Workgroup:
Benchmarking Methodology Working Group
Internet-Draft:
draft-dcn-bmwg-containerized-infra-10
Published: 12 March 2023
Intended Status: Informational
Expires: 13 September 2023
Authors: N. Tran
Soongsil University
S. Rao
The Linux Foundation
J. Lee
Soongsil University
Y. Kim
Soongsil University

Considerations for Benchmarking Network Performance in Containerized Infrastructures

Abstract

Recently, the Benchmarking Methodology Working Group has extended the laboratory characterization from physical network functions (PNFs) to virtual network functions (VNFs). Considering the network function implementation trend moving from virtual machine-based to container-based, system configurations and deployment scenarios for benchmarking will be partially changed by how the resource allocation and network technologies are specified for containerized VNFs. This draft describes additional considerations for benchmarking network performance when network functions are containerized and performed in general-purpose hardware.

Table of Contents

1. Introduction

The Benchmarking Methodology Working Group(BMWG) has recently expanded its benchmarking scope from Physical Network Function(PNF) running on a dedicated hardware system to Network Function Virtualization(NFV) infrastructure and Virtualized Network Function(VNF). [RFC8172] described considerations for configuring NFV infrastructure and benchmarking metrics, and [RFC8204] gives guidelines for benchmarking virtual switch which connects VNFs in Open Platform for NFV(OPNFV).

Recently NFV infrastructure has evolved to include a lightweight virtualized platform called the containerized infrastructure, where network functions are virtualized by using the host operating system (OS) virtualization instead of hardware virtualization in virtual machine (VM)-based infrastructure based on the hypervisor. In comparison to VMs, containers do not have a separate hardware and kernel. Containerized virtual network functions (C-VNF) share the same kernel space on the same host, while their resources are logically isolated in different namespaces. Considering this architecture difference between container-based and virtual-machine based NFV systems, containerized NFV network performance benchmarking might have different System Under Test(SUT) and Device Under Test(DUT) configurations compared with both black-box benchmarking and VM-based NFV infrastructure as described in [RFC8172].

In terms of networking, to route traffic between containers which are isolated in different network namespaces, virtual ethernet (vETH) interface pairs are used to create a tunnel to Linux bridge or virtual switch (vSwitch) instead of TAP virtual networking device in VM case. Besides, containerized network performance is also affected by multiple different packet acceleration techniques which have been applied recently in containerized infrastructure to achieve high throughput and line-rate transmission speed. Each kind of acceleration technique has different deployment location and usage of vSwitch, which is an important aspect of the NFV infrastructure as stated in [RFC8204]. Therefore, different networking models considerations based on the usage characteristic

of vSwitch in containerized infrastructure should be noticed while benchmarking containerized network performance.

This draft aims to provide additional considerations as specifications to guide containerized infrastructure benchmarking compared with the previous benchmarking methodology of common NFV infrastructure. These considerations include investigation of multiple networking models based on the usage of vSwitch in different packet acceleration techniques, and investigation of several resources configurations that might impact on containerized network performance such as CPU isolation, hugepages, CPU cores and memory allocation, service function chaining. The benchmark experiences of these mentioned considerations are also presented in this draft as references. Note that, although the detailed configurations of both infrastructures differ, the new benchmarks and metrics defined in [RFC8172] and [RFC8204] can be equally applied in containerized infrastructure from a generic-NFV point of view, and therefore defining additional evaluation metrics or methodologies are out of scope.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document is to be interpreted as described in [RFC2119]. This document uses the terminology described in [RFC8172], [RFC8204], [ETSI-TST-009].

3. Containerized Infrastructure Overview

With the proliferation of Kubernetes, in a common containerized infrastructure, pod is defined as a basic unit for orchestration and management that can host multiple containers, with shared storage and network resources. Kubernetes supports several run-time options for containers such as Docker, CRI-O and containerd. In this document, the terms container and pod are used interchangeably.

For benchmarking of the containerized infrastructure, as mentioned in [RFC8172], the basic approach is to reuse existing benchmarking methods developed within the BMWG. Various network function specifications defined in BMWG should still be applied to containerized VNF(C-VNF)s for the performance comparison with physical network functions and VM-based VNFs. A major distinction of the containerized infrastructure from the VM-based infrastructure is the absence of a hypervisor. Without hypervisor, all C- VNFs share the same host and kernel space. Storage, computing, and networking resources are logically isolated between containers via different namespaces.

Container networking is provided by Container Network Plugins (CNI). CNI creates the network link between containers and host's external (real) interfaces. Different kinds of CNI leverage different networking technologies and solutions to create this link. These include bringing host network device into container namespace, or creating vETH pairs with one side attached to container network namespace and the other attached to the host network namespace, either direct point-to-point, or via a bridge/switching function (Linux bridge, MACVLAN/IPVLAN sub-interfaces, kernel-space or user-space switch). SRIOV and eBPF are other available options. The architectural differences of these CNIs bring additional considerations when benchmarking network performance in containerized infrastructure.

4. Benchmarking Considerations

4.1. Networking Models

Container networking services in Kubernetes are provided by CNI plugins which describe network configuration in JSON format. Initially, when a pod or container is first instantiated, it has no network. CNI plugins insert a network interface into the isolated container network namespace, and performs other necessary tasks to connect the host and container network namespaces. It then allocates IP address to the interface, configures routing consistent with the IP address management plugin. Different CNIs use different networking technologies to implement this connection. Based on the chosen networking technologies, and how the packet is processed/ accelerated via the kernel-space and/or the user-space of the host, these CNIs can be categorized into different container networking models. The usage of each networking model and its corresponding CNIs can affect the container networking performance.

4.1.1. Kernel-space non-Acceleration Model

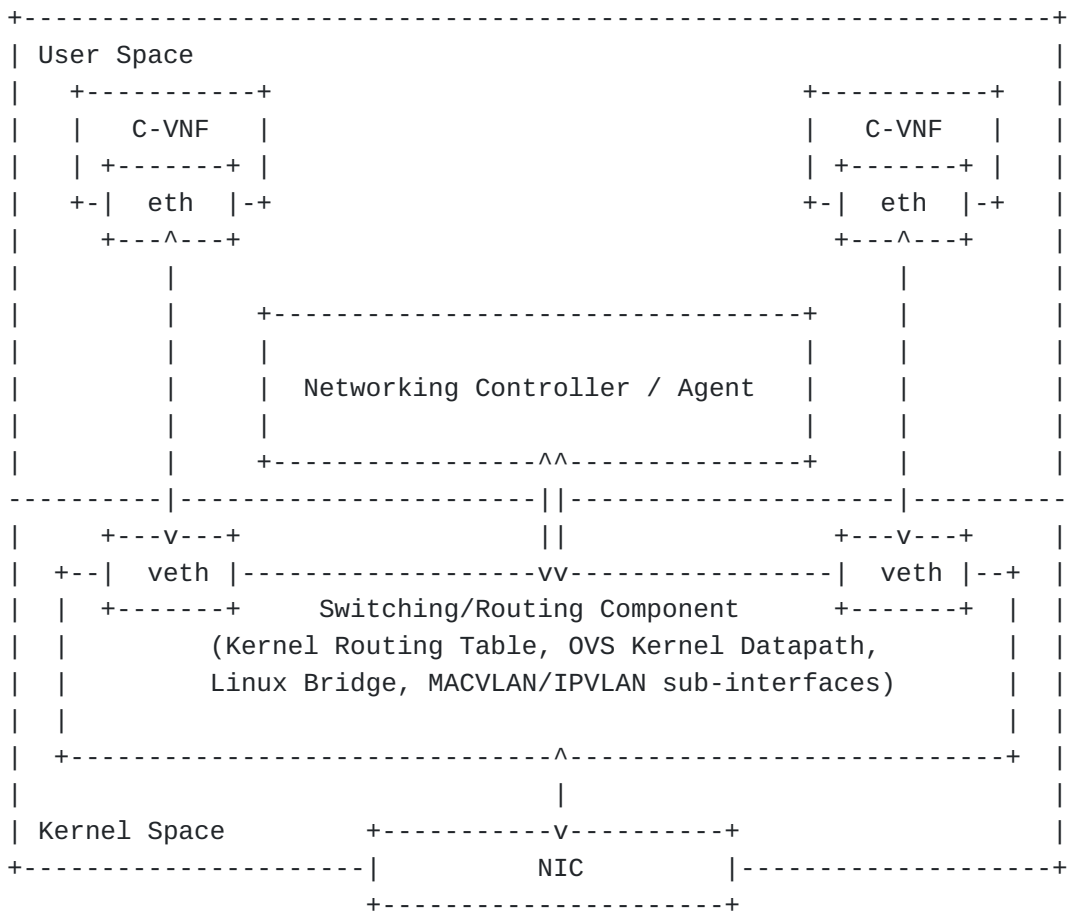

Figure 1: Example architecture of the Kernel-Space non-Acceleration Model

Figure 1 shows kernel-space non-Acceleration model. In this model, the vETH interface on the host side can be attached to different switching/routing components based on the chosen CNI. In the case of Calico, it is the direct point-to-point attachment to the host namespace then using Kernel routing table for routing between containers. For Flannel, it is the Linux Bridge. In the case of MACVLAN/IPVLAN, it is the corresponding virtual sub-interfaces. For dynamic networking configuration, the Forwarding policy can be pushed by the controller/agent located in the user-space. In the case of Open vSwitch (OVS) [OVS], configured with Kernel Datapath, the first packet of the 'non-matching' flow can be sent to the user space networking controller/agent (ovs-switchd) for dynamic forwarding decision.

