Survey on SR-IOV performance

Maximilian Fischer, Florian Wiedner*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: maximilian.fischer@in.tum.de, wiedner@net.in.tum.de

***Abstract*—Scalable, high performance VM networking is becoming increasingly important. Paravirtualized solutions like VIRTIO are not up to the task, since the overhead in latency and bandwidth is too high. Single-Root I/O Virtualization (SR-IOV) is a technology which eliminates the need to emulate NICs and could exceed VIRTIO and similar solutions in terms of performance. In this paper we give an overview over the performance of SR-IOV with ethernet, focusing on latency since it is especially important for applications like network function virtualization. We look at the current status of SR-IOV, as well as some optimizations that can be applied and how they actually impact performance and particularly latency. We discover that latency has not been the focus of recent research, but rather bandwidth. Additionally, the scalability of stock SR-IOV and of the shown optimizations is not examined enough, especially with regard to latency. We come to the conclusion that further research is necessary.**

***Index Terms*—SR-IOV, network function virtualzation, measurement, hight-speed networks, ethernet networks**

1. Introduction

As more and more services become virtualized and containerized, high performance networking becomes increasingly important. An application where this is especially critical are virtual network functions (VNF). Here the network functions which would normally be handled by separate devices, like a firewall, router or DNS resolver are instead put into virtual machines (VMs), with usually no specialized hardware. These are applications which all other clients in the network depend on, since, depending on the VNF, either traffic goes through the VM, e.g. a router, or future traffic depends on it, e.g. a DNS resolver. Low latency and high bandwidth are a must, since VNFs can act as a network-wide bottleneck. Since multiple VNFs can run on a single host, good scalability is also paramount, otherwise a major advantage of VNFs is lost.

At the moment, VM networking is implemented with different systems, depending on the hypervisor used. When using the Kernel-based Virtual Machine (KVM), a very popular paravirtualization standard is VIRTIO. The hypervisor Xen also has capabilities to use paravitualized network interfaces [1]. Unfortunately, neither KVM nor Xen can provide performance near that of native networking. Bandwidth is severely limited by the number of interrupts the CPU can handle. Latency is impacted negatively by tx batching, a technique where the hypervisor doesn't inform the VM about every packet that arrives, but rather does so in batches, thus reducing the number of context changes. When turning off tx batching, latency benefits but bandwidth suffers due to the number of interrupts [2] [3]. A technology which could solve all of this is Single Root I/O Virtualization (SR-IOV). An SR-IOV capable network interface controller (NIC) can present itself as multiple virtual PCIe devices. These devices are split into virtual functions (VF) and one physical function (PF). The VFs are passed through to the VMs and the PF is for the host. The PF has the capability to configure the NIC, how many VFs it has, the routing and much more, depending on the NIC [2]. In theory, this greatly improves performance, since the hypervisor doesn't have to emulate or paravitualize the NIC. There are a lot of works discussing the praxis, unfortunately most of them focus on bandwidth. If latency is ever talked about it is mostly in the context of InfiniBand. That is because latency is very important for most applications using InfiniBand, like the message passing interface (MPI) [4].

The goal of this paper is to give an overview of the current state of the performance of SR-IOV networking when using ethernet, with a focus on latency. First we will talk about the current, unoptimized state of SR-IOV. Afterwards we will show some optimizations which can be applied and how they actually impact the performance. Lastly we will show what conclusions we can draw from this.

2. Current Status

There are several works analysing and discussing the performance of SR-IOV as is, without any optimizations. In this section we take a look at some of them.

A general performance overview is given by Liu [2] with a 10 GbE connection between two servers. The half-roundtrip latency is shown in Figure 1. The 7 µs difference between SR-IOV and native is attributed to the interrupt virtualization needed for SR-IOV. The very high latency of VIRTIO can be traced back in part to tx batching. In this approach the hypervisor, while writing incoming packets into the buffer of the VM, does not instantly inform the VM about them. Instead the VM is only interrupted every few packets, which leads to a dramatically lower number of interrupts. When disabling the tx batching of the VIRTIO network interface, latency improves it to around 37 µs while negatively impacting performance under high tx load due to more interrupts. Figure 2 shows the bandwidth compared to the CPU usage when receving. For large messages, the performance of SR-IOV and native is almost

