilization, and reducing performance. We discuss in greater detail the performance advantages that Ethernet flow control provides in Section 8.

6 Improving SRIOV Security

The attack described in the previous sections 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. To secure SRIOV and eliminate the attack while keeping flow control functionality, we propose to extend SRIOV Ethernet NIC filtering capability to filter traffic transmitted by VFs, not only on the basis of source MAC and VLAN tags—the method currently employed by anti-spoofing features—but also on the basis of the MAC destination and Ethernet type fields of the frame. This filtering cannot be done by the host without some loss of performance [39] and has to be done before traffic hits the edge switch. Hence it must be done internally in the SRIOV NIC. We built a software-based prototype of an SRIOV Ethernet NIC with pause frame filtering. Before presenting the prototype, we begin by describing the internals of an SRIOV NIC.

6.1 SRIOV NIC Internals

Figure 8a shows a detailed schema of an SRIOV Ethernet NIC. The SRIOV device is connected to the external adjacent Ethernet switch on the bottom side and to the host's PCIe bus, internal to the host, on the top side.

Switching The NIC stores frames it receives from the external switch in its internal buffer. The size of this buffer is on the order of hundreds of KBytes, depending on the NIC model: 144KB in Intel I350 [43] and 512KB in Intel 82599 [42]. After receiving a packet, the SRIOV NIC looks up the frame's MAC in its MAC address table,

finds the destination VF according to the frame's destination MAC address, and copies the frame (using DMA) over the PCIe bus to the VF's buffer ring, which is allocated in the host's RAM. This is analogous to a standard Ethernet switch that receives a packet on an ingress port, looks up its MAC address, and chooses the right egress port to send it to. The data path of the frame is marked with a red dashed line in Figure 8a. In addition, SRIOV NIC is able to perform VM-to-VM switching internally, without sending the frames to the external switch.

Internal Buffer When Ethernet flow control is enabled, the SRIOV NIC starts monitoring its internal buffer. If the NIC cannot process received frames fast enough, for example due to an overloaded or slow PCIe link, the buffer fills up. Once it reaches a predefined threshold, the SRIOV NIC generates and sends pause frames to the external Ethernet switch. The switch then holds transmissions to the NIC for the requested time, storing these frames in its own internal buffers. While the switch is buffering frames, the NIC should continue copying the frames it buffered into the each VF's ring buffer, clearing up space in its internal buffer.

Ring Buffer The final destination for a received frame is in its VF's ring buffer, located in host RAM. The network stack in the VM driving the VF removes frames from its ring buffers at a rate that is limited by the CPU. If the VM does not get enough CPU cycles or is not efficient enough, the NIC may queue frames to the ring buffer faster than the VM can process them. When the ring buffer fills up, most Ethernet NICs (e.g., those of Intel's and Mellanox's) will simply drop incoming frames. Less commonly, a few NICs, such as Broadcom's NetXtreme II BCM57810 10GbE, can monitor each VF's ring buffer. When the ring buffer is exhausted, the NIC can send pause frames to the external switch to give the host CPU a chance to catch up with the sender. When available, this functionality is usually disabled by default.

Outbound Security Some SRIOV Ethernet NICs (e.g., Intel 82599 10GbE [42] or I350 1GbE [43] NICs) include anti-spoofing functionality. They can verify that the source MAC address and/or VLAN tag of each frame transmitted by the VF belongs to the transmitting VF. To this end, these NICs have an internal component that can inspect and even change frames transmitted from the VF. In addition, Intel SRIOV NICs have advanced inbound filtering capabilities, storm control, rate limiting, and port mirroring features, very much like any standard Ethernet switch.

As we can see, Ethernet SRIOV devices implement on-board a limited form of Ethernet switching. That is why such devices are also known as virtual Ethernet bridges (VEBs).

6.2 The VANFC design

The key requirement from VANFC is to filter outbound traffic transmitted by a VF. Ideally, VANFC would be implemented in a production SRIOV NIC. Unfortunately, all tested SRIOV NICs are proprietary with closed firmware. Furthermore, most frame processing logic is implemented in high speed ASIC hardware.

We opted instead to build VANFC as a software-based prototype of an SRIOV NIC that filters outbound traffic. VANFC takes advantage of the following two observations: `(1)` VEB embedded into the SRIOV NIC device replicates standard Ethernet switching behavior and can be considered as a virtual Ethernet switch; `(2)` all valid pause frames are generated by the NIC's hardware and have the PF's source MAC address, whereas invalid—malicious—pause frames are sent with source address of a VF. Should the adversary VM attempt to generate pause frames with the PF's source MAC address, the NIC's anti-spoofing will find and drop these frames.

