Unleashing SR-IOV Offload on Virtual Machines

Yui Washizu
NTT Open Source Software Center
Tokyo, Japan
yui.washidu@gmail.com

Abstract

Hardware offloading through Single Root I/O Virtualization(SR-IOV) is one of the methods to accelerate virtual networks, used by platforms such as OpenStack and Kubernetes. These platforms provide network features, such as routing, switching and encryption, through software implementation by default. Hardware offloading can radically eliminate overhead of network features implemented by software. Thus, it is super fast and useful for performance-sensitive applications. In spite of their usefulness, currently they can only offload Linux network functions on physical machines and cannot be used for Linux network functions on virtual machines (VMs). Thus, offloading virtual networks for containers running on VMs is not possible although there are cases where deploying containers to VMs rather than physical machines is preferred for high flexibility.

We aim to achieve offloading Linux networks on VMs to SR-IOV physical NICs. The reason why Linux networks on VMs cannot be offloaded is that network construction software for SR-IOV offload (e.g. SR-IOV CNI plug-in) cannot access PFs from within VMs. It is common that VFs are assigned to VMs when Linux networks on physical machines for VMs are offloaded by SR-IOV with PCI device assignment. Thus, when we deploy containers to virtual machines, the network construction software in the guest OS only can access VFs, not PFs. There are two existing solutions to resolve this problem, but they have drawbacks. One lacks scalability (one VM per PF) and also introduces security concerns and the other increases complexity in each network construction software.

We propose a solution which involves emulating PFs that have SR-IOV feature while offloading data plane to hardware. With this, it is possible to offload Linux network functions on VMs with high scalability of emulated PFs, enhanced security by not giving VMs full NIC hardware control, and minimum increase of complexity in network construction software. In order to accelerate data plane with this solution, we propose utilizing vDPA. Our PoC implementation employs an L2 switching feature in the SR-IOV legacy mode. We measured throughput and latency of container networks on VMs with our PoC. The results were several times better compared to when not using SR-IOV.

Keywords

Virtual Network, Container, Hardware Offload, Virtual Machine, SR-IOV, vDPA

Introduction

Hardware offloading is one of the methods to accelerate virtual networks, used by platforms such as OpenStack and Kubernetes. These platforms provide network features, such as routing, switching and encryption, through software implementation by default. These cause overhead on emulation layer and might become bottleneck to accelerate virtual networks [2] [3]. Examples of such overhead include CPU cache miss hits due to a network feature using hash tables and system calls when features are implemented in user space. Hardware offloading can radically eliminate overhead of network features implemented by software. Thus, it is super fast and useful for performance-sensitive applications.

The network construction software on platforms such as OpenStack and Kubernetes can offload network processing with Single Root I/O Virtualization (SR-IOV) [10], which is one of hardware offload technologies. SR-IOV enables a single PCI function (physical function, PF) to create multiple virtual PCI functions, called virtual functions (VFs). This feature is not only for networking acceleration, but we focus on network usage, where the PCI devices are NICs. Common NICs with the SR-IOV feature have embedded switches within themselves. Communication between VFs and PFs or with external networks is forwarded by the embedded switches. Also, depending on functionality of the NIC, it is possible to provide various network functions such as a firewall on the NIC. The platforms for VMs and containers can significantly improve performance by configuring these switching and network functions as it can reduce overhead by eliminating the need for software-based network function implementation. In order to use SR-IOV, administrators or network construction software needs to create VFs and configure switches in NICs by accessing PFs. One example of such network construction software is CNI-compliant [8] plugins, which are used to construct virtual network on Kubernetes. We assume that administrators create VFs and CNI plugins assign VFs to containers. Embedded switches may be configured by either of administrators or CNI plugins, but configuration by administrators is significantly limited and CNI plugins needs to configure embedded switches for common usage such as dynamic network configuration on Kubernetes platforms. SR-IOV CNI plugin, OVN CNI plugin and Antrea CNI plugin are examples of plugins that can use SR-IOV, all of which behave as assumed above and configure embedded

