Benchmarking Methodology Working Group
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

M. N. Tran
N. Ko
ETRI
S. Rao
The Linux Foundation
J. Lee
Y. Kim
Soongsil University

20 October 2025

Considerations for Benchmarking Network Performance in Containerized Infrastructures
draft-ietf-bmwg-containerized-infra-08

Abstract

Recently, the Benchmarking Methodology Working Group extended the laboratory characterization from physical network functions (PNFs) to virtual network functions (VNFs). With the ongoing shift in network function implementation from virtual machine-based to container-based approaches, system configurations and deployment scenarios for benchmarking will be partially influenced by how resource allocation and network technologies are specified for containerized network functions. This draft outlines additional considerations for benchmarking network performance when network functions are containerized and executed on 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 23 April 2026.

Table of Contents

1. Introduction 3
2. Requirements Language 4
3. Terminology 4
4. Scope 4
5. Benchmarking Considerations 5
5.1. Networking Models 5
5.1.1. Normal non-Acceleration Networking Model 5
5.1.2. Acceleration Networking Models 7
5.1.2.1. User-space Acceleration Model 8
5.1.2.2. eBPF Acceleration Model 10
5.1.2.3. Smart-NIC Acceleration Model 15
5.1.2.4. Model Combination 16
5.2. Resources Configuration 18
5.2.1. Container Deployment Environment 18
5.2.2. CPU Isolation / NUMA Affinity 18
5.2.3. Pod Hugepages 19
5.2.4. Pod CPU Cores and Memory Allocation 20
5.2.5. AF_XDP Configuration 21
5.2.6. Service Function Chaining 21
5.2.7. Other Considerations 22
6. IANA Considerations 22
7. Security Considerations 22
8. References 22
8.1. Informative References 22
Appendix A. Change Log (to be removed by RFC Editor before publication) 25
A.1. Since draft-ietf-bmwg-containerized-infra-07 25
A.2. Since draft-ietf-bmwg-containerized-infra-06 25
A.3. Since draft-ietf-bmwg-containerized-infra-03 25
A.4. Since draft-ietf-bmwg-containerized-infra-01 26
A.5. Since draft-ietf-bmwg-containerized-infra-00 26
A.6. Since draft-dcn-bmwg-containerized-infra-13 26

A.7. Since draft-dcn-bmwg-containerized-infra-12 26
A.8. Since draft-dcn-bmwg-containerized-infra-11 26
A.9. Since draft-dcn-bmwg-containerized-infra-10 27
A.10. Since draft-dcn-bmwg-containerized-infra-09 27
A.11. Since draft-dcn-bmwg-containerized-infra-08 28
A.12. Since draft-dcn-bmwg-containerized-infra-07 28
A.13. Since draft-dcn-bmwg-containerized-infra-06 28
A.14. Since draft-dcn-bmwg-containerized-infra-05 28
A.15. Since draft-dcn-bmwg-containerized-infra-04 29
A.16. Since draft-dcn-bmwg-containerized-infra-03 29
A.17. Since draft-dcn-bmwg-containerized-infra-02 29
A.18. Since draft-dcn-bmwg-containerized-infra-01 29
A.19. Since draft-dcn-bmwg-containerized-infra-00 29
Contributors 30
Acknowledgments 30
Authors' Addresses 30

1. Introduction

The Benchmarking Methodology Working Group (BMWG) has recently expanded its benchmarking scope from Physical Network Functions (PNFs), running on dedicated hardware systems, to Network Function Virtualization (NFV) infrastructure and Virtualized Network Functions (VNFs). [RFC8172] described considerations for configuring NFV infrastructure and benchmarking metrics, and [RFC8204] provided guidelines for benchmarking virtual switches that connect VNFs in the Open Platform for NFV (OPNFV).

Recently, NFV infrastructure has evolved to include a lightweight virtualized platform known as containerized infrastructure. Most benchmarking methodologies and configuration parameters specified in [RFC8172] and [RFC8204] can also be applied to benchmark container networking. However, significant architectural differences between virtual machine (VM)-based and container-based infrastructures necessitate additional considerations.

In terms of virtualization method, Containerized Network Functions (CNFs) utilize host Operating System (OS) virtualization rather than hypervisor-based hardware virtualization, as in VM-based infrastructures. Unlike VMs, containers do not have separate hardware or kernel instances. CNFs share the same kernel space on the host system, with resources logically isolated in different namespaces. Consequently, benchmarking container network performance may require different resource configuration settings.

In terms of networking, a Container Network Interface (CNI) plugin is required to route traffic between containers isolated in different network namespaces. When a pod or container is instantiated, it

initially has no network connectivity. The CNI plugin inserts a network interface into the container’ s isolated network namespace and performs tasks necessary to connect the host and container network namespaces. It also allocates an IP address to the interface and configures routing consistent with the IP address management plugin. Different CNI plugins employ various networking technologies to implement these connections. Based on the networking technologies used by the plugins and how packets are processed or accelerated through the host’ s kernel-space and/or user-space, these plugins can be categorized into different container networking models. These models should be taken into account when 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 are to be interpreted as described in [RFC2119].

3. Terminology

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

Besides, with the growing proliferation and popularity of Kubernetes as a container orchestration platform, this document adopts Kubernetes terminologies to describe general containerized infrastructure.

A Pod is defined as the basic and smallest unit of orchestration and management in Kubernetes. It can host multiple containers, sharing storage and network resources. Typically, each CNF is deployed as a container within a single pod. In this document, the terms container and pod are used interchangeably.

The Container Network Interface (CNI) plugin is a framework that dynamically creates and configures networks for containers.

4. Scope

The primary scope of this document is to fill in the gaps identified in BMWG’s previous NFV benchmarking consideration works ([RFC8172] and [RFC8204]) when applied to containerized NFV infrastructure. The first gap involves different network models and topologies configured by container network interfaces, particularly the extended Berkeley Packet Filter model, which was not covered in earlier documents. The second gap relates to resource configurations for containers. This document investigates these gaps as additional considerations for

benchmarking NFV infrastructure.

It is important to note that apart from these unique characteristics, the benchmarking tests and assessment methodologies defined in the aforementioned RFCs can be equally applied to containerized infrastructure from a general NFV perspective.

5. Benchmarking Considerations

5.1. Networking Models

Compared to VNFs, the selected CNI plugin is a critical software parameter for benchmarking containerized infrastructure. Different CNI plugins configure various network architectures for CNFs in terms of network interfaces, virtual switch usage, and packet acceleration techniques. This section categorizes container networking models based on the characteristics of CNI plugins.

Note that the CNI plugins mentioned in each category are notable examples. Other existing CNI plugins can also fall into one of the categories described in this section.

