Design and Implementation of eBPF-based Virtual TAP for Inter-VM Traffic Monitoring

Jibum Hong*, Seyeon Jeong*, Jae-Hyoung Yoo†, James Won-Ki Hong*
*Department of Computer Science and Engineering, POSTECH, Pohang, Korea
†Graduate school of Information Technology, POSTECH, Pohang, Korea
{hosewq, jsy0906, styoo, jwkhong}@postech.ac.kr

***Abstract*—With the proliferation of cloud computing and services, the internet traffic and the demand for better quality of service are increasing. To handle a huge amount of traffic using limited resources in a data center, server virtualization technology, which uses the resources of internal servers in the data center more efficiently, is receiving increased attention. However, the existing hardware test access port (TAP) equipment that duplicates packets for monitoring has many drawbacks, which make it unfit for deployment in the virtual datapaths configured for server virtualization. vTAP, which is a software version of the hardware TAP, overcomes this problem by duplicating packets in a virtual switch. However, implementation of vTAP in a virtual switch has a performance problem because it uses the computing resources of the host machines. To overcome this problem, we propose a vTAP implementation technique based on the extended Berkeley packet filter (eBPF), which is a high-speed packet processing technology. Finally, we compare its performance with that of the existing virtual TAP.**

***Index Terms*—virtual network monitoring, extended Berkeley Packet Filter (eBPF), Linux kernel networking**

I. Introduction

Cloud computing combines many technologies (Web services, virtualization, service-oriented architecture, and so on) and business models to deliver IT capabilities (software, platforms, hardware) [1]. As the cloud infrastructure and services are proliferating, the amount of traffic and requirements for higher service quality are increasing rapidly. In this situation, the server virtualization technology, which makes effective use of server resources using virtual machines (VMs), looks promising. In data centers, communication among VMs in the same server or VMs that are physically located in different servers occurs concurrently for processing various service requests. So, the east-west traffic among VMs in a data center is increasing significantly.

The existing hardware test access port (TAP) equipment duplicates packets that pass through a link between a terminal and a router (or a switch) and then transmits the duplicated packets to monitoring systems. However, in a server virtualization environment, links among VMs in the same host machine are virtual links configured by a virtual switch. Thus, it is impossible to install hardware TAP equipment. This makes it difficult to monitor traffic among VMs at a packet level.

To overcome this problem, the virtual test access port (vTAP), which is a software implementation of hardware TAP, provides inter-VM traffic monitoring at the packet level. The vTAP can be implemented by using virtual switches to connect VMs in the server for packet-level monitoring and traffic engineering.

In this paper, we present the design and implementation strategy of the vTAP. Because the vTAP uses the resources of the host machine for packet duplication and delivery, the performance of virtual switches and VMs, which share the same host machine resources, is degraded. To overcome this problem, we propose a vTAP based on the extended Berkeley Packet Filter (eBPF), which is a kernel-based high-speed packet processing technology for making optimum use of the limited resources of the host server.

The remainder of this paper is organized as follows: Section II provides the background technologies related to our vTAP and discusses the related issues. Section III details the design and implementation of the eBPF-based vTAP, and Section IV evaluates the performance by comparing the eBPF-based vTAP with the vTAP based on the virtual switch. Finally, in Section V, we conclude our paper and discuss future work.

II. Background and Related Work

A. Background

The vTAP commonly duplicates the traffic among VMs on host machines (servers) at the packet level and transfers the copies to a monitoring system to analyze the packets. Our design and implementation of vTAP is based on a kernel-based virtual machine (KVM) and eBPF, which is an open-source project to enable in-kernel high-speed packet processing and offers scalability without limitations.