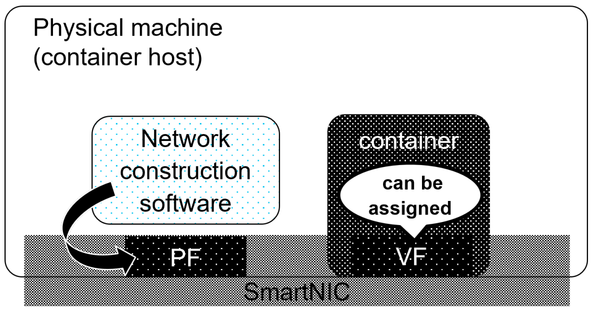


Figure 1: Deploying to physical machine

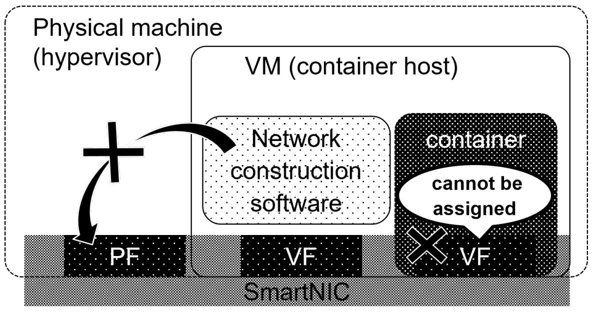


Figure 2: Deploying to virtual machine

switches themselves.

In spite of their usefulness, currently they can only offload Linux network functions on physical machines and cannot be used for Linux network functions on virtual machines (VMs). Thus, offloading virtual networks for containers running on VMs is not possible although there are cases where deploying containers to VMs rather than physical machines is preferred for high flexibility. The same is true for nested VMs.

Problem of Offload on VMs

As described in the previous section, network construction software needs to access PFs for common SR-IOV offload use cases. The reason why Linux networks on VMs cannot be offloaded is that network construction software for SR-IOV offload (e.g. SR-IOV CNI plug-in) cannot access PFs from within VMs. Let us now describe why the software running on VMs cannot access PFs. Here, we presume that Linux networks on physical machines for VMs are offloaded by ”PCI device assignment” of VFs to avoid overhead caused by software. PCI device assignment is a technology to allow guests to directly utilize physical PCI devices. It is common that VFs are assigned to VMs when Linux networks on physical machines for VMs are offloaded by SR-IOV. Thus, when we deploy containers to virtual machines as shown in figure 2, the network construction software in the guest OS only can access VFs, not PFs, in contrast to when deploying containers to physical machines as shown in figure 1 where the software can recognize and access PFs. Network construction software for SR-IOV does not work on VMs due to lack of access to PFs; consequently, virtual networks running on VMs cannot be offloaded by SR-IOV.

There are two existing solutions to resolve this problem:

1. Assign PFs to VMs and allow the guest to access PFs
2. Assign multiple VFs to VMs

However, both of these two methods have drawbacks.

Assign PFs to VMs and allow the guest to access PFs If we allow a VM to access a PF by PCI device assignment, the VM can directly control the physical device, which increases attack vectors from the VM. And, this method also lacks scalability because the VM exclusively use the specific NIC. Therefore, we should avoid direct access to PFs from VMs.

Assign multiple VFs to VMs This method involves assigning multiple VFs on physical machines to VMs and using some of them for containers running on VMs instead of creating VFs on VMs. In other words, this method delegates creation of VFs to hypervisors of VMs.

This requires VM administrators to have privilege in hypervisors or additional means to make hypervisors create and assign VFs to hot-plug them. Moreover, it is even more difficult to delegate configuration of embedded switches to hypervisors because in general it is network construction software that configures the switches. For the sake of delegation, the software must have implementation to collaborate with hypervisors, which increases complexity in each software. Also, we have not found any such software so far. It is possible that administrators configure switches in place of network construction software, but in such a case only limited networking functions, such as static network configuration, are available on VMs because even simple dynamic configuration is too complicated for administrators. Another example of switch features difficult for administrators is SR-IOV switchdev mode, with which we can use advanced switch features such as a firewall. The reason for the difficulty is that administrators are required to do more complicated handling than SR-IOV legacy (non-switchdev) mode. Therefore, this method is not a sufficient solution.

Consequently, we aim to achieve offloading Linux networks on VMs to SR-IOV physical NICs on the environment where guests cannot access or control PFs in order to assign VFs to VMs and to configure switches with more common configuration than static network.

