Benchmarking Methodology Working Group
Internet-Draft
Intended status: Informational
Expires: May 6, 2021

K. Sun
H. Yang
J. Lee
H. Nguyen
Y. Kim
Soongsil University

November 02, 2020

Considerations for Benchmarking Network Performance in Containerized Infrastructures
draft-dcn-bmwg-containerized-infra-05

Abstract

This draft describes considerations for benchmarking network performance in containerized infrastructures. In the containerized infrastructure, Virtualized Network Functions(VNFs) are deployed on operating-system-level virtualization platform by abstracting the user namespace as opposed to virtualization using a hypervisor. Leveraging this, the system configurations and networking scenarios for benchmarking will be partially changed by the way in which the resource allocation and network technologies specified for containerized VNFs. In this draft, we compare the state of the art in a container networking architecture with networking on VM-based virtualized systems, and provide several test scenarios for benchmarking network performance in containerized infrastructures.

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 VNFs share the same host Operating System(OS) and they are logically isolated by using a different namespace. While previous NFV infrastructure uses a hypervisor to allocate resources for Virtual Machine(VMs) and instantiate VNFs on it, the containerized infrastructure virtualizes resources without a hypervisor, therefore making containers very lightweight and more efficient in infrastructure resource utilization compared to the VM-based NFV infrastructure. When we consider benchmarking for VNFs in the containerized infrastructure, it may have a different System Under Test(SUT) and Device Under Test(DUT) configuration compared with both black-box benchmarking and VM-based NFV infrastructure as described in [RFC8172]. Accordingly, additional configuration parameters and testing strategies may be required.

In the containerized infrastructure, a VNF network is implemented by running both switch and router functions in the host system. For example, the internal communication between VNFs in the same host uses the L2 bridge function, while communication with external node(s) uses the L3 router function. For container networking, the host system may use a virtual switch(vSwitch), but other options exist. In the [ETSI-TST-009], they describe differences in networking structure between the VM-based and the containerized infrastructure. Occasioned by these differences, deployment scenarios for testing network performance described in [RFC8204] may be partially applied to the containerized infrastructure, but other scenarios may be required.

In this draft, we describe differences and additional considerations for benchmarking containerized infrastructure based on [RFC8172] and [RFC8204]. In particular, we focus on differences in system configuration parameters and networking configurations of the containerized infrastructure compared with VM-based NFV infrastructure. Note that, although the detailed configurations of both infrastructures differ, the new benchmarks and metrics defined in [RFC8172] can be equally applied in containerized infrastructure from a generic-NFV point of view, and therefore defining additional metrics or methodologies is 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. Benchmarking Considerations

3.1. Comparison with the VM-based Infrastructure

For the 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.

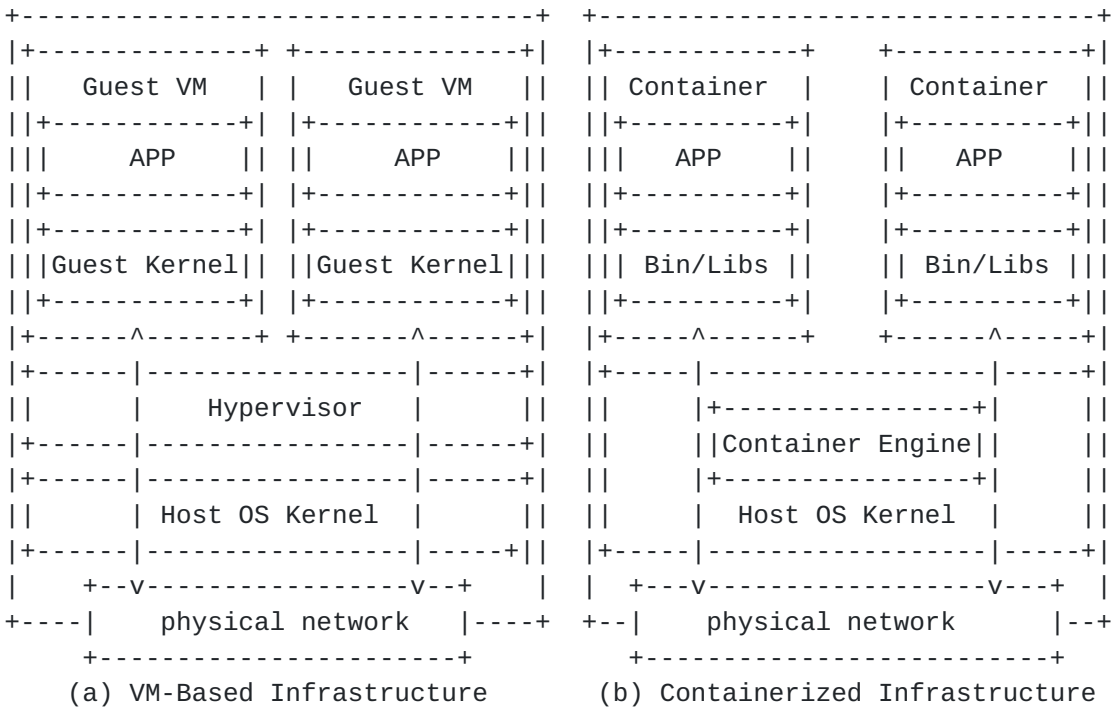

Figure 1: Comparison of NFV Infrastructures

In Figure 1, we describe two different NFV architectures: VM-based and Containerized. A major distinction between the containerized and the VM-based infrastructure is that with the former, all VNFs share the same host resources including but not limited to computing, storage and networking resources, as well as the host Operating System(OS), kernel and libraries. The absence of the guest OS and the hypervisor necessitates the following considerations that occur in the test environment:

o When we consider hardware configurations for the containerized infrastructure, all components described in [RFC8172] can be part of the test setup. While the capabilities of servers and storage should meet the minimum requirements for testing, it is possible to deploy a

test environment with fewer capabilities than in the VM-based infrastructure.

o About configuration parameters, the containerized infrastructure needs a specified management system instead of a hypervisor(e.g. Linux Container, Docker Engine).

o In the VM-based infrastructure, each VM manipulates packets in the kernel of the guest OS through its own CPU threads, virtualized and assigned by the hypervisor. On the other hand, C-VNFs use the host CPU without virtualization. Different CPU resource assignment methods may have different CPU utilization perspectives for performance benchmarking.

o From a Memory Management Unit(MMU) point of view, there are differences in how the paging process is conducted between two environments. The main difference lies in the isolated nature of the OS for VM-based VNFs. In the containerized infrastructure, memory paging which processes conversion between a physical address and the virtual address is affected by the host resource directly. Thus, memory usage of each C-VNFs is more dependent on the host resource capabilities than in VM-based VNFs.

3.2. Container Networking Classification

Container networking services are provided as network plugins. Basically, using them, network services are deployed by using an isolation environment from container runtime through the host namespace, creating a virtual interface, allocating interface and IP address to C-VNF. Since the containerized infrastructure has different network architecture depending on its using plugins, it is necessary to specify the plugin used in the infrastructure. There are two proposed models for configuring network interfaces for containers as below;

o CNM(Container Networking Model) proposed by Docker, using libnetwork which provides an interface between the Docker daemon and network drivers.

o CNI(Container Network Interface) proposed by CoreOS, describing network configuration files in JSON format and plugins are instantiated as new namespaces. Kubernetes uses CNI for providing network service.