In order to filter transmitted malicious pause frames, we first need to identify pause frames. In such frames the Ethernet type field is `0x8808` (MAC control type), the MAC opcode field is `0x0001` (pause opcode), and the destination MAC address is multicast `01-80-C2-00-00-01`. For any such packet, the VANFC filter should drop the frame if the source MAC is different than the PF's MAC address.

As mentioned previously, most SRIOV NICs already have a component that can filter outbound traffic; this component is a part of the SRIOV device's internal Ethernet switch and cannot be modified. Our prototype extends this switch in software by running the extension on the wire between the SRIOV NIC and the external switch.

Filtering Component For our Ethernet filtering device we use the standard Linux bridge configured on an x86-based commodity server running Ubuntu server 13.10 and equipped with two Intel 82599 10 Gigabit TN Network controllers installed in PCIe gen 2 slots. One NIC is connected to the host PF and the other is connected to the external Ethernet switch, as displayed in Figure 8b. Ethernet switching is performed by the Linux bridge [4] and filtering is done by the `ebtables` [32].

Performance model Bridge hardware is fast enough not to be a bottleneck for 10Gb Ethernet speed. However, by adding to the setup an Ethernet device implemented in software, we increased latency by a constant delay of approximately 55µs. An eventual implementa-

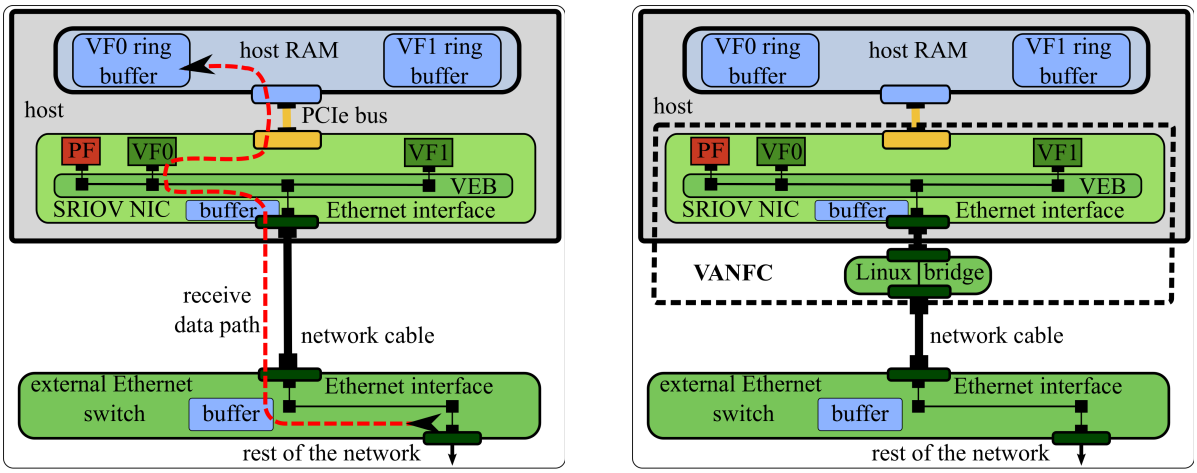


Figure 8: Fig. (a) shows schema of current SRIOV NIC internals; Fig. (b) shows VANFC schema.

tion of VANFC in hardware will eliminate this overhead; we therefore discount it in latency oriented performance tests.

We wanted to make this bridge completely transparent and not to interfere with passing traffic: the host should keep communicating with the external switch and the switch should keep communicating with the host as in the original setup without VANFC. VANFC should change neither the SRIOV software/hardware interface nor the Ethernet flow control protocol. To ensure this, we made a few modifications in Linux bridge code and in the Intel 82599 device driver used by the bridge device.

Bridge Modification 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 between the SRIOV device and the external switch, including the pause frames sent by the PF, we modify the Linux bridging code to forward MAC control frames and use `ebtables` to filter pause frames not sent from the PF. Our experiments use a static configuration for `ebtables` and for the Linux bridge.

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 [42], the flow control mechanism of the device receives pause frames when flow control is enabled; otherwise the device silently drops pause frames. In our setup, we disable the flow control feature of Intel NICs installed in the bridge machine and we configure them 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 [42].