Approach: Virtual PF

We propose a solution which involves emulating PFs that have SR-IOV feature. (This is called ”virtual PF”.) Hypervisors handle hardware control requests from VMs through virtual PFs to configure hardware. This enables creation of new VFs, called ”virtual VFs”, and allows SR-IOV to be controlled within VMs. The number of virtual PFs is not limited by PFs of NICs, so it scales well. Also, hypervisors can avoid giving VMs control of the entire NIC hardware because virtual PFs are emulated devices, leading to less security concerns. Another advantage of virtual PFs is that SR-IOV can be used in the same way as on physical machines.

Administrators on VMs can create VFs through virtual PFs in the same manner as normal PFs. Since network construction software does not need to communicate with hypervisors in this approach, existing network construction software works without adding special implementation for VMs. As a result, we can avoid the increase of complexity in each network construction software. Dynamic network configuration including SR-IOV switchdev mode is feasible because existing network construction software works in VMs and we do not need to rely on administrators to configure embedded switches in physical NICs. Also, in order to accelerate data plane with this solution, we propose utilizing virtio data path acceleration (vDPA) device [9], a technology added to Linux in 2020. vDPA is realized by a type of device (called vDPA device) whose datapath conforms to the virtio specification but whose control path is vendor specific. vDPA devices only allow data plane to be directly accessed from VMs and keep control plane emulated. This technology is originally used in live migration, etc. Figure 3 and 4 show control and data plane in this approach.

Another possible approach is combining SR-IOV and lightweight functions or devices that are non-compliant with PCI specification, such as SubFunctions (SFs) [7] and scalable IOV [6]. If we can create lightweight functions or devices through VFs assigned to VMs, just using such features could solve the problem. However, this is not a generic solution because such features are highly vendor-dependent, and we have not found any NICs with such features so far. This may be worth revisiting once NICs with such features become available in the future.

Implementation Details

As we have already explained in the previous section, we emulate PFs that have SR-IOV feature. We describe the implementation added to qemu and Linux kernel for this emulation.

The emulated device to be added the SR-IOV feature to is virtio-net because we utilized vDPA devices as backend devices and they require virtio-net devices. When administrators on VMs issue requests to add VFs, e.g., writing a number to sysfs sriov_numvfs file on Linux, Qemu

- creates emulated virtio-net devices for VFs,
- exposes them as PCI virtual functions to VMs, and
- assigns backend vDPA devices to them.

Also, because virtio-net driver did not have basic functions to get SR-IOV information, we added functions which we consider is minimum required to Linux kernel used in VMs. Specifically, two functions were added: one to get the number of VFs and another to get the configuration of VFs.

This approach needs vDPA devices as backend devices, while existing SR-IOV implementation, which is igb SR-IOV [1] whose embedded switch feature is emulated in Qemu, does not need backend devices. Since we do not hard-code vDPA devices as backend devices, backend devices are pluggable. So, it makes this approach useful for purposes other than acceleration. For example, using tap devices as backend devices is helpful to test and debug software using the SR-IOV feature on VMs, as shown in Figure 5. And it allows us to try virtio-net SR-IOV when we do not have physical NICs with vDPA feature.

We delegated configuration of embedded switch of emulated SR-IOV to management layer, not Qemu. The embedded switch of emulated SR-IOV may be the embedded switch of physical SR-IOV NICs in case of vDPA or Linux bridge for tap devices. Examples of what takes role of the management layer include libvirt and administrators.

This time we only implemented SR-IOV legacy mode. The solution for SR-IOV switchdev mode in VMs is discussed in the section of Future works.

Verification of Operation and Performance

We verified operation and performance of our PoC implementation [5]. The purpose of this verification is to confirm that network construction software correctly works and that performance is improved by vDPA. We tested the legacy SR-IOV implementation stated above.

Target Environments

We performed functional and performance verification in two environments. One is an environment to measure baseline performance where virtual networks for containers in VMs cannot be offloaded. This environment is called ”Without Virtual PF”. The other is an environment with our proposed method, ”virtual PF”, where we can offload virtual networks for containers in VMs. This environment is called ”With Virtual PF”. Hardware and software used to build the two environments is shown in Table 1.

The followings are conditions common to both environments:

- Utilize two virtual machines as Kubernetes nodes; one is master and the other is worker.
- Start containers as Kubernetes pods and verify communication between containers and physical machines different from hypervisors of Kubernetes nodes.
- Utilize legacy SR-IOV VFs in physical NICs on hypervisors of Kubernetes nodes.
- The netperf client process (netperf) and server process (netserver) run in a pod on the worker node.