The extended Berkeley Packet Filter [2] is an open-source IO Visor Project by the Linux community. The eBPF is an extension of the performance and universality of BPF used for packet filtering and monitoring in Linux. The simple BPF cannot call another BPF program because interaction is not available. Additionally, it has some elements that complicate the just-in-time (JIT) compilation and some disadvantages in program management. Compared with the existing Linux kernel-based implementation for networking applications and their limited capabilities in developing in-kernel networking functions, the eBPF converts a program written in high-level languages such as C and Python into bytecode, loads the program into the kernel, and calls kernel-level functions. As shown in Fig. 1, the verifier rejects the bytecode if it is deemed unsafe. The eBPF associates in-kernel event sources

978-3-903176-14-0

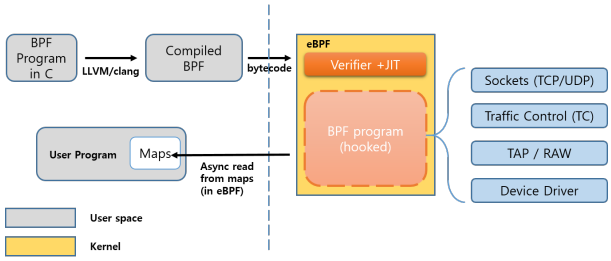

Fig. 1. Overall Process of eBPF

with desirable event handlers (sockets, traffic control, and so on). There is growing interest in kernel-based network and system monitoring through eBPF and utilization of high-speed packet processing.

Open vSwitch (OVS) [3] is a multi-layer Linux-based virtual switch that provides a virtual network construction environment among VMs. It is designed to enable massive network automation through programmatic extension. OVS can support many standard interfaces (e.g., OpenFlow protocol) and be used as a software-defined-network (SDN) switch to automate network configuration and management.

B. Related Work

Presently, there are some commercial vTAP solutions that provide visibility into inter-VM traffic by monitoring virtual networks in server virtualization environments.

Veryx vTAP is a pluggable software component that resides as a layer over the hypervisor and works with virtual switches that connect the virtual network interface controllers (vNICs) of the VMs. This vTAP monitors traffic, replicates it, and delivers it to external monitoring systems such as a network analyzer or an intrusion detection system (IDS) [4]. Gigamon vTAP, in which a monitoring agent is installed in each VM, duplicates the incoming and outgoing packets between VMs and transmits those to a monitoring platform for virtual traffic visibility [5]. Finally, IXIA vTAP is implemented by modifying the kernel of the host machine. It collects and transmits the packets between VMs to an external monitoring program [6].

All these commercial solutions provide high throughput and utilize a small amount of system resources. However, these solutions create dependencies because of their proprietary implementation, and the users (customers) face difficulties with their maintenance and scalability. In contrast, the design and implementation of vTAP proposed in this paper is based on the open-source eBPF that facilitates scalability.

Many studies have used vTAP-like functionalities to realize effective network monitoring in SDN/OpenFlow networks. Planck is a study of the network measurement system, which provides millisecond-scale monitoring for commodity networks. It was conducted to sample specific traffic flows using the port mirroring function of the hardware OpenFlow switch and to improve sampling accuracy and data acquisition speed [7]. NetAlytics is a study on the cloud-scale application performance analysis by processing data with Intels high-speed packet processing tool (DPDK) and port mirroring capabilities of the hardware OpenFlow switch to deliver copies of packets between VMs to a monitoring system. However, this study assumed only one VM per host machine [8].

Packet duplication can be useful for application in machine learning (ML) also. As many ML techniques are introduced in networking, especially for envisioning self-driving and intelligent networking, duplicated packets can offer various feature data (e.g., packet header values) to be learned in building an off-line or on-line network behavior model for different goals that are not easy for humans or have no predefined rules to go by. Amaral et al. [9] used port mirroring to extract feature data (e.g., timestamp, inter-arrival time, flow duration) from duplicated packets. This packet collection allowed the authors to achieve a better accuracy on packet classification compared to the existing rule- or signature-based packet inspection systems. Abubakar and Pranggono [10] also use port mirroring to collect packets and their metadata (flow-level features) to implement the ML-based intrusion detection system.

While all these studies dealt with network monitoring enabled by the port mirroring function of hardware switches, our research focuses on the implementation of the vTAP itself, which can duplicate packets between VMs at high speeds and effectively control user-defined monitoring policies by using the eBPF and Linux virtualization techniques.