To ensure repeatability of the container network setup for benchmarking each networking model, Kubernetes is recommended as the container orchestration platform. Kubernetes supports multiple CNIs, enabling benchmarking across all the networking models discussed in this document. In addition to installing the appropriate CNI, underlay network configurations of the Device Under Test/System Under Test (DUT/SUT) may be required, depending on the networking model. These details are outlined in each networking model sub-section.

5.1.1. Normal non-Acceleration Networking Model

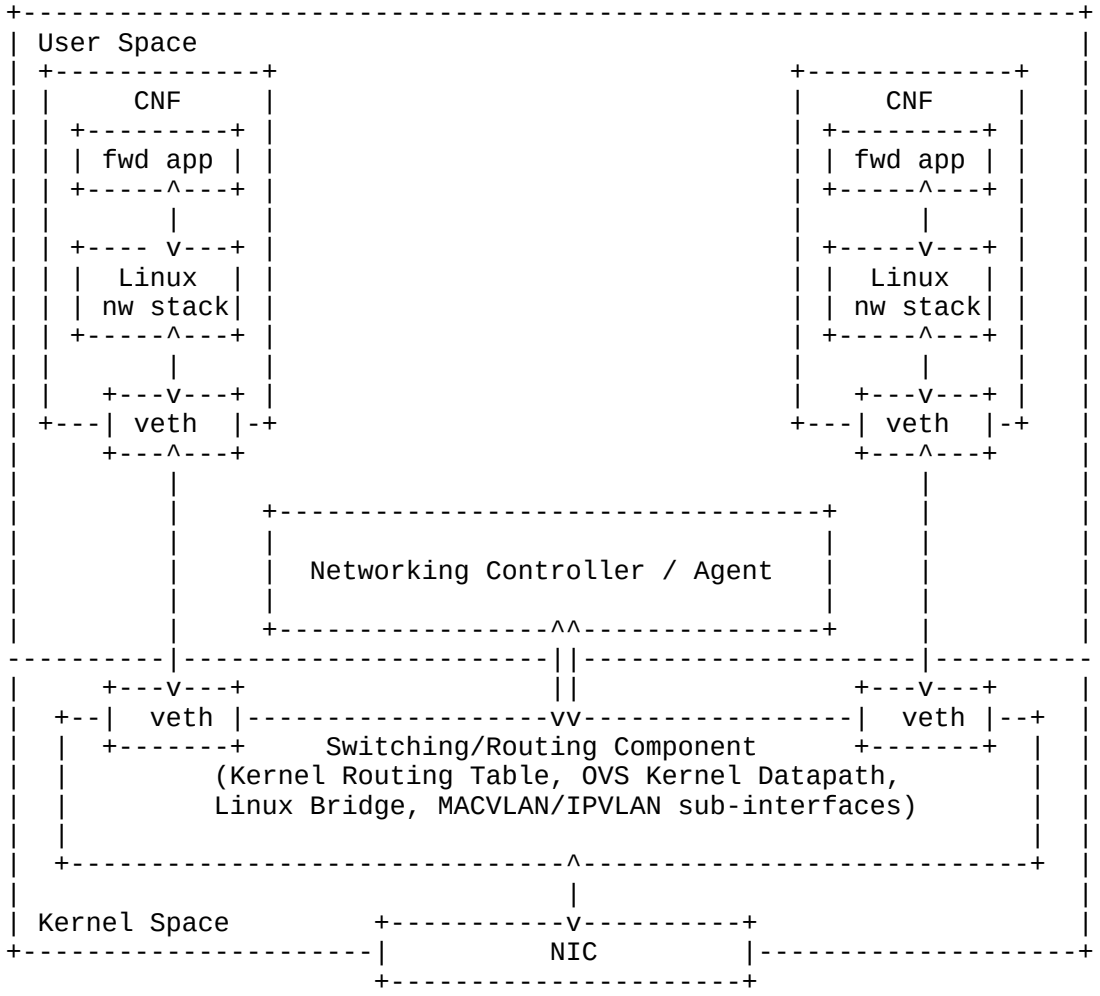

Figure 1: Example architecture of the Normal non-Acceleration Networking Model

Figure 1 illustrates the Normal Non-Acceleration Networking Model, which is typically deployed in general container infrastructure environments. A single CNI is used to configure the container network. The CNF operates in a separate network namespace from the host. The CNI configures virtual ethernet (veth) interface pairs to create a tunnel connecting these two separate network namespaces. The veth interface on the CNF side attaches to the vanilla Linux network stack within the CNF, while the veth interface on the host side attaches to the switching/routing component in the host kernel space.

The kernel's switching/routing component varies depending on the implemented CNI. In the case of Calico, it is the direct point-to-point attachment to the host namespace, using the 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. In the case of Open vSwitch (OVS) [OVS], when configured with the 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 this model, data packets are processed in the host kernel's network stack before being transferred to the CNF running in the user-space. This applies to both pod-to-external and pod-to-pod traffic. Additionally, data packets traverse through the CNF's network stack, moving between the application and the CNF's veth interface. This design results in lower networking performance compared to other models that leverage packet acceleration techniques (described in subsequent sections). Kernel-space vSwitch models are listed below:

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

The Normal Non-Acceleration Networking Model is the basic and default containerized networking model for general container deployment use cases. It can be set up by applying the appropriate YAML configuration file for the chosen CNI to the containerized cluster.

5.1.2. Acceleration Networking Models

Acceleration Networking Models includes different types of container networking model that leverage different packet acceleration techniques to address the performance bottlenecks of the kernel networking stack. Compared to the Normal Non-Acceleration Networking Model, which requires only a single CNI, Acceleration Networking Models typically require at least two CNIs to configure multiple network interfaces for each CNF. In general, a default CNF network

interface is configured by a standard normal non-Acceleration CNI to handle IP address management and control plane traffic, such as CNF management. Additional network interfaces are configured using CNIs that support packet acceleration techniques. These accelerated interfaces are dedicated to high-performance application traffic transmission. Multus CNI is a specialized CNI plugin that facilitates the attachment of multiple network interfaces to a CNF. In the Multus CNI configuration, the default CNI for IP address management (IPAM) and the chosen CNI for packet acceleration must be explicitly defined.

The Acceleration networking model architecture and characteristic differences caused by different implemented packet acceleration techniques are discussed below. For simplicity and clarity, the example architecture figures for these models focus exclusively on the accelerated application data path corresponding to the CNF network interface enabled by the CNI for packet acceleration. The CNF management data path, handled by the default CNI, is omitted from these illustrations.