In general, the switching/routing component is running on kernel space, data packets should be processed in-network stack of host kernel before transferring packets to the C-VNF running in user-space. Not only pod-to-External but also pod-to-pod traffic should be processed in the kernel space. This design makes networking

performance worse than other networking models which utilize packet acceleration techniques described in below sections. Kernel-space vSwitch models are listed below:

o Docker Network[Docker-network], Flannel Network[Flannel], Calico [Calico], OVS(OpenvSwitch)[OVS], OVN(Open Virtual Network)[OVN], MACVLAN, IPVLAN

4.1.2. User-space Acceleration Model

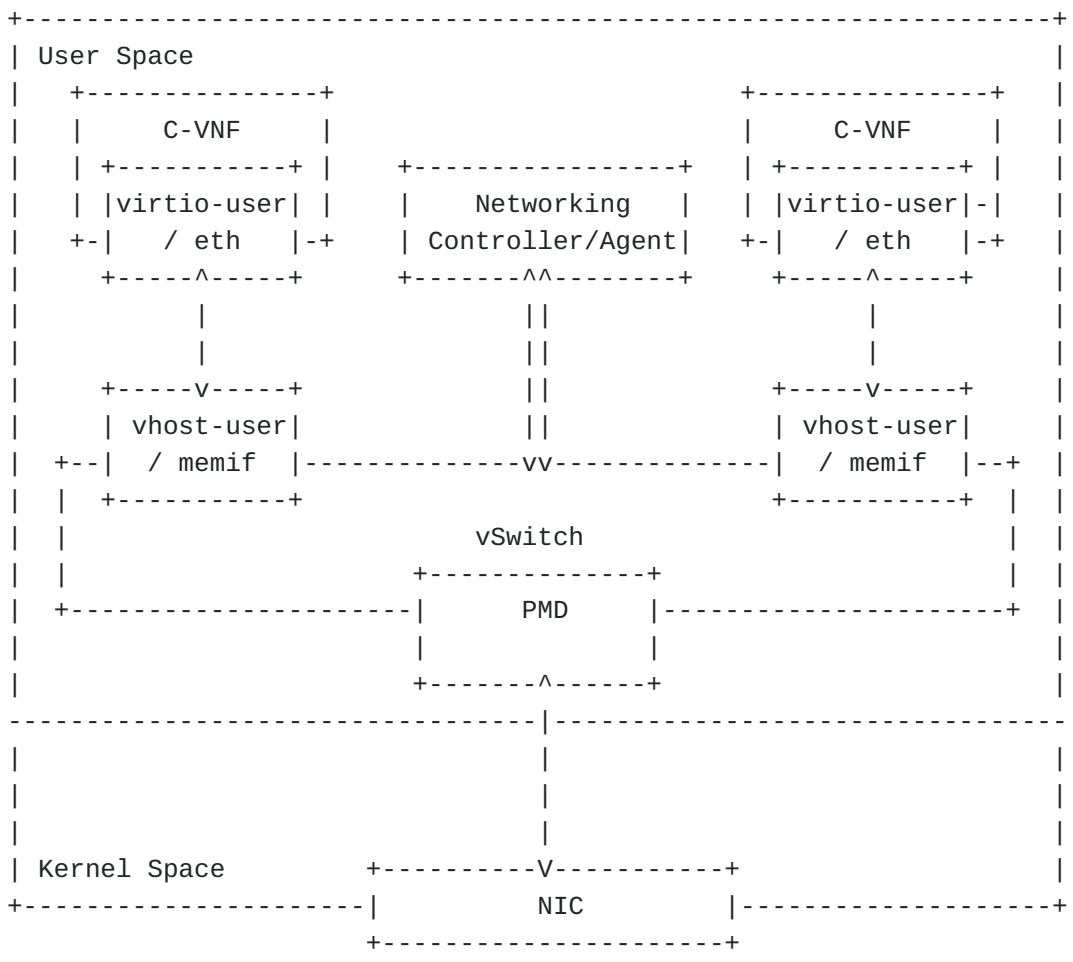

Figure 2: Example architecture of the User-Space Acceleration Model

Figure 2 shows user-space vSwitch model, in which data packets from physical network port are bypassed kernel processing and delivered directly to the vSwitch running on user-space. This model is commonly considered as Data Plane Acceleration (DPA) technology since it can achieve high-rate packet processing than a kernel-space network with limited packet throughput. For bypassing kernel and directly transferring the packet to vSwitch, Data Plane Development Kit (DPDK) is essentially required. With DPDK, an additional driver

called Pull-Mode Driver (PMD) is created on vSwtich. PMD driver must be created for each NIC separately. Userspace CNI [userspace-cni] is required to create user-space acceleration container networking. User-space vSwitch models are listed below;

o ovs-dpdk[ovs-dpdk], vpp[vpp]

4.1.3. eBPF Acceleration Model

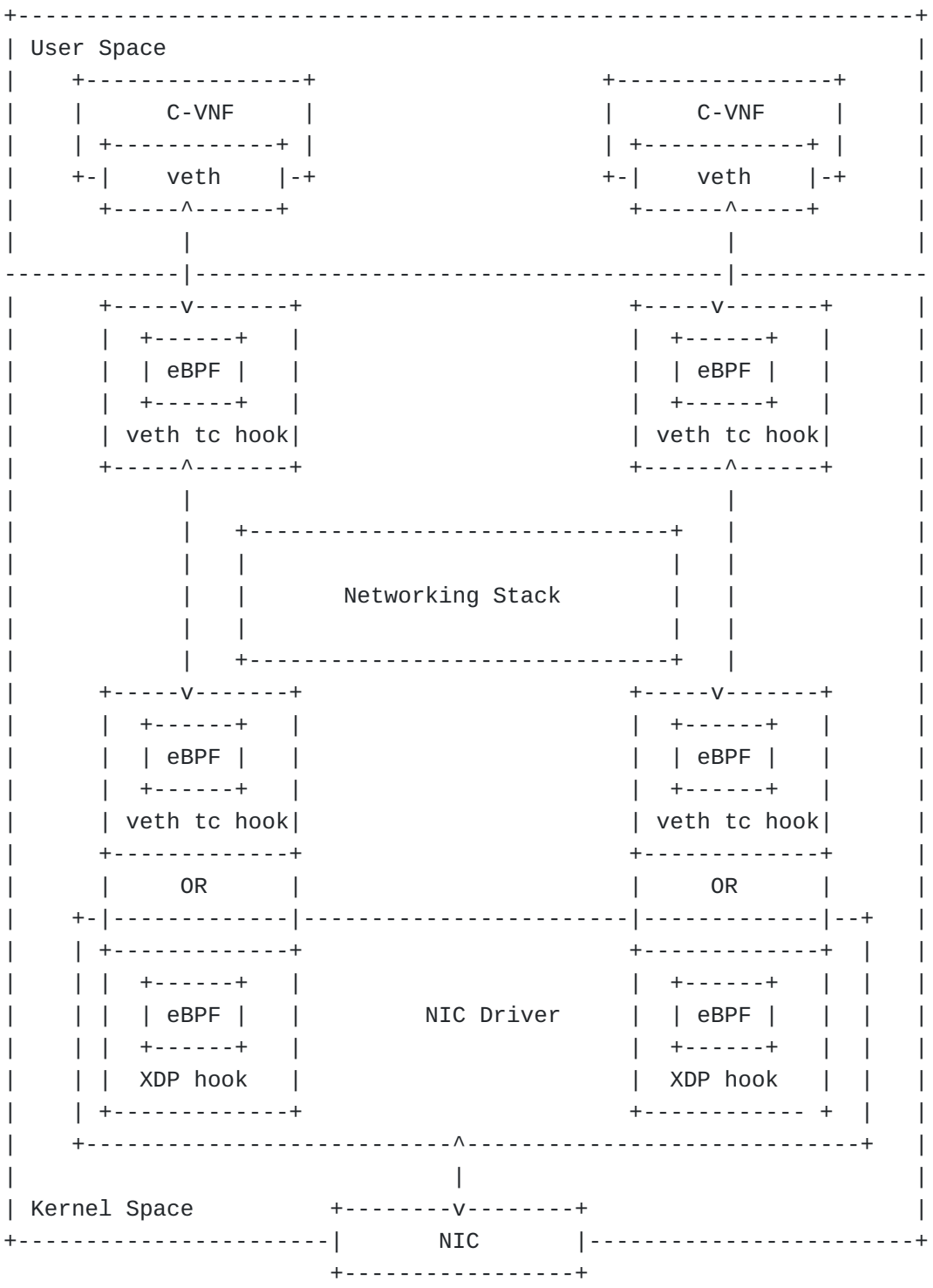

Figure 3: Example architecture of the eBPF Acceleration Model - non-AFXDP

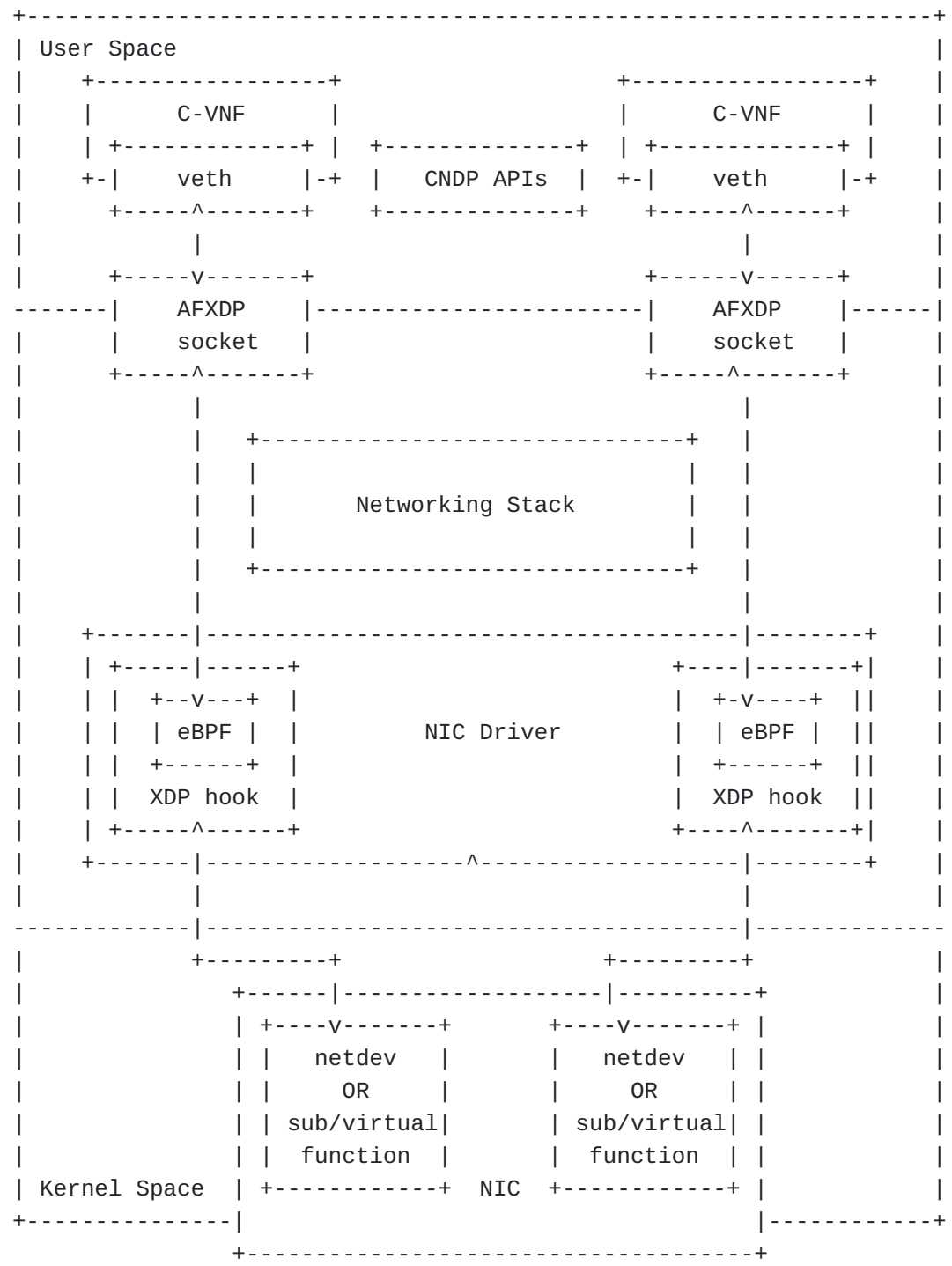

Figure 4: Example architecture of the eBPF Acceleration Model - using AFXDP supported CNI