III. Design and Implementation

Fig. 2 shows the operation and the function of the vTAP and the virtual switch compared to the limit of the hardware TAP equipment in the server virtualization environment. As mentioned in Introduction, there is performance degradation in the virtual-switch-based vTAP implementation because the switch has to use and share the physical resources with other VMs [11]. So we implemented the eBPF-based in-kernel vTAP function by making Linux host machines or VMs to take the role of packet copy instead of virtual switches.

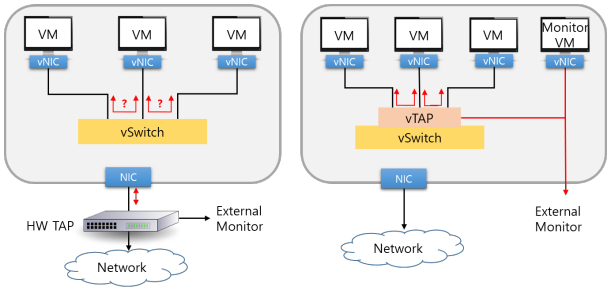

Fig. 2. Physical TAP and Virtual TAP

The conceptual design of our eBPF-based vTAP is shown in Fig. 3. The Linux machine has an input port connected to the VM, which sends input packets, and output ports connected to the destination VM (receiver VM) and monitoring VM (monitor VM). The traffic delivered through the Linux machine is

collected by the vTAP we implemented. The vTAP identifies and duplicates the packets to deliver them to the monitoring VM. The Python program of vTAP is configured to monitor the network interface, and the C program uses the BPF library to create logic that performs real vTAP functions, such as VLAN tagging used for packet duplication and identification of duplicates.

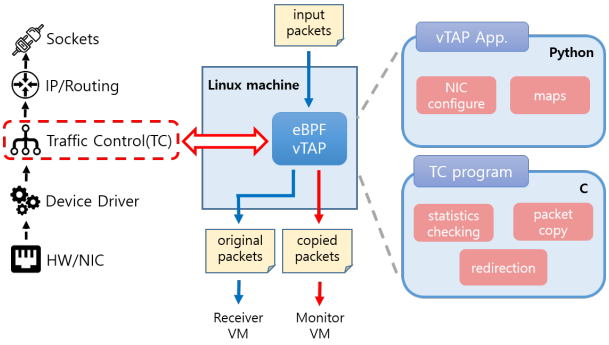

Fig. 3. Overview of eBPF-based vTAP

To elaborate, the vTAP application written in Python specifies the hooking points in the kernel networking stack to which the corresponding bytecode will be bound. The feature of vTAP binds to the traffic control (TC) because it can be implemented by calling the API for eBPF on the packet arriving at the Linux TC. The application also calls the eBPF program that will be hooked to a network event such as incoming and outgoing packets. At each event, the vTAP application calls the TC program, and the TC program reads the socket buffer. Then it looks up or initializes the key-value based hash map to identify the packet header, such as its source and destination IP/MAC addresses, by parsing the packet structure. If the source or destination IP address of the packet is in the list that we want to monitor, the packet is duplicated using the BPF library; otherwise the packet just passes through the vTAP. Next, the duplicated packet is redirected to the monitoring VM.

IV. Evaluation

To implement and verify our eBPF-based vTAP function described in the preceding section, in this section we will test the throughput of packets transmitted to the monitoring VM because the main purpose of our vTAP implementation is to improve the performance. Then we will perform the application level experiment using an intrusion detection system (IDS). The experimental environments are built on the physical and virtual machines as shown below.

We use Intel Core i7-4790 (3.6 GHz, 4 cores) processor, 16 GB of memory, and 256 GB SSD storage for the physical machine. The virtual environment is configured by Oracle VirtualBox version 5.2.12 with Ubuntu 18.04 (kernel 4.15.0), and VMs for testing are managed by QEMU/KVM hypervisor. OVS (version 2.9.0) is installed to compare performances in the same environment.