Regardless of both CNM and CNI, the container network model can be classified into the kernel-space network model and user-space network model according to the location of network service creation. In the case of the kernel-based network model, network interfaces are

created in kernel space so that data packets should be processed in network stack of host kernel before transferring packets to the C-VNF running in user-space. On the other hand, using user-based network model, data packets from physical network port are bypassed kernel processing and delivered directly to user-space. Specific technologies for each network model and example of network architecture are written as follows:

o Kernel space network model: Docker Network[Docker-network], Flannel Network[Flannel], Calico[Calico], OVS(OpenvSwitch)[OVS], OVN(Open Virtual Network)[OVN], eBPF[eBPF]

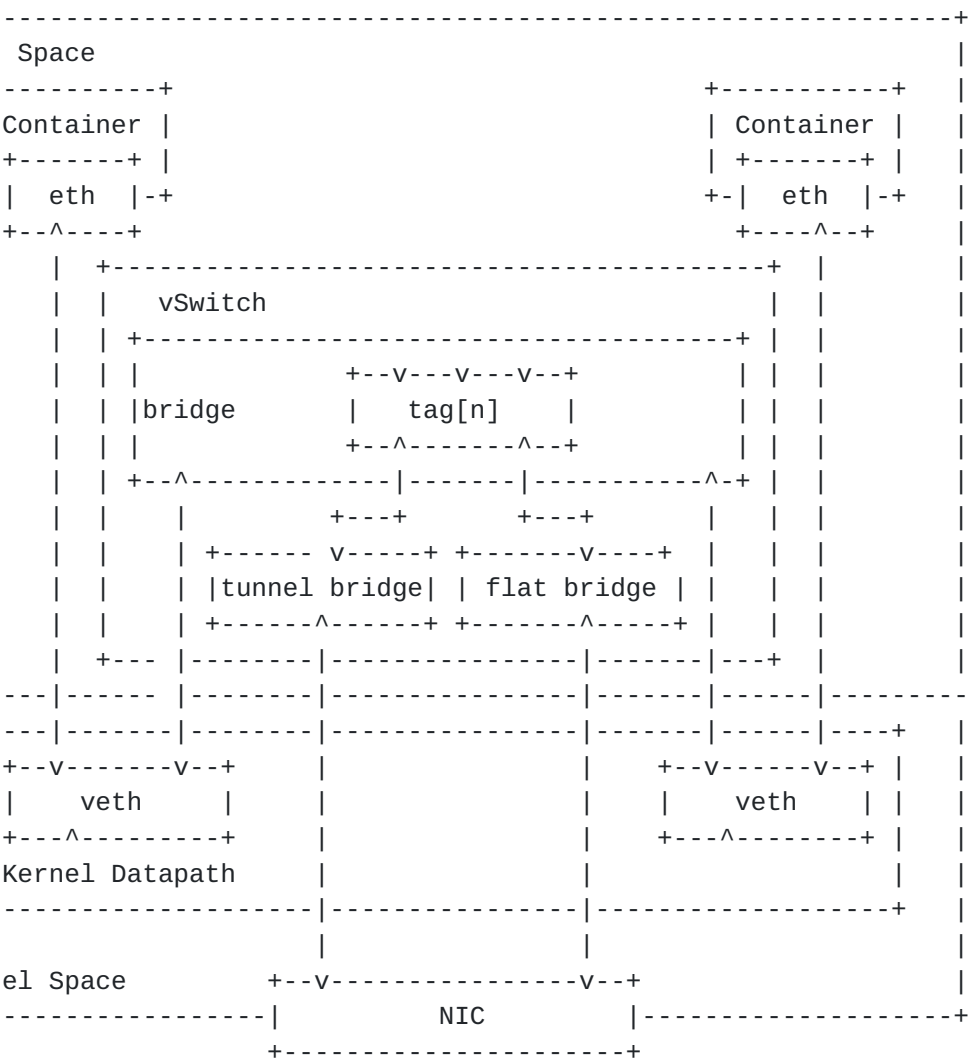

Figure 2: Examples of Kernel Space Network Model

o User space network model / Device pass-through model: SR-IOV[SR-IOV]

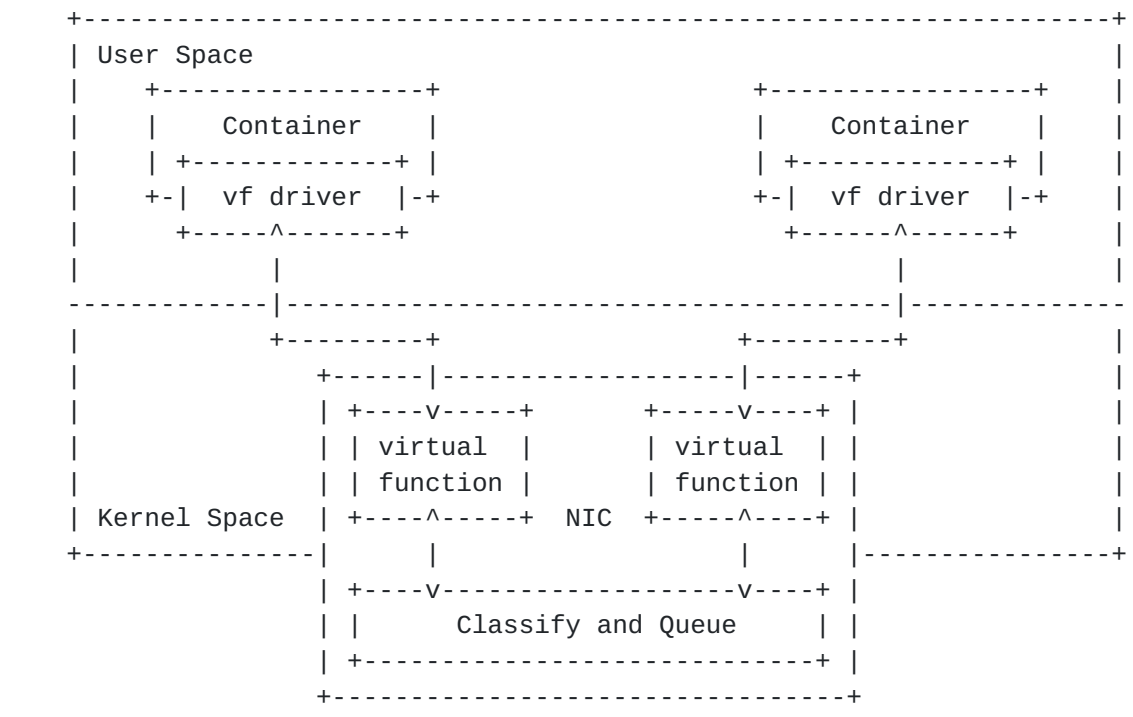

Figure 3: Examples of User Space Network Model - Device Pass-through

o User space network model / vSwitch model: ovs-dpdk[ovs-dpdk], vpp[vpp], netmap[netmap]

