NFV-VIPP: Catching Internal Figures of Packet Processing for Accelerating Development and Operations of NFV-nodes

Masahiro Dodare
Nagoya Institute of Technology
Nagoya-shi, Aichi, 466-8555, Japan
dodare@matlab.nitech.ac.jp

Yuki Taguchi
Nagoya Institute of Technology
Nagoya-shi, Aichi, 466-8555, Japan

Ryota Kawashima
Nagoya Institute of Technology
Nagoya-shi, Aichi, 466-8555, Japan
kawa1983@ieee.org

Hiroki Nakayama
BOSCO Technologies, Inc.
Minato-ku, Tokyo, 105-0003, Japan
nakayama@bosco-tech.com

Tsunemasa Hayashi
BOSCO Technologies, Inc.
Minato-ku, Tokyo, 105-0003, Japan
hayashi@bosco-tech.com

Hiroshi Matsuo
Nagoya Institute of Technology
Nagoya-shi, Aichi, 466-8555, Japan
matsuo@nitech.ac.jp

***Abstract*—Server-based NFV-nodes have disparate internals, such as simultaneous deployment of Virtual Network Functions (VNFs) and layered software abstractions including a virtual switch. The traditional operations tailored for function-hardware-coupled devices cannot cope with the increase of related components as well as complicated packet forwarding paths inside. Besides, self-development of VNFs attracting Telcos is still highly complicated work, due to lack of exact troubleshooting of internal NFV-nodes caused by exclusive resource management by Data-Plane Development Kit (DPDK). OPNFV Barometer provides means of stats acquisition, but internal figures of packet processing are still unveiled. In this paper, we propose an integrated metrics collection framework (NFV-VIPP) specialized to NFV-nodes. NFV-VIPP provides seamless understandings of system components in a node, and reveals the inside by transparently exposing implementation-related metrics. NFV-VIPP can be incorporated into Barometer/collectd via RESTful APIs to reinforce system visibility, meaning that our framework bridges NFV-node internals to existing management frameworks. We explore NFV-node management using intra-VNF metrics obtained by NFV-VIPP. Specifically, we prove that CPU-cycle consumption of inter-receive-polling is a driving force to estimate system load.**

***Index Terms*—NFV, DPDK, Barometer, Visualization, Network Monitoring, Load Measurement, DevOps**

I. Introduction

Decoupling traditional function-hardware mapping of network functions complicates management of themselves. First, deployment of multiple Virtual Network Functions (VNFs) on a server-based NFV-node needs to be monitored, although they are dynamically instantiated, terminated, or migrated. Second, layered software abstractions in NFV-nodes introduces further tracking targets, such as virtual switches and virtual NICs. This complicated structure obscures internal packet processing, which results in difficulty of identifying forwarding paths and locations of packet drops/delays for detailed troubleshooting.

Presently, Y. Taguchi is with LINE Corporation, Inc.

Telcos have expected self-development of VNFs for their networks (e.g. vVIG [1] by AT&T, and Kamuee [2] by NTT Communications); however, realizing highly-optimized VNFs is an arduous work even for experienced developers due to the obscure internal of NFV-nodes. There is a few developer/operator supportive tools specialized to NFV-nodes. NFVPerf [3] can detect a bottleneck of inter-components by packet capturing. OPNFV Barometer [4] can acquire basic I/O stats of NFV-nodes accelerated by Data Plane Development Kit (DPDK) [5]. Barometer paves the way for conventional stats-based monitoring of NFV-nodes, but there still remains uncollectible promising metrics as explained in VI.

We propose an integrated metrics collection framework, Visualizing Internals of Packet Processing (NFV-VIPP), for DPDK-accelerated NFV-nodes. One remarkable aspect is that intra-VNF metrics are exposed for exact understandings of internal packet processing. NFV-VIPP is implemented in DPDK as a dedicated library that collects internal metrics transparently to VNFs, and its instance within the host (hypervisor) manages other instances within VMs/containers for providing an integrated view to existing monitoring frameworks. In this paper, we describe details of our framework, and examine system load estimation of DPDK-accelerated NFV-nodes as a case study of intra-VNF metrics usage. We prove CPU-cycle consumption of inter-receive-polling alongside our novel analysis is a driving force to estimate the system load.