5.1.2.1. User-space Acceleration Model

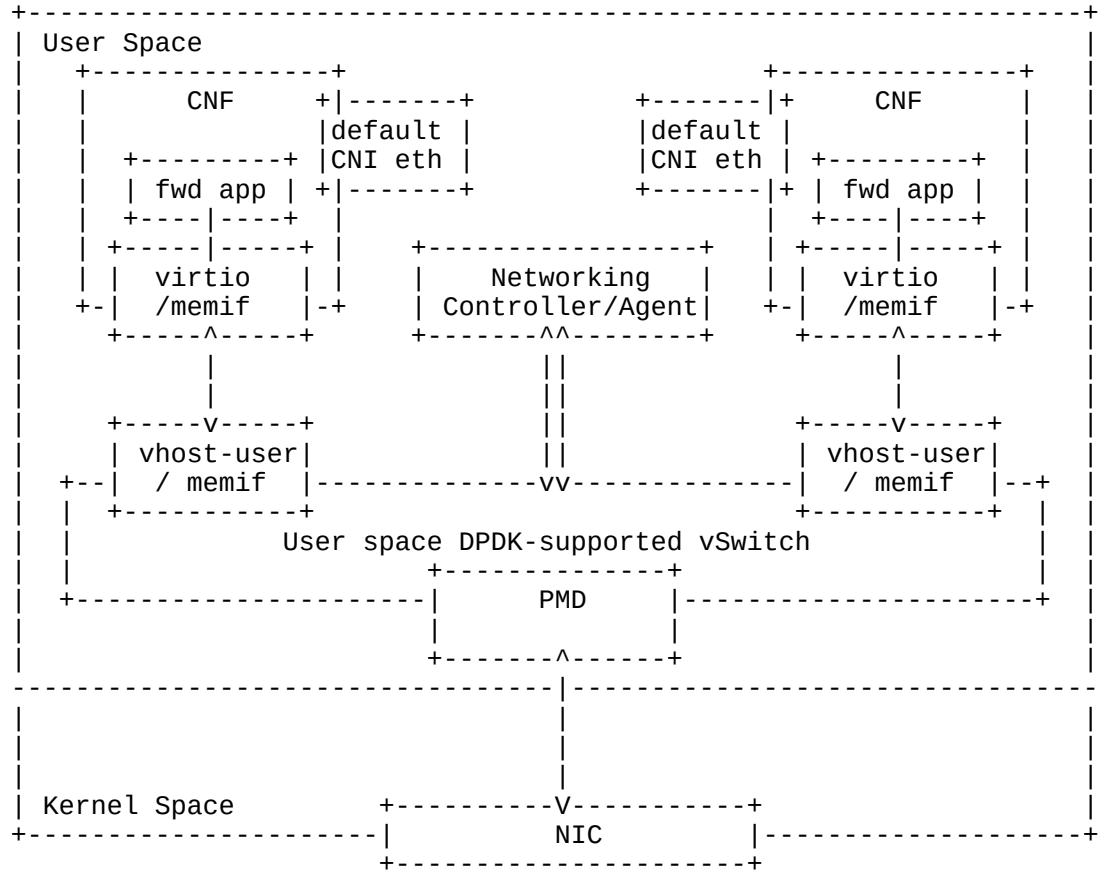

Figure 2: Example architecture of the User-Space Acceleration Model

Figure 2 shows the user-space vSwitch model, where data packets from the physical network port bypass the kernel-space network stack and are delivered directly to the vSwitch running in user-space. This model is commonly regarded as a form of Data Plane Acceleration (DPA) technology, as it achieves higher packet processing rates compared to kernel-space networks, which are constrained by limited packet throughput. To implement this user-space acceleration networking model, the user-space vSwitch must support Data Plane Development Kit (DPDK) libraries. DPDK enables the user-space vSwitch to utilize Poll Mode Drivers (PMDs) to poll incoming packets directly from NIC queues and transfer them to the user-space vSwitch.

Userspace CNI [userspace-cni] is required to create interfaces for packet transfer between the user-space vSwitch and pods. This CNI establishes shared-memory interfaces to enhance packet transfer performance. The two common types of shared-memory interfaces are vhost-user and memif. In case of vhost-user, the CNI creates a virtio PMD in the pod and links it to the vhost-user port in the DPDK-based vSwitch. In case of memif, the CNI creates a memif PMD in the pod and links it to the memif port in the DPDK-based vSwitch.

User-space Acceleration models are listed below based on the current available DPDK-based user-space vSwitches:

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

To set up the user-space acceleration model, mapping between NIC ports, vSwitch ports, and pod interfaces is required. For packet transfer between the NIC and vSwitch, DPDK libraries and DPDK-based user-space vSwitches need to be installed. Then, selected NIC ports for the user-space acceleration network must be bound to the vSwitch’s DPDK PMD using a DPDK-compatible driver such as VFIO or UIO. For packet transfer between the vSwitch and pods, vhost-user/ memif ports need to be added to the vSwitch via port configurations. Traffic routing paths between NIC polling PMD ports and these vhost-user/memif ports should be configured on the vSwitch. Finally, Userspace CNI should be installed and configured to map the pods’ virtio/memif interfaces to the vSwitch’s vhost-user/memif ports.