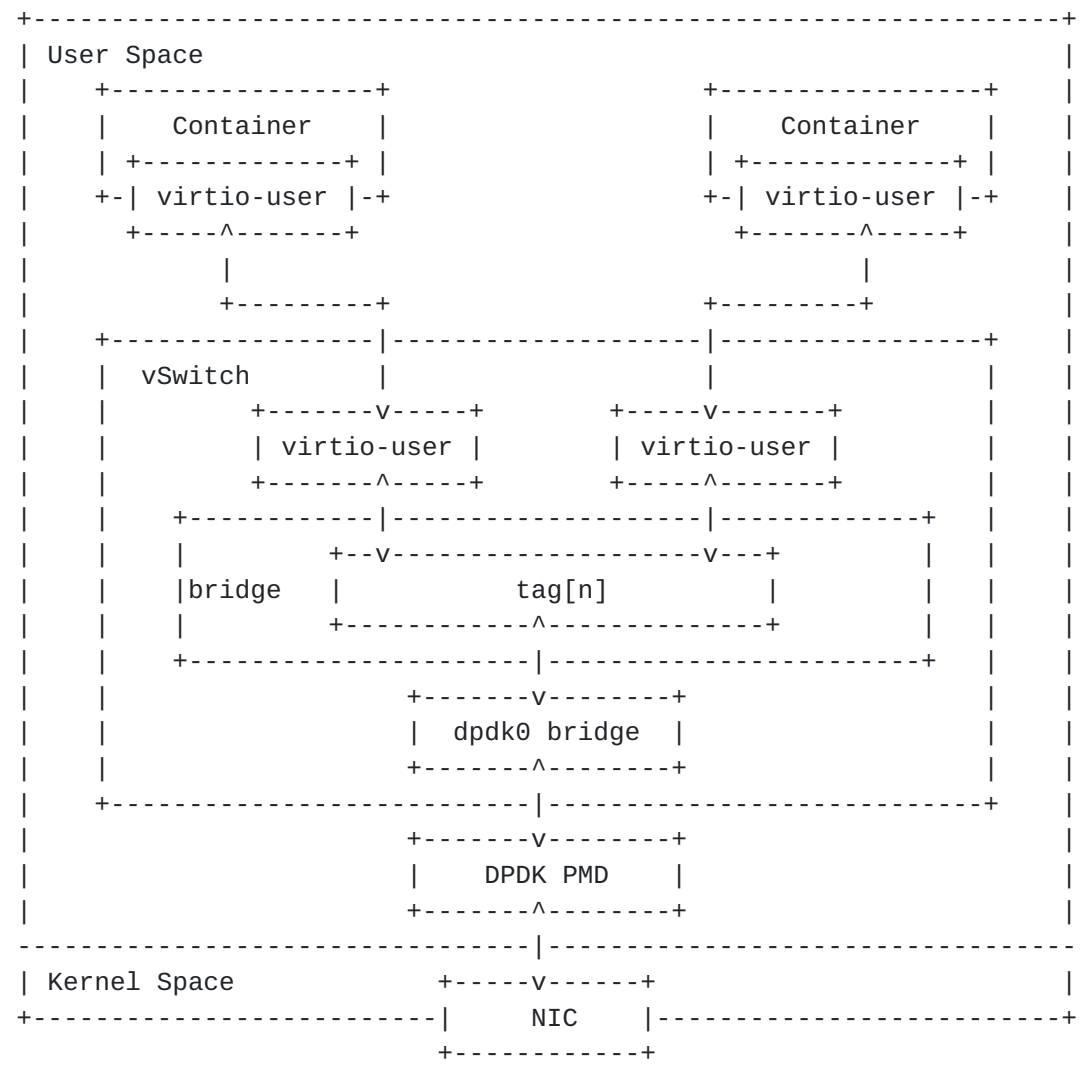

Figure 4: Examples of User Space Network Model - vSwitch Model using DPDK

3.3. Resource Considerations

In the containerized infrastructure, resource utilization and isolation may have different characteristics compared with the VM-based infrastructure. Some details are listed as follows:

o Hugepage

The huge page is that configuring 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 generally overall system performance. When using Cent OS or RedHat OS in the VM-based infrastructure, the huge

page should be set to at least 1G byte. In the VM-based infrastructure, the host OS and the hypervisor can configure a huge page depending on the guest OS. For example, guest VMs with the Linux OS requires to set huge pages at least 1G bytes. Even though it is a huge size, since this memory page is for not only its running application but also guest OS operation processes, actual memory pages for application is smaller.

In the containerized infrastructure, the container is isolated in 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 application use page more efficient way. Therefore, even if the page size is smaller than the VM, the effect of the huge page is large, which leads to the utilization of physical memory and the increasing number of functions in the host.

o NUMA

NUMA technology can be used both in the VM-based and containerized infrastructure. 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, it requires a strong understanding of VNF's memory requirements. If VNF uses more memory than a single NUMA node contains, the overhead will be occurred due to being spilled to another NUMA node.

In the VM-based infrastructure, the hypervisor can perform extracting NUMA topology and schedules VM workloads. In containerized infrastructure, however, it is more difficult to expose the NUMA topology to the container and currently, it is hard to guarantee the locality of memory when the container is deployed to host that has multiple NUMA nodes. For that reason, the instantiation of C-VNFs is somewhat non-deterministic and apparently NUMA-Node agnostic, which is one way of saying that performance will likely vary whenever this instantiation is performed. So, when we use NUMA in the containerized infrastructure, repeated instantiation and testing to quantify the performance variation is required.

o RX/TX Multiple-Queue

RX/TX Multiple-Queue technology[Multique], which enables packet sending/receiving processing to scale with the number of available vcpus of guest VM, may be used to enhance network performance in the VM-based infrastructure. However, RX/TX Multiple-Queue technology is not supported in the containerized infrastructure yet.

4. Benchmarking Scenarios for the Containerized Infrastructure

Figure 5 shows briefly differences of network architectures based on deployment models. Basically, on bare metal, C-VNFs can be deployed as a cluster called POD by Kubernetes. Otherwise each C-VNF can be deployed separately using Docker. In the former case, there is only one external network interface even a POD contains more than one C-VNF. An additional deployment model considers a scenario in which C-VNFs or PODs are running on VM. In our draft, we define new terminologies; BMP which is Pod on bare metal and VMP which is Pod on VM.

