Workgroup:
Benchmarking Methodology Working Group
Internet-Draft:
draft-dcn-bmwg-containerized-infra-13
Published: October 2023
Intended Status: Informational
Expires: 10 April 2024
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 network functions. This draft describes additional considerations for benchmarking network performance when network functions are containerized and performed in general-purpose hardware.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 April 2024.

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. Most benchmarking methodologies and configuration parameters specified in [RFC8172] and [RFC8204] can be equally applied to benchmark container networking. However, major architecture differences between virtual machine (VM)-based and container-based infrastructure cause additional considerations.

In terms of virtualization method, containerized network functions (CNF) are virtualized using the host operating system (OS) virtualization instead of hypervisor-based hardware virtualization in VM-based infrastructure. In comparison to VMs, containers do not have a separate hardware and kernel. CNFs share the same kernel space on the same host, while their resources are logically isolated in different namespaces. Hence, benchmarking container network performance might require different resources configuration settings.

In terms of networking, to route traffic between containers which are isolated in different network namespaces, a container network plugin is required. Initially, when a pod or container is first instantiated, it has no network. container network 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 plugins' networking technologies, and how the packet is processed/accelerated via the kernel-space and/or the user-space of the host, these plugins can be categorized into different container networking models. These models should be considered while benchmarking container network performance.

2. Requirements Language

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].

3. Terminology

This document uses the terminology described in [RFC8172], [RFC8204], [ETSI-TST-009].

Besides, with the proliferation and popularity of Kubernetes as a container orchestration platform, this document uses Kubernetes' terminologies for general containerized infrastructure.

Pod is defined as a basic and smallest unit for orchestration and management that can host multiple containers, with shared storage and network resources. Generally, each CNF is deployed as a container in a single pod. In this document, the terms container and pod are used interchangeably.

Container Network Interface (CNI) plugin is the framework that dynamically create and configure network for containers.

4. Scope

The primary scope of this document is to fill in the gaps of previous BMWG's NFV benchmarking consideration works ([RFC8172] and [RFC8204]) when applying to containerized NFV infrastructure. The first gap is different network models/topologies configured by container network interfaces (especially the extended Berkeley Packet Filter model which was not mentioned in previous documents). The other gap is resources configuration for containers. This document investigates these gaps as additional benchmarking considerations for NFV infrastructure.

Note that apart from the unique characteristics, benchmarking test and assessment methodologies defined in the above mentioned RFCs can be equally applied to containerized infrastructure from a generic-NFV point of view.

5. Benchmarking Considerations

5.1. Networking Models

Compared with VNFs, selected CNI Plugin is an important software detail parameter for containerized infrastructure benchmarking. Different CNI plugins configure different network architecture for CNFs in terms of network interfaces, virtual switch usage, and

packet acceleration techniques. This section categorizes container networking models based on CNI plugin characteristics.

Note that mentioned CNI plugins in each category are notable examples, and any other current CNI plugins can fall into one of the categories mentioned in this section.