Ring Buffer Exhaustion As mentioned, some SRIOV devices are capable of monitoring a VF's ring buffer and automatically generating pause frames when it is exhausted. In such a scenario, pause frames will be generated with the source MAC address of the PF and will not be recognized by the VANFC. We argue that such pause frame generation should be disabled in **any** SRIOV based setup, regardless of whether the VMs are trusted. Since the VM fully controls the VF's ring buffer, a malicious VM can modify its software stack (e.g., the VF device driver) to manipulate the ring buffer so that the SRIOV device monitoring the ring buffer will generate pause frames on the VM's behalf. Such pause frames will reach the external switch, which will stop its transmissions to the host and other VMs, leaving us with the same attack vector.

Automatic generation of pause frames on VF ring buffer exhaustion is problematic even if all VMs are trusted. Consider, for example, a VM that does not have enough CPU resources to process all incoming traffic and exhausts the VF's ring buffer. Sending pause frames to the switch may help this VM process the buffer but will halt the traffic to other VMs. Thus, to keep the SRIOV device secure, an SRIOV NIC should not automatically send pause frames when the VF's ring buffer is exhausted regardless of whether the VM is trusted.

Nevertheless, monitoring VF ring buffers can be use-

ful for keeping the Ethernet network lossless and avoiding dropped frames. We propose that the SRIOV device monitor ring buffers, but instead of automatically generating pause frames on ring buffer exhaustion, it should notify the hypervisor. The hypervisor, unlike the device, could then carefully consider whether the VM is malicious or simply slow. If the VM is simply slow, the hypervisor could give it a scheduling boost or assign more CPU resources to it, thereby giving it a chance to process its ring buffer before it fills up. We plan to explore this avenue in future work.

7 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 a constant rate of 150 frames/sec. In this scenario we measure the effectiveness of the attack on an unprotected system.

In the **protected system** scenario, VANFC, shown in Figure 8b, 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/sec. In this scenario we verify that VANFC indeed overcomes the attack.

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

As explained in Section 6.2, to filter malicious pause frames, our solution uses a software-based filtering device, which adds constant latency of 55µs. A production solution would filter these frames in hardware, obviating this constant latency overhead of software-based model. Thus, in latency-oriented performance tests of the VANFC, we reduced 55µs from the results.

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` [34] 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` [35] server version 1.4.14, installed from the Ubuntu repository with the default configuration file. On the client we run the `memslap` [78] 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, the client and server CPUs are almost idle.

From Figure 9 we conclude that VANFC completely blocks VM2's attack and introduces no performance penalty.

8 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 [28, 36, 46, 55, 66]. 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].

Certain traffic patterns that use the TCP protocol in high-bandwidth low-latency data center environments may suffer from catastrophic TCP throughput collapse, a phenomenon also known as the *incast* problem [58]. This problem occurs when many senders simultaneously transmit data to a single receiver, overflowing the network buffers of the Ethernet switches and the receiver, thus causing significant packet loss. Studies show that Ethernet flow control functionality, together with congestion control protocol, can mitigate the *incast* problem, thus improving the TCP performance [27, 62].

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) [19]. RoCE significantly reduces CPU utilization when compared with TCP.