This paper is an extended version of our technical report [6]. We describe the refined NFV-VIPP that co-works with Barometer as well as renewed evaluations conducted with upgraded environment, adding container-formed VNFs.

The rest of the paper is organized as follows: related work is introduced in II. We present requirements, architecture, and implementation details in III, IV, and V respectively. In VI, we explore the load estimation of NFV-nodes, and finally, we conclude this study and give future work in VII.

978-3-903176-24-9

II. Related work

VNF orchestration tools like OpenStack [7] and network-wide monitoring frameworks, such as NFV-Throttle [8], NetAlytics [9] and CloudHealth [10], have resource monitoring mechanisms inside, but they are not available for NFV-nodes powered-by DPDK because it invalidates traditional usage-based grasping of system status. NFVPerf [3] and ConMon [11] can detect performance anomalies by direct monitoring of inter-component communications within a node. Their approaches can be theoretically applied to DPDK-based systems; however, packet capturing is not suitable for short packet dominant high-speed traffics.

OPNFV Barometer [4] is a DPDK-dedicated monitoring framework based on collectd [12]. Barometer can be transparently attached to a target VNF as *secondary process* that can access shared variables holding various information including packet I/O stats and packet buffer usage. However, non-shared data cannot be acquired by other processes because of the system security, and therefore, we directly introduce dedicated metrics collecting feature within VNF processes.

J. Xie et al. have proposed a load estimation approach using length of reception queues in DPDK-enabled NFV-nodes [13]. The evaluations showed that overloaded condition can be detected, but the approach cannot predict of that state due to uprush of the queue length. Therefore, a yet another metric have to be considered.

III. System Requirements

We clarify system requirements of NFV-VIPP here. Followings have been derived from current situations of SDN/NFV where most high-performance VNFs depend on DPDK, and they do not have practical OAM features (our framework cannot rely on the features offered by VNFs). We suppose users of our system are not only network operators and but also developers (for optimizing VNFs), and therefore, both easy management and detailed metrics need to be satisfied.

A. Functional Requirements

Shedding light on implementation-related metrics is a key feature of NFV-VIPP providing far beyond visibility of NFV-nodes. Hence, NFV-VIPP instances should reside within each VNF to directly collect the metrics. Parallel monitoring of VNFs degrades both operation and performance costs. For instance, a monitoring server goes slow if multiple agents on an NFV-node send trap messages notifying a same syndrome. Therefore, multiple instances of NFV-VIPP must form a unified monitoring view and be seamlessly integrated into existing frameworks that can be categorized into metrics collector (e.g. collectd), metrics conveyer (e.g. SNMP [14] and Telemetry [15]), and metrics consumer (e.g. Zabbix [16]).

B. Non-functional Requirements

Implementation of NFV-VIPP needs to be covered by common software infrastructure like DPDK, to prevent existing VNFs from being modified. DPDK has brought 10 Gbps+ traffic to software-oriented nodes, and the metrics acquisition by NFV-VIPP should not degrade performance of NFV-nodes.

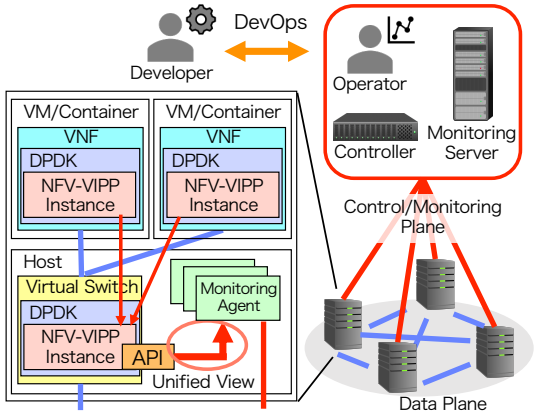

Fig. 1. An Architectural Overview of the Proposed System

IV. System Architecture

In this section, we explain architectural design of NFV-VIPP based on the aforementioned system requirements.

The feature of our framework is realized by a collaboration of NFV-VIPP *instances* deployed into a same NFV-node. Figure 1 illustrates how they collaborate each other to provide the unified monitoring view of the NFV-node to the existing frameworks. An NFV-VIPP instance is deployed into each VNF environment regardless of its forms (baremetal/VM/container), and each instance, realized as a DPDK subsystem, is dedicated to the VNF. most work of NFV-VIPP instances is not performed in the context of data-plane (except recording of CPU-cycle consumption), and they do not degrade packet processing efficiency of VNFs.