5.1.1. Kernel-space non-Acceleration Model

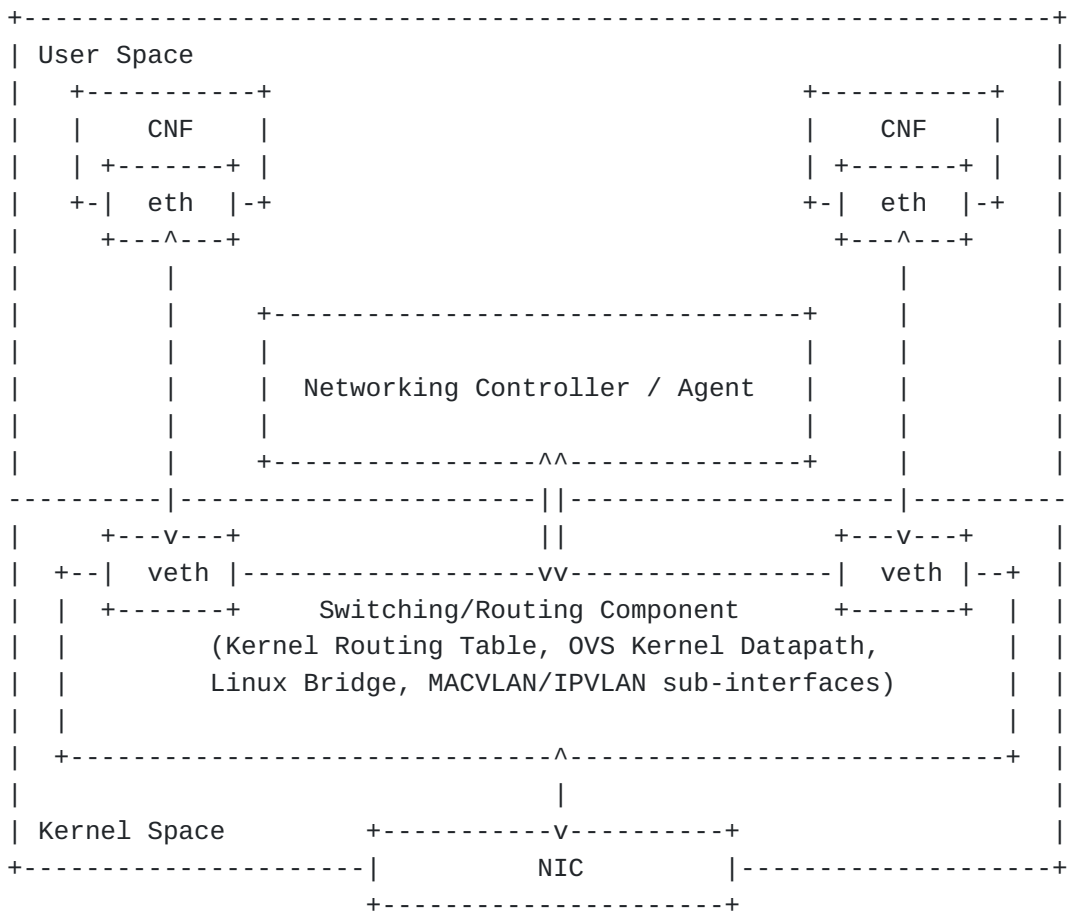

Figure 1: Example architecture of the Kernel-Space non-Acceleration Model

Figure 1 shows kernel-space non-Acceleration model. In this model, the virtual ethernet (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 CNF 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