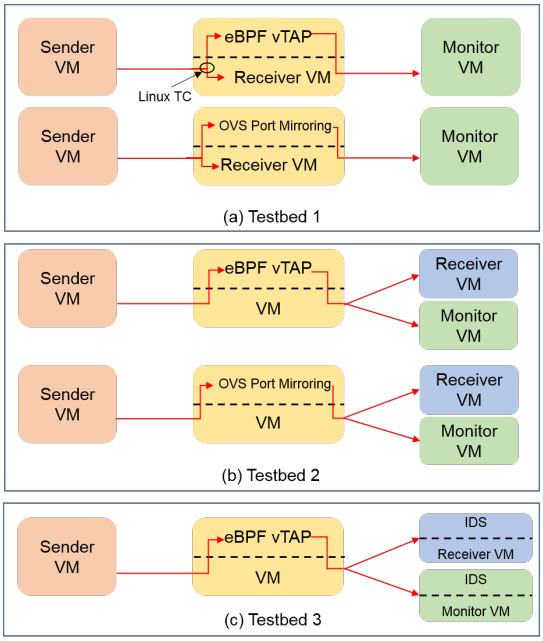

Fig. 4. (a) Testbed 1: Receiver has the role of packet copy. (b) Testbed 2: A dedicated VM that has the role of packet copy. (c) Testbed 3: Application of vTAP to Intrusion Detection System (IDS)

A. Testbed 1: Same location Receiver and vTAP

Packets are generated by Pktgen, which is a high- performance network testing tool in the Linux kernel [12]. To validate our eBPF-based vTAP, we compare the performance of the set-up presented in this paper with that of OVS port mirroringwhich is used to send a copy of packets to another monitoring port through the Open vSwitch Database (OVSDB) protocolby measuring the RX throughput of duplicated packets at Monitor VM. To use the OVS port mirroring function, we apply a port mirroring policy of the source port (Sender), destination port (Receiver), and output port (Monitor) for the duplicated packets.

In the first case, as shown in Fig. 4(a), Sender VM sends the UDP packets to Receiver VM, where the eBPF vTAP is operated. Then the vTAP on Receiver VM duplicates and redirects the packets to Monitor VM, which captures packets and checks performance. The metrics of throughput, packets per second (PPS), and CPU usage are measured every second. Fig. 5 and Table I show the measured average values of 20 trials for each case.

The OVS port mirroring processes an average of about 96,000 packets per second in case of the 64-byte packet, and this number decreases to about 87,000 as the packet size increases (Table I). As for the throughput, it shows an average of about 38 Mbps at the 64-byte packet size, and it increases in multiples up to about 1,002 Mbps as the packet size increases. On the other hand, our eBPF-based vTAP processes an average of 130,000 packets per second and shows about 25 Mbps more than OVS port mirroring in case of the 64-byte packet. The throughput increases as the packet size increases. The eBPF-