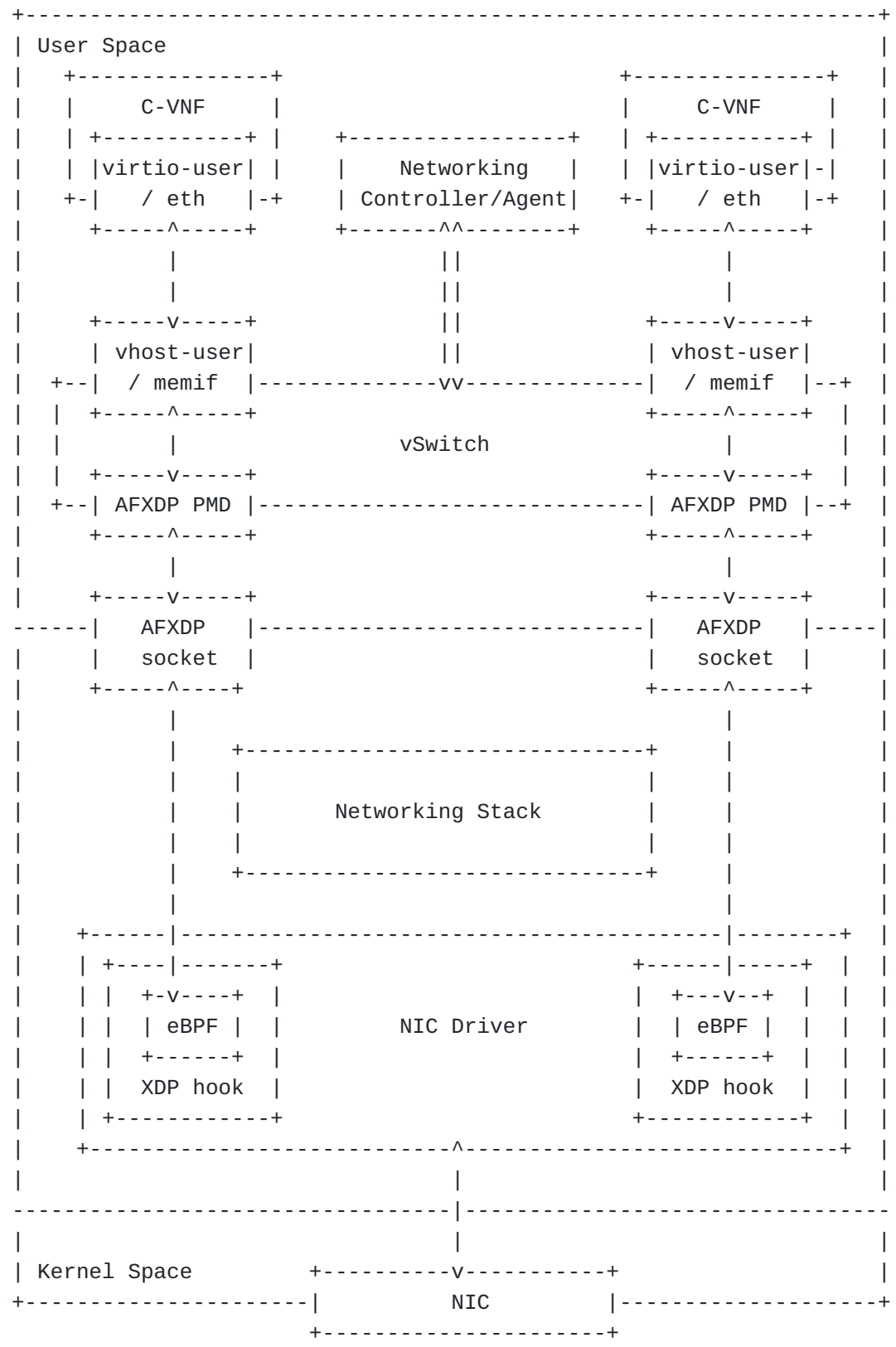

Figure 5: Example architecture of the eBPF Acceleration Model - using user-space vSwitch which support AFXDP PMD

The eBPF Acceleration model leverages the extended Berkeley Packet Filter (eBPF) technology [eBPF] to achieve high-performance packet processing. It enables execution of sandboxed programs inside abstract virtual machines within the Linux kernel without changing the kernel source code or loading the kernel module. To accelerate data plane performance, eBPF programs are attached to different BPF hooks inside the linux kernel stack.

One type of BPF hook is the eXpress Data Path (XDP) at the networking driver. It is the first hook that triggers eBPF program upon packet reception from external network. The other type of BPF hook is Traffic Control Ingress/Egress eBPF hook (tc eBPF). The eBPF program running at the tc hook enforce policy on all traffic exit the pod, while the eBPF program running at the XDP hook enforce policy on all traffic coming from NIC.

On the egress datapath side, whenever a packet exits the pod, it first goes through the pod’s vETH interface. Then, the destination that received the packet depends on the chosen CNI plugin that is used to create container networking. If the chosen CNI plugin is a non-AFXDP-based CNI, the packet is received by the eBPF program running at vETH interface tc hook. If the chosen CNI plugin is an AFXDP-supported CNI, the packet is received by the AFXDP socket [AFXDP]. AFXDP socket is a new Linux socket type which allows a fast packet delivery tunnel between itself and the XDP hook at the networking driver. This tunnel bypasses the network stack in kernel space to provide high-performance raw packet networking. Packets are transmitted between user space and AFXDP socket via a shared memory buffer. Once the egress packet arrived at the AFXDP socket or tc hook, it is directly forwarded to the NIC.

On the ingress datapath side, eBPF programs at the XDP hook/tc hook pick up packets from the NIC network devices (NIC ports). In case of using AFXDP CNI plugin [afxdp-cni], there are two operation modes: “primary” and “cdq”. In “primary” mode, NIC network devices can be directly allocated to pods. Meanwhile, in “cdq” mode, NIC network devices can be efficiently partioned to subfunctions or SR-IOV virtual functions, which enables multiple pods to share a primary network device. Then, from network devices, packets are directly delivered to the vETH interface pair or AFXDP socket (via or not via AFXDP socket depends on the chosen CNI), bypass all of the kernel network layer processing such as iptables. In case of Cilium CNI [Cilium], context-switching process to the pod network namespace can also be bypassed.

Notable eBPF Acceleration models can be classified into 3 categories below. Their corresponding model architecture are shown in Figure 3, Figure 4, Figure 5.

o non-AFXDP: eBPF supported CNI such as Calico [Calico], Cilium [Cilium]

o using AFXDP supported CNI: AFXDP K8s plugin [afxdp-cni] used by Cloud Native Data Plane project [CNDP]

o using user-space vSwitch which support AFXDP PMD: OVS-DPDK [ovs-dpdk] and VPP [vpp] are the vSwitches that have AFXDP device driver support. Userspace CNI [userspace-cni] is used to enable container networking via these vSwitches.

Container network performance of Cilium project is reported by the project itself in [cilium-benchmark]. Meanwhile, AFXDP performance and comparison against DPDK are reported in [intel-AFXDP] and [LPC18-DPDK-AFXDP], respectively.

4.1.4. Smart-NIC Acceleration Model

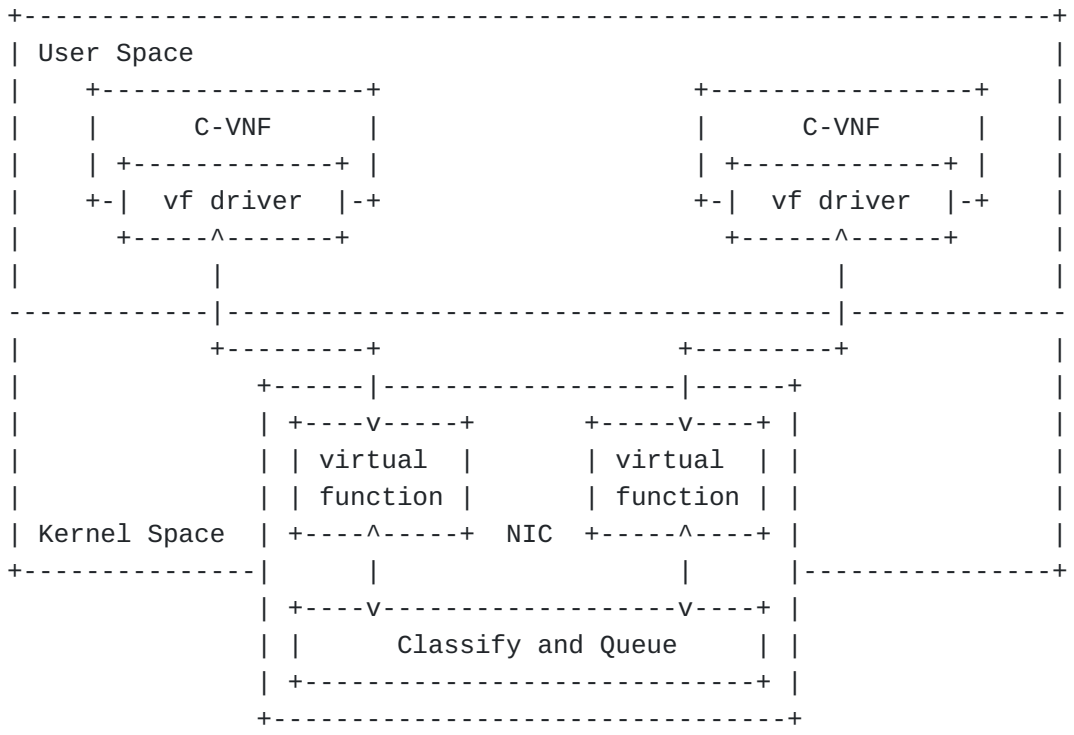

Figure 6: Examples of Smart-NIC Acceleration Model

Figure 6 shows Smart-NIC acceleration model, which does not use vSwitch component. This model can be separated into two technologies.

One is Single-Root I/O Virtualization (SR-IOV), which is an extension of PCIe specifications to enable multiple partitions running simultaneously within a system to share PCIe devices. In the NIC, there are virtual replicas of PCI functions known as virtual

functions (VF), and each of them is directly connected to each container's network interfaces. Using SR-IOV, data packets from external bypass both kernel and user space and are directly forwarded to container’s virtual network interface. SRIOV network device plugin for Kubernetes[SR-IOV] is recommended to create an SRIOV-based container networking.