5.1.2. User-space Acceleration Model

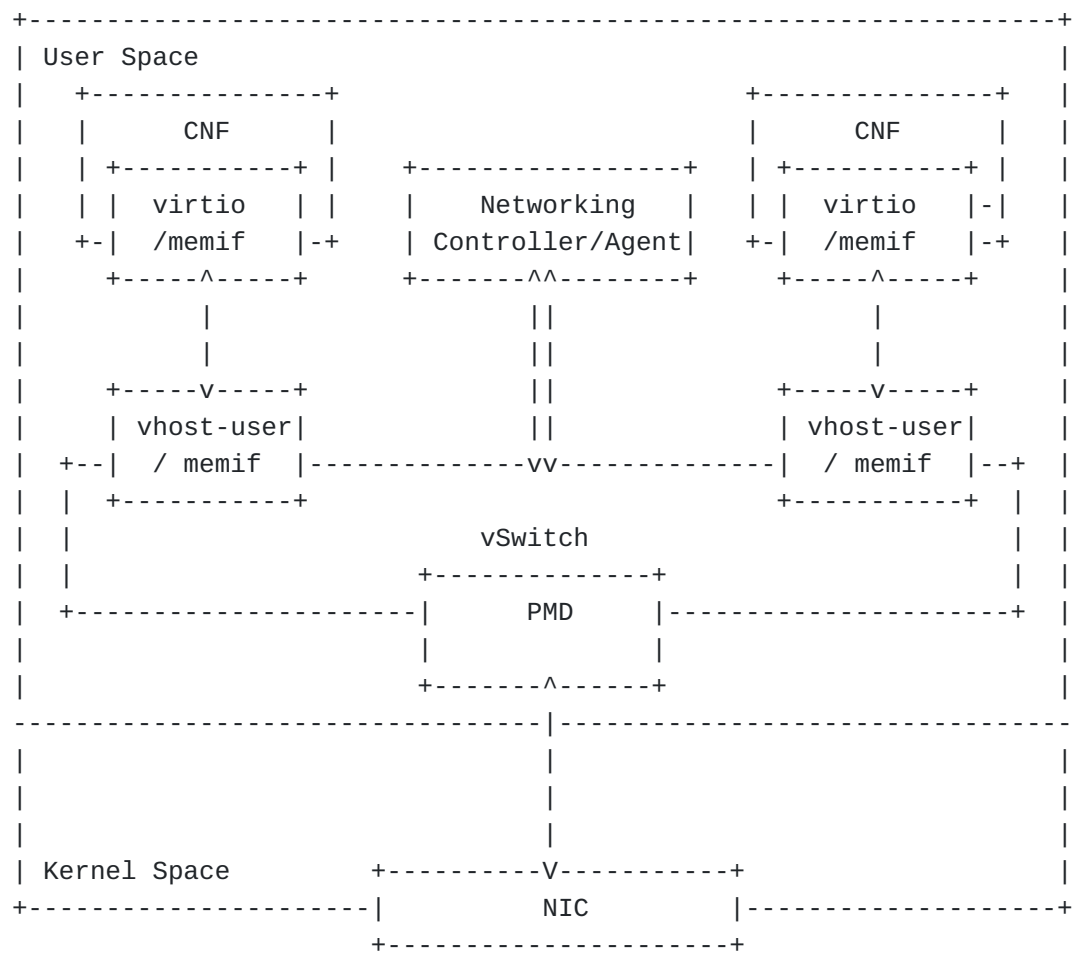

Figure 2: Example architecture of the User-Space Acceleration Model

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 network interface (virtio or memif) at each container. User-space vSwitch models are listed below:

o OVS-DPDK [ovs-dpdk], VPP [vpp]