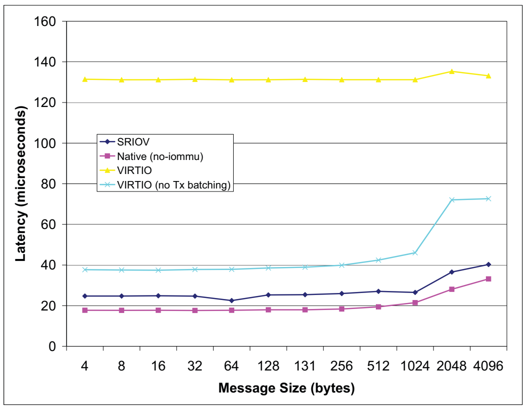

Figure 1: Latency [2]

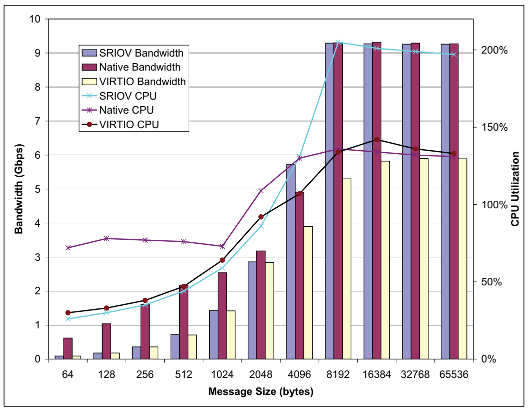

Figure 2: Rx bandwidth compared against CPU usage [2]

the same, at around 9.1 Gbit/s, while VIRTIO reaches its maximum at 5.9 Gbit/s, with a theoretical maximum of 10 Gbit/s. The CPU usage for SR-IOV, however, is significantly larger for messages bigger than 4096 B, at 200 % compared to 140 % for native or VIRTIO. This is important to note, since the CPU is often the bottleneck when using SR-IOV and also the point where a lot of papers discussed later begin their optimizations. The performance when receiving translates more or less to transmitting, except that native and SR-IOV already start to diverge for messages smaller than 8192 B. Also, the CPU usage of 110 % is closer to the VIRTIO CPU usage of 140 % than to the native of 60 %. A category in which VIRTIO pulls ahead of SR-IOV is VM exits and interrupt requests. The number of SR-IOV VM exits exceeds that of VIRTIOs 18000 by nearly eightfold [2].

Lockwood et al. [4] analyse the MPI performance of SR-IOV 10 Gbit/s ethernet on a VM compared to that of an VM with network virtualization, native 10 Gbit/s ethernet, SR-IOV InfiniBand on a VM and native InfiniBand. Since we focus on ethernet in this paper, InfiniBand is not discussed. The tests using 10 Gbit/s ethernet on a VM are carried out using Amazon Web Services and with no other VM running on the host. The benchmarks reveal that, as expected, native performs better than SR-IOV, which in turn performs better than fully virtualized networking. Latency stays about the same for message sizes of $\leq$1 KiB, with SR-IOV having about 40 % lower latency than fully virtualized networking. In numbers this means that fully virtualized networking has 65 µs of latency while SR-IOV has 40 µs. Unfortunately, this is still about two times as high as native. For larger messages the difference in ping between SR-IOV and fully virtualized starts to decrease, while the absolute latency of all three variants gets bigger. At the largest tested message size of 4 MB SR-IOV still performs 12 % better than without SR-IOV. SR-IOV also performs better when looking at latency variation, due to the fully virtualized networking being influenced by other tasks on the CPU. In numbers, that means SR-IOV has three to four times less latency variation. The bandwidth analysis shows that native provides nearly 6.4 Gbit/s for unidirectional and the full 10 Gbit/s for bidirectional traffic. The bandwidth of SR-IOV and fully virtualized never gets above about 3.2 Gbit/s for unidirectional, and 4 Gbit/s for bidirectional traffic. These differences to the previous paper are likely to the difference in benchmarks being used, as well as the usage of MPI in this paper [4].

The performance of SR-IOV is compared against that of Open vSwitch Data Plan Development Kit (OVS-DPDK) and Fast Data input/output Vector Packet Processing (FD.io VPP) for use with VNFs by Pitaev et al. [5]. The theoretical maximum interface speed is 20 Gbit/s. When the VNFs are under a light load, just doing IPv4 Forwarding, SR-IOV clearly pulls ahead. While the throughput of SR-IOV scales nearly linearly with every additional VNF added, up to about 19 Gbit/s, the throughput of FD.io VPP and OVS-DPDK stops increasing at two and three VNFs respectively, at about 12 Gbit/s to 16 Gbit/s. When using packet sizes of 128 B instead of the previous IMIX, the performance difference becomes even clearer, though the general development is the same. Loading the VNFs with NAT, Firewall, QoS and DPI and IMIX or 1500 B packet sizes yields much more interesting results. SR-IOV scales nearly linearly to the maximum bandwidth with both IMIX and 1500 B packets, while OVS-DPDK does not reach it at all and FD.io VPP only for 1500 B packets [5].