The other technology is eBPF/XDP programs offloading to Smart-NIC card as mentioned in the previous section. It enables general acceleration of eBPF. eBPF programs are attached to XDP and run at the Smart-NIC card, which allows server CPUs to perform more application-level work. However, not all Smart-NIC cards provide eBPF/XDP offloading support.

4.1.5. Model Combination

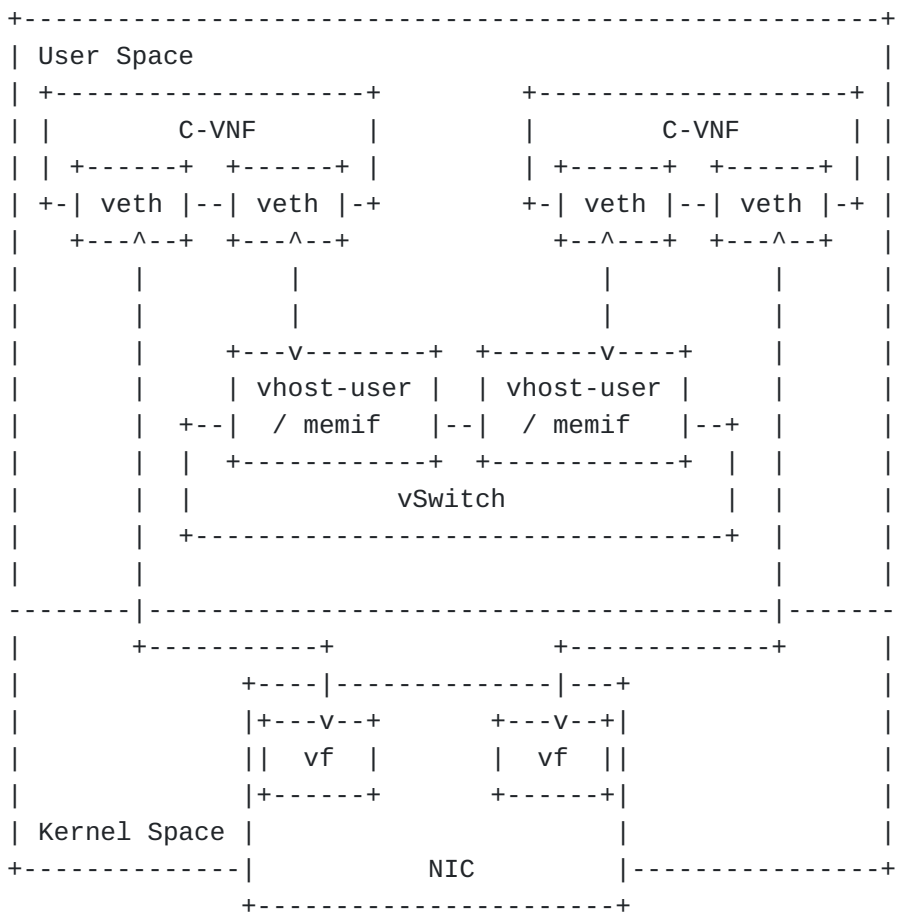

Figure 7: Examples of Model Combination deployment

Figure 7 shows the networking model when combining user-space vSwitch model and Smart-NIC acceleration model. This model is frequently considered in service function chain scenarios when two

different types of traffic flows are present. These two types are North/South traffic and East/West traffic.

North/South traffic is the type that packets are received from other servers and routed through VNF. For this traffic type, Smart-NIC model such as SR-IOV is preferred because packets always have to pass the NIC. User-space vSwitch involvement in north-south traffic will create more bottlenecks. On the other hand, East/West traffic is a form of sending and receiving data between containers deployed in the same server and can pass through multiple containers. For this type, user-space vSwitch models such as OVS-DPDK and VPP are preferred because packets are routed within the user space only and not through the NIC.

The throughput advantages of these different networking models with different traffic direction cases are reported in [Intel-SRIOV-NFV].

4.2. Resources Configuration

4.2.1. CPU Isolation / NUMA Affinity

CPU pinning enables benefits such as maximizing cache utilization, eliminating operating system thread scheduling overhead as well as coordinating network I/O by guaranteeing resources. This technology is very effective in avoiding the "noisy neighbor" problem, and it is already proved in existing experience [Intel-EPA].

Using NUMA, performance will be increasing not CPU and memory but also network since that network interface connected PCIe slot of specific NUMA node have locality. Using NUMA requires a strong understanding of VNF's memory requirements. If VNF uses more memory than a single NUMA node contains, the overhead will occurr due to being spilled to another NUMA node. Network performance can be changed depending on the location of the NUMA node whether it is the same NUMA node where the physical network interface and CNF are attached to. There is benchmarking experience for cross-NUMA performance impacts [cross-NUMA-vineperf]. In that tests, they consist of cross-NUMA performance with 3 scenarios depending on the location of the traffic generator and traffic endpoint. As the results, it was verified as below:

o A single NUMA Node serving multiple interfaces is worse than Cross-NUMA Node performance degradation

o Worse performance with VNF sharing CPUs across NUMA

4.2.2. Hugepages

Hugepage configures a large page size of memory to reduce Translation Lookaside Buffer(TLB) miss rate and increase the

application performance. This increases the performance of logical/ virtual to physical address lookups performed by a CPU's memory management unit, and overall system performance. In the containerized infrastructure, the container is isolated at the application level, and administrators can set huge pages more granular level (e.g., Kubernetes allows to use of 512M bytes huge pages for the container as default values). Moreover, this page is dedicated to the application but another process, so the application uses the page more efficiently way. From a network benchmark point of view, however, the impact on general packet processing can be relatively negligible, and it may be necessary to consider the application level to measure the impact together. In the case of using the DPDK application, as reported in [Intel-EPA], it was verified to improve network performance because packet handling processes are running in the application together.

4.2.3. CPU Cores and Memory Allocation

Different resources allocation choices may impact the container network performance. These include different CPU cores and RAM allocation to Pods, and different CPU cores allocation to the Poll Mode Driver and the vSwitch. Benchmarking experience from [ViNePERF] which was published in [GLOBECOM-21-benchmarking-kubernetes] verified that:

o 2 CPUs per Pod is insufficient for all packet frame sizes. With large packet frame sizes (over 1024), increasing CPU per pods significantly increases the throughput. Different RAM allocation to Pods also causes different throughput results

o Not assigning dedicated CPU cores to DPDK PMD causes significant performance dropss

o Increasing CPU core allocation to OVS-DPDK vSwitch does not affect its performance. However, increasing CPU core allocation to VPP vSwitch results in better latency.

Besides, regarding user-space acceleration model which uses PMD to poll packets to the user-space vSwitch, dedicated CPU cores assignment to PMD’s Rx Queues might improve the network performance.

4.2.4. Service Function Chaining

When we consider benchmarking for containerized and VM-based infrastructure and network functions, benchmarking scenarios may contain various operational use cases. Traditional black-box benchmarking focuses on measuring the in-out performance of packets from physical network ports since the hardware is tightly coupled with its function and only a single function is running on its dedicated hardware. However, in the NFV environment, the physical

network port commonly will be connected to multiple VNFs(i.e., Multiple PVP test setup architectures were described in [ETSI-TST-009]) rather than dedicated to a single VNF. This scenario is called Service Function Chaining. Therefore, benchmarking scenarios should reflect operational considerations such as the number of VNFs or network services defined by a set of VNFs in a single host. [service-density] proposed a way for measuring the performance of multiple NFV service instances at a varied service density on a single host, which is one example of these operational benchmarking aspects. Another aspect in benchmarking service function chaining scenario should be considered is different network acceleration technologies. Network performance differences may occur because of different traffic patterns based on the provided acceleration method.

4.2.5. Additional Considerations

Apart from the single-host test scenario, the multi-hosts scenario should also be considered in container network benchmarking, where container services are deployed across different servers. To provide network connectivity for container-based VNFs between different server nodes, inter-node networking is required. According to [ETSI-NFV-IFA-038], there are several technologies to enable inter-node network: overlay technologies using a tunnel endpoint (e.g. VXLAN, IP in IP), routing using Border Gateway Protocol (BGP), layer 2 underlay, direct network using dedicated NIC for each pod, or load balancer using LoadBalancer service type in Kubernetes. Different protocols from these technologies may cause performance differences in container networking.

5. Security Considerations

Benchmarking activities as described in this memo are limited to technology characterization of a Device Under Test/System Under Test (DUT/SUT) using controlled stimuli in a laboratory environment with dedicated address space and the constraints specified in the sections above.

The benchmarking network topology will be an independent test setup and MUST NOT be connected to devices that may forward the test traffic into a production network or misroute traffic to the test management network.

Further, benchmarking is performed on a "black-box" basis and relies solely on measurements observable external to the DUT/SUT.

Special capabilities SHOULD NOT exist in the DUT/SUT specifically for benchmarking purposes. Any implications for network security

arising from the DUT/SUT SHOULD be identical in the lab and in production networks.

6. References

6.1. Informative References

[AFXDP] "AF_XDP", September 2022, <https://www.kernel.org/doc/html/v4.19/networking/af_xdp.html>.

[afxdp-cni] "AF_XDP Plugins for Kubernetes", <https://github.com/intel/afxdp-plugins-for-kubernetes>.

[Calico] "Project Calico", July 2019, <https://docs.projectcalico.org/>.

[Cilium] "Cilium Documentation", March 2022, <https://docs.cilium.io/en/stable//>.

[cilium-benchmark] Cilium, "CNI Benchmark: Understanding Cilium Network Performance", May 2021, <https://cilium.io/blog/2021/05/11/cni-benchmark>.

[CNDP] "CNDP - Cloud Native Data Plane", September 2022, <https://cndp.io/>.

[cross-NUMA-vineperf] Anuket Project, "Cross-NUMA performance measurements with VSPERF", March 2019, <https://

wiki.anuket.io/display/HOME/Cross-NUMA+performance+measurements+with+VSPERF>.

[Docker-network] "Docker, Libnetwork design", July 2019, <https://github.com/docker/libnetwork/>.

[eBPF] "eBPF, extended Berkeley Packet Filter", July 2019, <https://www.iovisor.org/technology/ebpf>.

[ETSI-NFV-IFA-038] "Network Functions Virtualisation (NFV) Release 4; Architectural Framework; Report on network connectivity for container-based VNF", November 2021.

[ETSI-TST-009] "Network Functions Virtualisation (NFV) Release 3; Testing; Specification of Networking Benchmarks and Measurement Methods for NFVI", October 2018.

[Flannel] "flannel 0.10.0 Documentation", July 2019, <https://coreos.com/flannel/>.