5.1.2.2. eBPF Acceleration Model

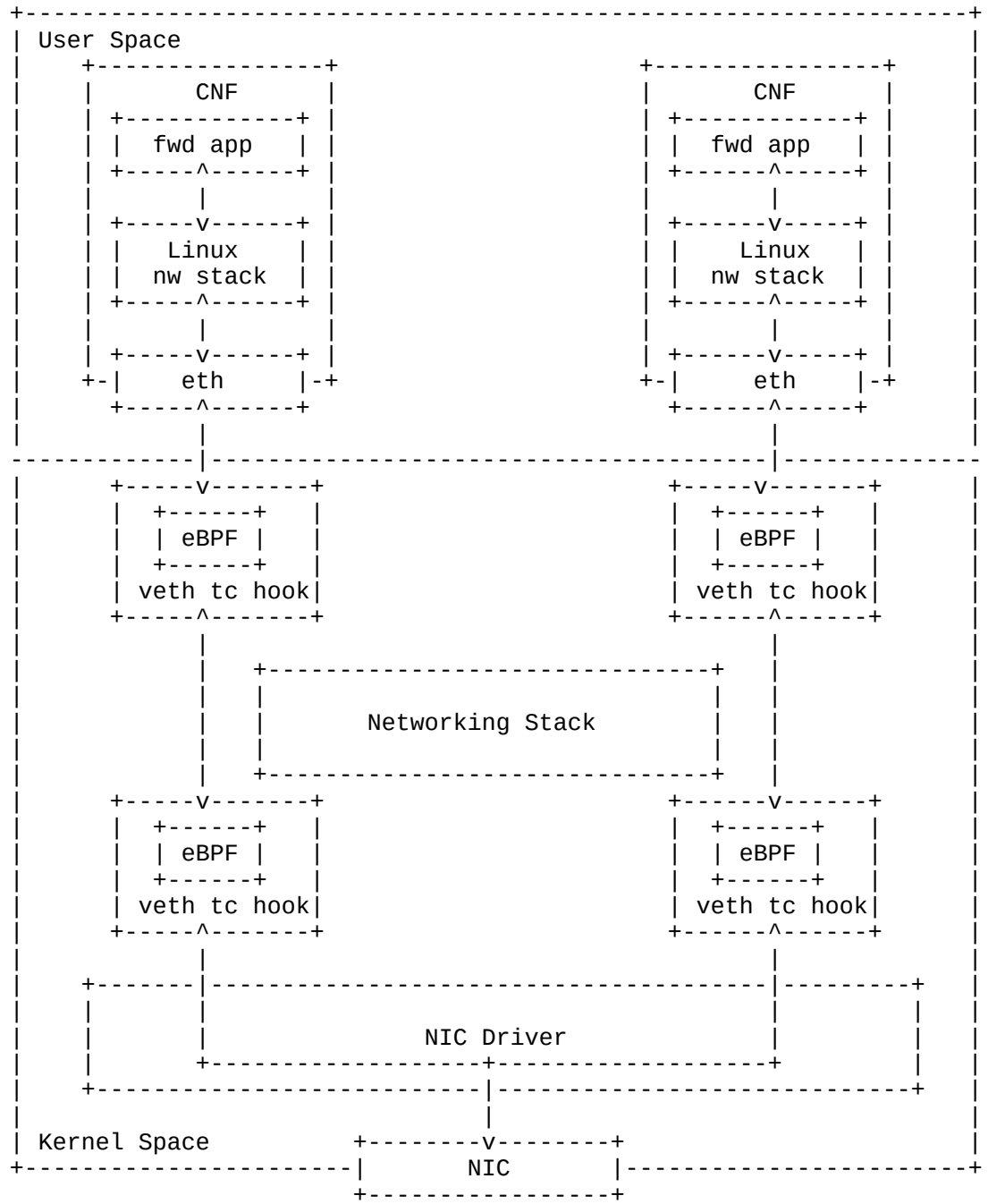
User Space
CNF
fwd app
Linux
nw stack
eth
CNF
fwd app
Linux
nw stack
eth
eBPF
veth tc hook
eBPF
veth tc hook
Networking Stack
eBPF
veth tc hook
eBPF
veth tc hook
NIC Driver
Kernel Space
NIC

Figure 3: Example architecture of the eBPF Acceleration Model - Forwarding via traffic control hook

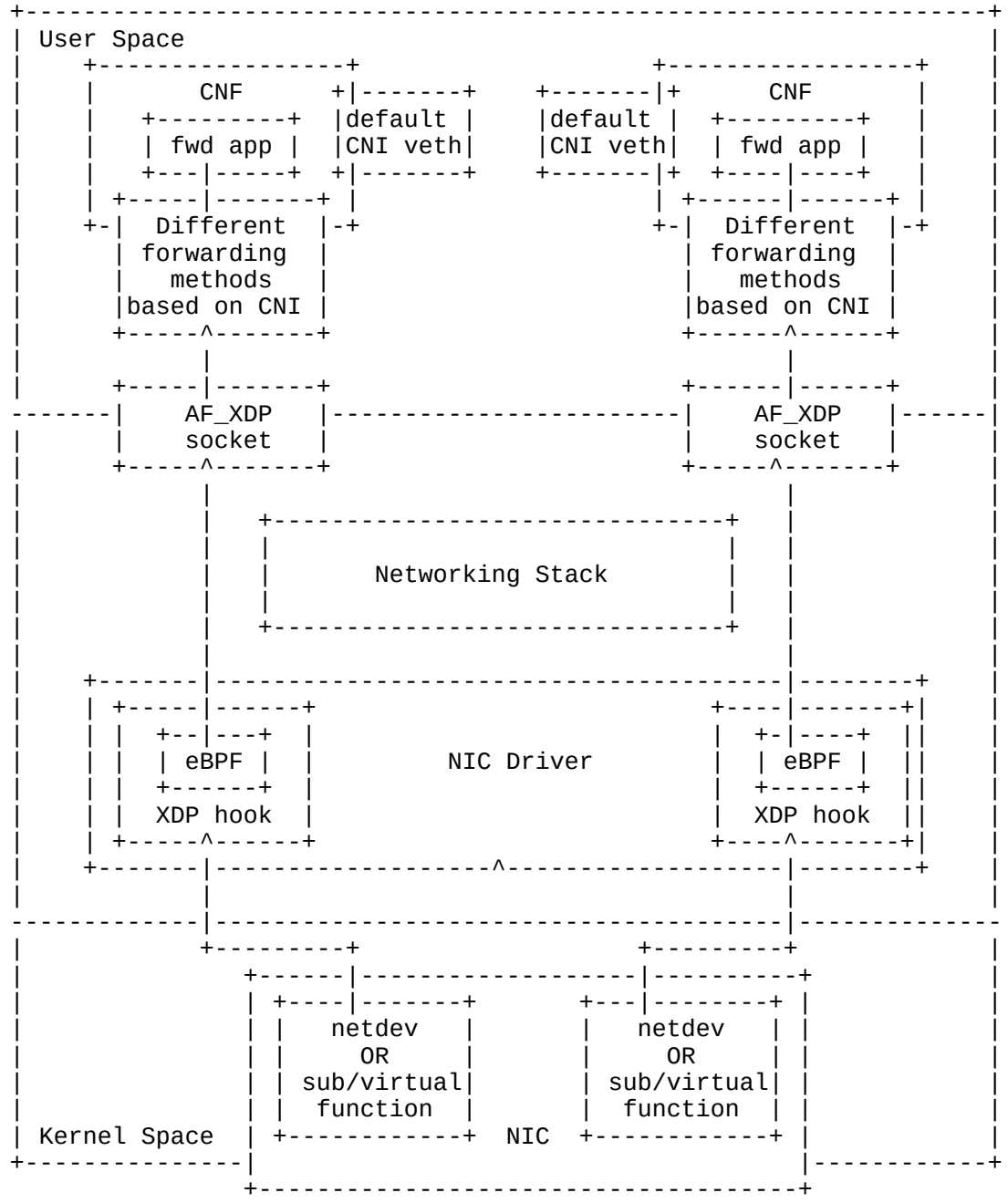
User Space
CNF
fwd app
default
CNI veth
Different
forwarding
methods
based on CNI
AF_XDP
socket
Networking Stack
NIC Driver
eBPF
XDP hook
netdev
OR
sub/virtual
function
NIC
Kernel Space

