Software Physical/Virtual Rx Queue Mapping Toward High-Performance Containerized Networking

Ryota Kawashima, *Member, IEEE*

***Abstract*—Softwarization of Network Functions (NFs) accelerates automated deployment and management of services on next-gen networks. Combining flexibility and high-performance is a vital requirement for Network Functions Virtualisation (NFV); however, many studies have demonstrated that containerization or virtualization of NFs severely degrades the fundamental efficiency of packet forwarding. Virtual network I/O, a mechanism of packet transferring between a guest and the host, has been seen as the performance bottleneck in the PVP (Physical-Virtual-Physical) datapath, and one of the main causes of this deterioration is packet copy between them. Various techniques, such as zero-copy, pass-through, and hardware offloading, have been examined to alleviate the performance overhead. However, existing designs and implementations incur pragmatic issues, such as compatibility, manageability, and insufficient quality of performance. We propose yet another design and implementation of zero-copy/pass-through acceleration (named IOVTee) to resolve real-world problems as well as to enhance the forwarding efficiency. IOVTee takes advantage of pre-processing of virtual switches with achieving zero-copy on the receive (Rx) path. The pluggable style of IOVTee for vhost-user (the de-facto virtual network I/O) enables our approach to be transparent to both containers/VMs and virtual switches. In this article, we explain the heart of IOVTee, a fully software-based Rx queue mapping mechanism (between physical and virtual) that enables a concept of Virtual DMA Write-through (to the NF). Our evaluation results showed that applying IOVTee to vhost-user drastically increased efficiency of packet forwarding in the PVP datapath (by 45% and 98% for traffic of 64-byte and 1514-byte packets respectively).**

***Index Terms*—Network functions virtualization, container, DPDK, Vhost-user, zero-copy, pass-through.**

I. Introduction

NETWORK Functions Virtualisation (NFV) [1] is a mission-critical technology for 5G network slicing, and tailored virtual networks (*slices*) are agilely provided on the substrate network in a multi-tenant manner [2]. Such an extremely flexible nature comes from NFV accelerates innovative services requiring diverse network characteristics [3].

Manuscript received June 1, 2020; revised October 28, 2020 and December 23, 2020; accepted December 29, 2020. Date of publication January 4, 2021; date of current version March 11, 2021. This research and development work was supported by the MIC/SCOPE #182106107. The associate editor coordinating the review of this article and approving it for publication was T. Zinner.

The author is with the Department of Computer Science and Engineering, Nagoya Institute of Technology, Nagoya 466-8555, Japan (e-mail: kawa1983@ieee.org).

Digital Object Identifier 10.1109/TNSM.2020.3049053

In this softwarization era, various cloud native technologies as represented by Kubernetes [4] have grown in real services. Telco applications are required to be container-based microservices [5], [6]. These technologies have a remarkable affinity to NFV in terms of lifecycle management of containerized (cloud native) or virtualized network functions (CNFs/VNFs); however, many studies have demonstrated substantially lower packet forwarding efficiency of CNFs/VNFs than both hardware-based and (baremetal-formed) software-based implementations [7], [8], [9], [10], [11].

Now performance-critical NFs are deployed on white-box switches equipped with a programmable ASIC. P4 language [12], [13] and open domain specific architectures (e.g., Open Compute Project [14]) alleviate the traditional compatibility and vendor lock-in issues. Regarding network slicing, *pure* softwarization on commodity servers is promising for agility and flexibility, while they are less suitable for performance-critical functions. Containerized NFs running on commodity servers have definitive fortes for both developers and administrators in that better programmability and manageability. Therefore, we believe that further work is necessary to accelerate the fundamental performance of softwarized CNFs/VNFs.

Virtual network I/O, a mechanism of packet transferring between a virtual switch and CNFs/VNFs on the host, has been considered as the performance bottleneck in the PVP (Physical-Virtual-Physical) datapath. In practice, packet copy degrades performance of virtual network I/O, and its overhead cannot be reduced even with Data Plane Development Kit (DPDK) [15], a widely-used fast packet I/O framework. A major approach to achieve *zero-copy* at virtual network I/O [16], [17], [18] is to keep packet data being stored in a shared packet pool of the host side during the PVP datapath. However, accessing the memory region of the host requires alternations of existing virtual network I/O and also requires CNFs/VNFs to be aware of the host environment. Hardware-assisted pass-through like SR-IOV [19] avoids the packet copy in a safe manner thanks to IOMMU [20], but beneficial packet pre-processing of virtual switches is completely bypassed. Furthermore, we have demonstrated that SR-IOV obviously degrades performance stability [7], [21]. As a result, *copy-full* vhost-user [22] still has been the de-facto virtual network I/O technology.

We propose a novel zero-copy/pass-through design and implementation, named IOVTee, for not only boosting up

packet forwarding efficiency in the PVP datapath, but also resolving pragmatic concerns such as compatibility, manageability, and inexplicable performance characteristics. IOVTee takes advantages of the packet pre-processing with achieving zero-copy on the Rx path. We have designed and implemented IOVTee as an internal add-on to DPDK/vhost-user of the host side to hide IOVTee from both virtual switches and CNFs/VNFs. Such a transparency is essential for 5G network slicing in that user services can receive performance gain from IOVTee without modifications of them. In this article, we explain system requirements of IOVTee toward real world softwarized networks as well as the architecture and implementation. Specifically, we have developed a fully software-based *Rx Queue Mapping* mechanism (between physical and virtual) that enables a novel zero-copy concept (*Virtual DMA Write-through*). We evaluate and quantitatively analyze the performance effect of IOVTee on vhost-user in Section V.

This article is an extended version of our conference paper [21]. In the extended paper, we have fully re-implemented IOVTee from scratch because our previous implementation showed suboptimal performance. Our new implementation has become more cache aware and supported *Packed* Virtqueue of vhost-user, which results in considerable performance enhancement. Evaluation becomes more fulfilling in that further mechanisms are examined such as container-formed NFs, *Packed* Virtqueue, OpenNetVM [17], and CPU caching behaviors are quantitatively analyzed.

The remainder of this article is as follows. System requirements of IOVTee are explained in Section II in view of practical usage, and we describe architectural design and implementation in Sections III and IV respectively. In Section V, we show the results of comprehensive evaluation of IOVTee-enabled NFV-nodes. Related work for performance improvement of the PVP datapath is referred to in Section VI. We conclude this study and give future work in Section VII.

II. System Requirements

Cutting-edge performance acceleration mechanisms for virtual network I/O have pragmatic issues as described in Section VI, and many of them have not proved that the maximum throughput (for minimum-size packet traffic) can be boosted to over 10 Gbps (14.88 Mpps). Thus, we have to design and implement IOVTee to satisfy the following system requirements.

- *DPDK/Vhost-User Compatible:* DPDK is commonly seen in software-based NFV-nodes as a packet I/O base, and vhost-user is the de-facto virtual network I/O for containerized or virtualized NFs on the DPDK-enabled host environment. This requires IOVTee to be DPDK/vhost-user compatible to take advantage of existing NFV ecosystems like OPNFV [40].
- *Fully Software-Based:* Flexible and agile composition of various network services is the heart of the NFV concept and this nature comes from *softwarization*. Special purpose hardware devices (e.g., SmartNICs) have been developed for accelerating a specific software like Open vSwitch by offloading its heavier packet processing to the devices, but other applications cannot enjoy such a feature without being tailored to them. SR-IOV, the hardware-based pass-through mechanism, does not depend on applications in nature, but containers/VMs have to be physical NIC-aware to use virtual functions (VFs) of the NIC. We have to adopt a fully software-based approach for IOVTee to make it transparent to not only CNFs/VNFs but also virtual switches and physical NICs. Moreover, IOVTee should be as orthogonal to the dedicated hardware features as possible.
- *Minimum Modification:* Generally, performance accelerations involve modifications of existing components, such as NFs, packet I/O frameworks, virtual switches, virtual network I/O, virtual machine monitors, operating systems, packet formats, and underlying hardware devices. Modifying wide spread components spoils the acceleration technology in network service operations, even if ideal performance is achieved. Furthermore, critical system components like OS and hardware devices are required to be stable and mature in the real environment. We confine the modification to DPDK internals, and other existing components can be used without any modification.
- *Container/VM Support:* Automated lifecycle management of NFs is the key for advanced network slicing. In particular, container-based (cloud-native) NFs have been getting attentions for agile deployment and less performance overhead [41]. Therefore, IOVTee has to support not only VM-based NFs but also container-based NFs. DPDK has provided network drivers of vhost-user for both containers (*virtio-user* [42]) and VMs (*virtio-net*), which indicates that IOVTee has to be vhost-user compatible.
- *Near-Baremetal Level Performance:* Vhost-user now achieves over 10 Gbps throughput for any size of packet traffic as shown in Section V thanks to the efforts, but there is considerable performance gap with *baremetal* (NFs are directly deployed on commodity servers without containerization/virtualization). IOVTee has to significantly close the gap with optimized implementation.