Any acceleration, including the proposed method and the technologies to accelerate virtual network on VM, is only applied to the worker node because netperf processes for communication of this verification runs in the worker node.

This verification was performed with a single flow. We stopped the operation of irqbalance service on the hypervisor to ensure that the cores generating interrupts were not changed during the verification.

The performance verification metrics are throughput and latency. The latency is the round trip time between the container and the external physical machine.

Setup: Without Virtual PF

Figure 6 shows the overview of this environment. In this environment, we utilize CNI plugin Calico for data plane. Therefore, the followings included in Kubernetes virtual network are software implementation:

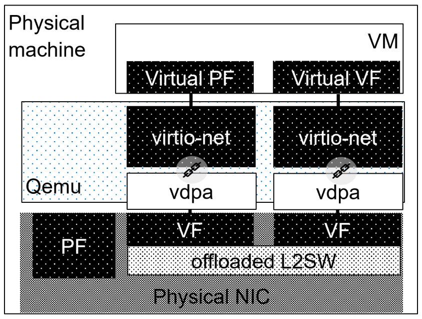


Figure 3: Control Plane of Approach

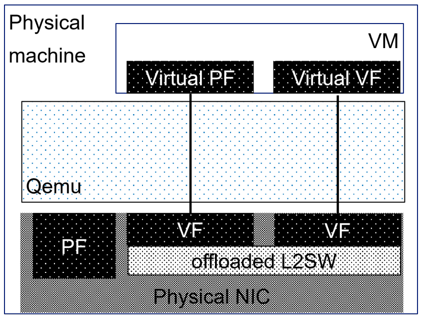


Figure 4: Data Plane of Approach

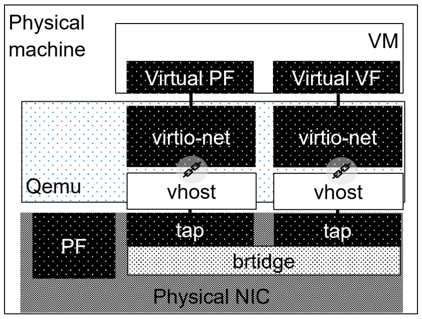
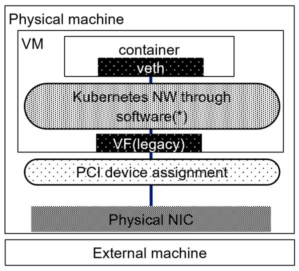


Figure 5: For Testing

Table 1: The hardware and software used to build the environment

Server model	HPE ProLiant DL360 Gen9
CPU	Intel Xeon CPU E5-2600 @2.3GHz (CPU x2, 10 cores/CPU)
NIC	Mellanox Technologies MT27710 family ConnectX-6 Dx (100G)
host/guest OS	Rocky Linux 9.2
host/guest kernel	6.5.7
qemu version	Qemu 8.1.1 (w/ virtio-net legacy SR-IOV support PoC patch applied)
Kubernetes version	1.27.6
CNI plugin	Calico v3.26.3
SR-IOV Device Plugin	2.7.0
netperf	2.7



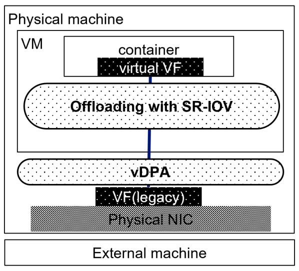


Figure 6: Without Virtual PF Figure 7: With Virtual PF

- L4LB/NAT: iptables implementation of kube-proxy
- Routing: CNI plugin Calico
- Firewall: NetworkPolicy function of CNI plugin Calico

We accelerated virtual network between VMs and external physical machines with PCI device assignment.

Setup: With Virtual PF

Figure 7 shows the overview of this environments. We applied a patch [5] to qemu to enable virtual PF and utilized the patched qemu in this environment. We used SR-IOV CNI plugin for data plane in contrast to ”without virtual PF” which uses CNI plugin Calico. Because SR-IOV CNI plugin only manages virtual VFs assigned to pods, following plugins were required:

- SR-IOV Network Device plugin: discover and advertise networking resources
- CNI meta plugin: retrieve allocated network device information of a Pod and give SR-IOV CNI plugin

In this verification, we used Multus CNI plugin as the CNI meta plugin. Multus is a CNI plugin which enables multiple network interfaces to be attached to a Kubernetes pod. Since SR-IOV CNI plugin is used by Multus to attach an additional interface, Multus requires another CNI plugin called ”master CNI plugin” to create the primary interface which is used for Kubernetes cluster network. We used Calico CNI plugin as the master CNI plugin.