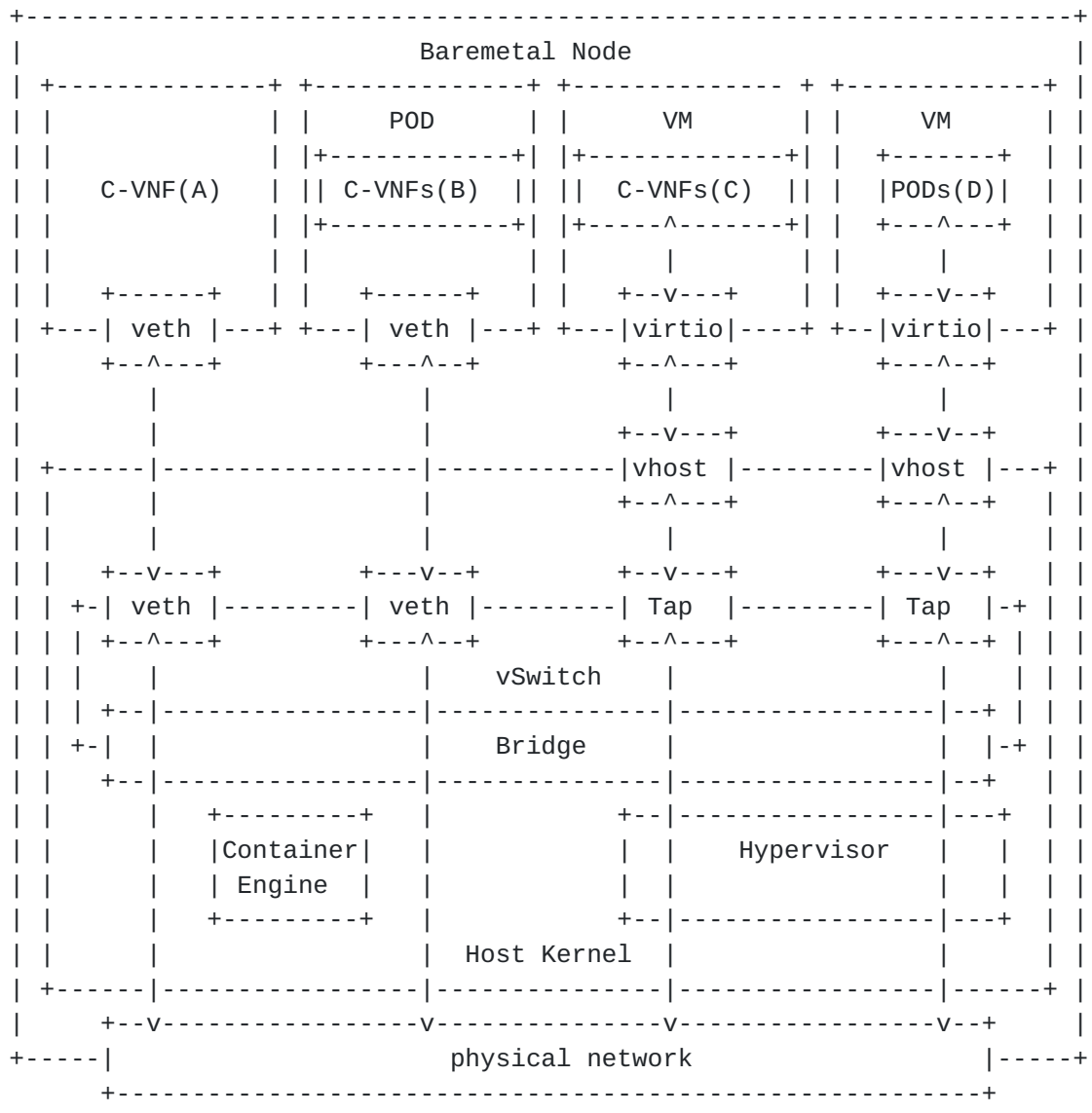

Figure 5: Examples of Networking Architecture based on Deployment Models - (A)C-VNF on Baremetal (B)Pod on Baremetal(BMP) (C)C-VNF on VM (D)Pod on VM(VMP)

In [ETSI-TST-009], they described data plane test scenarios in a single host. In that document, there are two scenarios for containerized infrastructure; Container2Container which is internal communication between two containers in the same Pod, and the Pod2Pod model which is communication between two containers running in different Pods. According to our new terminologies, we can call the Pod2Pod model as the BMP2BMP scenario. When we consider container running on VM as an additional deployment option, there can be more single host test scenarios as follows;