III. IOVTee Architecture

We have designed our novel zero-copy/pass-through virtual network I/O mechanism, IOVTee, with satisfying the above requirements. IOVTee can combine *DMA-to-the-CNF/VNF* and *virtual switch intervention* approaches by one-to-one mapping of Rx queues between a container/VM and the host (*Rx Queue Mapping*). Apparently, the *DMA-to-the-CNF/VNF* feature is alike SR-IOV in that incoming packets are directly stored in the memory space of the container or VM to avoid packet copy between the host and the guest. However, IOVTee enables the *virtual switch intervention* feature too unlike SR-IOV, meaning that modern highly functional virtual switches, such as Open vSwitch, BESS [43], and Lagopus [44], are given chances to perform their jobs as pre-processing of CNF/VNF's work. Such a pre-processing is effective in real network service

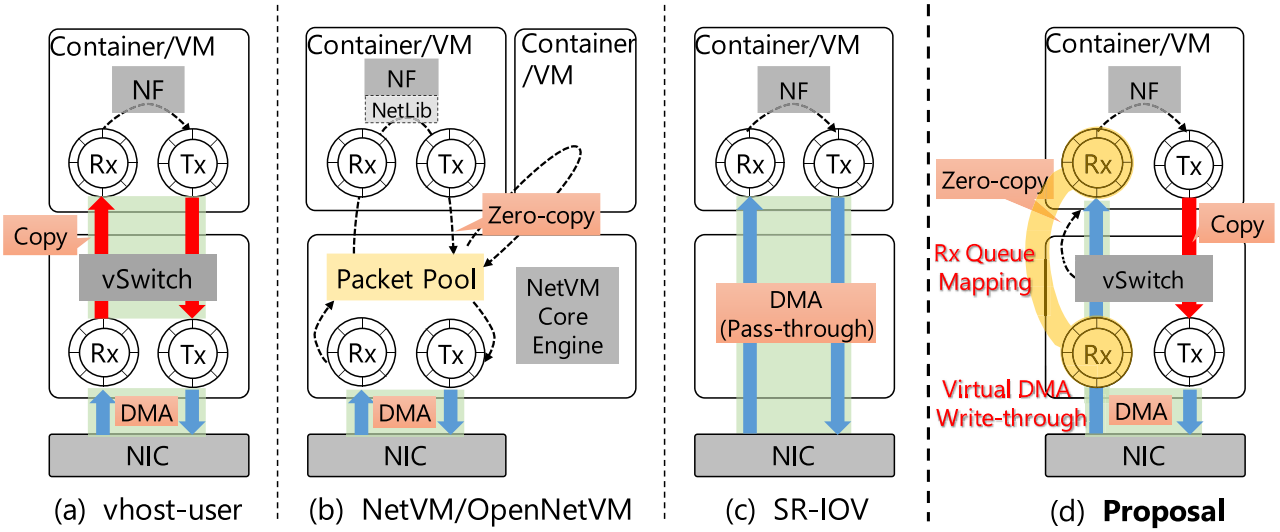


Fig. 1. Overview of Virtual Network I/O Technologies.

operations, for instance, Lagopus has been developed to support carrier-specific features like MPLS, EVPN with VXLAN, IPsec, and BGP. IOVTee is a fully software-based composable approach that can coexist with various virtual switches. We realize this mechanism in a transparent way to both virtual switches and containers/VMs.

Here, we show conceptual differences between IOVTee and other similar technologies in Fig. 1. In type (a), the original vhost-user protocol is used for both the Rx and Tx paths. Vhost-user enables *in-user-space* packet transferring between the virtual switch and containers/VMs, but packet copy is involved in each direction.[1] Type (b) expresses the architecture of NetVM/OpenNetVM, a representative approach of the big shared packet pool. *NetVM Core Engine* (vSwitch) provides a shared packet pool in the host memory space, and the NFs can directly access the pool with the help of a dedicated library (*NetLib*). The switch has to tell exact locations of packet buffers need to be processed to the NFs at the right timing using a vhost-user-like protocol. Type (c) is a case of SR-IOV, and entire packet processing at the host is omitted (Pass-through). Incoming packets are directly DMAed to the guest memory space using I/O virtualization technology, and outgoing packets are transferred to the physical NIC from the guest memory space likewise. Architectural overview of IOVTee is illustrated in (d). Our proposal follows system composition of type (a), but removes packet copy on the Rx path by introducing the two concepts, *Rx Queue Mapping* and *Virtual DMA Write-through*. These are based on a design pattern of performance-oriented virtual network I/O. In case of vhost-user, a container/VM side provisions both Rx/Tx queues and corresponding packet buffers on its memory region, and makes the region accessible from the host side. IOVTee acquires valid physical memory addresses of empty packet buffers of

[1]DPDK provides an optional zero-copy implementation for the Tx-path (*dequeue-zero-copy*), but our previous study [21] revealed that the feature decreased throughput about 15% for 64-byte packet traffic due to a long-term nature of packet staying in the virtual Tx queue.

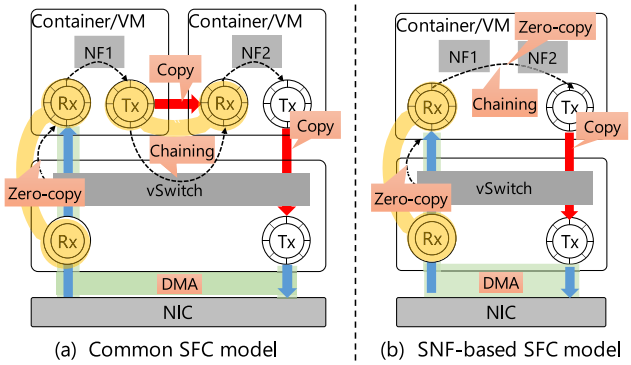
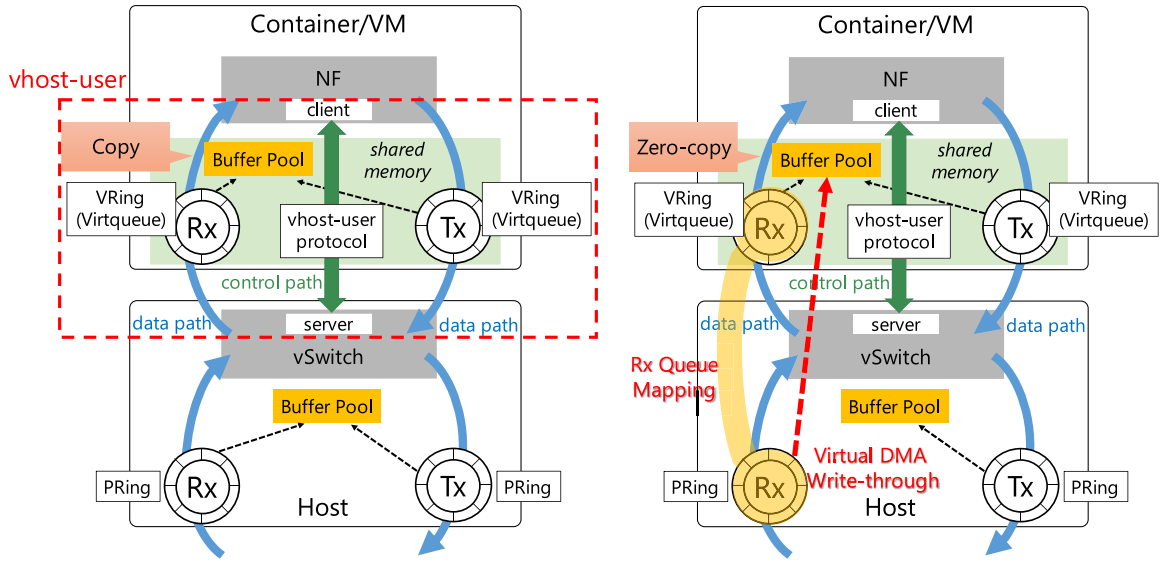


Fig. 2. Rx Queue Mapping in Service Function Chaining.

the CNF/VNF via the Rx queue, and logically maps entries of physical/virtual Rx queues (*Rx Queue Mapping*). In terms of DMA, replacing packet buffers in the host region with the acquired ones results in the elimination of packet copy, because both the host (virtual switch) and the CNF/VNF use the same packet buffers for their own processing (*Virtual DMA Write-through*). IOVTee is a dedicated optimization for the Rx-path, and therefore, packet copy is still necessary for the Tx-path. This copy could be performance concern, but our approach resulted in great throughput increase as shown in Section V.

The notion of Rx Queue Mapping can be applied to an arbitrary pair of queues as long as they can be accessible from the host. For instance, performance of service function chaining can be improved as illustrated in Fig. 2 (a). In this scenario, the Rx queue of the second NF is mapped to the Tx queue of the first NF in advance. This mapping is transparent to the NFs such that the virtual switch directly copies outgoing packet data in the Tx queue to the Rx queue of the next NF, which reduces the number of packet copy per chain (2→1). And now, chaining multiple CNFs/VNFs via virtual network I/O comes under question, because inter-CPU core packet transfer obviously degrades performance [45]. Synthesizing network functions into a single CNF/VNF [46] is a promising approach



(a) vhost-user (b) Proposed

Fig. 3. Internals of NFV-node with vhost-user.

for pursing hardware-level performance, and IOVTee can be integrated into synthesized NFs as shown in Fig. 2 (b).

IOVTee supports two modes for the Rx-path, *Bridge* and *Pass-through*. The bridge mode is useful when pre-processing of the virtual switch is necessary, such as VXLAN decapsulation. Such a pre-processing can be bypassed like SR-IOV in the pass-through mode.

IV. Implementation

We explain details of internal IOVTee implementation in this section. We added an IOVTee library (*librte_iovtee*) into DPDK 19.11, and modified both vhost-user (*librte_vhost*) and Mellanox ConnectX-5 driver (*mlx5*) of the DPDK framework. We have totally revised the previous implementation of IOVTee [21] for further optimization and support of *Packed* Virtqueue, though we keep its conceptual design.