Xu and Davda [6] talk about SRVM, which provides VM live migration support for SR-IOV to the VMware ESXi hypervisor. Comparing the performance of SR-IOV to that of the VMXNET3 driver, SR-IOV performs, as expected, better. All the tests were conducted with a 10 GbE NIC. The average latency of the system normalized to the native latency is 113 % for SR-IOV and 207.7 % for VMXNET3. The minimum and maximum latency of SR-IOV is even better than native, at 95.2 % and 66.2 %, while that of VMXNET3 is 185 % and 636.7 % respectively. As expected, the throughput of SR-IOV is nearly on par with native at 99.8 % for packet sizes of 256 B, 512 B and 1024 B, whereas VMXNET3 is at 16.2 %, 33.5 % and 49.7 % [6].

Hwang et al. [7] present NetVM, a tool which competes with SR-IOV and makes use of the data plane development kit and KVM. It consists of three parts, NetVM manager, NetVM core engine and NetLib. NetVM manager is the interface to the hypervisor, receiving events from it. It then notifies NetVM core engine which actually implements those events. NetVM core engine is also responsible for receiving packets and forwards it to the VM via shared memory over an emulated PCI device. NetLib provides the interface for the user application in the VM. The test setup consists of two servers with a 10 GbE connection between them. When comparing the roundtrip latency of NetVM and SR-IOV while forwarding packets, both behave mostly the same for lower latencies, at about 40 µs to 50 µs. But the latency of SR-IOV starts to rise shortly above 5 Gbit/s of load, to 70 µs, while NetVM stays more or less the same. Although not mentioned explicitly, this is probably due to the limitations of SR-IOV seen in other papers discussed previously. Since here SR-IOV is not optimized in any way, it presumably starts to reach the limits of what the CPU can handle interrupt-wise. Unfortunately CPU load is not measured here [7].

Bauer et al. [8] compare the performance of SR-IOV software function chaining (SFC) using a single PF to that of SFC using VFs. When just comparing the performance of the ixgbe and the ixgbevf drivers, latency over an Open vSwitch OvS bridge is also examined. For bandwidths $\leq$500 Mbit/s both are around 10^4 µs. Above that, ixgbe is at about 0.8×10^7 µs, while ixgbevf is at 0.8×10^6 µs. This is presumably due to the tasks which would normally be performed by the OS now being performed by the NIC directly, which is especially beneficial for SFCs, since the forwarding between two SFs can be offloaded onto the NIC. When looking at interrupts, VF and PF mostly behave the same for loads below 700 Mbit/s. Above that, the number of interrupts for PF decreases drastically, from 10^4 Interrupts/s to 10^2 Interrupts/s, while the interrupt numbers for VF stay mostly the same at 10^4 Interrupts/s. As already seen previously, this is due to the lack of a number of features concerning the controlling of interrupts, like dynamic interrupt throttle rate. When comparing the scalability of SCFs using SR-IOV to that of SFCs using virtual ethernet (vEth) interfaces, the results show that using vEth yields higher throughput for the tested chain lengths from 1 to 5 SFs. This is explained with VFs having less efficient drivers and that there are more I/O operations required. Examining the throughput chains of length one, two and four with varying load, vEth performs better initially. For chains with length two and four, SR-IOV still takes back the lead after a certain threshold. Due to the software nature of vEth, as soon as all resources are taken up, the whole system suffers. Since with SR-IOV, a big part of the processing is offloaded onto the NIC, this is not a problem [8].

3. Optimizations

There are several different optimization options. A lot of them focus on decreasing the CPU load on high throughput, though there are of course others. In this section we discuss several of them.