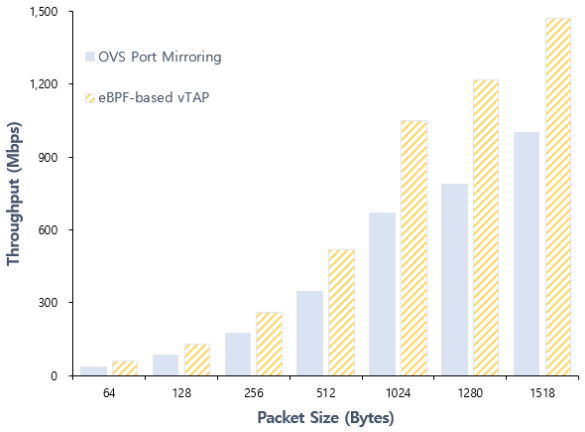

Fig. 5. Throughput Comparison of Testbed 1

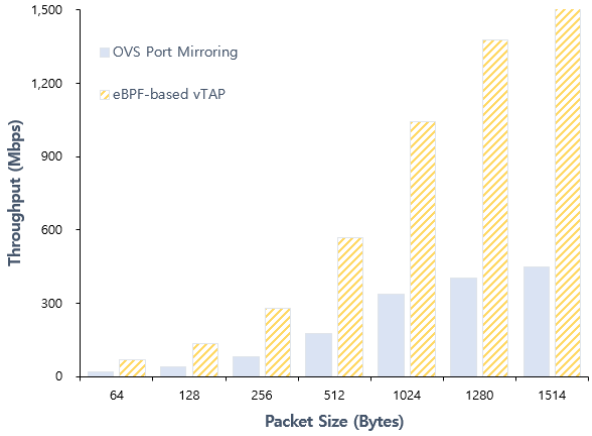

Fig. 6. Throughput Comparison of Testbed 2

TABLE I
DETAIL PERFORMANCE MEASUREMENT OF TESTBED 1

Packet Size (bytes)	OVS Port Mirroring		eBPF-based vTAP	
	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)
64	96,502	37.99	127,504	62.09
128	97,966	85.16	134,980	131.72
256	94,425	175.57	132,575	259.00
512	90,755	348.17	132,510	518.66
1024	87,158	669.23	134,326	1,050.34
1280	81,541	787.96	124,588	1,217.92
1518	87,475	1,002.42	126,952	1,473.10

TABLE II
DETAIL PERFORMANCE MEASUREMENT OF TESTBED 2

Packet Size (bytes)	OVS Port Mirroring		eBPF-based vTAP	
	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)
64	45,719	20.71	139,853	66.55
128	45,976	41.18	141,582	135.80
256	42,641	79.25	141,350	278.48
512	48,457	176.16	147,748	569.34
1024	43,652	337.67	133,010	1,040.92
1280	41,505	403.45	135,927	1,375.43
1514	38,928	449.51	128,091	1,507.60

based vTAP shows less CPU usage (about 1.6%) than OVS port mirroring (about 2.3%).

B. Testbed 2: Different location Receiver and vTAP

In the second case, as shown in Fig. 4(b), Sender VM sends the packets to Receiver VM, which is separated from the vTAP VM. Then vTAP duplicates the packets and redirects them to Monitor VM. The metrics used in this case are the same as those used in Testbed 1 and are measured at both Receiver VM and Monitor VM. While Testbed 1 uses 2 CPUs per VM, Testbed2 uses only 1 CPU due to the resource limitation of the physical machine. The maximum packet size is also slightly less than that in Testbed 1.

OVS port mirroring is heavily influenced by resource limits. It processes an average of about 46,719 packets per second in case of the 64-byte packet, and the PPS value decreases below 40,000 when the packet size is 1514 byte (Table II). Throughput also decreases by almost 50% that of Testbed 1. The throughput shows an average of about 20 Mbps at the 64-byte packet size, and about 450 Mbps at the 1,514-byte packet size. In contrast, our eBPF-based vTAP processes packets almost similarly to the performance of Testbed 1. The CPU usage of OVS (about 13.5%) is double of our eBPF-based vTAP.

We observe that our eBPF-based vTAP outperforms the OVS port mirroring in our test scenarios. Because eBPF can be easily mapped to native instructions for JIT compilation and the TC program intercepts data when it reaches the kernel traffic control function in the RX or TX mode, our eBPF approach can process the packets faster than OVS port mirroring.

C. Testbed 3: Application of vTAP to Intrusion Detection System (IDS) analysis

In this experiment, we further use the proposed vTAP function to feed copied packets into the IDS instance operated in the Monitor VM (Fig. 4(c)). IDS is one of typical applications of vTAP where its capability of packet-level monitoring is useful for security guarantee in internal networks.

We choose Suricata [13] as the target IDS because it is a free IDS and intrusion protection system (IPS). Additionally, it offers high performance of multi-threading and updated rule sets for new network attacks. In our case, it operates in IDS mode to which packets are usually offered by TAP and which generates alert reports on anomalous packets. Otherwise, IPS is usually located in alignment with a packet delivery path, similarly to firewall.