There is a special NFV-VIPP instance, attached to the baremetal virtual switch, that works as an integrator of monitoring planes within the node. That is, the instance provides a unified monitoring view to the *agents* of existing frameworks via RESTful APIs. Such an integrated design not only simplifies the agents to support our framework, but also bundles the monitored metrics from the perspective of an NFV-node (not a VNF) for the monitoring servers like Zabbix.

In terms of the control plane, the SDN controller can use NFV-node status information notified by a monitoring server. For instance, the controller can determine update of current structure of the network by re-routing network paths (to avoid overloaded areas), or adjusting the number of VNF instances depending on load of each NFV-node.

V. Implementation

We describe implementation details of NFV-VIPP in this section. As explained in the previous section, NFV-VIPP instances are directly deployed into VNFs as a DPDK subsystem. We have achieved this by adding a yet another thread group into DPDK, *monitoring plane threads*, that are dedicated to manage acquired metrics inside the VNF process.

Figure 2 shows NFV-node internal with NFV-VIPP implemented in DPDK v19.05. There are three NFV-VIPP instances in the NFV-node. Two of which reside in container-formed

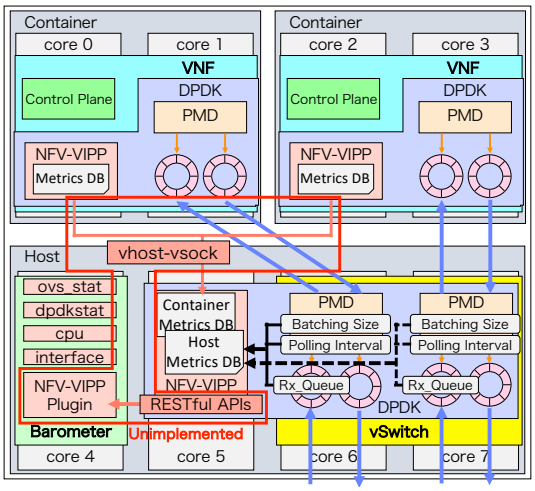

Fig. 2. NFV-node internals with NFV-VIPP

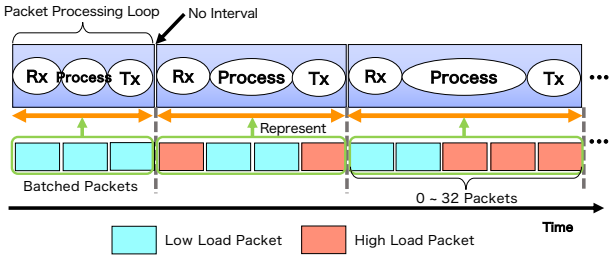

Fig. 3. Packet Processing Loop Model of DPDK

TABLE I
SPECIFICATIONS OF THE MACHINES

CPU	Intel Core i7-6900K 3.2 GHz (8 cores, HT: off)
Memory	64 GB
NIC	Intel XXV710-DA2 (25 GbE, 2 ports)
OS	CentOS 7.6
DPDK	v19.05
Open vSwitch [19]	v2.11.1

VNFs and the remaining one is for the virtual switch. Each instance has a single monitoring plane thread and they run on separate CPU cores from data plane (PMD) threads for performance reasons. The NFV-VIPP instance of the virtual switch also has roles of both the monitoring plane integrator and the RESTful API server. We have implemented these roles (still under development) using a vhost-vsock mechanism [17] (that realizes socket communications between the host and the guest) and the C++ REST SDK [18]. We have also extended PMD threads to acquire CPU-cycle consumption of inter-receive-polling. Specifically, each invoking interval of an `rte_eth_rx_burst` function is recorded using *RDTSC*. For Barometer integration, we have added a dedicated plugin to communicate with NFV-VIPP via RESTful APIs.

VI. LOAD MEASURING

In this section, we examine the intra-metrics acquisition for estimating system load of DPDK-accelerated NFV-nodes as a case study of NFV-VIPP usage. Precise understanding of the system load is a crucial requirement for VNF management in that it is a key indicator for auto-scaling/migration as well as troubleshooting of packet losses and processing delays.