A. Internals of Vhost-User

Before looking into IOVTee internals, we explain the inside of vhost-user to clarify how IOVTee takes advantage of its mechanism while keeping the system interface. Figure 3 shows system internals of an NFV-node with vhost-user (a) and with vhost-user/IOVTee (b). Vhost-user consists of two parts, VRings (Virtqueues) and a vhost-user *protocol*. VRings are one-way ring buffers for data plane communications between the host (vSwitch) and the guest (CNF/VNF), and are provided on a shared memory region offered by the guest for allowing the host to access them. The vhost-user protocol is used for control plane communications to share configurations of VRings between them at startup.[2]

A VRing (Split Virtqueue) is specifically composed of three ring buffers, *avail*, *used*, and *desc* as shown in Fig. 4 (default ring size is 256). Each entry of *desc* points to a virtual memory address of a packet buffer in the guest region and contains its size too. The producer (vSwitch) checks the *avail* ring to know available *desc* entries for enqueuing packets. Likewise, the consumer (CNF/VNF) checks the *used* ring to know available *desc* entries for dequeuing packets. The three rings are synthesized to a *packed desc* ring for *Packed* Virtqueue, aiming at the enhancement of cache hit ratio.

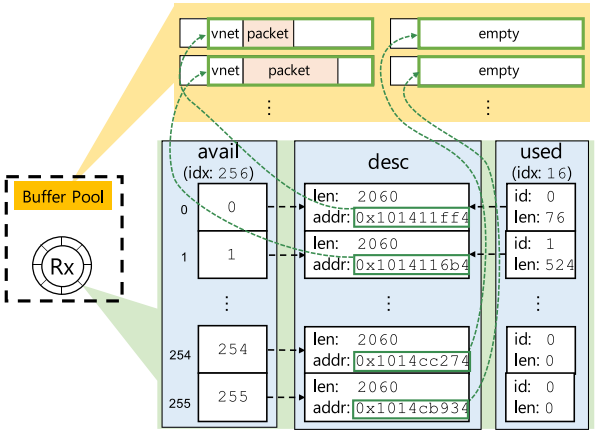


Fig. 4. Three internal rings of the VRing (Virtqueue). (The CNF/VNF has provided 256 empty packet buffers and the vSwitch has sent 16 packets toward the CNF/VNF).

B. Implementation of IOVTee

As shown in Fig. 3 (b), IOVTee redirects the *pointers* toward empty packet buffers for DMA via PRing such that memory addresses in the guest region[3] are set as DMA destinations (*Rx Queue Mapping*). This redirection enables received packets to be directly stored in the shared memory region offered by the guest, and both the vSwitch and the CNF/VNF handle same packet buffers on the Rx-path (*Virtual DMA Write-through*).

[2]We use the term "vhost-user" to refer to data plane communication via VRings in this article.

[3]The host can orthodoxly get these memory addresses via *avail* and *desc* rings of the VRing (Rx).

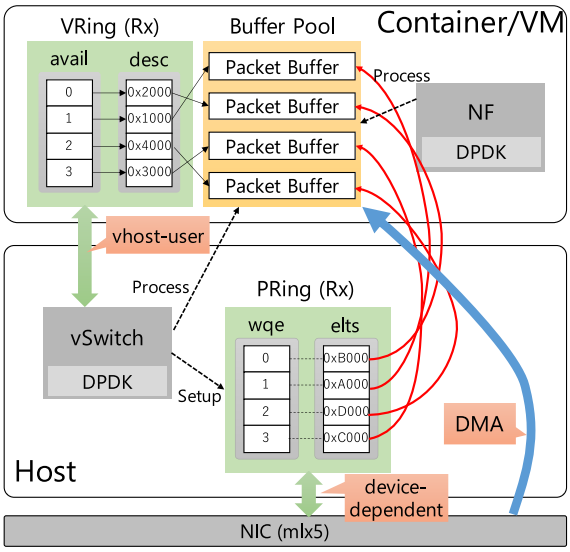


Fig. 5. Details of *Rx Queue Mapping*.

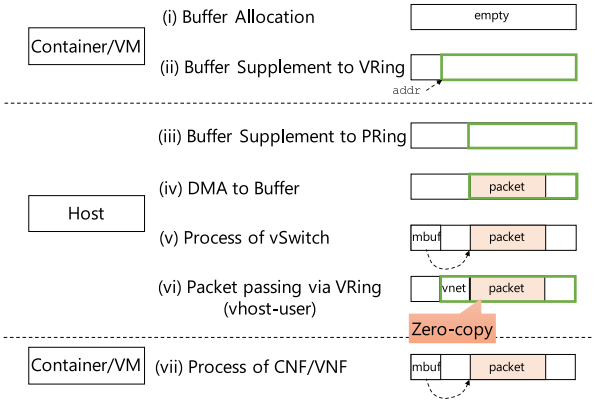


Fig. 6. Packet buffer usage (Rx path). (The same packet buffer is used for i to vii)

Figure 5 illustrates the details of *Rx Queue Mapping*. IOVTee works on the host side and looks into the VRing (Rx) to acquire memory addresses of empty packet buffers as DMA destination addresses. Physical NIC drivers including *mlx5* have a similar ring structure to VRing (e.g., *wqe* and *elts* correspond to *avail* and *desc* respectively), and therefore, IOVTee can logically one-to-one map each entry of PRing to VRing. Then, IOVTee provides adequately transformed physical addresses of packet buffers (in the guest region) to each entry of *elts* that contains a DMA destination address. As a result, inbound packet data are directly stored into the packet buffers.

Next, we explain how packet buffers are handled without involving the packet copy on the Rx-path. Figure 6 depicts a transition of a buffer content. A physically consecutive memory buffer is created and supplied to VRing at the guest side as a normal procedure of DPDK/vhost-user (i and ii). IOVTee acquires the memory address of the buffer via the VRing, and then provides its physical address to the PRing (iii). A received packet is DMAed to the buffer (iv), and the driver sets up an *MBuf* header of DPDK at the beginning of the buffer. Note that the CNF/VNF does not notice the existence of packet data in the buffer because the *used* ring has not been updated. The MBuf-formed packet buffer is passed to the virtual switch as the result of the DPDK's *polling* function, and the switch can arbitrarily process the packet (v). Then, the switch forwards the packet to the CNF/VNF under the context of vhost-user (The *used* ring is updated) (vi). Here, memory copy operations for packet data can be eliminated (zero-copy) because source and destination addresses are the same. At last, a *virtio-net* driver of the guest sets up an *MBuf* header and the CNF/VNF can handle the packet (vii).

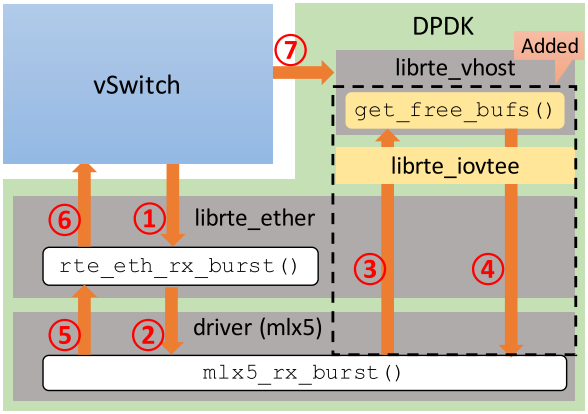


Fig. 7. A call sequence of packet reception at the host.

Figure 7 expresses a function call sequence of packet reception at the host. Suppose that the `elts` array has already pointed to packet buffers in a container/VM. (1) the virtual switch calls `rte_eth_rx_burst` function for polling the Rx queue. (2) The device-dependent receive function (`mlx5_rx_burst`) is invoked, and an `MBuf` header is arranged for each packet data DMAed into memory region of the guest. (3)(4) The newly added `get_free_bufs` function is called to obtain memory addresses of empty packet buffers from the container/VM to fill the Rx queue of the host for subsequent packet reception. The *librte_iovtee* library works as a glue interface between *mlx5* and *librte_vhost*. (5) The received packets have already been stored in (previously acquired) packet buffers, and are returned back to the caller as a set of MBufs. (6) The virtual switch receives the stuffed MBufs, and finally, informs CNFs/VNFs of packet arrival using vhost-user (7). Phase 3 and 4 are the additional steps for `mlx5_rx_burst`, but the original acquiring process of empty packet buffers in *librte_vhost* (for copying transferring packets) is bypassed instead. Therefore, our implementation does not add extra processing overhead to the Rx-path. We applied various optimization techniques for cache utilization [47] to enhance the performance of our previous implementation [21]. We evaluate the performance of our new implementation of IOVTee in the next section.