5.1.3. eBPF Acceleration Model

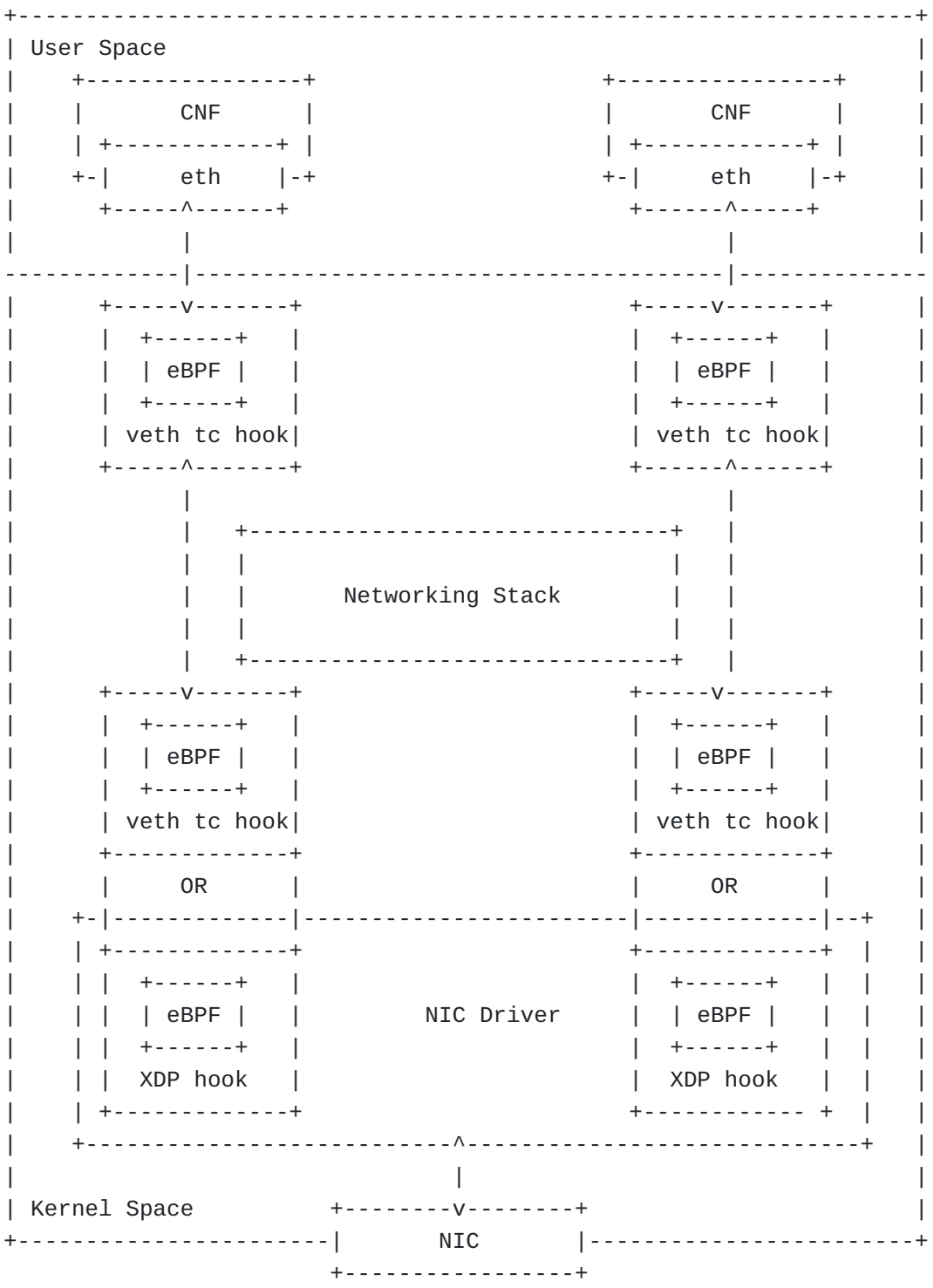

Figure 3: Example architecture of the eBPF Acceleration Model - non-AFXDP