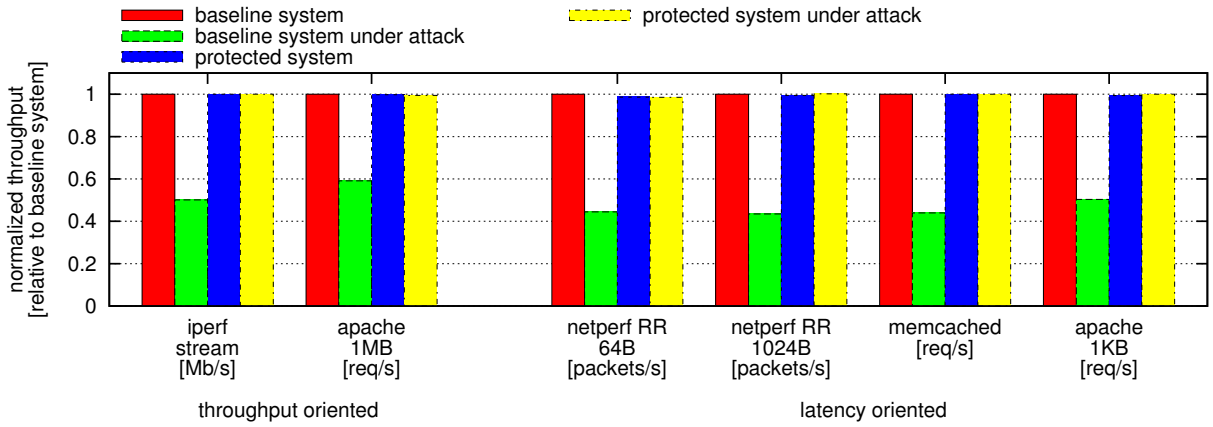
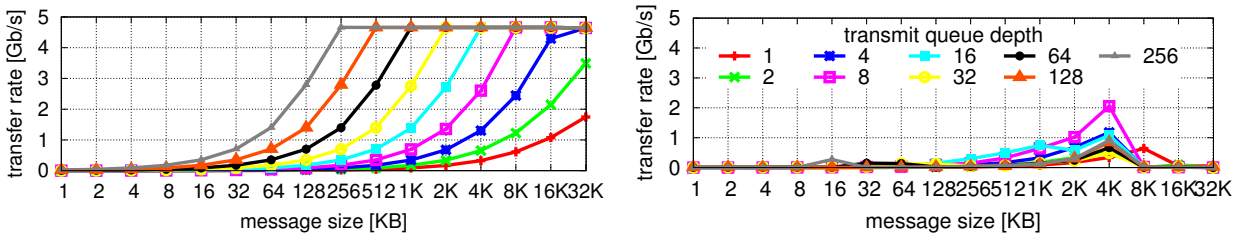


Figure 9: VANFC performance evaluation results



(a) (b)

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 few recent studies that evaluate performance of different data transfer protocols over high speed links have been published [48, 49, 67, 72]. Kissel et al. [49] 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 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 [24, 37, 54].

Studies show that RoCE cannot function properly without flow control [48, 49, 67, 72]. Figure 10, taken from Kissel et al. [49], 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. We have also independently reproduced and verified these results.

Kissel et al. also show [49] 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.

9 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 Ethernet switch, by programming the edge switch to filter flow control frames from VFs' MAC addresses. 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, upgrading 48 SRIOV devices connected to the 48 port switch requires considerably more resources than single switch upgrade.

Having said that, we argue that proper implementation of the solution to the described problem is in the SRIOV NIC and not in the edge Ethernet switch. The problem we discuss is strictly related to the virtualization platform and caused by a design flaw in the SRIOV NIC's internal switching implementation. Mitigating the problem in the edge switch, an external device whose purpose is not handle virtualization problems of the host, would force the edge switch to learn about each VF's MAC address and to distinguish PFs from VFs, coupling the edge switch too closely with the NICs.

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.

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 [29], 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. Besides not solving the flow control attack, it uses, again, the edge Ethernet switch, which is external to the source of the problem–SRIOV NIC. Thus, a VEPA extension should not be considered for the solution and the problem should be solved in the SRIOV NIC.

10 Related Work

Several recent works discussed the security of self-virtualizing devices. Pék et al. [61] 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. [68] 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 [59].

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.

11 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 NIC 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 or 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; looking at the security of direct-assigned self-virtualizing devices other than NICs, such as high-end NVMe SSDs and GPGPUs; developing VF co-residency detection techniques; and using the hypervisor to solve the problem of VM ring buffer exhaustion. Handling this with software without losing performance will be challenging. On VANFC specifically, we plan to continue our evaluation and to explore what an eventual hardware-based implementation would look like.

Acknowledgments

We would like to thank the anonymous reviewers, our shepherd, Srdjan Capkun, Shachar Raindel from Mellanox, David H. Lorenz and Ilya Lesokhin from Technion, and Sharon Kessler for insightful comments. This research was supported, in part, by the Ministry of Science and Technology, Israel, grant #3-9609. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

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)* (1997), 1–324.

[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)* (Aug 2011), 1–1365.

[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)* (Sept 2011), 1–40.

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

[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)* (July 2012), 1–191.

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

[12] AGMON BEN-YEHUDA, O., BEN-YEHUDA, M., SCHUSTER, A., AND TSAFRIR, D. The Rise of RaaS: The Resource-as-a-Service Cloud. *Communications of the ACM (CACM)* (2014).