[GLOBECOM-21-benchmarking-kubernetes] Sridhar, R., Paganelli, F., and A. Morton, "Benchmarking Kubernetes Container-Networking for Telco Usecases", December 2021.

[intel-AFXDP] Karlsson, M., "AF_XDP Sockets: High Performance Networking for Cloud-Native Networking Technology Guide", January 2021.

[Intel-EPA] Intel, "Enhanced Platform Awareness in Kubernetes", 2018, <https://builders.intel.com/docs/networkbuilders/enhanced-platform-awareness-feature-brief.pdf>.

[Intel-SRIOV-NFV] Patrick, K. and J. Brian, "SR-IOV for NFV Solutions Practical Considerations and Thoughts", February 2017.

[LPC18-DPDK-AFXDP] Karlsson, M. and B. Topel, "The Path to DPDK Speeds for AF_XDP", November 2018.

[OVN] "How to use Open Virtual Networking with Kubernetes", July 2019, <https://github.com/ovn-org/ovn-kubernetes>.

[OVS] "Open Virtual Switch", July 2019, <https://www.openvswitch.org/>.

[ovs-dpdk] "Open vSwitch with DPDK", July 2019, <http://docs.openvswitch.org/en/latest/intro/install/dpdk/>.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8172] Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", RFC 8172, July 2017, <https://www.rfc-editor.org/rfc/rfc8172>.

[RFC8204] Tahhan, M., O'Mahony, B., and A. Morton, "Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV)", RFC 8204, September 2017, <https://www.rfc-editor.org/rfc/rfc8204>.

[service-density] Konstantynowicz, M. and P. Mikus, "NFV Service Density Benchmarking", March 2019, <https://tools.ietf.org/html/draft-mkonstan-nf-service-density-00>.

[SR-IOV] "SRIOV for Container-networking", July 2019, <https://github.com/intel/sriov-cni>.

[userspace-cni] "Userspace CNI Plugin", August 2021, <https://github.com/intel/userspace-cni-network-plugin>.

[ViNePERF] "Project: Virtual Network Performance for Telco NFV", <https://wiki.anuket.io/display/HOME/ViNePERF>.

[vpp] "VPP with Containers", July 2019, <https://fdio-vpp.readthedocs.io/en/latest/usecases/containers.html>.

Appendix A. Benchmarking Experience(Contiv-VPP)

A.1. Benchmarking Environment

In this test, our purpose is to test the performance of user-space based model for container infrastructure and figure out the relationship between resource allocation and network performance. With respect to this, we set up Contiv-VPP, one of the user-space based network solutions in container infrastructure and tested like below.

o Three physical server for benchmarking

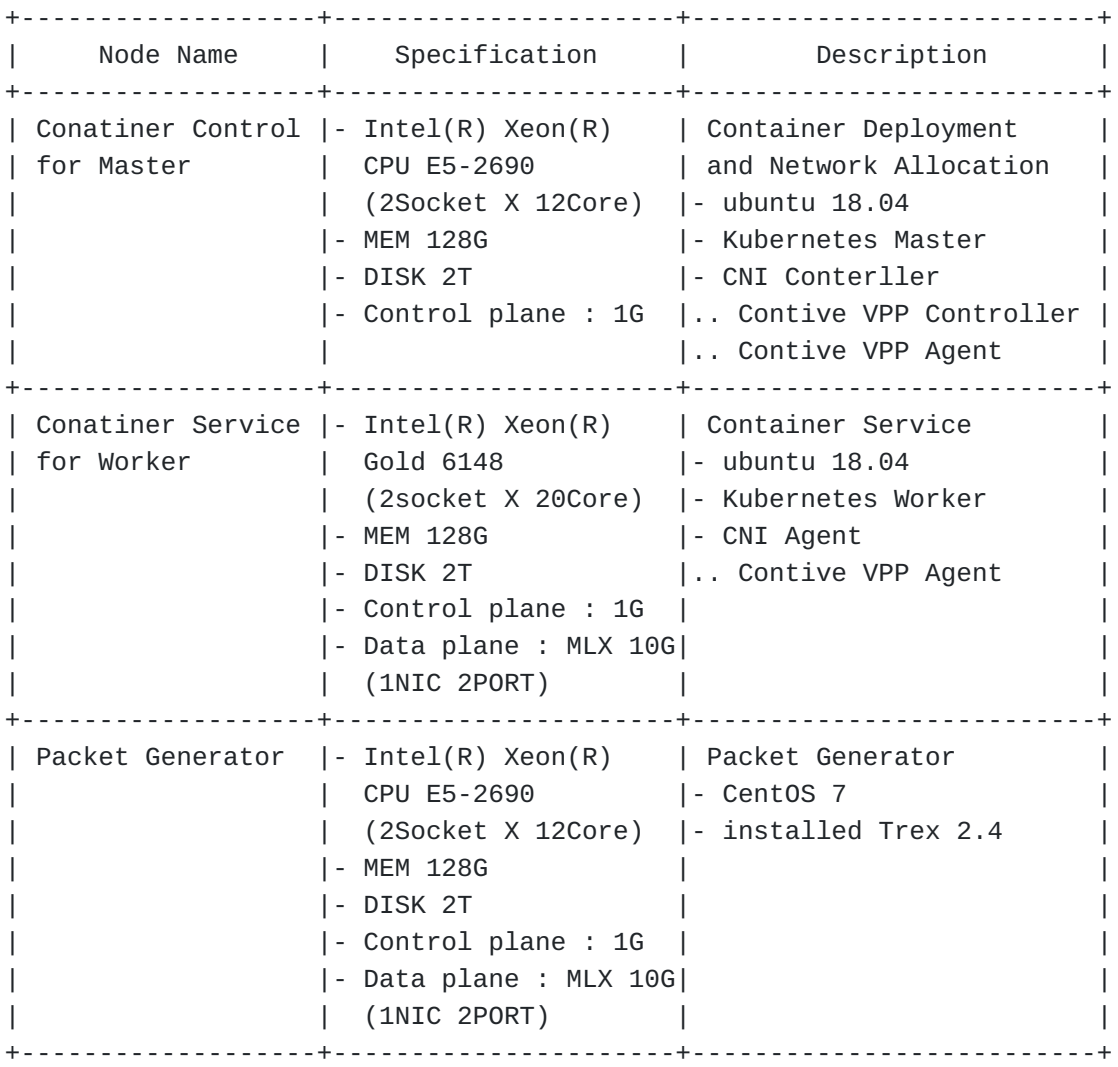

Figure 8: Test Environment-Server Specification

o The architecture of benchmarking

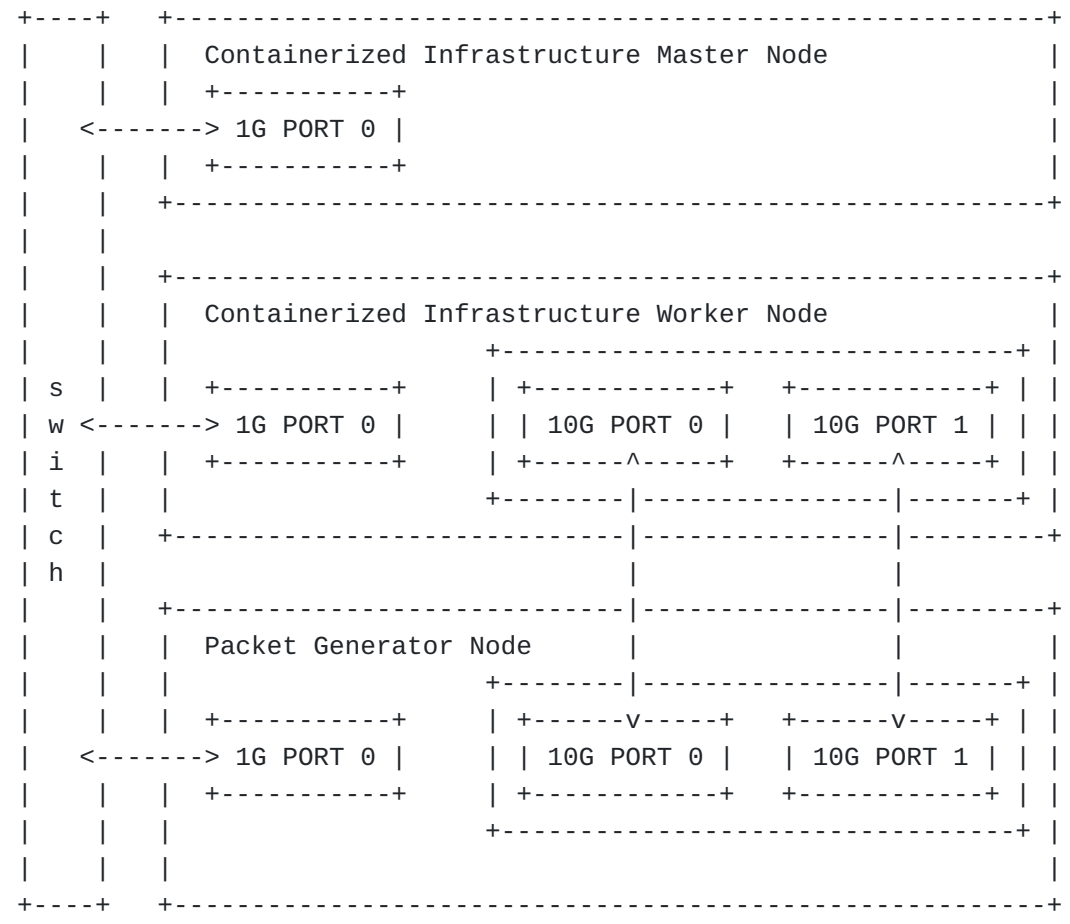

Figure 9: Test Environment-Architecture

o Network model of Containerized Infrastructure(User space Model)