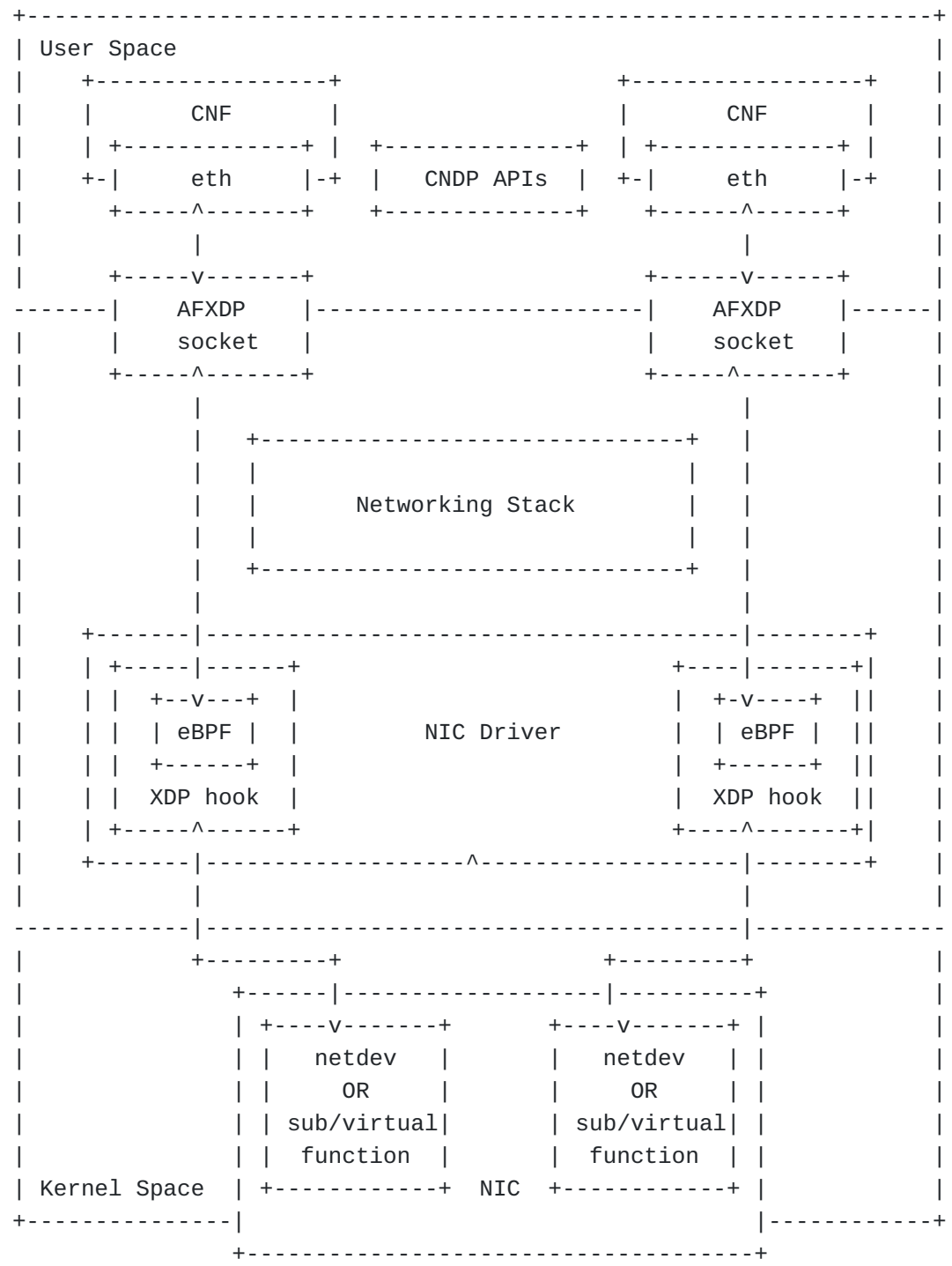

Figure 4: Example architecture of the eBPF Acceleration Model - using AFXDP supported CNI