C. Supporting Multiple NFs

There are two types of scenarios of involving multiple NFs, *independent* and *service functions chaining*. In the former case, the Rx queue (VRing) of each NF has to be mapped to a different Rx queue (PRing) because our Rx Queue Mapping requires

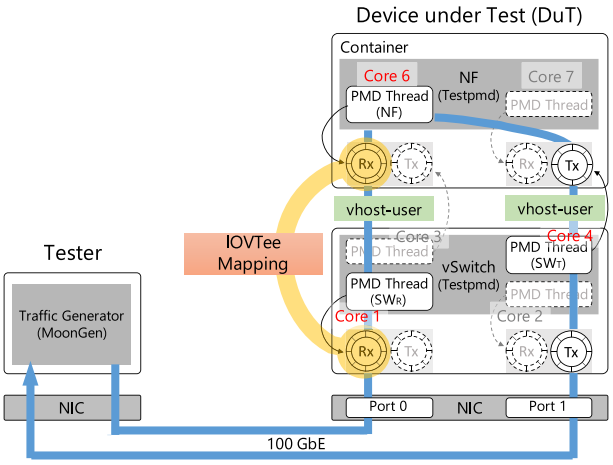


Fig. 8. Evaluation environment.

exclusive one-to-one association. As a result, performance of each PVP datapath does not affect each other. For common service function chaining, the additional vhost-user communications can decrease throughput even though one packet copy is to be avoided.[4] The SNF-based approach is more preferable for extremely high-performance service functions chaining with IOVTee because additional vhost-user communications are avoided.

Receive Side Scaling (RSS) and multi-queue features will be used in real situation to fully use hardware resources. In the nature of DPDK, increasing the number of queues results in additions of datapaths in a Lock-Free Multi-Threading or embarrassingly parallel way. IOVTee can be applied to each datapath independently, and therefore, entire performance of the NFV-node can scale out.

V. Evaluation

In this section, we evaluate the effect of IOVTee on packet forwarding performance of a standard NFV-node, with comparing to the original (copy-full) vhost-user implementation in the PVP datapath. We performed two-round experiments to analyze performance of both IOVTee's zero-copy nature and fundamental vhost-user's communication mechanism. In the first round, we investigated how Rx/Tx ring sizes affected the forwarding efficiency. Our previous study [21] revealed that the ring sizes have measurable impact on the performance. Narrowing the range of the ring sizes is necessary before precise performance analysis because four rings are involved in the datapath (physical and virtual Rx/Tx rings. In the second round, we precisely analyzed actual effect of zero-copy at virtual network I/O and performance characteristics of *split* and *packed* Virtqueues by unveiling CPU caching behaviors during packet forwarding in the NFV-node.

Evaluation environment is shown in Fig. 8. We used two physical servers that were back-to-back connected with 100 GbE links. The right-hand machine was a device under test (DuT) where an IOVTee-supported containerized NF ran on, and the other was a tester node where a traffic generator (MoonGen [48]) was executed on. Table I gives the machine specification of the NFV-node (DuT). In the experiments, MoonGen on the Tester node generated fixed-rate UDP traffic (single flow) for 60 seconds, and average throughput of the received traffic was derived. On the DuT node, the single containerized NF was running, and the virtual switch in the host forwarded incoming/outgoing packets to/from the container. We adopted Testpmd [49], a DPDK sample application that can be used as a lightweight packet forwarder, as both the virtual switch and the NF to avoid becoming performance bottleneck in the datapath.

TABLE I
Machine Specifications

Device under Test (DuT)	
Hardware	
CPU	Intel Core i9-7940X 3.1 GHz (14 cores w/o HT)
Memory	32 GB (DDR4-2666)
NIC	Mellanox ConnectX-5 Ex EN (MCX516A-CDAT)
Software	
vSwitch	Testpmd (DPDK 19.11) (Pinned cores: 1,2,3,4)
OS	CentOS 7.7 (kernel-3.10.0-1062.9.1.x86_64)
Network Driver	mlx5 (RSS: off)
Container	
Network Function	Testpmd (DPDK 19.11) (Pinned cores: 6,7)
OS	CentOS 8.0
Network Driver	virtio-user (multiqueue: off)

Multiple PMD threads (exclusively assigned to different CPU cores) were used to poll each queue each as shown in Fig. 8. Specifically, the virtual switch created four PMD threads (core 1-4) and the NF created two PMD threads (core 6-7), but only three PMD threads (core 1, 4, and 6) were involved in the packet forwarding.[5] IOVTee was applied to the Rx-path corresponding to Port 0 in Bridge mode because the switching overhead of Testpmd was negligible.

We summarize updates of the evaluation scenarios from the previous ones [21] as follows:

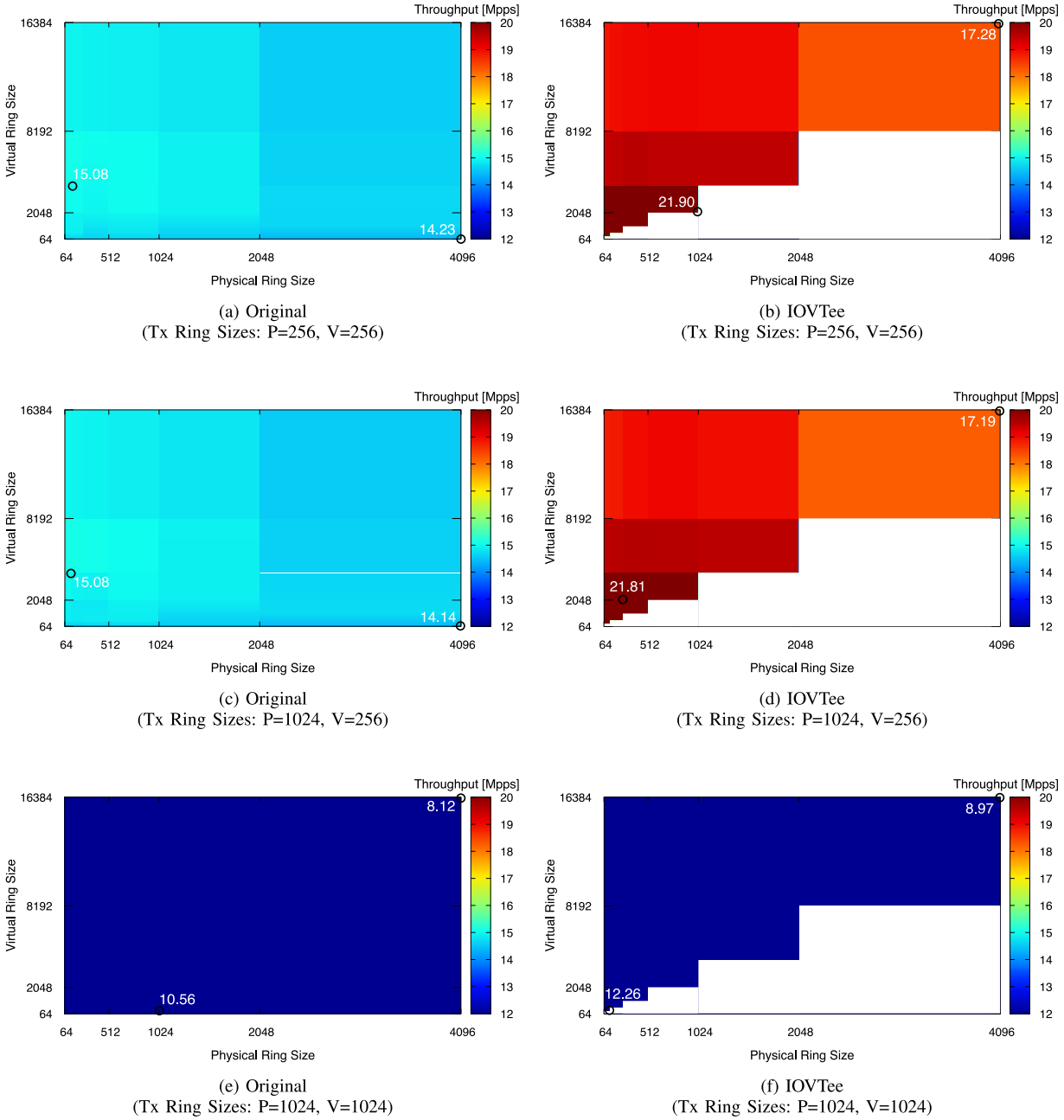
- NF-forms was changed from VM to container
- Tx Ring sizes were also varied
- 1514-byte packets were also examined for ring size effect
- Effect of CPU caching was quantitatively analyzed
- *Packed* Virtqueue was evaluated
- OpenNetVM [17] was evaluated

A. Round-1: Determining Effective Range of Ring Sizes

In the first experiment, we evaluate a relationship between throughput and sizes of four Rx/Tx rings to understand effective sizes in performance for the later experiments. We

[4]This mapping feature for an arbitrary pair of VRings is under development.

[5]The other three PMD threads (core 2, 3, and 7) kept empty polling during the experiment.



(a) Original
(Tx Ring Sizes: P=256, V=256)

(b) IOVTee
(Tx Ring Sizes: P=256, V=256)

(c) Original
(Tx Ring Sizes: P=1024, V=256)

(d) IOVTee
(Tx Ring Sizes: P=1024, V=256)

(e) Original
(Tx Ring Sizes: P=1024, V=1024)

(f) IOVTee
(Tx Ring Sizes: P=1024, V=1024)

Fig. 9. Rx Ring Sizes vs. Throughput (Packet size: 64 bytes).