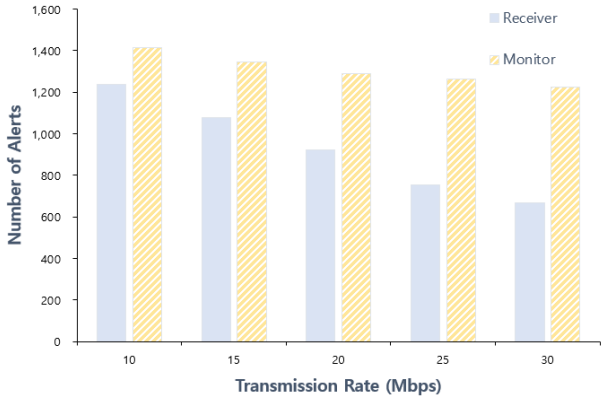

Fig. 7. Suricata IDS Alerts Comparison of Testbed 3

TABLE III
SURICATA IDS EXPERIMENTAL RESULTS OF TESTBED 3

Transmission Rate (Mbps)	Receiver		Monitor	
	Alerts (#)	Drop Rate (%)	Alerts (#)	Drop Rate (%)
10	1,238	59.31	1,413	0.00
15	1,080	57.42	1,345	0.00
20	924	55.89	1,289	0.74
25	753	55.61	1,263	0.01
30	666	58.50	1,223	0.00

To compare both results on original packets at Receiver VM and duplicated packets at Monitor VM, we also install the Suricata IDS instance in Receiver VM. So, packets generated by Sender VM are sent to eBPF vTAP running on the dedicated VM where they are duplicated, forwarding originals to Receiver VM and duplicates to Monitor VM. Each IDS instance generates an analysis result on its packet received. Due to the resource limitation of the physical machine, each VM has 1 vCPU and 2 GB memory.

More specifically on the testbed setup, we use the pytbull [14] IDS/IPS testing framework to generate anomalous packets from Sender VM to Receiver VM. This tool allows one to generate various network attacks such as DoS, shell-code, and others. We capture packets in one run of the whole generation of those attacks at Sender VM to replay them several times in the same network configuration with various parameters (e.g., bandwidth, loop) using Tcpreplay [15].

Fig. 7 shows the results of replays of the captured network attacks with different transmission rates. Receiver IDS (the IDS on Receiver VM) generated an average of 1,238 alerts at a transmission rate of 10 Mbps. As the transmission rate increased, the alerts generated by Receiver IDS decreased to 666 at 30 Mbps. Otherwise, Monitor IDS (the IDS on Monitor VM) generated more alerts overall than did Receiver IDS. It generated 1,413 alerts at a transmission rate of 10 Mbps and 1,223 alerts at 30 Mbps as the transmission rate increased. The difference between the number of alerts according to the transmission rate was smaller for Monitor IDS than for Receiver IDS.

Along with the alerts comparison shown in Fig. 7, we also compared the drop rate of packets of each Suricata IDS (Table III). Packets arriving at Receiver IDS showed an average drop rate of 59.31% at a transmission rate of 10 Mbps. Despite increase in the transmission rate, the drop rate was 58.50% at 30 Mbps, so it did not show such drastic differences in the case of alert generation. In contrast, the drop rate of Monitor IDS packets remained zero regardless of transmission rates.

From the comparison of drop rates between Receiver and Monitor IDS, Receiver VM suffered more packet drops than did the Monitor VM. Through further analysis on several resource usage statistics (e.g., CPU utilization, packet count) in the testbed, we observed that although Receiver VM received around two times of packets more than that of Monitor VM, Receiver VM showed drastic packet drops, which are mainly caused by CPU saturation as a result of a large amount of per-packet interruptions (e.g., softirq) in the VM. This leads to two interpretations: (a) the use of 1 vCPU in the VM was a performance bottleneck and (b) eBPF vTAP had additional packet processing overhead on the duplicated packets for Monitor VM, showing a lower packet throughput than original packets for Receiver VM. To improve the result, we can consider allocating more resources on the VMs and improving the vTAP business logic with various offerings from eBPF (e.g., egress queue discipline).