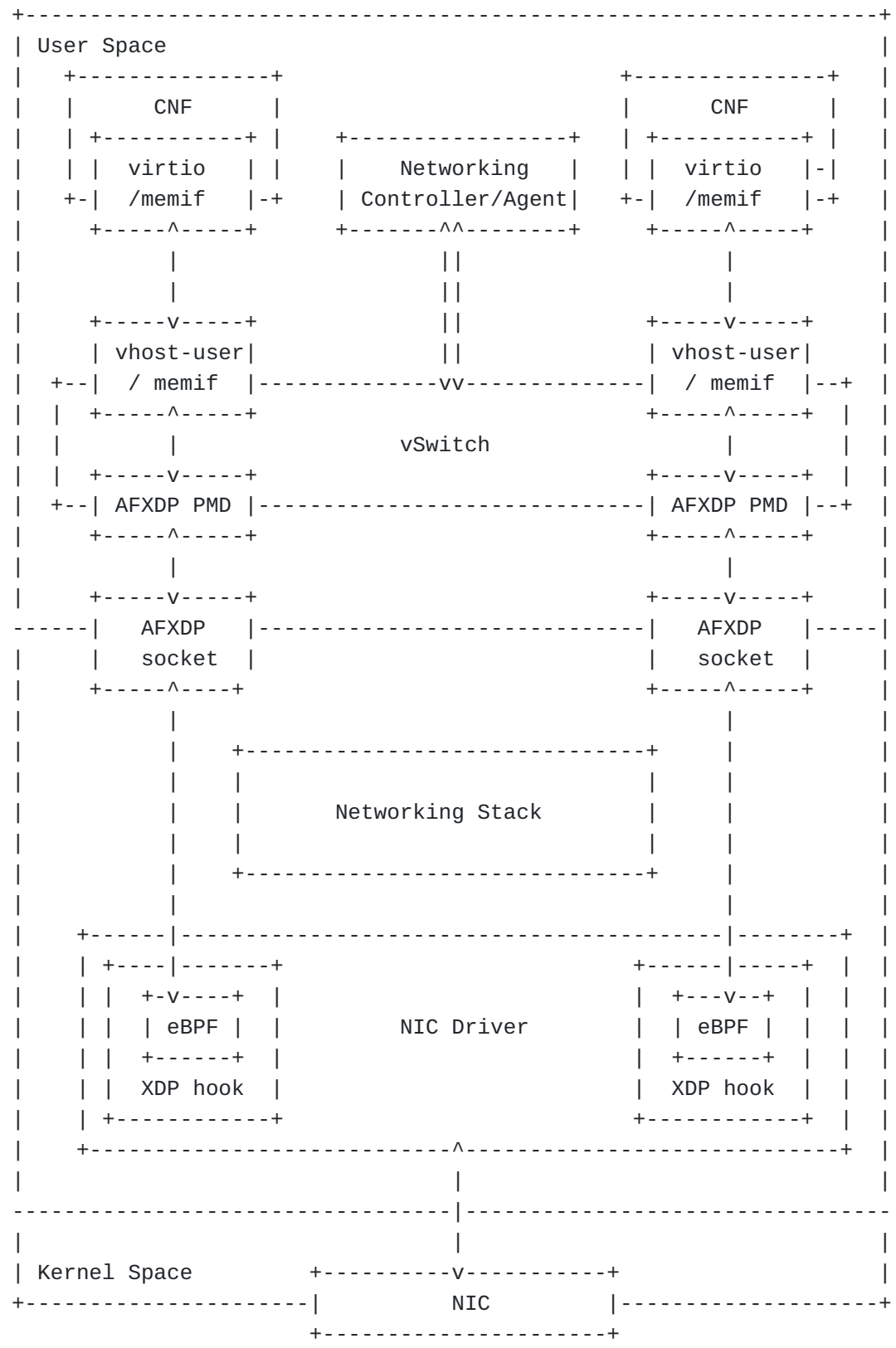

Figure 5: Example architecture of the eBPF Acceleration Model - using user-space vSwitch which support AFXDP PMD

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.