Dong et al. [3] explore the performance of SR-IOV together with the hypervisor Xen. Three major optimisation methods are proposed and tested, all concentrating on interrupts and the minimisation of performance hits from them. One of them being the moving of message signalled interrupts (MSI) from the device model in userspace into the hypervisor. The emulation of a virtual end of interrupt (EOI) is also identified as a hotspot and simplified massively. Thirdly, an adaptive interrupt coalescing (AIC) method is developed, which adjusts the interrupt rate dynamically based on a set of equations to reduce load on the CPU. This optimization presumably has the most impact on latency, though unfortunately this is not measured. A 9.6 % drop in TCP throughput is observed when using an interrupt frequency of 1 kHz, which is attributed to TCPs latency sensitivity, though it is mitigated by the AIC optimisation. Since the bandwidth is at its maximum of around 1 Gbit/s per guest even before the optimizations, it is barely affected by them. As is expected, overhead of the CPU is heavily reduced, together the optimizations reduce it from 499 % to around 227 %. This mostly affects the host, but some improvements are also made in the CPU overhead of the guests. When testing the scalability of the three optimizations no major issues are detected, even at 60 VMs, although it might be important to mention that there are still only a total of 10 1 Gbit/s connections available [3].

Li et al. [9] explore the method of throttling the frequency of these interrupts on the fly to increase the throughput. The method of using a fixed interrupt rate (FIR) implemented in some device drivers is used as a baseline for measurements with the baseline fixed at 8000 Interrupts/s. Two new approaches of regulating interrupts are proposed. Course grained interrupt rate control (CGR) classifies the traffic into four categories, depending on the packet size. Based on that, the interrupt rate is set, higher for smaller packets and lower for larger packets. Packets with sizes from 64 B to 300 B are classified as latency sensitive traffic and the interrupt rate is set to 20 kInterrupts/s. Packets smaller than 64 B are classified as latency critical traffic, the interrupt rate is set to 100 kInterrupts/s. How this actually translates to praxis is not measured. The idea behind Adaptive interrupt rate control (AIR) is that there is always an optimal interrupt rate, depending on the currently used bandwidth and average packet size, which can be calculated. It is set based on the current bandwidth, average packet size, number of packets received on each interrupt and the current interrupt rate. Comparing these three approaches using netperf TCP and UDP stream, it becomes clear that for 4 VMs or less the CPU overhead is large enough that the CPU performance is worse using AIR and CGR compared to FIR. When using TCP stream, a packet size of 1472 B and four VMs, the CPU usage for AIR, CGR and FIR is ~500 %, ~400 % and ~395 % respectively. That said, the throughput when using AIR and CGR is consistently on par or higher than when using FIR, for higher VM numbers higher than 8 AIR even outperforms CGR [9].

Huang et al. [10] talk about optimizing SR-IOV with the AIR also discussed by Li et al. [9] as well as an approach using multi-threaded NAPI. New API (NAPI) is an API in the Linux kernel allowing the driver using it to mask some of the interrupts produced by incoming packets. It normally is single-threaded, which makes it more and more of at bottleneck with increasing VM counts, since they all are limited by the capacity of this single thread [11]. A multi-threaded NAPI is proposed, consisting of a dispatcher and n worker threads. This increases the throughput but also the CPU usage. For example when using TCP with a packet size of 1472 B, throughput increased by 38 %, from ~5.8 Gbit/s to ~8 Gbit/s, but CPU usage also increased by 73 %, from ~100 % to ~175 %. For smaller packets, the performance gain is barely noticeable, it is up to about 0.6 Gbit/s from 0.5 Gbit/s. The increase in CPU usage is similarly small, from 60 % to 65 %. UDP performance is behaves similar, but it is lower in general, due to UDP [10].

4. Analysis

In some works, the latency of SR-IOV is very close to that of native networking, in others it is not. In some it is not compared to native. The absolute numbers differ greatly as well, by up to 20 µs. Liu [2] measures an absolute latency of 24 µs for SR-IOV and 17 µs for native with packet sizes $\leq$1 KiB. Normalized to the latency of native networking that equates to 141 % for SR-IOV. For the same packet sizes Lockwood et al. [4] measure the latency of SR-IOV at 40 µs, which is nearly twice as much. When normalizing the latencies, SR-IOV is at about 200 % to 250 % compared to that of native. These numbers do not at all match those measured by Liu [2]. Even though the setups seem similar at first, they differ in many aspects, not even the benchmarks used to measure the latency are similar. This means that even though the numbers seem comparable, they actually are not.