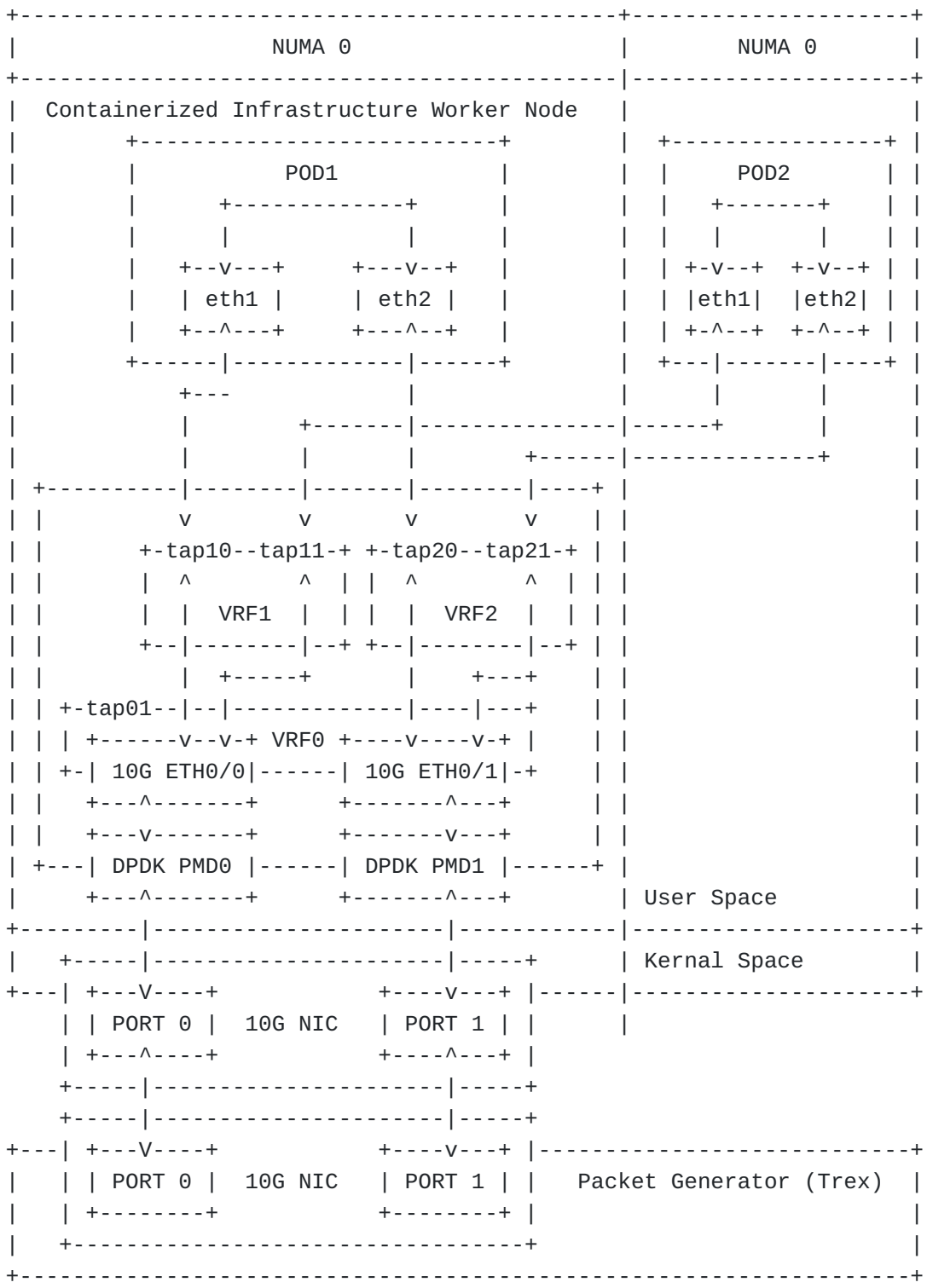

Figure 10: Test Environment-Network Architecture

We set up a Contive-VPP network to benchmark the user space container network model in the containerized infrastructure worker node. We set up network interface at NUMA0, and we created different network subnets VRF1, VRF2 to classify input and output data traffic, respectively. And then, we assigned two interfaces which

connected to VRF1, VRF2 and, we setup routing table to route Trex packet from eth1 interface to eth2 interface in POD.

A.2. Trouble shooting and Result

In this environment, we confirmed that the routing table doesn't work when we send packets using Trex packet generator. The reason is that when kernel space based network configured, ip forwarding rule is processed to kernel stack level while 'ip packet forwarding rule' is processed only in vrf0, which is the default virtual routing and forwarding (VRF0) in VPP. The above testing architecture makes problem since vrf1 and vrf2 interface couldn't route packet. According to above result, we assigned vrf0 and vrf1 to POD and, data flow is like below.

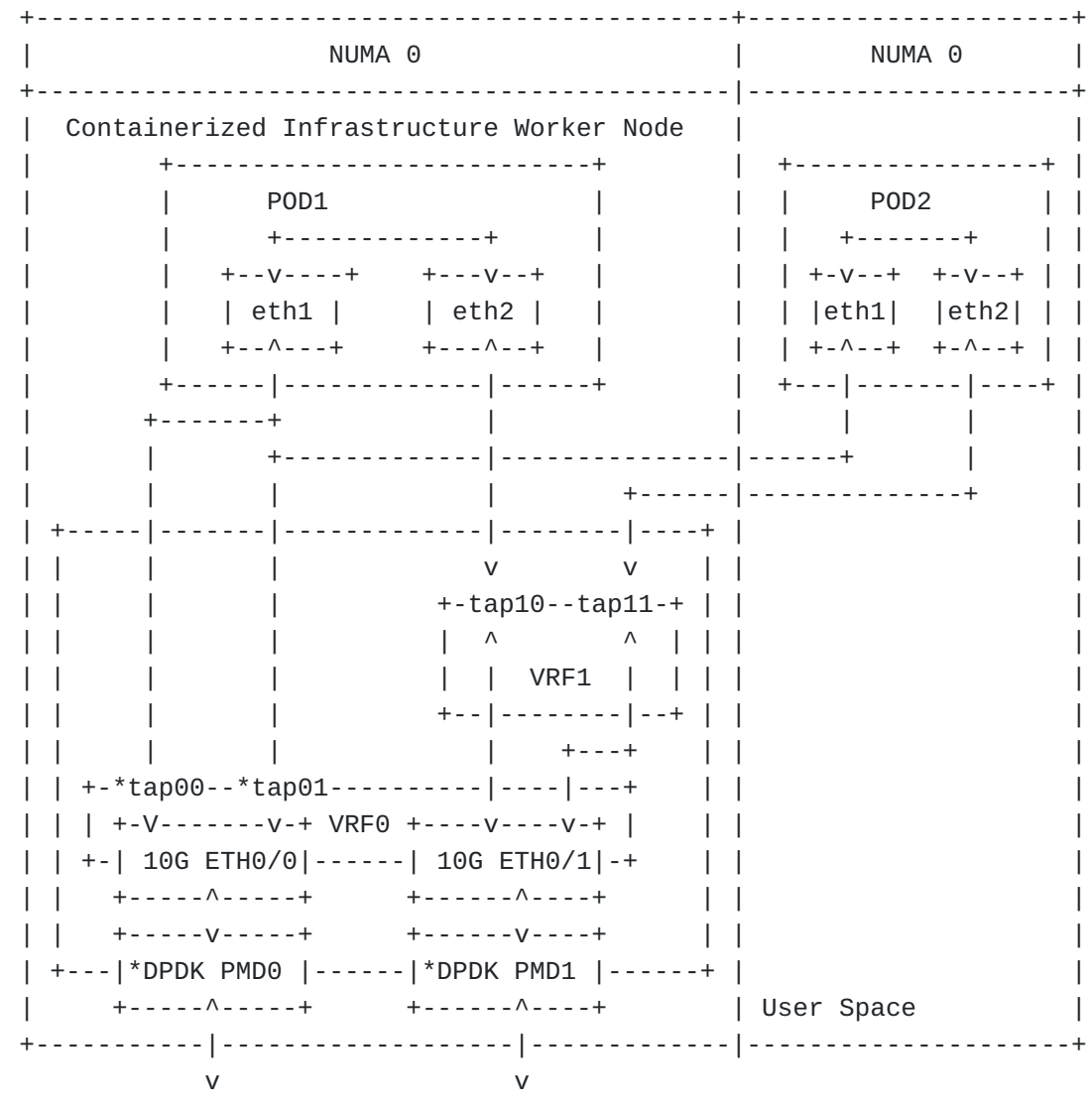

Figure 11: Test Environment-Network Architecture(CPU Pinning)

We conducted benchmarking with three conditions. The test environments are as follows. - Basic VPP switch - General kubernetes (No CPU Pining) - Shared Mode / Exclusive mode. In the basic Kubernetes environment, all PODs share a host's CPU. Shared mode is that some POD share a pool of CPU assigned to specific PODs. Exclusive mode is that a specific POD dedicates a specific CPU to use. In shared mode, we assigned two CPUs for several PODs, in exclusive mode, we dedicated one CPU for one POD, independently. The result is like Figure 12. First, the test was conducted to figure out the line rate of the VPP switch, and the basic Kubernetes performance. After that, we applied NUMA to the network interface using Shared Mode and Exclusive Mode in the same node and different node. In Exclusive and Shared mode tests, we confirmed that Exclusive mode showed better performance than Shared mode when same NUMA CPU was assigned, respectively. However, we confirmed that performance is reduced at the section between the vpp switch and the POD, affecting the total result.

Model	NUMA Mode (pinning)	Result(Gbps)
Maximum Line Rate	N/A	3.1
	same NUMA	9.8
Without CMK	N/A	1.5
CMK-Exclusive Mode	same NUMA	4.7
	Different NUMA	3.1
CMK-shared Mode	same NUMA	3.5
	Different NUMA	2.3

Figure 12: Test Results

Appendix B. Benchmarking Experience(SR-IOV with DPDK)

B.1. Benchmarking Environment

In this test, our purpose is to test the performance of Smart-NIC acceleration model for container infrastructure and figure out relationship between resource allocation and network performance. With respect to this, we setup SRIOV combining with DPDK to bypass the Kernel space in container infrastructure and tested based on that.

o Three physical server for benchmarking

Node Name	Specification	Description
Conatiner Control for Master	- Intel(R) Core(TM) i5-6200U CPU (1socket x 4Core) - MEM 8G - DISK 500GB - Control plane : 1G	Container Deployment and Network Allocation - ubuntu 18.04 - Kubernetes Master - CNI Conterller MULTUS CNI SRIOV plugin with DPDK
Conatiner Service for Worker	- Intel(R) Xeon(R) E5-2620 v3 @ 2.4Ghz (1socket X 6Core) - MEM 128G - DISK 2T - Control plane : 1G - Data plane : XL710-qda2 (1NIC 2PORT- 40Gb)	Container Service - Centos 7.7 - Kubernetes Worker - CNI Agent MULTUS CNI SRIOV plugin with DPDK
Packet Generator	- Intel(R) Xeon(R) Gold 6148 @ 2.4Ghz (2Socket X 20Core) - MEM 128G - DISK 2T - Control plane : 1G - Data plane : XL710-qda2 (1NIC 2PORT- 40Gb)	Packet Generator - CentOS 7.7 - installed Trex 2.4