Figure 4: Example architecture of the eBPF Acceleration Model - Forwarding via XDP hook

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

eXpress Data Path (XDP) and Traffic Control Ingress/Egress (tc) are the eBPF hook types used by different eBPF acceleration CNIs. XDP is the hook located at the NIC driver, representing the earliest point in the networking stack where a BPF hook can be attached. Traffic Control Ingress/Egress (tc) is the hook at the network interface on the container’s incoming/outgoing packet path. When a packet reaches these hooks, an eBPF program is triggered to process it. The packet forwarding paths differ depending on the hooks used:

In case of packets are forwarded via tc hook, packets between the pod’s network interface and the NIC are forwarded through eBPF programs running at the tc hooks attached to both sides, with forwarding facilitated via eBPF maps.

In case of packets are forwarded via XDP hook, packets are forwarded from the eBPF XDP hook at the NIC driver to an AF_XDP socket [AFXDP]. AF_XDP socket is a new Linux socket type that provides an in-kernel short path between user-space and the XDP hook at the networking driver. The eBPF program at the XDP hook redirects packets from the NIC to the AF_XDP socket, bypassing the kernel networking stack. Packets are exchanged between user-space and the AF_XDP socket through a shared memory buffer. From the AF_XDP socket, packets can be forwarded to the pod via various methods depending on the selected CNI, such as: user-space packet forwarding libraries as in AF_XDP CNI plugin [afxdp-cni], normal pod network interface as in Cilium [Cilium], or an AF_XDP-supported user-space vSwitch such as OVS-DPDK. This AF_XDP in-kernel short path is only applicable to NIC ingress packets. Pod egress packets are forwarded via eBPF programs attached to traffic control hooks.

Figure 3, Figure 4 show the corresponding model architectures of 2 eBPF Acceleration Model types: Forwarding via traffic control hook and Forwarding via XDP hook. Notable examples of these models are listed below:

o Forwarding via traffic control hook: Calico [Calico], Cilium [Cilium]

o Forwarding via XDP hook: AF_XDP K8s plugin [afxdp-cni], Cilium with XDP forwarding enabled, Userspace CNI [userspace-cni] with an AF_XDP-supported user-space vSwitch such as OVS-DPDK [ovs-dpdk] or VPP [vpp]

To set up these kinds of eBPF Acceleration networking model, the corresponding CNIs of each model need to be installed and configured to map pod interfaces to NIC ports. In case of using user-space vSwitch, the AF_XDP-supported version of the vSwitch needs to be installed. The NIC ports can be bound to the vSwitch's AF_XDP PMD via vSwitch port configurations. Then, packet transfer between pods and vSwitch is configured via Userspace CNI.

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

5.1.2.3. Smart-NIC Acceleration Model

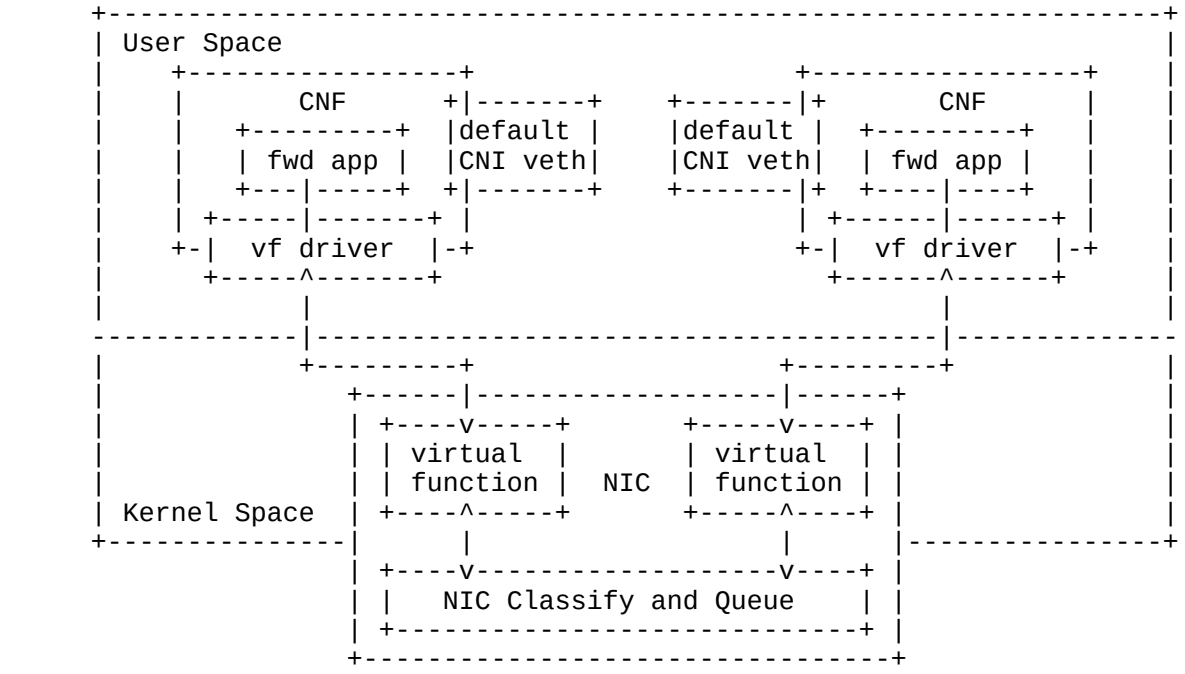

Figure 5: Examples of Smart-NIC Acceleration Model

Figure 5 shows the Smart-NIC acceleration model, which leverages the packet acceleration features of Smart-NIC cards.

Single-Root I/O Virtualization (SR-IOV) is a commonly used packet acceleration technique supported by Smart-NIC cards from various vendors. SR-IOV extends PCIe specifications to enable multiple partitions running simultaneously within a system to share PCIe devices. In this context, the NIC creates virtual replicas of PCI functions, known as virtual functions (VFs), each of which is directly connected to a container's network interface. Using SR-IOV, data packets from external sources bypass both the kernel and user-space, and are directly forwarded to the container's virtual network interface. The SR-IOV network device plugin for Kubernetes [SR-IOV] is recommended for creating a special interface in each container, controlled by the VF driver.