Since the impact of the previously mentioned optimizations is also often not measured, we try to make some educated guesses on how it could behave The MSI and the EOI optimizations proposed by Dong et al. [3] presumably have little to no impact on latency. This is because both do not really touch when an incoming packet is processed but rather how it is processed. Since both optimizations do not massively change what actually happens when a packet arrives, we can assume that the latency stays about the same. The optimizations with the presumably largest impact on latency are the various interrupt coalescing optimizations. Since they reduce load on the CPU by coalescing interrupts together, an arriving packet might not immediately get processed. This could lead to increased latency, with the severity depending on the intricacies of the optimization. It is even touched upon by Dong et al. [3] where the degraded TCP performance after applying the AIC optimization is attributed to the higher latency sensitivity of TCP. We can assume that the other coalescing optimizations behave similarly. The CGR and AIR optimizations mentioned by Li et al. [9] and Huang et al. [10] are very similar to the previously mentioned interrupt coalescing optimizations, but they adjust the interrupt rate on the fly based on the traffic. This could mean that latency sensitive traffic is still delivered fast enough and large traffic volumes are not bottlenecked by the CPU. Though due to the way the current interrupt rate is calculated, the latency sensitive traffic would have to dominate either in number of packets or in used bandwidth. This could mean that the applications running on the machine have to be carefully chosen and matched to not interfere with each other.

5. Conclusion

As mentioned previously, the very few latency measurements which are provided paint a very inconclusive picture. The measurements which are provided can only be generalised to a certain point. We did make some educated guesses about how the optimizations mentioned before could impact latency, but those are only guesses and do not replace actual measurements. More latency measurements are definitely needed, with and without optimizations.

One question that is still left completely unanswered by any of the surveyed papers is how latency behaves with a growing number of VMs. Scalability is touched upon by Dong et al. [3] and Bauer et al. [8], but not in regards to latency. Bauer et al. [8] show what is to be expected, above a certain number of VFs the CPU becomes a bottleneck due to the huge amount of interrupts that need to be handled. Since they did not use VMs, but only VFs used by applications directly, this effect would presumably get worse since the interrupts not only need to be handled but then also virtualized. These problems could maybe be mitigated by the optimizations proposed by Dong et al. [3]. Their optimizations make VMs with SR-IOV scale nearly perfectly, at least bandwidth- and CPU-wise. Unfortunately latency is measured by neither. Either way, unfortunately, more data is also absolutely necessary.

Although we tried to focus on latency in this paper, bandwidth still needs to be talked about. There is some conflicting data regarding this. While most of the mentioned papers come to the conclusion that bandwidth does not seem to be a problem for the tested systems (mostly 10 GbE), Lockwood et al. [4] discover that SR-IOV cannot quite keep up with native. Though in this case, it could be due to the fact that MPI bandwidth is tested, not "normal" bandwidth. In most cases, SR-IOV can keep up with native, even at speeds up to 20 Gbit/s [5].

References

[1] "Xen Networking," https://wiki.xenproject.org/wiki/Xen_Networking, Last Accessed: 2021-06-07.

[2] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[3] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–10.

[4] G. K. Lockwood, M. Tatineni, and R. Wagner, "SR-IOV: Performance Benefits for Virtualized Interconnects," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, 2014.

[5] N. Pitaev, M. Falkner, A. Leivadeas, and I. Lambadaris, "Characterizing the Performance of Concurrent Virtualized Network Functions with OVS-DPDK, FD.IO VPP and SR-IOV," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 285–292.

[6] X. Xu and B. Davda, "SRVM: Hypervisor Support for Live Migration with Passthrough SR-IOV Network Devices," *SIGPLAN Not.*, vol. 51, no. 7, pp. 65–77, 2016.

[7] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, 2014, pp. 445–458.

[8] S. Bauer, D. Raumer, P. Emmerich, and G. Carle, "Intra-Node Resource Isolation for SFC with SR-IOV," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, 2018, pp. 1–6.

[9] J. Li, S. Xue, W. Zhang, R. Ma, Z. Qi, and H. Guan, "When I/O Interrupt Becomes System Bottleneck: Efficiency and Scalability Enhancement for SR-IOV Network Virtualization," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1183–1196, 2019.

[10] Z. Huang, R. Ma, J. Li, Z. Chang, and H. Guan, "Adaptive and Scalable Optimizations for High Performance SR-IOV," in *2012 IEEE International Conference on Cluster Computing*, 2012, pp. 459–467.

[11] "NAPI," https://wiki.linuxfoundation.org/networking/napi, Last Accessed: 2021-07-30.