[13] ALIZADEH, M., ATIKOGLU, B., KABBANI, A., LAKSHMIKANTHA, A., PAN, R., PRABHAKAR, B., AND SEAMAN, M. Data Center Transport Mechanisms: Congestion Control Theory and IEEE Standardization. In *46th Annual Allerton Conference on Communication, Control, and Computing* (2008), IEEE.

[14] ALTUNBASAK, H., KRASSER, S., OWEN, H., GRIMMINGER, J., HUTH, H.-P., AND SOKOL, J. Securing Layer 2 in Local Area Networks. In *4th International Conference on Networking*, Lecture Notes in Computer Science. Springer, 2005.

[15] AMIT, N., BEN-YEHUDA, M., TSAFRIR, D., AND SCHUSTER, A. vIOMMU: efficient IOMMU emulation. In *USENIX Annual Technical Conference (ATC)* (2011).

[16] AMIT, N., BEN-YEHUDA, M., AND YASSOUR, B.-A. IOMMU: Strategies for Mitigating the IOTLB Bottleneck. In *Workshop on Interaction between Operating Systems & Computer Architecture (WIOSCA)* (2010).

[17] ARTEMJEV, O. K., AND MYASNYANKIN, V. V. Fun with the Spanning Tree Protocol. *Phrack 11* (2003), 61.

[18] ASSOCIATION, I. T. InfiniBand Architecture Specification Release 1.2.1, Volume 1. *InfiniBand Trade Association* (2007).

[19] ASSOCIATION, I. T. InfiniBand Architecture Specification Release 1.2.1, Volume 1, Annex A16: RoCE. *InfiniBand Trade Association* (2010).

[20] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review* (2003).

[21] BEN-YEHUDA, M., BOROVIK, E., FACTOR, M., ROM, E., TRAEGER, A., AND YASSOUR, B.-A. Adding Advanced Storage Controller Functionality via Low-Overhead Virtualization. In *USENIX Conference on File & Storage Technologies (FAST)* (2012).

[22] BEN-YEHUDA, M., MASON, J., KRIEGER, O., XENIDIS, J., VAN DOORN, L., MALLICK, A., NAKAJIMA, J., AND WAHLIG, E. Utilizing IOMMUs for Virtualization in Linux and Xen. In *Ottawa Linux Symposium (OLS)* (2006).

[23] BEN-YEHUDA, M., XENIDIS, J., OSTROWSKI, M., RISTER, K., BRUEMMER, A., AND VAN DOORN, L. The Price of Safety: Evaluating IOMMU Performance. In *Ottawa Linux Symposium (OLS)* (2007).

[24] BIRKENHEUER, G., BRINKMANN, A., KAISER, J., KELLER, A., KELLER, M., KLEINEWEBER, C., KONERSMANN, C., NIEHRSTER, O., SCHFER, T., SIMON, J., AND WILHELM, M. Virtualized HPC: a contradiction in terms? *Software: Practice and Experience* (2012).

[25] BRADNER, S., AND MCQUAID, J. Benchmarking methodology for network interconnect devices. RFC 2544, Internet Engineering Task Force, Mar. 1999.

[26] BROADCOM CORPORATION. *Broadcom BCM57810S NetXtreme II Converged Controller*, 2010. [Accessed February 2015].

[27] CHEN, Y., GRIFFITH, R., LIU, J., KATZ, R. H., AND JOSEPH, A. D. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *1st ACM workshop on Research on Enterprise Networking* (2009), ACM.

[28] CLARK, D. D., JACOBSON, V., ROMKEY, J., AND SALWEN, H. An analysis of TCP processing overhead. *Communications Magazine, IEEE*, 6 (1989).

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

[30] CONGDON, P., FISCHER, A., AND MOHAPATRA, P. A Case for VEPA: Virtual Ethernet Port Aggregator. In *2nd Workshop on Data CenterConverged and Virtual Ethernet Switching* (2010).

[31] CONGDON, P., AND HUDSON, C. 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.

[32] DE SCHUYMER, B., AND FEDCHIK, N. Ebtables/Iptables Interaction On A Linux-Based Bridge. http://ebtables.sourceforge.net, 2003. [Accessed Jul, 2014].

[33] DONG, Y., YANG, X., LI, X., LI, J., TIAN, K., AND GUAN, H. High performance network virtualization with SR-IOV. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2010).