Smart-NICs from some vendors also support the XDP eBPF packet acceleration technique. This approach retains the same eBPF acceleration networking model, with the key difference being that the XDP eBPF program runs on the Smart-NIC card instead of the host machine. This setup frees the host machine's CPU resources for application workloads.

To set up the SR-IOV acceleration container network, SR-IOV capable NIC cards and BIOS support for creating VFs on the NIC are required. After creating the VFs, they must be bound to a DPDK-compatible driver, such as VFIO. Pods can then be configured to use these VFs via SR-IOV network plugin configurations. In the case of the eBPF/XDP offloading container network, Cilium CNI with its eBPF offloading feature must be installed and properly configured.

It is worth noting that Smart-NIC features are also available in a new class of programmable networking device called the "any" Processing Units (xPU), such as Data Processing Units (DPUs) and Infrastructure Processing Units (IPUs). An xPU combines Smart-NIC capabilities, CPU cores, and acceleration engines, enabling it to offload, accelerate, and isolate networking and data processing tasks from the host machine' s CPU resources. When using an xPU for container networking, control plane functions (e.g., Kubernetes kube-proxy) can be offloaded to the xPU. The accelerated datapath between the CNF and the xPU data plane can be implemented using techniques like SR-IOV or its enhanced version, Scalable IOV. Currently, xPU-based container networking is a nascent technology with limited implementation and documentation. Configuring such networks may require xPU-supported CNIs and xPU network operators (e.g [Bluefield]).

5.1.2.4. Model Combination

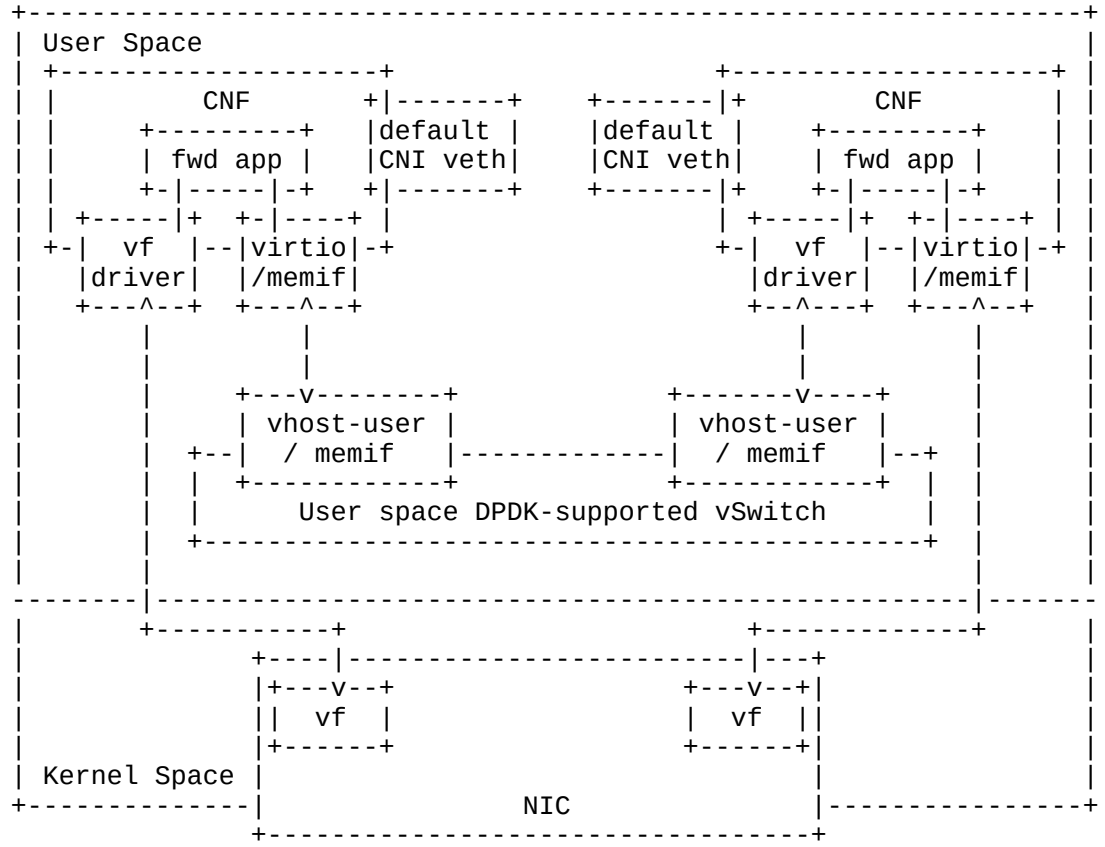

Figure 6: Examples of Model Combination deployment

Figure 6 depicts the networking model that combines the user-space vSwitch model and the Smart-NIC acceleration model. This model is often used in service function chain scenarios where two different types of traffic flows are present: North/South traffic and East/West traffic.

North/South traffic is traffic betwen the NIC and CNFs. Meanwhile, East/West traffic is traffic between multiple CNFs within the same host machine. An example of this packet acceleration technique combination involves using user-space vSwitch models, such as OVS-DPDK or VPP, for East/West traffic and Smart-NIC models, such as SR-IOV, for North/South traffic. The SR-IOV datapath can help avoid potential performance bottlenecks in the North/South direction, which might otherwise be caused by the user-space vSwitch. Meanwhile, the user-space vSwitch facilitates East/West traffic transmission entirely in the user-space.

To set up this combined networking model, Multus CNI must be correctly set up to enable the appropriate CNI for each pod's interface. For example, the interface for North/South traffic should be configured using SR-IOV network plugin. The interface for East/West traffic should be configured using Userspace CNI plugin.

The throughput advantages of these combined networking models in different traffic direction scenarios are documented in [Intel-SRIOV-NFV].

5.2. Resources Configuration

The resource configuration considerations listed here apply not only to the CNF but also to other components in a containerized System Under Test (SUT). A containerized SUT comprises NICs, possible cables between hosts, the kernel and/or vSwitch, and CNFs.

Note that this section describes only the new resource configuration parameters introduced by containerized infrastructures. The configuration parameters for vSwitch and VNF environments, as defined in [RFC8172] and [RFC8204], remain applicable.