A. Problem Statement

The DPDK's implementation styles, exclusive CPU core occupation by PMDs for polling-based packet reception, invalidate common CPU-usage-based estimation of system load due to 100%-fixed CPU usage. Memory usage or packet amount obtained by Barometer could be available for load estimation (how many packets in processing now is understandable); however, how much each packet processing costs is obscure. Heavy packet processing, such as tunnel encapsulation, can cost multiple times more than simple forwarding, and therefore, we are first noticed of overloaded conditions of NFV-nodes when consecutive packet losses have occurred in realistic situations. Hence, a novel means of precisely understanding the system load of NFV-nodes is necessary.

B. CPU-cycle Consumption in Packet Processing Loop

A required load indicator should react to changes in both traffic amount and processing cost of each packet. Conventional CPU usage, a ratio of CPU-cycles consumed by a processor in a given time unit, satisfies this requirement. We have found an analogous metric in DPDK internal that switches the notion of meaningful CPU-cycle consumption. Figure 3 illustrates a common packet processing loop model of DPDK. PMDs continuously repeat a packet processing loop, consisting of reception (Rx), main processing (Process), and transmission (Tx), involving a batch of certain packets (up to 32). Total CPU-cycles consumed in a single loop satisfies the above requirement as follows: (i) increase of traffic rate lengthens the batch size in a loop (resulting in larger CPU-cycle consumption) (ii) increase of processing load of each packet involves additional CPU-cycles to be consumed. Besides, adapting CPU-cycle consumption of each loop as the indicator can distinguish meaningful packet processing from idle receive polling that also consumes reasonable CPU-cycles.

C. Estimation of the Load using CPU-cycle Consumption

Catching a sign of pre-overloaded conditions prevents performance issues before they occur. Here, we explain how the newly introduced load indicator (average CPU-cycle consumption per packet processing loop) is used to grasp the signs.

First we consider a relationship between the CPU-cycle consumption and input traffic load. There is conceptual equivalence in terms of CPU-cycle consumption between the two types of loads derived from traffic amount increase and per-packet processing cost growth. In other words, varying traffic rate (amount) and varying per-packet processing cost, ranging from zero to a maximum point where the NFV-node can accept without causing packet losses, result in similar fluctuation of CPU-cycle consumption. Therefore, we can understand the relationship in advance of real operation by examining various traffic rate with constant per-packet processing cost.

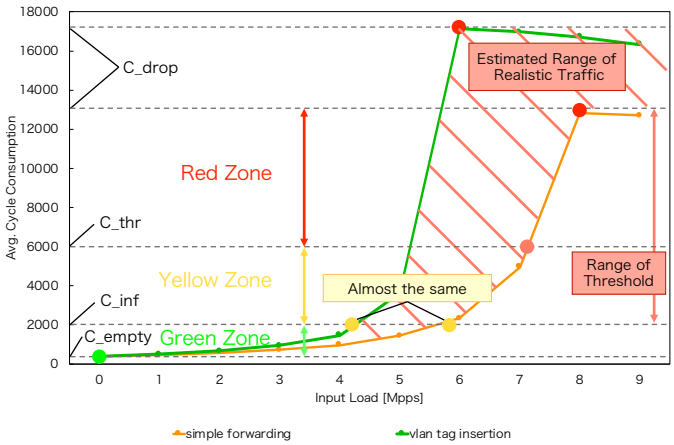

Fig. 4. Input Load vs. Cycles Consumption

Next, we demonstrate the load estimation using CPU-cycle consumption. The device under test NFV-node hosted a container-formed VNF (OVS-DPDK) over a baremetal OVS-DPDK in a P2V2P fashion. The machine specification is shown in Table I. We periodically varied traffic rate per 1 Mpps ranging from zero to the maximum, and the NFV-VIPP instances calculated average CPU-cycle consumption values of a million samples for each rate. In the experiment, the VNF performs two types of packet processing, simple forwarding and 802.1Q vlan tag insertion, respectively.

Figure 4 shows the result of the experiments. The horizontal axis represents the input load (incoming traffic rate), and the other axis represents the average CPU-cycle consumption of per packet processing loop. When the node keeps idle receive polling, the system load is 0% (C_empty) and likewise, the system load is 100% (C_drop) when the node cannot accept slightest bit of load further. The figure indicates us the existence of an inflection point (C_inf) where the consumption values drastically react to slight increase of input load when the load exceeds the point. The inflection point allows the Green/Yellow/Red alarming scheme in the operation of NFV-nodes. For instance, the SDN controller can assign additional flows to an NFV-node with a green label, while the controller should avoid populating further flows to a node with a yellow label. If a node has a red label, the controller will migrate some of existing flows to another NFV-node.