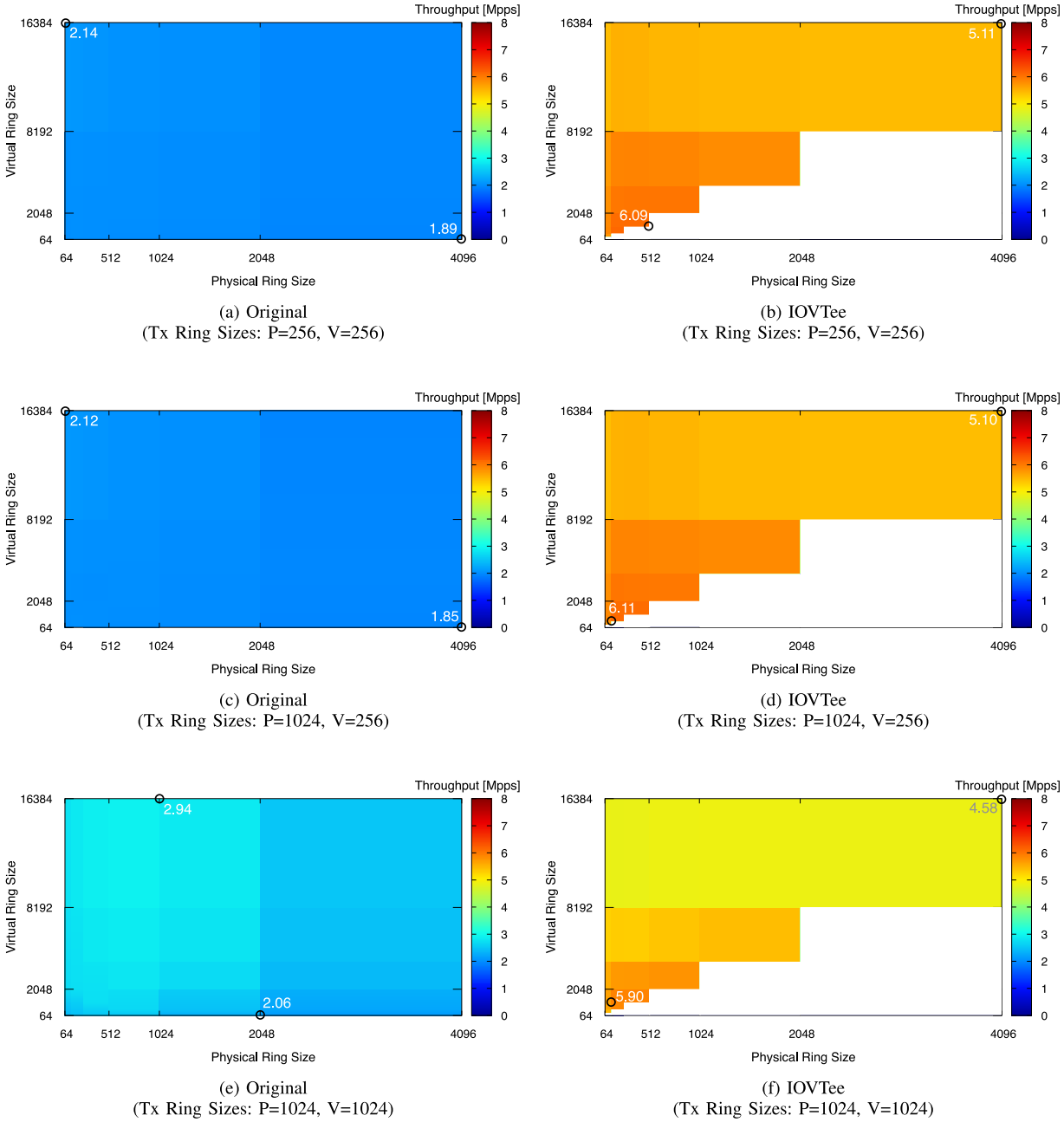
TABLE II
EXAMINED RING SIZES

Physical	Rx	64, 128, 256, 512, 1024, 2048, 4096, 8192
	Tx	256, 1024, 4096
Virtual	Rx	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384
	Tx	256, 1024, 4096

examined every combination of ring sizes in the experiment as in Table II for both 64-byte and 1514-byte packet traffic. The patterns of Tx ring sizes were fewer than that of Rx ring sizes to prevent combinational explosion of the measurements. This limitation does not affect the later experiments because this experiment showed that proper Tx ring sizes can be easily chosen regardless of implementations.

1) Results of 64-Byte Packet Traffic: We show the results of the experiment for 64-byte packet traffic in Fig. 9. The three figures of the left-hand side ((a), (c), and (e)) represent the results of the default NFV-node using the original copy-full vhost-user (packet copy occurred on both the Rx/Tx paths), and the other side of figures ((b), (d), and (f)) show that of IOVTee-enabled one (packet copy occurred on the Tx path only). The upper figures ((a) and (b)) show the effect of varying Rx ring sizes on throughput with fixing the Tx ring sizes (Physical: 256, Virtual: 256). The remaining figures represent the effect in a similar way with different Tx ring sizes.

In terms of *Original*, its throughput is steady (about 14–15 Mpps) regardless of Rx ring sizes (Physical/Virtual) and Tx ring size (Physical) as long as Tx ring size (Virtual)



(a) Original
(Tx Ring Sizes: P=256, V=256)

(b) IOVTee
(Tx Ring Sizes: P=256, V=256)

(c) Original
(Tx Ring Sizes: P=1024, V=256)

(d) IOVTee
(Tx Ring Sizes: P=1024, V=256)

(e) Original
(Tx Ring Sizes: P=1024, V=1024)

(f) IOVTee
(Tx Ring Sizes: P=1024, V=1024)

Fig. 10. Rx Ring Sizes vs. Throughput (Packet size: 1514 bytes).

is 256. More precisely, slightly higher throughput can be seen when Rx ring size (Physical) is in a range of 64–1024 and Virtual one is in a range of 512–8192. Surprisingly throughput becomes totally lower (8–10 Mpps) when Tx ring size (Virtual) is enlarged from 256 to 1024 as shown in (c).

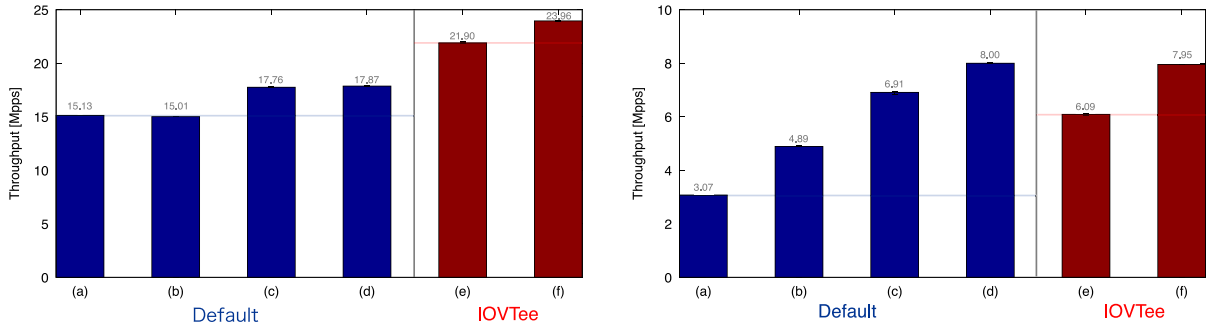
On the other hand, *IOVTee* recorded higher performance at every measurement point.[6] In the figures (b) and (d), its throughput is visibly higher (about 18–22 Mpps) than that of *Original*, while IOVTee is more sensitive to Rx ring sizes. Most effective Rx ring size (Physical) is in a range of 64–1024 and Virtual one is in a range of 128–2048. The overall performance degradation can be seen like *Original*, when Tx ring size (Virtual) is 1024. We will further discuss this phenomenon to identify the cause in Section V-B3.

According to the results, we can understand the picture of the effect of ring sizes on throughput as follows:

- Tx ring size (Physical) has little impact on performance.
- Increasing Tx ring size (Virtual) from 256 severely degrades overall performance.
- Better performance can be gained when Rx ring size (Physical) is in a range of 64-1024.
- Effective Rx ring size (Virtual) differs for each implementation.

Note that we omit the figures of the cases where Tx ring size is 4096 for the space because we have gotten similar results to the case where the ring size was 1024.

[6]Throughputs were not measured (values were set to 0) when the following condition was not satisfied due to the nature of IOVTee design:
Rx ring size (*Physical*) $<$ *Rx ring size* (*Virtual*).



(a) Packet Size: 64 bytes

(b) Packet Size: 1514 bytes

Fig. 11. Throughput of various combinations of Rx/Tx implementations.

2) Results of 1514-Byte Packet Traffic: Figure 10 show the results of the effect of ring sizes on performance for 1514-byte packet traffic. The results are presented in the same way as in Fig. 9. For *Original*, its throughput is highly stable again (about 1.8–2.1 Mpps) regardless of Rx ring sizes when Tx ring size (Virtual) is 256. Enlarging the size of the virtual Tx ring (1024) slightly enhances entire throughput (about 2.0–2.9 Mpps) contrary to the previous experiment. In addition, the effective range of Rx ring size (Virtual) is also enlarged (4096–16384) from the result for 64-byte packet traffic.

IOVTee indicates much higher performance when Tx ring size (Virtual) is 256 (about 5.0–6.1 Mpps), but unlike *Original*, the performance enhancement due to the enlarged ring size cannot be seen (about 4.6–5.9 Mpps). Effective range of Rx ring size (Virtual) is 512–4096 which is also enlarged from the previous result.

B. Round-2: Analyzing Performance Characteristics

In the next round, we analyze the fundamental performance characteristics of vhost-user as well as the actual effect of IOVTee on performance. Specifically, we analyze following three concerns, (i) performance effect of IOVTee's zero-copy Rx-path and copy-full Tx-path, (ii) maximum gain of zero-copying in vhost-user, and (iii) effect of cache-aware implementation of Packed Virtqueue, in this evaluation. From the following experiments, we set Tx ring sizes to 1024 (Physical) and 256 (Virtual) respectively if not specified.