5.2.1. Container Deployment Environment

Compared with a VM-based VNF benchmarking environment, a CNF benchmarking environment introduces additional deployment-specific factors that must be explicitly specified to ensure reproducibility and comparability of results:

- Container hosting infrastructure (whether deployed on a virtualized environment or bare-metal hardware)

- Kubernetes details (distribution and version)

- CNI plugin details (selected plugin, version, and its specific configuration parameters)

- NIC offload capabilties

5.2.2. CPU Isolation / NUMA Affinity

CPU pinning offers benefits such as maximizing cache utilization, eliminating operating system thread scheduling overhead, and coordinating network I/O by guaranteeing resources. An example of CPU pinning technology in containerized infrastructure is the CPU Manager for Kubernetes (CMK) [CMK]. This technology has proven effective in avoiding the "noisy neighbor" problem, as demonstrated in previous experiences [Intel-EPA]. Additionally, the benefits of

CPU isolation techniques extend beyond addressing the "noisy neighbor" problem. Different CNFs can also interfere with each other and with the vSwitch if used.

NUMA impacts the speed of CPU cores accessing memory. CPU cores in the same NUMA node can access shared memory in that node locally, which is faster than accessing memory in a different NUMA node. In a containerized network, packet forwarding occurs through the NIC, CNF, and possibly a vSwitch, depending on the chosen networking model. The NUMA node alignment of an NIC can be determined via the PCI device's node affinity. Specific CPU cores can also be directly assigned to CNFs and vSwitches via configuration settings. Network performance may vary depending on whether the NUMA node hosting the physical network interface, vSwitch, and CNF is the same or different. Benchmarking experience for cross-NUMA performance impacts [cross-NUMA-vineperf] revealed the following:

o Performance degradation is worse when a single NUMA node serves multiple interfaces than in cross-NUMA scenarios.

o Sharing CPUs across NUMA nodes results in significant performance drops.

Note that CPU pinning and NUMA affinity considerations also apply to VM-based VNFs. Dedicated CPU cores from a specific NUMA node can be assigned to VNFs and vSwitches via their runtime configurations. The NUMA node of an NIC can be identified from PCI device information, and a host's NUMA nodes can be scheduled to virtual machines by specifying the desired nodes in their settings.

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

- Selected CPU Isolation level

- NUMA cores allocation to pods/CNFs, vSwitch, NIC

5.2.3. Pod Hugepages

Hugepages configure large memory page sizes to reduce Translation Lookaside Buffer (TLB) miss rates and increase application performance. This boosts the efficiency of logical/virtual to physical address lookups performed by a CPU's memory management unit and enhances overall system performance. In containerized infrastructures, containers are isolated at the application level, allowing administrators to configure hugepages more granularly (e.g., Kubernetes supports 2 MB or 1 GB hugepages for containers). These pages are dedicated to the application and cannot be used by other

processes, enabling more efficient utilization. From a network benchmarking perspective, the impact of hugepages on general packet processing may be relatively minor. However, application-level considerations are necessary to fully evaluate their impact. For example, in DPDK-based applications, it has been demonstrated in [Intel-EPA] that hugepages improve network performance because packet handling processes run within the application.

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

- Pod hugepage size

5.2.4. Pod CPU Cores and Memory Allocation

Resource allocation choices, such as CPU cores and RAM allocation to pods, PMD and vSwitch, can significantly impact container network performance. Benchmarking experiences reported in [GLOBECOM-21-benchmarking-kubernetes] by [ViNePERF] verified that:

o Allocating 2 CPUs per pod is insufficient for all packet frame sizes. For larger packet sizes (over 1024 bytes), increasing CPU per pods significantly improves the throughput. Varying RAM allocation to Pods also results in different throughput performances.

o Not assigning dedicated CPU cores to DPDK PMD leads to significant performance degradation.

o Increasing CPU core allocation to OVS-DPDK vSwitch does not improve performance, but increasing cores for VPP vSwitch reduces latency.

Additionally, assigning dedicated CPU cores to PMD' s Rx queues in user-space vSwitch can enhance network performance.

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

- Pod CPU cores allocation

- Pod RAM allocation

5.2.5. AF_XDP Configuration

AF_XDP operates in two packet polling modes: busy and non-busy. In busy polling mode, AF_XDP uses the same CPU cores for application and packet Rx/Tx processing. In non-busy polling mode, it uses separate CPU cores for these tasks. The choice of AF_XDP mode and CPU core configuration for application and packet processing in non-busy polling mode can affect benchmarking performance. [LPC18-DPDK-AFXDP]

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

- AF_XDP busy polling mode

- Number of CPU cores allocated in non-busy polling mode

5.2.6. Service Function Chaining

Benchmarking scenarios for containerized and VM-based infrastructures may include various operational use cases. Traditional black-box benchmarking measures packet in-out performance at physical network ports, as hardware typically performs a single function. However, in NFV environments, physical network ports are often connected to multiple CNFs (e.g., multiple PVP test setups described in [ETSI-TST-009]), a scenario referred to as Service Function Chaining. In addition to the differences among CNI plugin networking models discussed above, benchmarking scenarios should also account for factors such as the number of CNFs or network services deployed on a single host. Another aspect to consider in a service function chaining scenario is that a single CNF or VNF may be part of multiple service chains. In such cases, varying resource scheduling policies during runtime (e.g., CNF activation scheduling, CPU scheduling) may also lead to network function throughput and loss improvement as described in [NFVnice].

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

- Number of CNFs

- Service chain CNF composition (e.g., CNF belongs to multiple concurrent chains)

- Service chain resource scheduling policies (e.g., CNF activation time, CPU allocation time, etc.)

- Selected CNI Plugin

5.2.7. Other Considerations

In addition to intra-node test scenarios, inter-node scenarios should also be considered for container network benchmarking, where container services are deployed across different nodes. Inter-node networking is required to provide connectivity for CNFs between nodes. According to [ETSI-NFV-IFA-038], several technologies enable inter-node networking, including overlay technologies (e.g., VXLAN, IP-in-IP), BGP routing, Layer 2 underlay, direct networking using dedicated NICs for each pod, or Kubernetes LoadBalancer services. Different protocols and technologies may lead to varying performance outcomes.

6. IANA Considerations

This document does not require any IANA actions.

7. Security Considerations

Benchmarking activities as described in this memo are limited to technology characterization of a 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.

8. References

8.1. Informative References

[AFXDP] The Linux Kernel, "AF_XDP", <https://docs.kernel.org/networking/af_xdp.html>.

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

[Bluefield]
RedHat, "OVN/OVS offloading with OpenShift on NVIDIA BlueField-2 DPUs", May 2023, <https://access.redhat.com/articles/6804281>.

[Calico] Tigera, "Calico", <https://docs.tigera.io/calico/latest/about>.

[Cilium] Cilium, "Cilium", <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, "CPU Manager for Kubernetes", <https://github.com/intel/CPU-Manager-for-Kubernetes>.

[cross-NUMA-vineperf]
Anuket, "Cross-NUMA performance measurements with VSPERF", <https://lf-anuket.atlassian.net/wiki/spaces/HOME/pages/21859213/Cross-NUMA+performance+measurements+with+VSPERF>.