From the result, CPU-cycle consumption in case of VLAN tag insertion more quickly reached its maximum than that of simple forwarding, but the fluctuation trend of CPU-cycle consumption and the value of the inflection point (C_inf) are equivalent. These imply that the threshold value (C_thr) of the simplest packet processing is reasonable even for more realistic one, and therefore the understanding these key values by pre-measurement before real operation is possible.

VII. Conclusion

Introducing the concept of softwarization has drastically changed the means of networking. Such a wave is also affecting network operators by posing programmability of their network nodes. The combination of server-based NFV-nodes and DPDK is a practical scenario in various commercial networks, but the complexity of their structures prevents them from being easily managed and highly optimized. Nowadays both network operators and VNF developers need to collaborate each other to tackle the issues in the 5G era.

In this paper, we have proposed a DevOps friendly NFV-VIPP for seamless understanding of NFV-node internals, and demonstrated the usefulness of CPU-cycle consumption in the polling loop to estimate the system load. We are planning to extend the framework to identify packet loss points within the node by exploring further promising internal metrics.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP19K11940.

References

[1] "vVIG: virtual VPN Internet Gateway [Online]," (Sep. 30, 2019), https://www.sdxcentral.com/articles/news/att-intel-push-pull-strategy-nfv/2017/08/.
[2] Y. Ohara, H. Shirokura, A. D. Banik, Y. Yamagishi, and K. Kyunghwan, "Kamuee: An IP Packet Forwarding Engine for Multi-Hundred-Gigabit Software-based Networks."
[3] P. Naik, D. K. Shaw, and M. Vutukuru, "NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Palo Alto, CA, USA, Nov 2016, pp. 154–160.
[4] "Barometer [Online]," (Sep. 30, 2019), https://wiki.opnfv.org/display/fastpath/Barometer+Home.
[5] "DPDK: Data Plane Development Kit [Online]," (Sep. 30, 2019), https://dpdk.org.
[6] M.Dodare, Y. Taguchi, R. Kawashima, H. Nakayama, T. Hayashi, and H. Matsuo, "Visualizing the NFV Node Conditions based on DPDK's Processing Model -A Case Study of Load Measurement-," *IEICE Technical Report*, vol. 118, no. 303, pp. 33–38, Nov 2018, (in Japanese).
[7] "OpenStack [Online]," https://www.openstack.org/.
[8] D. Cotroneo, R. Natella, and S. Rosiello, "NFV-Throttle: An Overload Control Framework for Network Function Virtualization," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 949–963, Dec 2017.
[9] G. Liu and T. Wood, "Cloud-Scale Application Performance Monitoring with SDN and NFV," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 440–445.
[10] A. Shatnawi, M. Orrù, M. Mobilio, O. Riganelli, and L. Mariani, "CloudHealth: A Model-Driven Approach to Watch the Health of Cloud Services," in *2018 IEEE/ACM 1st International Workshop on Software Health (SoHeal)*. IEEE, 2018, pp. 40–47.
[11] F. Moradi, C. Flinta, A. Johnsson, and C. Meirosu, "ConMon: an Automated Container Based Network Performance Monitoring System," in *Proc. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, May 2017, pp. 54–62.
[12] "collectd [Online]," https://collectd.org/.
[13] J. Xie, M. Miao, F. Ren, W. Cheng, R. Shu, and T. Zhang, "Overload Detecting in High Performance Network I/O Frameworks," in *Proc. 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, Australia, Dec 2016, pp. 999–1006.
[14] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple network management protocol (snmp)," Tech. Rep., 1990.
[15] "Telemetry [Online]," (Sep. 30, 2019), https://tools.ietf.org/html/draft-song-opsawg-ntf-03.
[16] "ZABBIX [Online]," https://www.zabbix.com/features.
[17] S. Hajnoczi, "virtio-vsock: Zero-configuration host/guest communication," in *KVM Forum*, 2015.
[18] "The C++ REST SDK [Online]," (Sep. 30, 2019), https://microsoft.github.io/cpprestsdk/index.html.
[19] "Open vSwitch [Online]," http://openvswitch.org/.