o BMP2VMP scenario

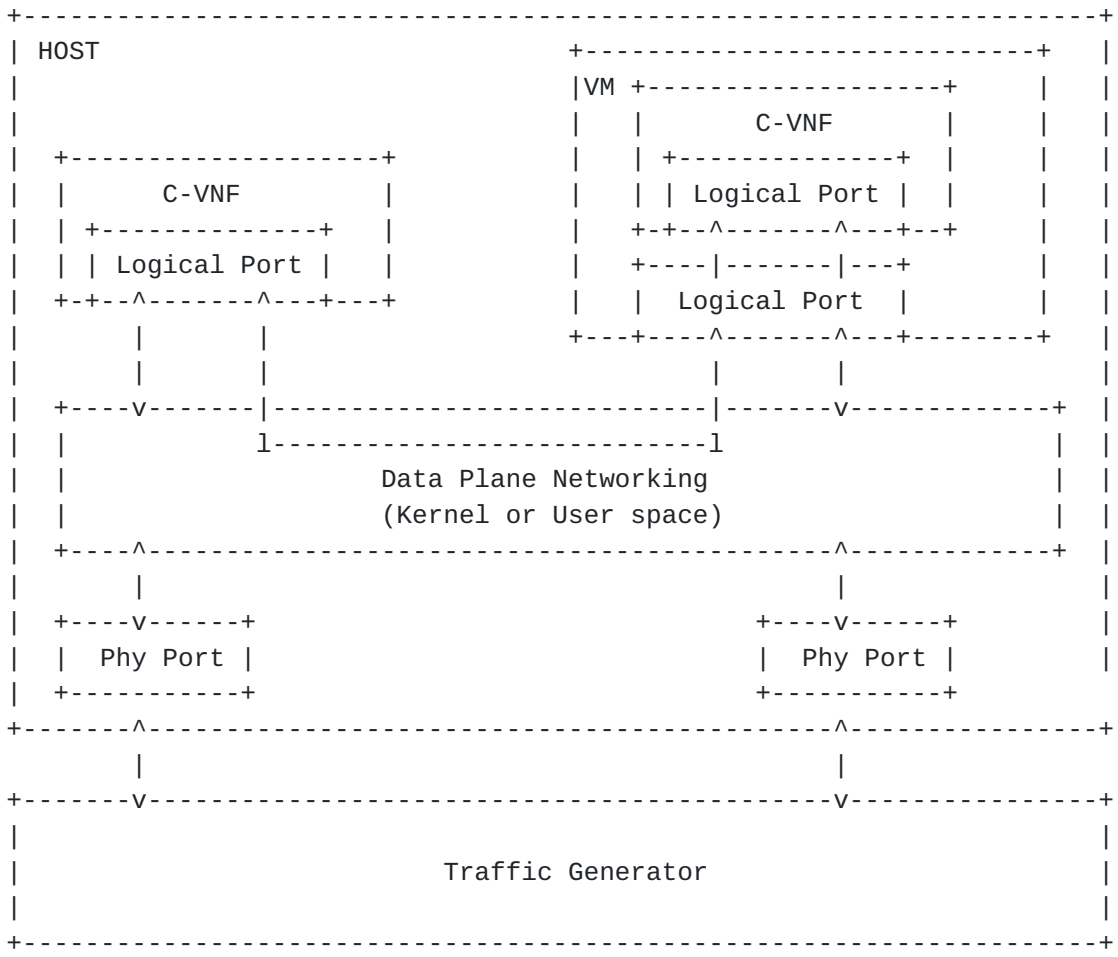

Figure 6: Single Host Test Scenario - BMP2VMP

o VMP2VMP scenario

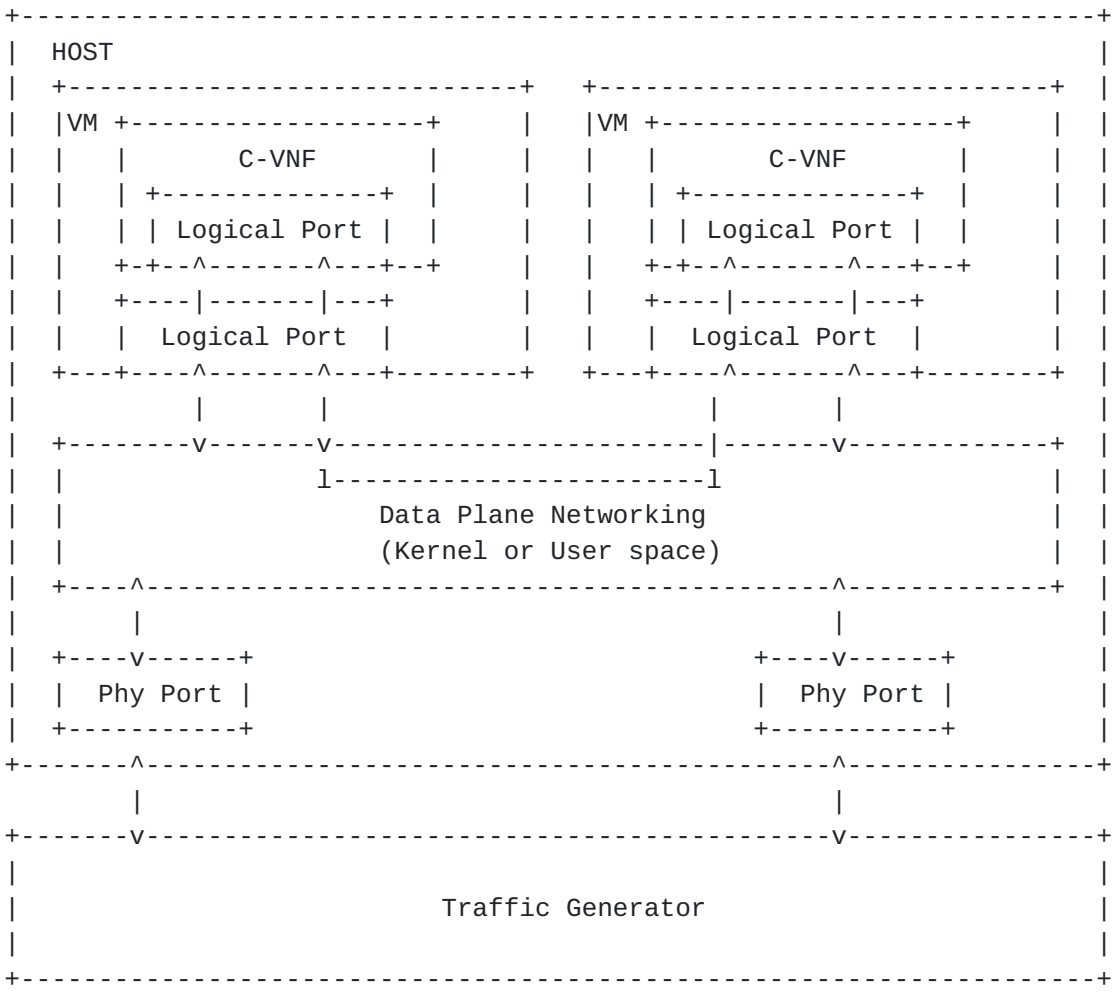

Figure 7: Single Host Test Scenario - VMP2VMP

5. Additional Considerations

When we consider benchmarking for not only containerized but also VM-based infrastructure and network functions, benchmarking scenarios may contain various operational use cases. Traditional black-box benchmarking is focused to measure in-out performance of packet 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. Therefore, benchmarking scenarios should reflect operational considerations such as number of VNFs or network services defined by a set of VNFs in a single host. [service-density], which proposed a way for measuring the performance of multiple NFV service instances at a varied service density on a

single host, is one example of these operational benchmarking aspects.

Regarding the above draft, it can be classified into two types of traffic for benchmark testing. One is North/South traffic and the other is East/West traffic. North/South has a architecture that receives data from other servers and routes them through VNF. 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. The one of the example is Service Function Chaining. Since network acceleration technology in a container environment has different accelerated areas depending on the method provided, performance differences may occur depending on traffic patterns.