5.1.4. Smart-NIC Acceleration Model

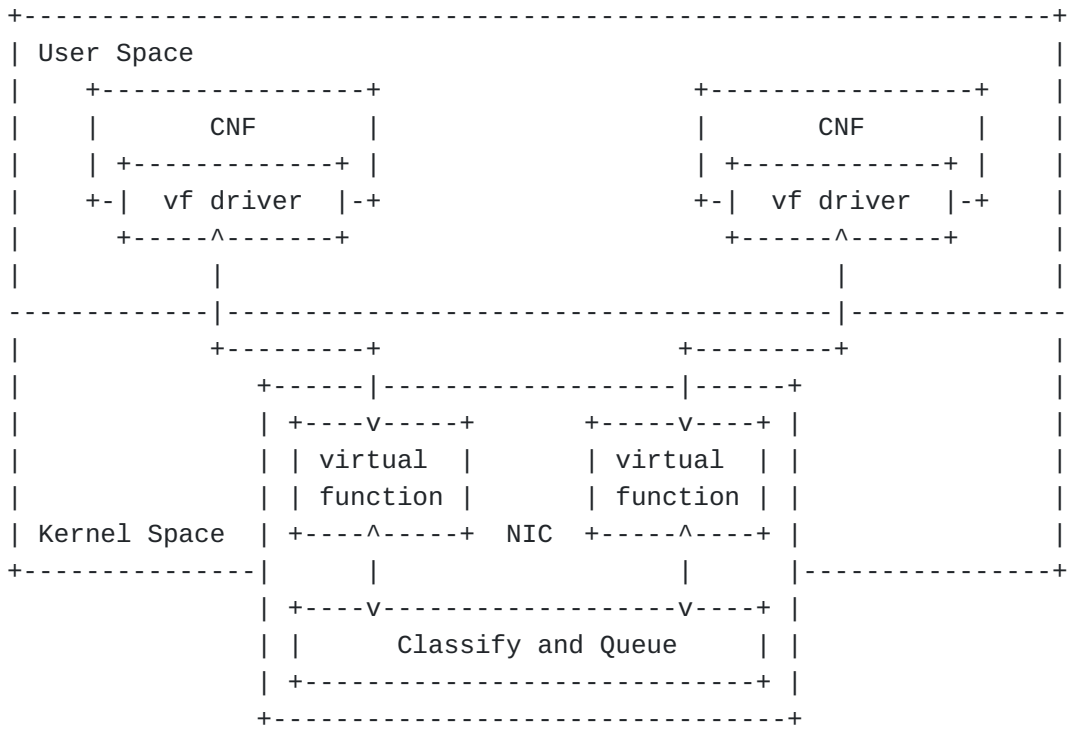

Figure 6: Examples of Smart-NIC Acceleration Model

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 special interface at each container controlled by the VF driver.

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.

5.1.5. Model Combination

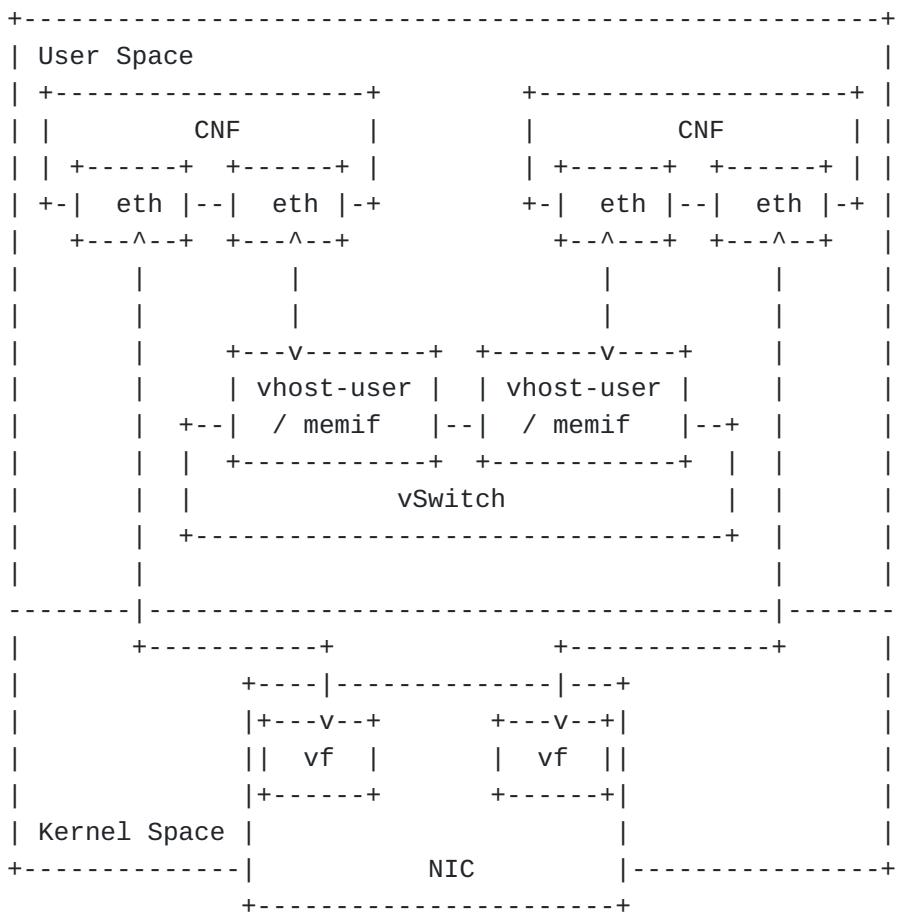

Figure 7: Examples of Model Combination deployment

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 CNF. 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].

5.2. Resources Configuration

The resources configuration consideration list here is not only applied for the CNF but also other components in a containerized SUT. A Containerized SUT is composed of NICs, possible cables between hosts, kernel and/or vSwitch, and CNFs.

5.2.1. 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. One example technology of CPU Pinning in containerized infrastructure is the CPU Manager for Kubernetes (CMK) [CMK]. This technology was proved to be effective in avoiding the "noisy neighbor" problem, as shown in an existing experience [Intel-EPA]. Besides, CPU Isolation techniques' benefits are not only applied for "noisy neighbor" problem. Different CNFs also neighbor each other and neighbor vSwitch if used.