Operation

Verification method We checked the followings in the environment with Virtual PF:

1. New virtio-net devices (virtual VFs) are created and recognized by kernel when an administrator in a VM writes the number of VFs to sysfs sriov_numvfs file of virtual PFs on Linux.
2. Network construction software correctly works.
   (a) SR-IOV network device plugin recognizes virtual VFs as VF devices.
   (b) SR-IOV CNI plugin can assign virtual VFs to containers on the worker node.
   (c) The external physical machine and containers on the worker node can communicate with each other.

Result We checked there was no problem with 1, 2(a), 2(c). 2(b) did not work without changes to SR-IOV CNI plugin mainly because virtio-net is subtly different from other NICs.

Table 2: Performance comparison on a single core [Mbps]

	transmission	reception
without Virtual PF	1635	968.1
with Virtual PF	1919.1	5421.9

The difference resides in sysfs, the interface to get information of relation between VFs and PFs. Nevertheless, since the basic mechanism using Netlink to configure the embedded switch is kept unchanged, we consider that these changes are slight and increase little complexity in CNI plugins. Let us now concretely describe the difference and these changes added to software.

Unlike other network devices, virtio-net-pci devices are configured such that PCI devices themselves are not directly recognized as network devices. Instead, virtio-net-pci devices have virtio buses under themselves. Because of this difference in the hardware structure, the sysfs directory structure of the Linux kernel is different from other network devices, and SR-IOV CNI plugin was not compatible with this. So, we added modifications to SR-IOV CNI plugin used in this environment to accommodate this.

There was another pitfall in the VLAN setting of virtio devices. Even though we do not configure VLAN, SR-IOV CNI plugin automatically attempts to configure VLAN. This caused an EOPNOTSUPP error because virtio-net driver in Linux kernel does not have a function to handle network attribute IFLA_VF_VLAN which is used for VLAN configuration. We modified SR-IOV CNI plugin to ignore EOPNOTSUPP as a workaround. Notice that this problem will disappear once the VLAN feature is added to virtio-net driver in guest kernel in the future.

Performance: Throughput

Verification method We conducted a performance validation in each target environments. This validation involved 60-second UDP bulk transfer between a container and an external physical machine. We performed the measurements five times and the average values of these are the validation results of both transmission and reception throughput.

We conducted UDP bulk transfers from the container to the external machine for validation of transmission performance, and conducted it from the external machine to the container for validation of reception performance.

Regarding transmission performance, we measured it without applying bandwidth control because we confirmed that the Linux kernel correctly applied backpressure from the NIC to the socket. On the other hand, for reception performance, we gradually increased the bandwidth limit on the sender side with TC HTB filter to measure the maximum throughput to determine the upper limit performance.

Result The results are shown in Table 2. In terms of transmission performance in "With Virtual PF", we observed a performance increase of about 1.2 times compared to the performance in "Without Virtual PF", and the reception performance improved about 6 times than "Without Virtual PF", indicating a significant performance improvement.

Table 3: Performance comparison on a single core [μsec]

	latency
without Virtual PF	320.2
with Virtual PF	221.8

The transmission result of 1.2 times improvement was lower than we expected. Since we assigned emulation work of guest interrupts to the same CPU as the vCPU was running on in this test, we consider that overhead of the interrupt emulation caused the lower performance. In fact, when we assigned the emulation work to a different CPU, the performance drastically improved.

Performance: Latency

Verification method In both environments, we measured latency by conducting multiple Request/Response communications over UDP for 60 seconds between a container and an external machine. We conducted the measurements five times, and the average of 99th percentile latencies was used as the validation result for latency.

Result The results are shown in Table 3. Compared to the latency without Virtual PF, the one with Virtual PF has decreased by 100 μsec.

Future works

We are considering a potential solution for realizing switchdev mode on VMs. For switchdev mode, tc flower filters in VMs must be offloaded. In other words, a flow-based switch needs to be emulated in hypervisors. Also, in order to accelerate virtual networks with vDPA, the flow-based switch must be offloaded to the embedded switch in physical NICs. We consider Open vSwitch (OVS) [4] fits emulation of the flow-based switch because switchdev mode was originally introduced for offloading OVS.