6. Benchmarking Experience(Contiv-VPP)

6.1. Benchmarking Environment(Contiv-VPP)

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

o Three physical server for benchmarking

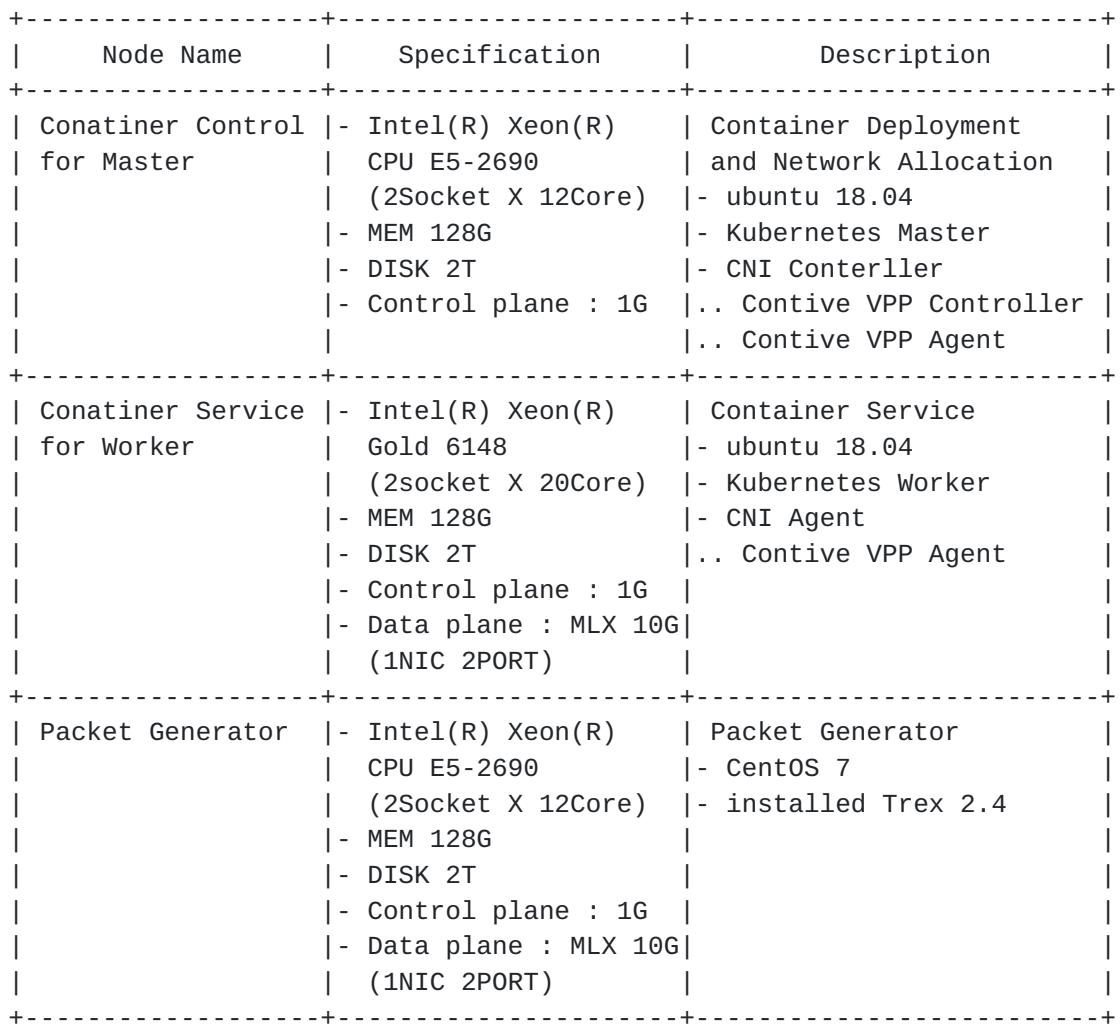

Figure 8: Test Environment-Server Specification

o The architecture of benchmarking

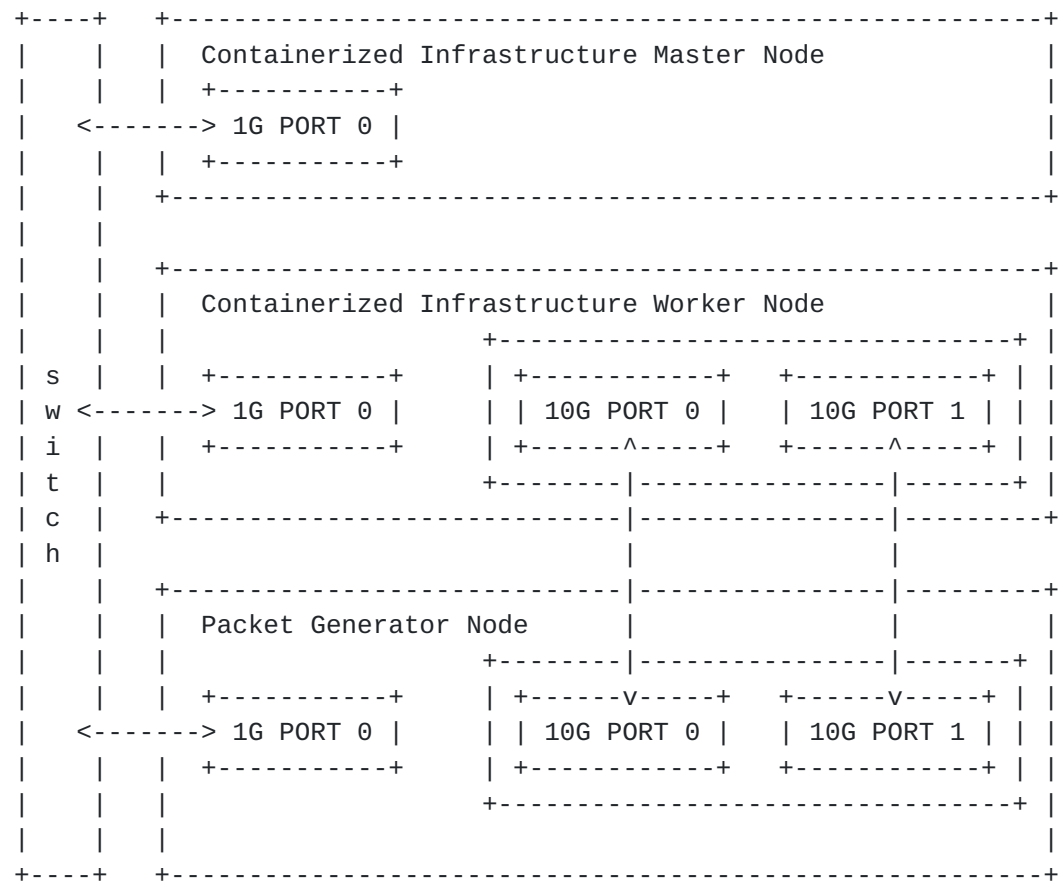

Figure 9: Test Environment-Architecture

o Network model of Containerized Infrastructure(User space Model)

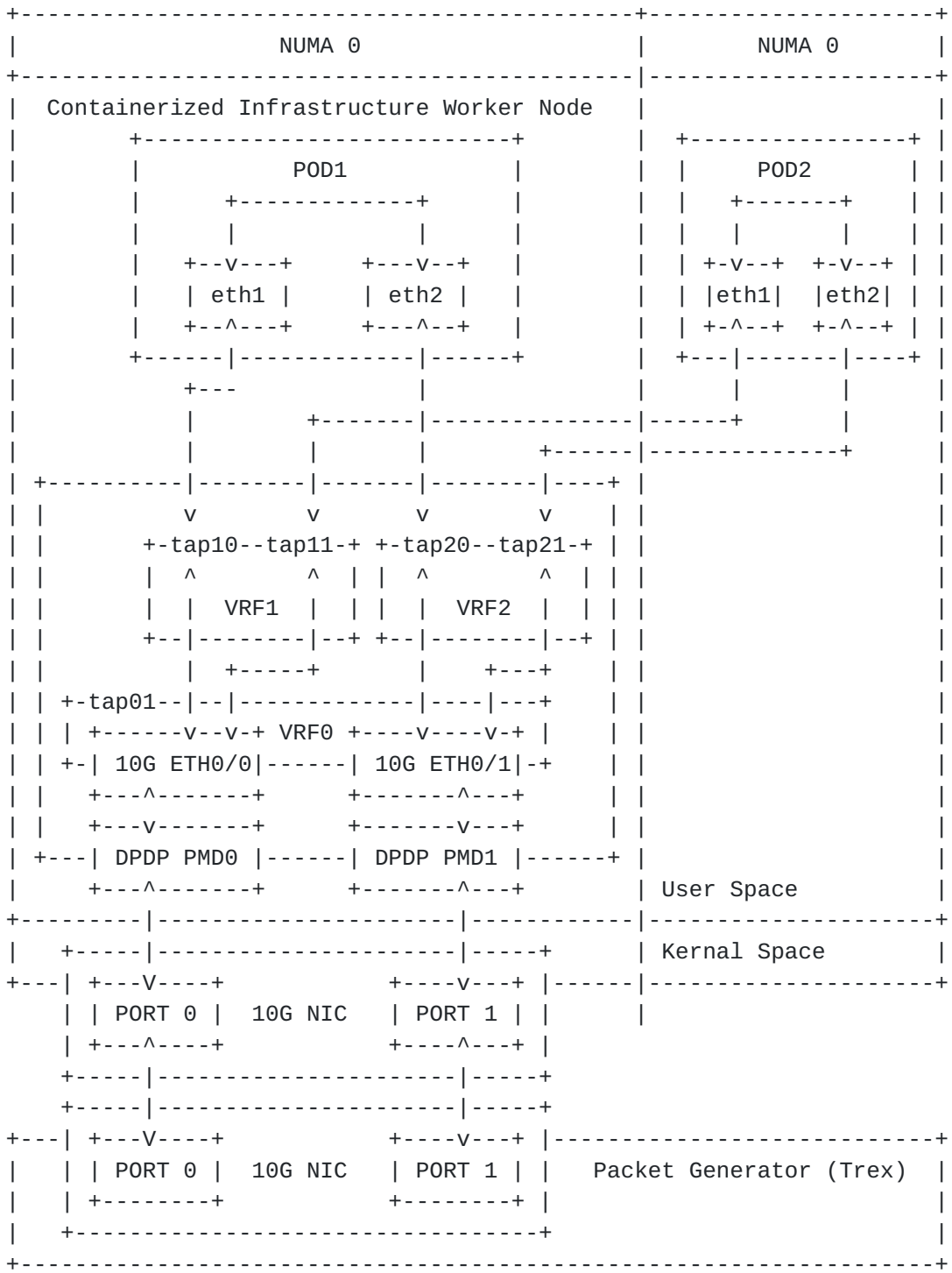

Figure 10: Test Environment-Network Architecture

We setup a Contive-VPP network to benchmark the user space container network model in the containerized infrastructure worker node. We setup network interface at NUMA0, and we created different network subnet VRF1, VRF2 to classify input and output data traffic,

respectively. And then, we assigned two interface which connected to VRF1, VRF2 and, we setup routing table to route Trex packet from eth1 interface to eth2 interface in POD.

6.2. Trouble shooting and Result

In this environment, we confirmed that the routing table doesn't work when we send packet 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. That is, 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.