1) Effect of Zero-Copy and Maximum Gain: In this experiment, we evaluate the actual effect of zero-copy vhost-user on performance with assessing the overhead of IOVTee. We measured throughput of various patterns of Rx or Tx-path implementations as described in Table III. Pattern (a) denotes the original copy-full implementation of vhost-user and pattern (e) denotes the case where IOVTee was enabled. In addition, pattern (b), (c), (d), and (f) were examined to understand the degree of packet copy overhead.[7] We chose most effective Rx ring sizes for each pattern in the experiment. Note that the actual zero-copy feature of vhost-user for the Tx-path (*dequeue-zero-copy*) was not examined unlike our previous paper [21] because its implementation caused memory-related errors for the examined version of mlx5 driver (v19.11).

[7]*Fake Zero Copy*: Every `rte_memcpy` function for copying packet data in vhost-user implementation is forcibly skipped (but actual length of each packet was notified to the counterpart via the *used* ring), and therefore, packet forwarding can be continued even though each packet data became dummy.

TABLE III
PACKET COPY PATTERNS

Pattern	Packet Copy		Ring Size (Rx)	
	Rx	Tx	Physical	Virtual
(a)	Copy	Copy	512	4096
(b)	Copy	Fake Zero Copy		
(c)	Fake Zero Copy	Copy		
(d)	Fake Zero Copy	Fake Zero Copy		
(e)	Zero Copy (IOVTee)	Copy	1024	2048
(f)	Zero Copy (IOVTee)	Fake Zero Copy	128	1024

Figure 11 shows throughput of each Rx/Tx pattern for both 64-byte and 1514-byte packet traffic. First, removing packet copy in the Rx-path (*Zero-Copy* of IOVTee and *Fake-Zero-Copy*) is demonstrably more effective than in the Tx-path, which can be a proof of reasonableness of IOVTee. This characteristic can be explained as follow. In the common PVP datapath as illustrated in Fig. 8, the Rx-path of vhost-user becomes the performance bottleneck and lots of received packets are actually dropped at the point. Hence, removing packet copy in the Tx-path does not prevent these packets from being dropped. The effect of zero-copy in the Tx-path appears when the packet loss is alleviated in the Rx-path, and there can be such situations; forwarding efficiency on the Rx-path is highly improved (the case of IOVTee) or the number of incoming packets is low (the case of 1514-byte packet traffic). Second, *Fake Zero Copy* for the Rx-path (c) shows lower throughput than that of IOVTee (e) for 64-byte packet traffic. This is because not only our implementation of IOVTee does not add extra packet processing cost, but also static mapping of the Rx queues enables certain optimization of packet transfer in vhost-user. Third, the effect of zero-copy on performance becomes larger as increasing packet size. Finally, entirely removing packet copy in both Rx/Tx paths (d) does not dramatically boost up the throughput for 64-byte packet traffic. This implies that manipulations of VRings involve measurable performance overhead, and therefore, optimization of the way of virtqueue handling is indispensable for further performance increase.

2) Effect of Cache-Aware Virtqueue Manipulations: Next, we evaluate the effect of *Packed* Virtqueue on throughput with varying packet sizes. We examined following packet sizes

TABLE IV
L1 CACHE MISS LATENCY [NS]

PRing (Rx:Tx) (512:1024) Pattern	Thread	VRing (Rx:Tx)					
		64 bytes			1514 bytes		
		1024:256 (a)	8192:256 (b)	1024:1024 (c)	1024:256 (d)	8192:256 (e)	1024:1024 (f)
Default (Split)	SW_R	5.90	5.29	9.39	8.70	8.88	10.44
	SW_T	6.98	6.96	7.41	28.30	31.92	23.41
	NF	3.50	6.22	10.26	12.28	15.40	19.04
Default (Packed)	SW_R	5.58	4.68	9.30	8.79	8.04	9.86
	SW_T	6.53	7.08	7.61	41.75	39.27	40.03
	NF	3.04	5.39	8.87	15.77	20.49	21.98
IOVTee (Split)	SW_R	4.52	4.34	4.86	6.65	6.40	6.52
	SW_T	3.84	5.69	13.79	11.78	12.43	11.82
	NF	3.40	6.06	12.26	6.48	11.21	6.09
IOVTee (Packed)	SW_R	4.62	4.66	5.09	6.88	6.54	9.18
	SW_T	3.92	6.15	14.00	12.57	13.26	17.81
	NF	2.86	5.46	11.06	4.61	8.07	11.41

(64, 128, 256, 512, 1024, 1280, and 1514 bytes), and also measured throughput of OpenNetVM [17] for comparison. The result is presented in Fig. 12. Each of the three dotted lines indicates maximum theoretical throughput for 10 GbE, 40 GbE, or 100 GbE respectively. The aqua-colored line with no points (*Baremetal*) denotes fundamental packet forwarding efficiency of the virtual switch in our environment, and the values are useful to estimate the performance overhead of vhost-user. In terms of *Default*, the throughputs of both models surpassed the theoretical values of 10 GbE at any packet size, and surprisingly using *Packed* Virtqueue pushed the maximum throughput up to 20 Mpps (for the traffic of 64-byte packets). However, its cache-aware nature showed a negative impact on throughput for larger packet sizes (256–). On the other hand, *Packed* Virtqueue was effective for *IOVTee* at any packet size, and the maximum throughput was 24.66 Mpps when the packet size was 64 bytes. This difference (whether the throughput is increased or decreased) implies that the cache-aware implementation of *Packed* Virtqueue is negatively sensitive to packet copy (memory copy). While *IOVTee* certainly reduces the performance overhead of vhost-user, there is still much performance gap to that of *Baremetal* in regard to short-packet traffic.[8] Additional acceleration mechanisms like packet aggregation [51] are needed to fill the gap, rather than further optimize the way of virtqueue manipulations (*Packed* Virtqueue has been continually optimized so far).

For *OpenNetVM*, the maximum throughput saturated around 9 Mpps for short packet traffic. Disabling a flow lookup feature[9] improved the performance to 13 Mpps, but optimized *OpenNetVM* still underperformed *Default* (64-bytes). However, *OpenNetVM* outperformed even *IOVTee* for larger packet traffic due to the zero-copy effect on the Tx path.

3) Analyzing CPU Caching Behaviors: Finally, we analyze caching behaviors of CPU (*L1*, *L2*, and *L3* caches) during packet forwarding on the PVP datapath to understand the performance characteristics of IOVTee. We used Processor Counter Monitor (PCM) [50] to measure cache-related values, such as L1-cache miss latency (penalty), the number of L2 cache misses, and L3 cache hit ratio.

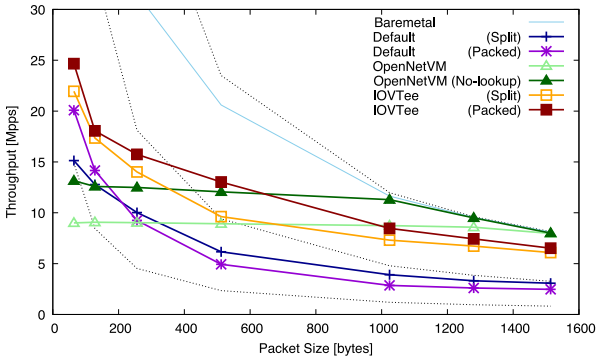


Fig. 12. Effect of *Packed* Virtqueue on throughput.

Table IV shows L1 cache miss latency. The thread SW_R is the PMD thread of the virtual switch (see Fig. 8). The thread forwarded the received packets from port 0 to the container, and SW_T transferred the packets from the container to port 1. *NF* routed the packets in the container. For the pattern (a), the effect of L1 caching was enhanced by using *Packed* Virtqueue for every thread. In addition, *IOVTee* obviously alleviated the cache miss penalty compared to *Default* in each Virtqueue. Increasing Rx ring size (Virtual) (b) slightly worsened the cache usage, especially for *NF*. This indicates that iterating the longer ring descriptors caused more frequent purging of cache lines. Increasing Tx ring size (Virtual) (c) caused far large penalty for every thread, and this was the cause of the severe performance degradation mentioned in Section V-A. Unlike packet reception, packet transferring involves memory accesses to the ring descriptors twice, for enqueuing packets and for discarding packets. Since the later accesses occur after a long time in a batched manner, smaller Tx ring is preferable in terms of effective memory caching. For larger

[8]The maximum rate of *Baremetal* was 38.48 Mpps (64 bytes).

[9]Each received packet is matched against a flow table to determine the first NF on the host side (default action), and this behavior can be disabled by invalidating `FLOW_LOOKUP` macro.