NUMA affects the speed of different CPU cores when accessing different memory regions. CPU cores in the same NUMA nodes can locally access to the shared memory in that node, which is faster than remotely accessing the memory in a different NUMA node. In containerized network, packet forwarding is processed through NIC, CNF and a possible vSwitch based on chosen networking model. NIC's NUMA node alignment can be checked via the PCI devices' node affinity. Meanwhile, specific CPU cores can be direclty assigned to CNF and vSwtich via their configuration settings. 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, vSwitch 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 CNF sharing CPUs across NUMA

Note that CPU Pinning and NUMA Affinity configurations considerations might also applied to VM-based VNF. As mentioned above, dedicated CPU cores of a specific NUMA node can be assigned to VNF and vSwitch via their own running configurations. NIC's NUMA node can be checked from the PCI devices' infomration. Host's NUMA nodes can be scheduled to virtual machines by specifying in their settings the chosen nodes.

For this consideration, the additional configuration parameters should be considered for containerized infrastructure benchmarking are:

- Selected CPU Isolation level

- NUMA cores allocation to pod

5.2.2. Pod 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 2M bytes or 1G bytes huge pages for the container). 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.

For this consideration, the additional configuration parameters should be considered for containerized infrastructure benchmarking are:

- Pod's hugepage size

5.2.3. Pod 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.

For this consideration, the additional configuration parameters should be considered for containerized infrastructure benchmarking are:

- Pod's CPU cores allocation

- Pod's RAM allocation

5.2.4. 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 CNFs(i.e., Multiple PVP test setup architectures were described in [ETSI-TST-009]) rather than dedicated to a single CNF. This scenario is called Service Function Chaining. Therefore, benchmarking scenarios should reflect operational considerations such as the number of CNFs 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.

For this consideration, the additional configuration parameters should be considered for containerized infrastructure benchmarking are:

- Number of CNFs/pod

- Selected CNI Plugin

5.2.5. 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 CNFs 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.

6. 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.

7. References

7.1. 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>.

[CMK] Intel, "Userspace CNI Plugin", February 2021, <https://github.com/intel/CPU-Manager-for-Kubernetes>.

[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>.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, March 1999, <https://www.rfc-editor.org/rfc/rfc2544>.

[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] Intel, "CPU Manager for Kubernetes", 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. Change Log (to be removed by RFC Editor before publication)

A.1. Since draft-dcn-bmwg-containerized-infra-12

Updated scope to clearly specify the gaps of related RFCs.

A.2. Since draft-dcn-bmwg-containerized-infra-11

Merged Containerized infrastructure overview into Introduction section

Added Scope section which briefly explains the draft contribution in a clear way.

Mentioned the additional benchmarking configuration parameters for containerized infrastructure benchmarking in each Benchmarking Consideration sub-sections.

Removed Benchmarking Experiences Appendixes

A.3. Since draft-dcn-bmwg-containerized-infra-10

Updated Benchmarking Experience appendixes with latest results from Hackathon events.

Re-orgianized Benchmarking Experience appendixes to match with the the proposed benchmarking consideration inside the draft (Networking Models and Resources Configuration)

Minor enhancement changes to Introduction and Resource Configuration consideration sections such as general description for container network plugin, which resource can also be applied for VM-VNF.

A.4. Since draft-dcn-bmwg-containerized-infra-09

Removed 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

Removed 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

Renamed 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

A.5. 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.

A.6. 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

A.7. Since draft-dcn-bmwg-containerized-infra-06

Added Benchmarking Experience of Multi-pod Test

A.8. 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

A.9. Since draft-dcn-bmwg-containerized-infra-04

Added Benchmarking Experience of SRIOV-DPDK.

A.10. Since draft-dcn-bmwg-containerized-infra-03

Added Benchmarking Experience of Contiv-VPP.

A.11. Since draft-dcn-bmwg-containerized-infra-02

Editorial changes only.

A.12. Since draft-dcn-bmwg-containerized-infra-01

Editorial changes only.

A.13. 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

Minh-Ngoc Tran
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