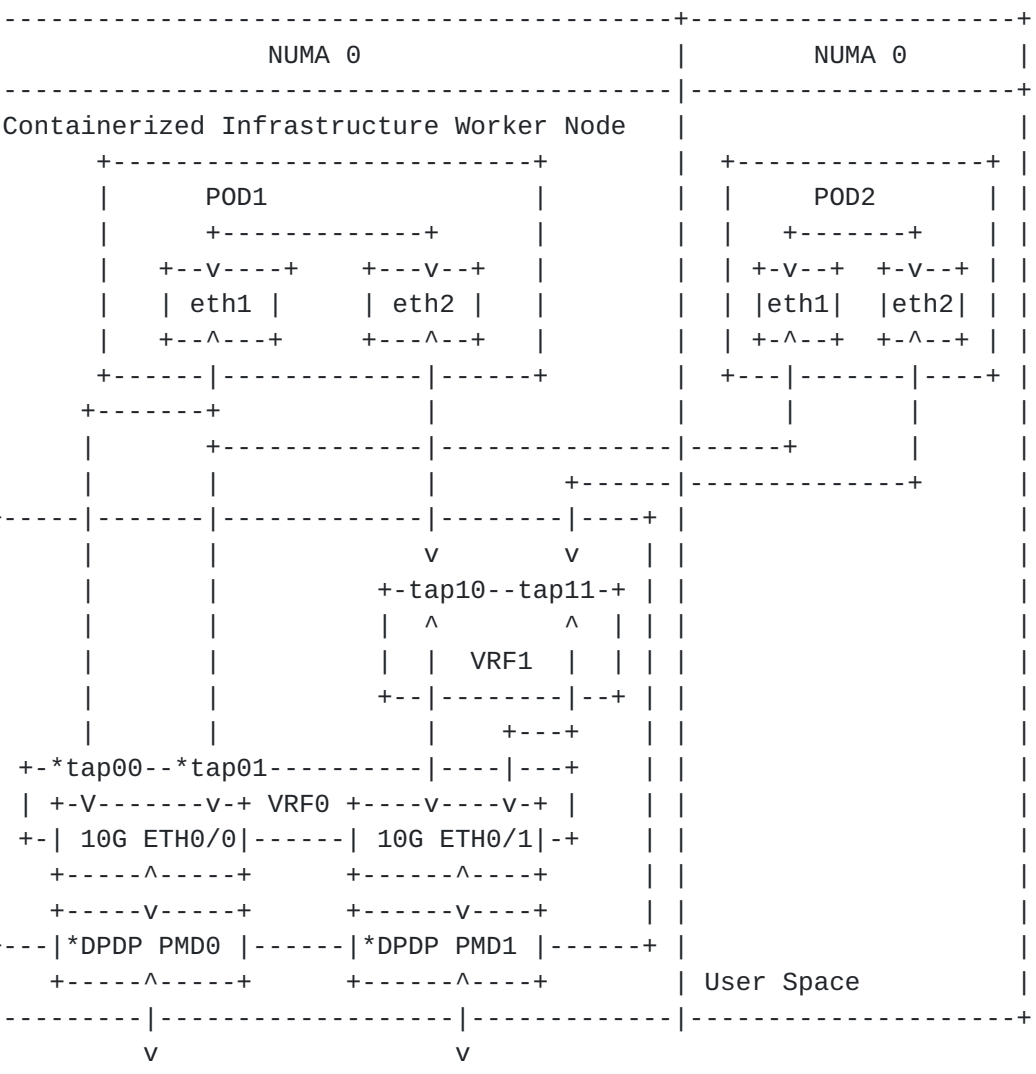

Figure 11: Test Environment-Network Architecture(CPU Pinning)

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 a specific PODs. Exclusive mode is that a specific POD dedicates a specific CPU to use. In shared mode, we assigned two CPU for several POD, 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 network interface using Shared Mode and Exclusive Mode in the same node and different node respectively. In Exclusive and Shared mode tests, we confirmed that Exclusive mode showed better performance than Shared mode when same

NUMA cpu assigned, respectively. However, we confirmed that performance is reduced at the section between the vpp switch and the POD, so that it affect to total result.

Model	NUMA Mode (pinning)	Result(Gbps)
Switch only	N/A	3.1
	same NUMA	9.8
K8S Scheduler	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

7. Benchmarking Experiment(SR-IoV-DPDK)

7.1. Benchmarking Environment(SR-IoV-DPDK)

In this test, our purpose is that we test performance of user space based 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