TABLE V
THE NUMBER OF L2-CACHE MISSES (PER 1M PACKETS)

PRing (Rx:Tx) (512:1024) **Pattern**	**Thread**	**VRing (Rx:Tx)**					
		64 bytes			**1514 bytes**		
		1024:256 (a)	8192:256 (b)	1024:1024 (c)	1024:256 (d)	8192:256 (e)	1024:1024 (f)
Default (Split)	SW_R	2.0M	2.0M	1.8M	24M	21M	39M
	SW_T	2.0M	2.0M	2.1M	34M	30M	39M
	NF	0.29M	5.1M	4.6M	3.5M	5.2M	7.0M
Default (Packed)	SW_R	2.4M	1.9M	4.3M	56M	46M	40M
	SW_T	2.4M	2.1M	2.8M	82M	70M	41M
	NF	0.15M	4.7M	5.7M	7.7M	11M	5.0M
IOVTee (Split)	SW_R	2.5M	2.2M	2.3M	2.5M	2.5M	2.4M
	SW_T	0.20M	1.5M	1.8M	29M	25M	27M
	NF	0.19M	4.8M	4.6M	0.35M	4.7M	0.33M
IOVTee (Packed)	SW_R	2.1M	2.2M	2.1M	2.6M	2.7M	3.2M
	SW_T	0.13M	1.4M	1.9M	30M	29M	25M
	NF	0.05M	4.5M	4.1M	0.35M	5.1M	3.4M

TABLE VI
L3-CACHE HIT RATIO

PRing (Rx:Tx) (512:1024) **Pattern**	**Thread**	**VRing (Rx:Tx)**					
		64 bytes			**1514 bytes**		
		1024:256 (a)	8192:256 (b)	1024:1024 (c)	1024:256 (d)	8192:256 (e)	1024:1024 (f)
Default (Split)	SW_R	1.00	1.00	1.00	1.00	0.99	0.97
	SW_T	1.00	1.00	1.00	1.00	1.00	0.29
	NF	0.99	1.00	0.39	1.00	0.96	0.26
Default (Packed)	SW_R	1.00	1.00	1.00	1.00	0.99	0.98
	SW_T	1.00	1.00	1.00	1.00	1.00	0.70
	NF	0.97	1.00	0.39	1.00	0.94	0.26
IOVTee (Split)	SW_R	1.00	1.00	0.93	1.00	0.99	1.00
	SW_T	1.00	1.00	0.99	1.00	0.97	0.98
	NF	0.85	0.99	0.22	0.97	0.92	0.89
IOVTee (Packed)	SW_R	1.00	1.00	0.93	1.00	0.99	0.99
	SW_T	1.00	1.00	0.99	1.00	0.99	0.72
	NF	0.61	0.99	0.20	0.99	0.93	0.06

packet sizes (d)–(f), cache miss penalties were entirely heavier because a lot more cache lines were purged during packet copy. *Default* took the worst penalty when *Packed* Virtqueue was used because of its cache-aware implementation, while *IOVTee* mitigated the purging by avoiding packet copy in the Rx-path.

Table V and VI show the number of L2 cache misses and hit ratio of L3 cache, respectively. These results indicate again that *Packed* Virtqueue was effective when the negative effect of packet copy was minified, and therefore, IOVTee took better advantage of the caching technology. Note that the hit ratio of the *NF* thread for *IOVTee* was apparently lower in pattern (a), because L1 and L2 caches worked well considering the much higher throughput.

VI. RELATED WORK

In this section, we categorize related work that tackled performance enhancement of virtual network I/O into software-based and hardware-based approaches.

A. Software-Based Approaches

1) Big Shared Packet Pool: This type of technology uses packet buffers of the virtual switch as a big packet pool accessed by CNFs/VNFs to achieve zero-copy at the virtual network I/O for efficient service function chaining on the same host. Multiple CNFs or VNFs can communicate each other via the underlying virtual switch without involving packet copy. NetVM [16] and OpenNetVM [17] are representative DPDK-based frameworks for fast service function chaining. They provide a custom NF manager (virtual switch) and an NFLib library (dedicated virtual network driver for NFs) for fine-grained trust-based packet routing among the CNFs/VNFs. OVSPP [18] is an enhancement of Open vSwitch (OVS-DPDK) [24] for carrier-grade service function chaining in CORD [25] systems, and Soft Patch Panel (SPP) [26] plays a key role in communications between the switch and VNFs. SPP supports two types of modes for its virtual network I/O, zero-copy based *ring* and traditional virtio-based *vhost*,

with a view to efficiency and compatibility. IVSHMEM [27] is internally used for realizing the shared memory feature of the *ring* mode.

A big shared packet pool eliminates packet copy at virtual network I/O, and is especially effective for inter-NF communications on the same NFV-node. However, a custom-made mechanism is necessary in both the host and the guests to allow sharing the packet pool between each other. This forces CNFs/VNFs to be host-aware and is undesirable for public cloud environment where lots of users deploy their own network functions on demand.

2) Others: There are other software-based approaches to achieve zero-copy at virtual network I/O. ClickOS [28] realizes a Click [29] like service function chaining framework for virtualized NFs based on the netmap packet I/O framework [30] and the VALE switch [31]. Each NF is realized on top of a thin Xen-compatible VM (*ClickOS*) for agile and flexible chaining. Per-port ring buffers of the switch are mapped to the VM space for optimizing Xen's virtual network I/O, but its performance is underrun copy-full vhost-user [32].

A fully software-based pass-through has been designed using *ptnetmap* [33] or extended *ptnet* [34] on the netmap framework (Netmap Passthrough). Netmap-ready applications in guest VMs are allowed to directly access VALE ports in the host kernel, which eliminates not only packet copy between the host and the guests, but also extra overhead involved in network device emulation. Such a queue mapping approach realizes efficient packet forwarding at virtual packet I/O (over 20 Mpps). While the notion of queue mapping is similar with our IOVTee, our queue mapping is transparent to both CNFs/VNFs and virtual switches thanks to the de-facto standard virtio mechanism.

An RDMA-to-the-VM method has been adopted in vSocket [35] to avoid packet copy in the Rx path. Comprehensive vSocket framework has been provided to apply the RDMA technology to virtualized applications in public clouds, and they receive performance benefits of RDMA via dedicated virtual packet I/O. However, adoption of wide-range of custom system components is required in both the host and the guests in addition to deployment of RDMA framework in the substrate network. On the other hand, IOVTee has been designed to fit into common NFV-nodes on Ethernet composed of de-facto DPDK/vhost-user and variety of virtual switches including OVS-DPDK as well as containerized or virtualized NFs. Furthermore, IOVTee can be applied into the nodes without modifying virtual switches or containers/VMs, thanks to the unmodified vhost-user interface.

ZCopy-Vhost [36] adopts a page swapping technique to achieve zero-copy at virtual packet I/O. Each packet is stored and well-aligned in a dedicated 4 KB memory page, and ZCopy-Vhost modifies Extended Page Table (EPT) to internally swap the pages between the hypervisor and the guests. ZCopy-Vhost requires modifications in both the host including the virtual switch and the guests to hack page handling, and such a swapping of huge number of pages causes frequent TLB flushing and degrades packets-per-second (pps) based throughput to about 4 Mpps, while the effect of zero-copy appears for large packet sizes.

B. Hardware-Based Approaches

Modern CPUs and NICs support a hardware-level I/O virtualization mechanism (e.g., Intel VT-d [37]), and SR-IOV is the de-facto application of this technology. With SR-IOV, VMs or containers are permitted to directly access the physical NIC in a safe way via dedicated virtual functions (VFs) of the virtual NICs. As a result, the communications involving packet copy between the virtual switch and the NFs can be avoided (Hardware Passthrough). However, several studies have reported that throughput of SR-IOV is lower than that of wire-rate of 10 GbE (14.88 Mpps) for 64-byte packets [16], while the effect of packet copy avoidance appears for large packets [21]. Moreover, evaluation results presented in our past papers [7], [21] showed that its performance was unstable and incomprehensible. Hardware pass-through makes difficult of investigation of packet forwarding process of NFV systems. Besides, VMs/containers have to be physical NIC-aware, which causes service continuity problems when they are lively migrated to another NFV-node. Netronome has developed Express Virtio (XVIO) [38] that balances the pass-through feature and VM mobility, but this implementation is only available on their SmartNIC series now. PPAP (Packet Processing Acceleration Platform) [39] uses FPGA to process stateless functions (e.g., table lookup) of arbitrary VNFs, but state-ful VNFs are not supported.

VII. Conclusion

Performance enhancement virtual packet I/O is a critical challenge to boost up packet forwarding efficiency of software-based NFV-nodes consist of commodity servers and containerized networking functions. Vhost-user is now the de-facto virtual packet I/O mechanism, and various enhancement has been applied to its implementation including cache-aware *Packed* Virtqueue. However, vhost-user is still copy-full and packet copy is a major performance concern for performance of NFV-nodes. Various zero-copy/pass-through techniques have been proposed so far; however, they have pragmatic problems in that compatibility, manageability, and degraded quality of performance. In this article, we have proposed a yet another approach named IOVTee that resolves above concerns with achieving remarkable performance. The design/implementation of IOVTee can be characterized by the fully software-based receive (Rx) queue mapping mechanism enabling virtual DMA write-through as explained in Section III and IV. We have evaluated both the effect of both IOVTee on performance and the fundamental performance characteristics of vhost-user from various points of view. The evaluation results have shown that IOVTee pushes up throughput of the PVP datapath about 45% for 64-byte packet traffic and 98% for 1514-byte packet traffic respectively at maximum, compared to the original implementation of vhost-user. For future work, we are planning to integrate an NFV-oriented packet aggregation mechanism (PA-Flow [51]) to IOVTee for boosting up the performance of software-based NFV-nodes. Further, porting the concept of IOVTee to another virtual network I/O (e.g., io_uring [52]) is also under consideration.