Figure 13: Test Environment-Server Specification

o The architecture of benchmarking

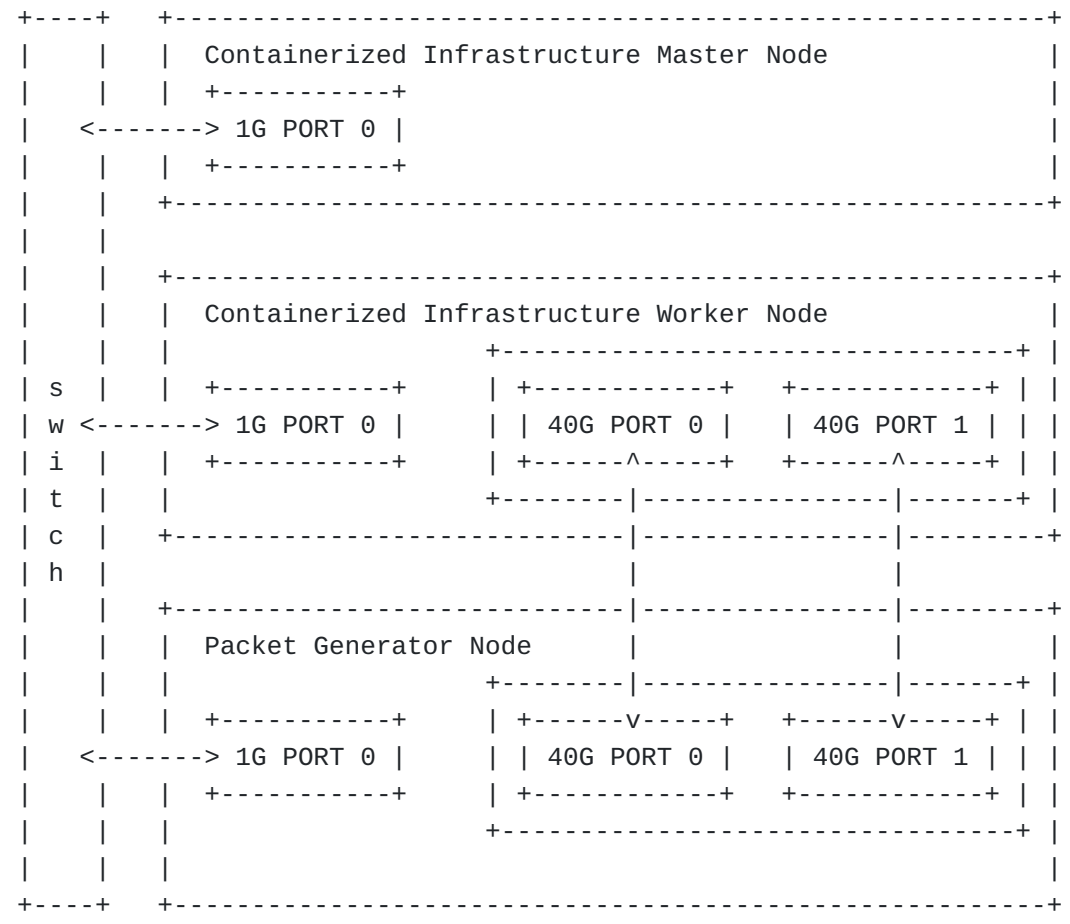

Figure 14: Test Environment-Architecture

o Network model of Containerized Infrastructure(User space Model)

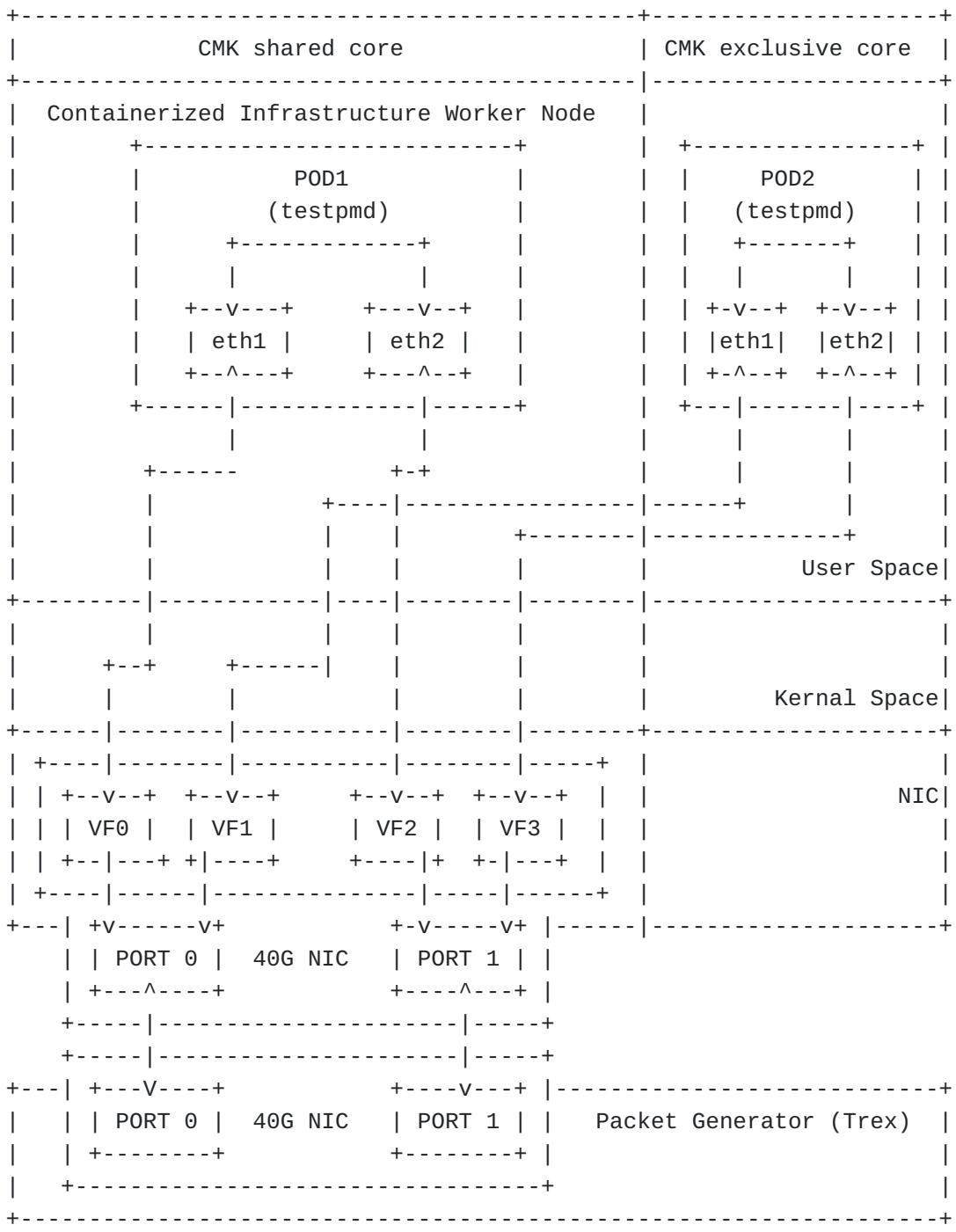

Figure 15: Test Environment-Network Architecture

We set up a Multus CNI, SRIOV CNI with DPDK to benchmark the user-space container network model in the containerized infrastructure worker node. The Multus CNI support creates multiple interfaces for a container. The traffic is bypassed the Kernel space by SRIOV with DPDK. We established two modes of CMK: shared core and exclusive core. We created VFs for each network interface of a container. Then, we set up TREX to route packet from eth1 to eth2 in a POD.

B.2. Trouble shooting and Results

Figure 16 shows the test results when using 1518 bytes packet traffic from the T-Rex traffic generator. First, we get the maximum line rate of the system using SR-IOV as the packet acceleration technique. Then we measured throughput when applying the CMK feature. We observed similar results as VPP CPU Pinning test. The default Kubernetes system without CMK feature enabled had the worst performance as the CPU resources are shared without any isolation. When the CMK feature is enabled, Exclusive Mode performed better than Shared Mode because each pod had its own dedicated CPU.

Model	Result(Gbps)
Maximum Line Rate	39.3
Without CMK	11.5
CMK-Exclusive Mode	39.2
CMK-shared Mode	29.6

Figure 16: SR-IOV CPU Pinning Test Results

Appendix C. Benchmarking Experience(Multi-pod Test)

C.1. Benchmarking Overview

The main goal of this experience was to benchmark the multi-pod scenario, in which packets are traversed through two pods. To create additional interfaces for forwarding packets between two pods, Multus CNI was used. We compared two userspace-vSwitch model network technologies: OVS/DPDK and VPP-memif. Since that vpp-memif has a different packet forwarding mechanism by using shared memory interface, it is expected that vpp-memif may provide higher performance that OVS-DPDK. Also, we consider NUMA impact for both cases, and made 6 scenarios depending on CPU location of vSwitch and two pods. Figure 17 is packet forwarding scenario in this test, where two pods run on the same host and vSwitch delivers packets between two pods.