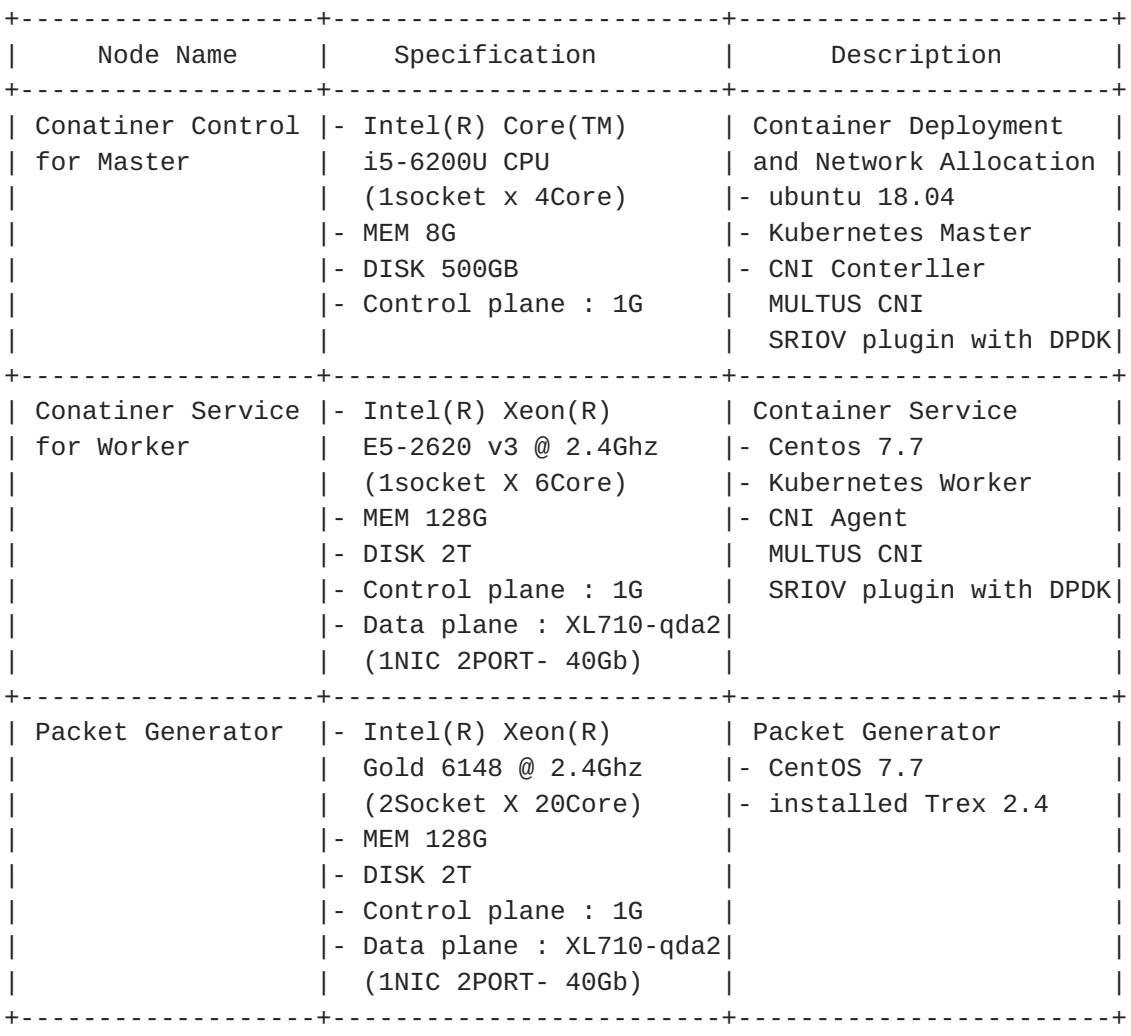

Figure 13: Test Environment-Server Specification

o The architecture of benchmarking

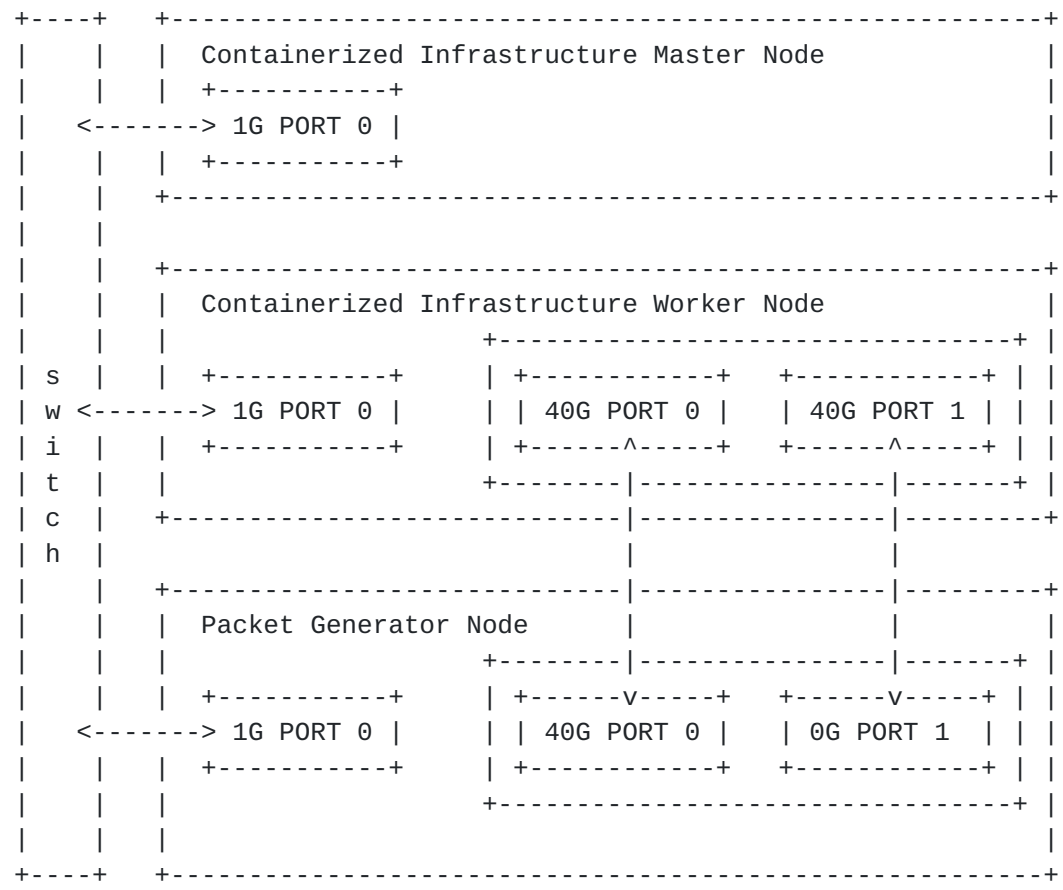

Figure 14: Test Environment-Architecture

o Network model of Containerized Infrastructure(User space Model)

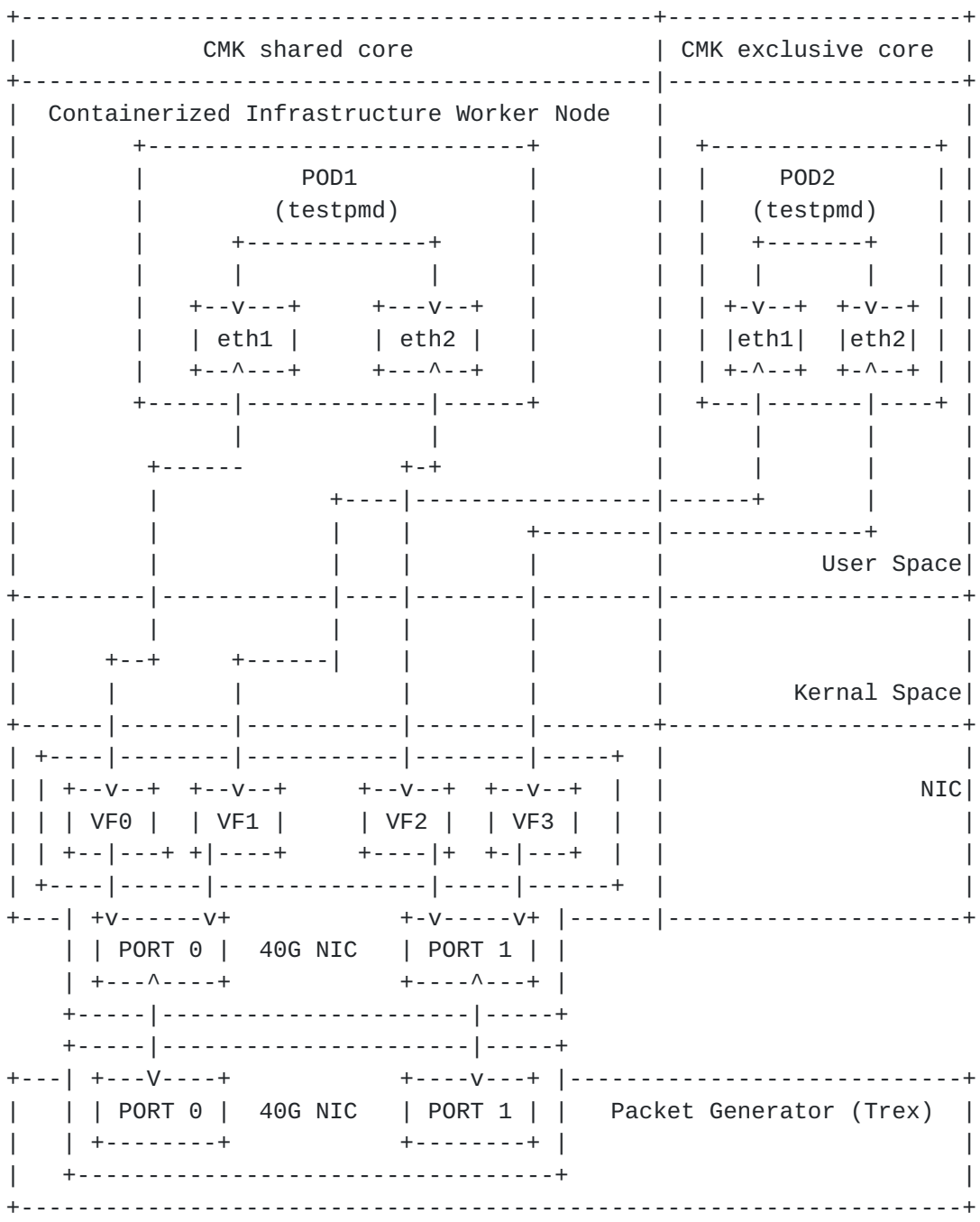

Figure 15: Test Environment-Network Architecture

We setup 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 to create 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 setup TREX to route packet from eth1 to eth2 in a POD.

7.2. Trouble shooting and Result(SR-IoV-DPDK)

TBD

8. Security Considerations

TBD

9. Acknkowledgement

We would like to thank Al, Maciek and Luis who reviewed and gave comments of previous draft.

10. Informative References

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

[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-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/>.

[Multique] "Multiqueue virtio-net", July 2019, <https://www.linux-kvm.org/page/Multiqueue>.

[netmap] "Netmap: a framework for fast packet I/O", July 2019, <https://github.com/luigirizzo/netmap>.

[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.

[RFC8172] Morton, A., "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure", RFC 8172, July 2017.

[RFC8204] Tahhan, M., O'Mahony, B., and A. Morton, "Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV)", RFC 8204, September 2017.

[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>.

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

Authors' Addresses

Kyoungjae Sun
School of Electronic Engineering
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul, Seoul 06978
Republic of Korea

Phone: +82 10 3643 5627
EMail: gomjae@dcn.ssu.ac.kr

Hyunsik Yang
School of Electronic Engineering
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul, Seoul 06978
Republic of Korea

Phone: +82 10 9005 7439
EMail: yangun@dcn.ssu.ac.kr

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

Phone: +82 10 7448 4664
EMail: jangwon.lee@dcn.ssu.ac.kr

Quang Huy Nguyen
School of Electronic Engineering
Soongsil University
369, Sangdo-ro, Dongjak-gu
Seoul, Seoul 06978
Republic of Korea

Phone: +82 10 4281 0720
EMail: huynq@dcn.ssu.ac.kr

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

Phone: +82 10 2691 0904
EMail: younghak@ssu.ac.kr