We illustrate how to realize switchdev mode SR-IOV on VMs with OVS. OVS is able to have multiple virtual bridges. We can use one of these bridges as an emulated flow-based switch for a VM. Qemu and management layer configure the virtual bridge by translating offloaded TC flower to OVS commands corresponding to ovs-ofctl. OVS has a feature to offload virtual bridges to physical NICs, with which we can offload the virtual bridge for the emulated flow-based switch to offload vDPA datapath as shown in Figure 8.

Conclusion

We proposed "Virtual PF," as a solution to offload Linux network on VMs with SR-IOV. This approach emulates PFs with SR-IOV features, creates "virtual VFs," and allows VMs to control SR-IOV. Benefits include high virtual PF scalability, enhanced security by not giving VMs full NIC hardware control, and same SR-IOV usage as on physical machines. In order to accelerate data plane with this solution, we propose utilizing vDPA.

In this solution, when administrators on VMs issue requests to add VFs, e.g., writing a number to sysfs

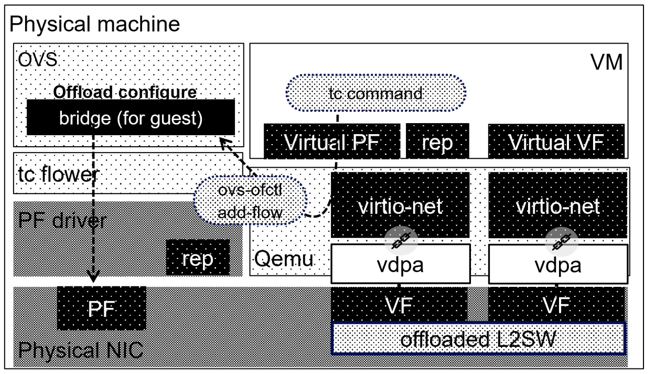


Figure 8: Concept for Switchdev mode

sriov_numvfs file on Linux, Qemu creates emulated virtio-net devices for VFs, exposes them as PCI virtual functions to VMs, and assigns backend vDPA devices to them. Since we do not hard-code vDPA devices as backend devices, backend devices are pluggable. Thus, it makes this approach useful for purposes other than acceleration. This time we only implemented SR-IOV legacy mode.

We verified operation and performance of our PoC implementation. In the verification of operation, there was almost no problem although we needed to add slight changes, which involve sysfs for VF and PF relation information, to SR-IOV CNI plugin. However, the basic mechanism to configure emulated switches remains unchanged, minimizing impact on CNI plugins. Also, we observed performance improvement with transmission throughput by 1.2 times and reception by 6 times and latency reduced by 100 μsec. The transmission result was lower than expected due to the overhead of emulation work of guest interrupts.

In conclusion, the proposed approach shows significant performance improvements when applied to container platforms deployed on VMs. We are considering potential solution with OVS for realization of switchdev mode on VMs.

References

[1] [PATCH v6 0/9] Introduce igb. https://www.mail-archive.com/qemu-devel@nongnu.org/msg936193.html.

[2] Istio. Performance and Scalability. https://istio.io/latest/docs/ops/deployment/performance-and-scalability/.

[3] Kapočius, N. 2020. Performance studies of kubernetes network solutions. In *2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, 1–6.

[4] Open vSwitch. https://www.openvswitch.org/.

[5] QEMU patches - virtio-net: add support for SR-IOV emulation. https://patchwork.kernel.org/project/qemu-devel/cover/1689731808-3009-1-git-send-email-yui.washidu@gmail.com/.

[6] Scalable Function (SF). https://docs.nvidia.com/doca/archive/doca-v1.1/scalable-functions/index.html.

[7] Mellanox ConnectX(R) mlx5 core VPI Network Driver - mlx5 subfunction. https://docs.kernel.org/next/networking/device_drivers/ethernet/mellanox/mlx5.html#mlx5-subfunction.

[8] The CNI Authors, and The Linux Foundation. CNI. https://www.cni.dev/.

[9] Wang, J. [PATCH V9 0/9] vDPA support. https://lkml.org/lkml/2020/3/26/481.

[10] Yu Zhao, D. D. The Linux Kernel documentation PCI Express I/O Virtualization Howto. https://www.kernel.org/doc/html/v5.15/PCI/pci-iov-howto.html.