REFERENCES

[1] *ETSI—NFV*. Accessed: May 26, 2020. [Online]. Available: https://www.etsi.org/technologies/nfv

[2] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[3] 5G-PPP Architecture Working Group. (Jul. 2016). *View on 5G Architecture, White Paper Version 1.0*. Accessed: May 26, 2020. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-July-2016.pdf

[4] *Kubernetes*. Accessed: May 26, 2020. [Online]. Available: https://kubernetes.io/

[5] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May/Jun. 2018.

[6] 5G-PPP Software Network Working Group. (Jul. 2018). *From Webscale to Telco, the Cloud Native Journey, White Paper, Version 1.0*. Accessed: May 26, 2020. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2018/07/5GPPP-Software-Network-WG-White-Paper-23052018-V5.pdf

[7] R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo, "Evaluation of forwarding efficiency in NFV-nodes toward predictable service chain performance," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 920–933, Dec. 2017.

[8] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, L. Iannone, and J. Roberts, "Comparing the performance of state-of-the-art software switches for NFV," in *Proc. 15th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Orlando, FL, USA, Dec. 2019, pp. 68–81.

[9] R. Leira, G. J.-Moreno, I. González, F. J. Gómez-Arribas, and J. E. López de Vergara, "Performance assessment of 40 Gbit/s off-the-shelf network cards for virtual network probes in 5G networks," *Comput. Netw.*, vol. 152, pp. 133–143, Apr. 2019.

[10] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, Savannah, GA, USA, Nov. 2016, pp. 203–216.

[11] *Intel Open Network Platform Server Release 1.5 Performance Test Report*, Intel, Santa Clara, CA, USA, Oct. 2015.

[12] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[13] *P4*. Accessed: May 26, 2020. [Online]. Available: https://p4.org/

[14] *Open Compute Project*. Accessed: May 26, 2020. [Online]. Available: https://www.opencompute.org/

[15] *DPDK*. Accessed: May 26, 2020. [Online]. Available: https://www.dpdk.org/

[16] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 34–47, Mar. 2015.

[17] W. Zhang *et al.*, "OpenNetVM: A platform for high performance network service chains," in *Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Funct. Virtualization (HotMiddlebox)*, Florianópolis, Brazil, Aug. 2016, pp. 26–31.

[18] T. Trinh and D. Hara, *OVSPP: Hybrid Enhanced vSwitch for CORD-Based Telco SFC*, ONF Connect, Santa Clara, CA, USA, 2019. Accessed: May 26, 2020. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2019/09/2.30pm-Tri-Trinh-and-Daisuke-Hara-OVSPP-Hybrid-Enhanced-vSwitch-for-CORD-Based-Telco-SFC.pdf

[19] PCI-SIG. (Jan. 2010). *Single Root I/O Virtualization and Sharing Specification Revision 1.1*. Accessed: May 26, 2020. [Online]. Available: https://pcisig.com/single-root-io-virtualization-and-sharing-specification-revision-11

[20] AMD. (Dec. 2016). *AMD I/O Virtualization Technology (IOMMU) Specification*. Accessed: May 26, 2020. [Online]. Available: http://developer.amd.com/wordpress/media/2013/12/48882_IOMMU.pdf

[21] R. Kawashima and H. Matsuo, "IOVTee: A fast and pragmatic software-based zero-copy/pass-through mechanism for NFV-nodes," in *Proc. 4th IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, Verona, Italy, Nov. 2018, pp. 1–6.

[22] *Vhost-User Feature for QEMU*. Accessed: May 26, 2020. [Online]. Available: http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/

[23] OASIS. (Apr. 2019). *Virtual I/O Device (VIRTIO) Version 1.1, Committee Specification*. Accessed: May 26, 2020. [Online]. Available: https://docs.oasis-open.org/virtio/virtio/v1.1/cs01/virtio-v1.1-cs01.pdf

[24] *Open vSwitch*. Accessed: May 26, 2020. [Online]. Available: http://www.openvswitch.org/

[25] L. Peterson , "Central office re-architected as a data center," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[26] T. Nakamura, Y. Ogawa, N. Takada, and H. Nakamura, "MAGONIA (Soft Patch Panel): High-speed inter-function technique," *NTT Tech. Rev.*, vol. 14, no. 10, pp. 1–4, 2016.

[27] *QEMU*. Accessed: May 26, 2020. [Online]. Available: https://github.com/qemu/qemu/blob/master/docs/specs/ivshmem-spec.txt

[28] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Seattle, WA, USA, Apr. 2014, pp. 459–473.

[29] E. Kohler, R. T. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[30] L. Rizzo, M. Carbone, and G. Gatalli, "Transparent acceleration of software packet forwarding using netmap," in *Proc. IEEE INFOCOM*, 2012, pp. 2471–2479.

[31] L. Rizzo and G. Lettieri, "VALE, a switched Ethernet for virtual machines," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 61–72.

[32] Z. Niu, X. Xu, L. Liu, Y. Tian, P. Wang, and Z. Li, "Unveiling performance of NFV software dataplanes," in *Proc. 2nd Workshop Cloud Assist. Netw. (CAN)*, Incheon, Republic of Korea, Dec. 2017, pp. 13–18.

[33] V. Maffione, L. Rizzo, and G. Lettieri, "Flexible virtual machine networking using netmap passthrough," in *Proc. 23rd IEEE Int. Symp. Local Metropolitan Area Netw. (LANMAN)*, Osaka, Japan, Jun. 2016, pp. 1–6.

[34] S. Garzarella, G. Lettieri, and L. Rizzo, "Virtual device passthrough for high speed VM networking," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oakland, CA, USA, May 2015, pp. 99–110.

[35] D. Wang, B. Fu, G. Lu, K. Tan, and B. Hua, "vSocket: Virtual socket interface for RDMA in public clouds," in *Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ. (VEE)*, Providence, RI, USA, Apr. 2019, pp. 179–192.

[36] D. Wang and B. Hua, "ZCopy-Vhost: Replacing data copy with page remapping in virtual packet I/O," *IEEE Access*, vol. 7, pp. 51047–51057, 2019.

[37] Intel. (Jun. 2016). *Intel Virtualization Technology for Directed I/O*. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf

[38] *Express Virtio (XVIO) Technology*. Accessed: May 26, 2020. [Online]. Available: https://www.netronome.com/solutions/xvio/overview/

[39] T. Lan, Q. Han, H. Fan, and J. Lan, "FPGA-based packets processing acceleration platform for VNF," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, Nov. 2017, pp. 314–317.

[40] *OPNFV*. Accessed: May 26, 2020. [Online]. Available: https://www.opnfv.org/

[41] S. Natarajan, R. Krishnan, A. Ghanwani, D. Krishnaswamy, P. Willis, and A. Chaudhary, "An analysis of container-based platforms for NFV," Internet Eng. Task Force, Fremont, CA, USA, Internet-Draft IETF-94 NFVRG, Nov. 2015. Accessed: May 26, 2020. [Online]. Available: https://www.ietf.org/proceedings/94/slides/slides-94-nfvrg-11.pdf

[42] J. Tan, C. Liang *et al.*, "VIRTIO-USER: A new versatile channel for kernel-bypass networks," in *Proc. ACM SIGCOMM Workshop Kernel Bypass Netw. (KBNets)*, Los Angeles, CA, USA, Aug. 2017, pp. 13–18.

[43] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2015-155, 2015. Accessed: May 26, 2020. [Online]. Available: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-155.html

[44] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, *Scalable, High-Performance, Elastic Software OpenFlow Switch in Userspace for Wide-Area Network*, Open Networking Summit (ONS), Santa Clara, CA, USA, Mar. 2014.

[45] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire, Jr., "Metron: NFV service chains at the true speed of the underlying hardware," in *Proc. 15th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Renton, WA, USA, Apr. 2018, pp. 171–186.

[46] G. P. Katsikas, M. Enguehard, M. Kuźniar, G. Q. Maguire, and D. Kostić, "SNF: Synthesizing high performance NFV service chains," *PeerJ Comput. Sci.*, vol. 2, p. e98, Nov. 2016.
[47] D. Barach, L. Linguaglossa, D. Marion, P. Pfister, S. Pontarelli, and D. Rossi, "High-speed software data plane via vectorized packet processing," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 97–103, Dec. 2018.
[48] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A scriptable high-speed packet generator," in *Proc. ACM Conf. Internet Meas. Conf. (IMC)*, 2015, pp. 275–287.
[49] *Testpmd Application User Guide*. Accessed: May 26, 2020. [Online]. Available: https://doc.dpdk.org/guides/testpmd_app_ug/index.html
[50] *Processor Counter Monitor*. Accessed: May 26, 2020. [Online]. Available: https://github.com/opcm/pcm
[51] Y. Taguchi, R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo, "Fast datapath processing based on hop-by-hop packet aggregation for service function chaining," *IEICE Trans. Inf. Syst.*, vol. E102-D, no. 11, pp. 2184–2194, 2019.
[52] *Efficient IO with io_uring*. Accessed: Oct. 28, 2020. [Online]. Available: https://kernel.dk/io_uring.pdf

Ryota Kawashima (Member, IEEE) was born in 1983. He received the M.S. degree from Iwate Prefectural University in 2007, and the Ph.D. degree from the Graduate University for Advanced Studies (SOKENDAI) in 2010. He has worked as a Software Engineer with ACCESS Company Ltd. and Stratosphere, Inc. He became an Assistant Professor with the Nagoya Institute of Technology in 2013, and an Associate Professor in 2020. His research interest is high-performance system softwares for SDN and NFV. He received the Best Paper Awards for 2016 IEICE Communications Society and IEEE NFV-SDN 2018.