[34] FIELDING, R. T., AND KAISER, G. The Apache HTTP Server Project. *IEEE Internet Computing*, 4 (1997).

[35] FITZPATRICK, B. Distributed Caching with Memcached. *Linux Journal*, 124 (2004).

[36] FOONG, A. P., HUFF, T. R., HUM, H. H., PATWARDHAN, J. P., AND REGNIER, G. J. TCP Performance Re-visited. In *International Symposium on Performance Analysis of Systems and Software* (2003), IEEE.

[37] GAVRILOVSKA, A., KUMAR, S., RAJ, H., SCHWAN, K., GUPTA, V., NATHUJI, R., NIRANJAN, R., RANADIVE, A., AND SARAIYA, P. High-Performance Hypervisor Architectures: Virtualization in HPC Systems. In *Workshop on System-level Virtualization for HPC (HPCVirt)* (2007).

[38] GORDON, A., AMIT, N., HAR'EL, N., BEN-YEHUDA, M., LANDAU, A., SCHUSTER, A., AND TSAFRIR, D. ELI: bare-metal performance for I/O virtualization. In *ACM Architectural Support for Programming Languages & Operating Systems (ASPLOS)* (2012), ACM.

[39] HAR'EL, N., GORDON, A., LANDAU, A., BEN-YEHUDA, M., TRAEGER, A., AND LADELSKY, R. Efficient and Scalable Paravirtual I/O System. In *USENIX Annual Technical Conference (ATC)* (2013).

[40] HAWLEY, A., AND EILAT, Y. Oracle Exalogic Elastic Cloud: Advanced I/O Virtualization Architecture for Consolidating High-Performance Workloads. *An Oracle White Paper* (2012).

[41] HUANG, S., AND BALDINE, I. Performance Evaluation of 10GE NICs with SR-IOV Support: I/O Virtualization and Network Stack Optimizations. In *16th International Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance* (2012), Springer-Verlag.

[42] INTEL CORPORATION. *Intel 82599 10 GbE Controller Datasheet*, 2014. Revision 2.9. [Accessed August 2014].

[43] INTEL CORPORATION. *Intel I350 10 GbE Controller Datasheet*, 2014. Revision 2.2. [Accessed February 2015].

[44] JACOBSON, V., LERES, C., AND MCCANNE, S. Tcpdump: a powerful command-line packet analyzer. http://www.tcpdump.org. [Accessed Jul, 2014].

[45] JONES, R. The Netperf Benchmark. http://www.netperf.org. [Accessed Jul, 2014].

[46] KAY, J., AND PASQUALE, J. The importance of non-data touching processing overheads in TCP/IP. *ACM SIGCOMM Computer Communication Review* (1993).

[47] KIRAVUO, T., SARELA, M., AND MANNER, J. A Survey of Ethernet LAN Security. *Communications Surveys Tutorials, IEEE* (2013).

[48] KISSEL, E., AND SWANY, M. Evaluating High Performance Data Transfer with RDMA-based Protocols in Wide-Area Networks. In *14th International Conference on High Performance Computing and Communication & 9th International Conference on Embedded Software and Systems (HPCC-ICESS)* (2012), IEEE.

[49] KISSEL, E., SWANY, M., TIERNEY, B., AND POUYOUL, E. Efficient Wide Area Data Transfer Protocols for 100 Gbps Networks and Beyond. In *3rd International Workshop on Network-Aware Data Management* (2013), ACM.

[50] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. 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].

[51] KO, M., AND RECIO, R. Virtual Ethernet Bridging. http://www.ieee802.org/1/files/public/docs2008/new-dcb-ko-VEB-0708.pdf, 2008.

[52] LEVASSEUR, J., UHLIG, V., STOESS, J., AND GÖTZ, S. Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines. In *Symposium on Operating Systems Design & Implementation (OSDI)* (2004).

[53] LIU, J. Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)* (2010).

[54] LOCKWOOD, G. 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].

[55] MARKATOS, E. P. Speeding up TCP/IP: faster processors are not enough. In *21st International Conference on Performance, Computing, and Communications* (2002), IEEE.

[56] MARRO, G. M. Attacks at the Data Link Layer. Master's thesis, University of California, Davis, 2003.

[57] MELLANOX TECHNOLOGIES. *Mellanox OFED for Linux User Manual*, 2014. Revision 2.2-1.0.1. [Accessed July 2014].