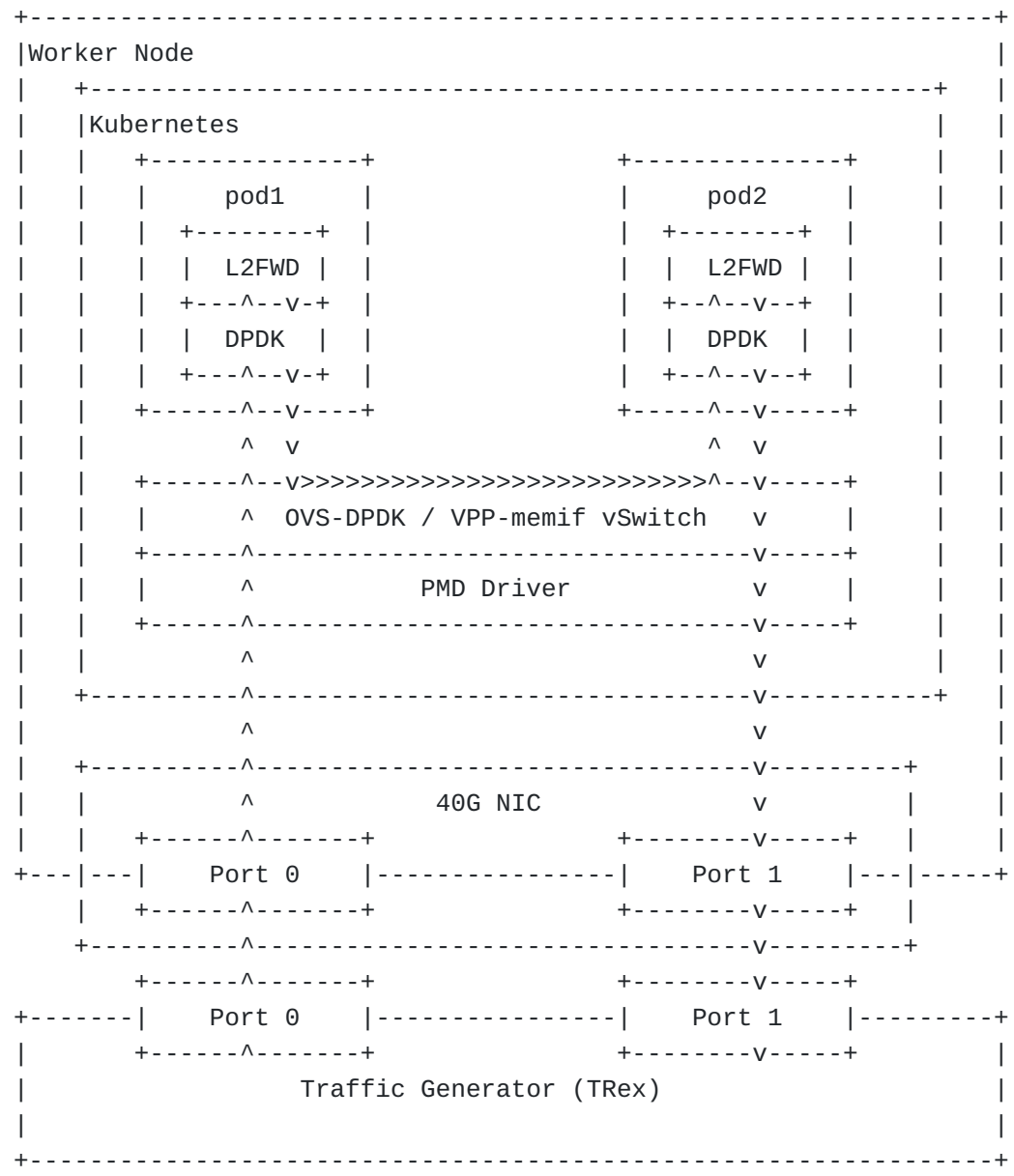

Figure 17: Multi-pod Benchmarking Scenario

C.2. Hardware Configurations

Node Name	Specification	Description
Conatiner Control for Master	- Intel(R) Core(TM) E5-2620v3 @ 2.40GHz (1socket x 12Cores) - MEM 32GB - DISK 1TB - NIC: Control plane: 1G - OS: CentOS Linux7.9	Container Deployment and Network Allocation - ubuntu 18.04 - Kubernetes Master - CNI Controller - MULTUS CNI - DPDK-OVS/VPP-memif
Conatiner Service for Worker	- Intel(R) Xeon(R) Gold 6148 @ 2.40GHz (2socket X 40Cores) - MEM 256GB - DISK 2TB - NIC - Control plane: 1G - Data plane: XL710-qda2 (1NIC 2PORT- 40Gb) - OS: CentOS Linux 7.9	- Container dpdk-L2fwd - Kubernetes Worker - CNI Agent - Multus CNI - DPDK-OVS/VPP-memif
Packet Generator	- Intel(R) Xeon(R) Gold 6148 @ 2.4Ghz (2Socket X 40Core) - MEM 256GB - DISK 2TB - NIC - Data plane: XL710-qda2 (1NIC 2PORT - 40Gb) - OS: CentOS Lunix 7.9	Packet Generator - Installed Trex v2.92

Figure 18: Hardware Configurations for Multi-pod Benchmarking

For installations and configurations of CNIs, we used userspace-cni network plugin. Among this CNI, multus provides to create multiple interfaces for each pod. Both OVS-DPDK and VPP-memif bypass kernel with DPDK PMD driver. For CPU isolation and NUMA allocation, we used Intel CMK with exclusive mode. Since Trex generator is upgraded to the new version, we used the latest version of Trex.

C.3. NUMA Allocation Scenario

To analyze benchmarking impacts of different NUMA allocation, we set 6 scenarios depending on CPU location allocating to two pods and vSwich. For this scenario, we did not consider cross-NUMA case, which allocates CPUs to pod or switch in a manner that two cores are

located in different NUMA nodes. 6 scenarios we considered are listed in Table 1. Note that, NIC is attached to the NUMA1.

Scenario #	vSwtich	pod1	pod2
S1	NUMA1	NUMA0	NUMA0
S2	NUMA1	NUMA1	NUMA1
S3	NUMA0	NUMA0	NUMA0
S4	NUMA0	NUMA1	NUMA1
S5	NUMA1	NUMA1	NUMA0
S6	NUMA0	NUMA0	NUMA1

Table 1: NUMA Allocation Scenarios

C.4. Traffic Generator Configurations

For multi-pod benchmarking, we discovered Non Drop Rate (NDR) with binary search algorithm. In Trex, it supports command to discover NDR for each testing. Also, we test for different ethernet frame sizes from 64bytes to 1518bytes. For running Trex, we used command as follows;

```
./ndr --stl --port 0 1 -v --profile stl/bench.py --prof-tun size=x
--opt-bin-search
```

C.5. Benchmark Results and Trouble-shootings

As the benchmarking results, Table 2 shows packet loss ratio using 1518 bytes packet in OVS-DPDK/vpp-memif. From that result, we can say that the vpp-memif has better performance that OVS-DPDK, which is came from the difference in the way to forward packets between vswitch and pod. Also, the impact of NUMA is bigger when vswitch and both pods are located in the same node than when allocating CPU to the node where NIC is attached.

Networking Model	S1	S2	S3	S4	S5	S6
OVS-DPDK	21.29	13.17	6.32	19.76	12.43	6.38
vpp-memif	59.96	34.17	45.13	57.1	33.47	44.92

Table 2: Multi-pod Benchmarking Results (% of Line Rate)

Appendix D. Change Log (to be removed by RFC Editor before publication)

D.1. Since draft-dcn-bmwg-containerized-infra-09

Remove Additional Deployment Scenarios (section 4.1 of version 09). We agreed with reviews from VinePerf that performance difference between with-VM and without-VM scenarios are negligible

Remove Additional Configuration Parameters (section 4.2 of version 09). We agreed with reviews from VinePerf that these parameters are explained in Performance Impacts/Resources Configuration section

As VinePerf suggestion to categorize the networking models based on how they can accelerate the network performances, rename titles of section 4.3.1 and 4.3.2 of version 09: Kernel-space vSwitch model and User-space vSwitch model to Kernel-space non-Acceleration model and User-space Acceleration model. Update corresponding explanation of kernel-space non-Acceleration model

VinePerf suggested to replace the general architecture of eBPF Acceleration model with 3 seperate architecture for 3 different eBPF Acceleration model: non-AFXDP, using AFXDP supported CNI, and using user-space vSwitch which support AFXDP PMD. Update corresponding explanation of eBPF Acceleration model

Rename Performance Impacts section (section 4.4 of version 09) to Resources Configuration.

We agreed with VinePerf reviews to add "CPU Cores and Memory Allocation" consideration into Resources Configuration section

D.2. Since draft-dcn-bmwg-containerized-infra-08

Added new Section 4. Benchmarking Considerations. Previous Section 4. Networking Models in Containerized Infrastructure was moved into this new Section 4 as a subsection

Re-organized Additional Deployment Scenarios for containerized network benchmarking contents from Section 3. Containerized Infrastructure Overview to new Section 4. Benchmarking Considerations as the Addtional Deployment Scenarios subsection

Added new Addtional Configuration Parameters subsection to new Section 4. Benchmarking Considerations

Moved previous Section 5. Performance Impacts into new Section 4. Benchmarking Considerations as the Deployment settings impact on network performance section

Updated eBPF Acceleration Model with AFXDP deployment option

Enhanced Abstract and Introduction's description about the draft's motivation and contribution.

D.3. Since draft-dcn-bmwg-containerized-infra-07

Added eBPF Acceleration Model in Section 4. Networking Models in Containerized Infrastructure

Added Model Combination in Section 4. Networking Models in Containerized Infrastructure

Added Service Function Chaining in Section 5. Performance Impacts

Added Troubleshooting and Results for SRIOV-DPDK Benchmarking Experience

D.4. Since draft-dcn-bmwg-containerized-infra-06

Added Benchmarking Experience of Multi-pod Test

D.5. Since draft-dcn-bmwg-containerized-infra-05

Removed Section 3. Benchmarking Considerations, Removed Section 4. Benchmarking Scenarios for the Containerized Infrastructure

Added new Section 3. Containerized Infrastructure Overview, Added new Section 4. Networking Models in Containerized Infrastructure. Added new Section 5. Performance Impacts

Re-organized Subsection Comparison with the VM-based Infrastructure of previous Section 3. Benchmarking Considerations and previous Section 4.Benchmarking Scenarios for the Containerized Infrastructure to new Section 3. Containerized Infrastructure Overview

Re-organized Subsection Container Networking Classification of previous Section 3. Benchmarking Considerations to new Section 4. Networking Models in Containerized Infrastructure. Kernel-space vSwitch models and User-space vSwitch models were presented as seperate subsections in this new Section 4.

Re-organized Subsection Resource Considerations of previous Section 3. Benchmarking Considerations to new Section 5. Performance Impacts as 2 seperate subsections CPU Isolation / NUMA Affinity and Hugepages. Previous Section 5. Additional Considerations was moved into this new Section 5 as the Additional Considerations subsection.

Moved Benchmarking Experience contents to Appendix

D.6. Since draft-dcn-bmwg-containerized-infra-04

Added Benchmarking Experience of SRIOV-DPDK.

D.7. Since draft-dcn-bmwg-containerized-infra-03

Added Benchmarking Experience of Contiv-VPP.

D.8. Since draft-dcn-bmwg-containerized-infra-02

Editorial changes only.

D.9. Since draft-dcn-bmwg-containerized-infra-01

Editorial changes only.

D.10. Since draft-dcn-bmwg-containerized-infra-00

Added Container Networking Classification in Section 3.Benchmarking Considerations (Kernel Space network model and User Space network model).

Added Resource Considerations in Section 3.Benchmarking Considerations(Hugepage, NUMA, RX/TX Multiple-Queue).

Renamed Section 4.Test Scenarios to Benchmarking Scenarios for the Containerized Infrastructure, added 2 additional scenarios BMP2VMP and VMP2VMP.

Added Additional Consideration as new Section 5.

Contributors

Kyoungjae Sun - ETRI - Republic of Korea

Email: kjsun@etri.re.kr

Hyunsik Yang - KT - Republic of Korea

Email: yangun@dcn.ssu.ac.kr

Acknowledgments

The authors would like to thank Al Morton for their valuable ideas and comments for this work.

Authors' Addresses

Tran Minh Ngoc
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea

Phone: +82 28200841
Email: mipearlska1307@dcn.ssu.ac.kr

Sridhar Rao
The Linux Foundation
B801, Renaissance Temple Bells, Yeshwantpur
Bangalore 560022
India

Phone: +91 9900088064
Email: srao@linuxfoundation.org

Jangwon Lee
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea

Phone: +82 1074484664
Email: jangwon.lee@dcn.ssu.ac.kr

Younghan Kim
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul
06978
Republic of Korea

Phone: +82 1026910904
Email: younghak@ssu.ac.kr