[eBPF] Linux Foundation, "eBPF, extended Berkeley Packet Filter", <https://ebpf.io/>.

[ETSI-NFV-IFA-038]
ETSI, "Network Functions Virtualisation (NFV) Release 4; Architectural Framework; Report on network connectivity for container-based VNF", ETSI GS NFV-IFA 038, Version 4.1.1, November 2021.

[ETSI-TST-009]
ETSI, "Network Functions Virtualisation (NFV) Release 3; Testing; Specification of Networking Benchmarks and Measurement Methods for NFVI", ETSI GS NFV-TST 009, Version 3.4.1, December 2020.

[Flannel] Flannel, "flannel", <https://github.com/flannel-io/flannel>.

[GLOBECOM-21-benchmarking-kubernetes]
Sridhar, R., Paganelli, F., and A. Morton, "Benchmarking Kubernetes Container-Networking for Telco Usecases", DOI 10.1109/GLOBECOM46510.2021.9685803, December 2021, <https://ieeexplore.ieee.org/document/9685803>.

[intel-AFXDP]
Intel, "AF_XDP Sockets: High Performance Networking for Cloud-Native Networking Technology Guide", January 2021, <https://builders.intel.com/docs/networkbuilders/af-xdp-sockets-high-performance-networking-for-cloud-native-networking-technology-guide.pdf>.

[Intel-EPA]
Intel, "Enhanced Platform Awareness in Kubernetes", January 2018, <https://networkbuilders.intel.com/docs/networkbuilders/enhanced-platform-awareness-in-kubernetes-performance-benchmark-report.pdf>.

[Intel-SRIOV-NFV]
Intel, "SR-IOV for NFV Solutions Practical Considerations and Thoughts", February 2017, <https://www.intel.com/content/www/us/en/content-details/335625/sr-iov-for-nfv-solutions-practical-considerations-and-thoughts-technical-brief.html>.

[LPC18-DPDK-AFXDP]
Karlsson, M. and B. Topel, "The Path to DPDK Speeds for AF_XDP", November 2018, <http://oldvger.kernel.org/lpc_net2018_talks/lpc18_paper_af_xdp_perf-v2.pdf>.

[NFVnice] Kulkarni, S., Zhang, W., Hwang, J., Rajagopalan, S., Ramakrishnan, K., and T. Wood, "NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains", DOI 10.1109/TNET.2020.2969971, April 2020, <https://ieeexplore.ieee.org/document/9007052>.

[OVN] Cloud Native Computing Foundation, "ovn-kubernetes", <https://ovn-kubernetes.io/>.

[OVS] Linux Foundation, "Open vSwitch", <https://docs.openvswitch.org/en/latest/>.

[ovs-dpdk] Linux Foundation, "Open vSwitch with DPDK", <https://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>.

[SR-IOV] Intel, "SR-IOV CNI plugin", <https://github.com/k8snetworkplumbingwg/sriov-cni>.

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

[ViNePERF] Anuket, "Project: Virtual Network Performance for Telco NFV", <https://lf-anuket.atlassian.net/wiki/spaces/HOME/pages/21896595/ViNePERF>.

[vpp] Fast Data Project, "VPP - Vector Packet Processor", <https://s3-docs.fd.io/vpp/25.10/>.

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

A.1. Since draft-ietf-bmwg-containerized-infra-07

Added Container Deployment Environment configurations

A.2. Since draft-ietf-bmwg-containerized-infra-06

Added CNF service chain resource scheduling policy consideration parameters Added Container Deployment Environment configurations

A.3. Since draft-ietf-bmwg-containerized-infra-03

Changed eBPF acceleration model categorization into: Fowarding via tc hook and Forwarding via XDP hook.

Update latest information for references.

Enhance English grammar of the whole document.

A.4. Since draft-ietf-bmwg-containerized-infra-01

Addressed comments and feedbacks from the related RFC 8204 document author - Maryam Tahhan:

Updated general description for all accelerated networking models: using Multus CNI to enable multiple CNI configurations for hybrid container networking stack.

Updated illustration of the default CNI network interface in all accelerated networking models.

Updated illustration of the separated networking stack inside CNF in normal non-acceleration networking model.

Made minor correction on the descriptions of AF_XDP and eBPF datapath.

Added xPU networking device mentioning in the Smart-NIC acceleration model.

Added AF_XDP busy/non-busy polling mode resource configuration consideration.

A.5. Since draft-ietf-bmwg-containerized-infra-00

Minor editorial changes and nits correction.

A.6. Since draft-dcn-bmwg-containerized-infra-13

Updated environment setup repeatability guidance for all mentioned container networking models.

A.7. Since draft-dcn-bmwg-containerized-infra-12

Updated scope to clearly specify the gaps of related RFCs.

A.8. 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.9. 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 Resources Configuration consideration sections such as general description for container network plugin, which resources can also be applied for VM-VNF.

A.10. 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-AF_XDP, using AF_XDP supported CNI, and using user-space vSwitch which support AF_XDP 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.11. 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 AF_XDP deployment option

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

A.12. 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.13. Since draft-dcn-bmwg-containerized-infra-06

Added Benchmarking Experience of Multi-pod Test

A.14. 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 Resources 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.15. Since draft-dcn-bmwg-containerized-infra-04

Added Benchmarking Experience of SRIOV-DPDK.

A.16. Since draft-dcn-bmwg-containerized-infra-03

Added Benchmarking Experience of Contiv-VPP.

A.17. Since draft-dcn-bmwg-containerized-infra-02

Editorial changes only.

A.18. Since draft-dcn-bmwg-containerized-infra-01

Editorial changes only.

A.19. 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 Resources 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

The following people have substantially contributed to this document:

Kyoungjae Sun
ETRI
Email: kjsun@etri.re.kr

Hyunsik Yang
Interdigital
Email: hyunsik.yang@interdigital.com

Rute C. Sofia
fortiss
Email: sofia@fortiss.org

Tina Samizadeh
fortiss
Email: samizadeh@fortiss.org

George Koukis
ATHENA
Email: george.koukis@athenarc.gr

Acknowledgments

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

Authors' Addresses

Minh-Ngoc Tran
Electronics and Telecommunications Research Institute
218 Gajeong-ro, Yuseong-gu
Daejeon
34129
Republic of Korea
Phone: +82 1073874420
Email: mipearlska@etri.re.kr

Namseok Ko
Electronics and Telecommunications Research Institute
218 Gajeong-ro, Yuseong-gu
Daejeon
34129
Republic of Korea
Phone: +82 428605560
Email: nsko@etri.re.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