[58] NAGLE, D., SERENYI, D., AND MATTHEWS, A. The Panasas Activescale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *ACM/IEEE conference on Supercomputing* (2004), IEEE.

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

[60] PCI SIG. Single Root I/O Virtualization and Sharing 1.1 Specification, 2010.

[61] PÉK, G., LANZI, A., SRIVASTAVA, A., BALZAROTTI, D., FRANCILLON, A., AND NEUMANN, C. On the Feasibility of Software Attacks on Commodity Virtual Machine Monitors via Direct Device Assignment. In *9th ACM Symposium on Information, Computer and Communications Security* (2014), ACM.

[62] PHANISHAYEE, A., KREVAT, E., VASUDEVAN, V., ANDERSEN, D. G., GANGER, G. R., GIBSON, G. A., AND SESHAN, S. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. In *USENIX Conference on File & Storage Technologies (FAST)* (2008).

[63] POSTEL, J. B. Transmission control protocol. RFC 793, Internet Engineering Task Force, Sept. 1981.

[64] RAJ, H., AND SCHWAN, K. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. In *International Symposium on High Performance Distributed Computer (HPDC)* (2007).

[65] RAM, K. K., SANTOS, J. R., TURNER, Y., COX, A. L., AND RIXNER, S. Achieving 10Gbps using Safe and Transparent Network Interface Virtualization. In *ACM/USENIX International Conference on Virtual Execution Environments (VEE)* (2009).

[66] REGNIER, G., MAKINENI, S., ILLIKKAL, R., IYER, R., MINTURN, D., HUGGAHALLI, R., NEWELL, D., CLINE, L., AND FOONG, A. TCP Onloading for Data Center Servers. *Computer Magazine, IEEE* (2004).

[67] REN, Y., LI, T., YU, D., JIN, S., ROBERTAZZI, T., TIERNEY, B., AND POUYOUL, E. Protocols for Wide-Area Data-Intensive Applications: Design and Performance Issues. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)* (2012).

[68] RICHTER, A., HERBER, C., RAUCHFUSS, H., WILD, T., AND HERKERSDORF, A. Performance Isolation Exposure in Virtualized Platforms with PCI Passthrough I/O Sharing. In *Architecture of Computing Systems (ARCS)*. Springer International Publishing, 2014.

[69] RUSSELL, R. virtio: towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review (OSR)* (2008).

[70] STEPHENS, B., COX, A. L., SINGLA, A., CARTER, J., DIXON, C., AND FELTER, W. Practical DCB for Improved Data Center Networks. In *International Conference on Computer Communications (INFOCOM)* (2014), IEEE.

[71] SUGERMAN, J., VENKITACHALAM, G., AND LIM, B.-H. Virtualizing I/O Devices on Vmware Workstation's Hosted Virtual Machine Monitor. In *USENIX Annual Technical Conference (ATC)* (2001).

[72] TIERNEY, B., KISSEL, E., SWANY, M., AND POUYOUL, E. Efficient Data Transfer Protocols for Big Data. In *8th International Conference on E-Science* (2012), IEEE Computer Society.

[73] TREJO, L. A., MONROY, R., AND MONSALVO, R. L. Spanning Tree Protocol and Ethernet PAUSE Frames DDoS Attacks: Their Efficient Mitigation. Tech. rep., Instituto Tecnolgico de Estudios Superiores de Monterrey, ITESM-CEM, 2006.

[74] WILLMANN, P., SHAFER, J., CARR, D., MENON, A., RIXNER, S., COX, A. L., AND ZWAENEPOEL, W. Concurrent Direct Network Access for Virtual Machine Monitors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2007).

[75] WONG, A., AND YEUNG, A. Network Infrastructure Security. In *Network Infrastructure Security*. Springer, 2009.

[76] YASSOUR, B.-A., BEN-YEHUDA, M., AND WASSERMAN, O. Direct Device Assignment for Untrusted Fully-Virtualized Virtual Machines. Tech. Rep. H-0263, IBM Research, 2008.

[77] YASSOUR, B.-A., BEN-YEHUDA, M., AND WASSERMAN, O. On the DMA Mapping Problem in Direct Device Assignment. In *Haifa Experimental Systems Conference (SYSTOR)* (2010), ACM.

[78] ZHUANG, M., AND AKER, B. Memslap: Load Testing and Benchmarking Tool for memcached. http://docs.libmemcached.org/bin/bin/memslap.html. [Accessed Jul, 2014].