V. CONCLUSION

In a server virtualization environment, VMs are connected through virtual network interfaces and virtual switches. Because the existing hardware TAP cannot be used for inter-VM packet monitoring (duplication), we need a vTAP function somewhere in a point such as a virtual switch. However, a virtual-switch-based vTAP degrades performance because it shares resources of the host machine with other VMs. To overcome this problem, we designed and implemented a vTAP using eBPF, which uses the latest open-source high-speed packet processing technology. The experimental results show that our eBPF-based vTAP performs better than OVS port mirroring in terms of PPS, RX throughput, and CPU usage. Furthermore, eBPF can provide better extendibility in developing additional functions, for instance, TC-based queue scheduling.

Presently, all experiments were performed on the host machine with constrained resource. In the future work, we will extend our eBPF-based vTAP to make it more practical at the application level. Furthermore, we will extend the testbed with various scenarios and validate our eBPF-based vTAP by comparing the result to other high-speed vTAP functions, such as OVS-DPDK [16] and other commercial solutions. Furthermore, we will try to extend our vTAP with XDP (eXpress DataPath), which is a high-speed packet processing technology that does not require kernel bypass [17].

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Ministry of Science and ICT (MSIT), Korea (No.2018-0-00749, Devel-opment of virtual network management technology based on artificial intelligence), and the Korea government (MSIT) (No.2015-0-00575, Global SDN/NFV Open-Source Software Core Module/Function Development)

REFERENCES

[1] O. Sefraoui, M. Aissaoui, and M. Eleuldj, Open-stack: Toward an open-source solution for cloud computing, Int. J. Comp. Appl. (0975-8887), vol. 55(03), October 2012.

[2] IO Visor Project, extended Berkeley Packet Filter. [Online]. Available at https://www.iovisor.org/technology/ebpf.

[3] Ben Pfaff et al., The design and implementation of Open vSwitch, In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) , 2015, pp. 117-130.

[4] VeryX, Veryx vTAP datasheet, Tech. Report, 2017. [Online]. Available at http://www.veryxtech.com/wp-content/uploads/2017/11/Datasheet-Veryx-vTAP_20171115.pdf.

[5] Gigamon, GigaVUE-VM datasheet, Tech. Report, 2016. [Online]. Available at https://www.gigamon.com/content/dam/resource-library/english/data-sheet/ds-gigavue-vm-virtual-machine.pdf.

[6] IXIA, Ixia Panthom vTAP with tapflow filtering, Tech. Report, 2016. [Online]. Available at https://www.viavisolutions.com/pt-br/literature/ixia-phantom-vtap-tapflow-filtering-data-sheet-en.pdf.

[7] J. Rasley et al., Planck: Millisecond-scale monitoring and control for commodity networks, In ACM Conference on SIGCOMM, 2014, pp. 407-418.

[8] G. Liu and T. Wood, Cloud-scale application performance monitoring with SDN and NFV, In 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, 2015, pp. 440-445.

[9] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, Machine learning in software defined networks: Data collection and traffic classification, In 2016 IEEE 24th International Conference on Network Protocols (ICNP), Singapore, 2016, pp. 1-5.

[10] A. Abubakar and B. Pranggono, Machine learning based intrusion detection system for software defined networks, In 2017 Seventh International Conference on Emerging Security Technologies (EST), Canterbury, 2017, pp. 138-143.

[11] S. Jeong, D. Lee, J. Li, and J. W. Hong, OpenFlow-based virtual TAP using open vSwitch and DPDK, In 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, 2018, pp. 1-9.

[12] Robert Olsson, Pktgen the linux packet generator, In Proceedings of the Linux Symposium, Ottawa, Canada, 2005, p. 11-24.

[13] Suricata, Open source IDS/IPS/NSM engine. [Online]. Available at https://suricata-ids.org/.

[14] pytbull, IDS/IPS testing framework. [Online]. Available at http://pytbull.sourceforge.net/?page=documentation.

[15] Tcpreplay, Pcap editing and replaying utilities. [Online]. Available at https://tcpreplay.appneta.com/wiki/overview.html.

[16] Open vSwitch, Open vSwitch with DPDK. [Online]. Available at http://docs.openvswitch.org/en/latest/intro/install/dpdk/.

[17] IO Visor Project, eXpress data path. [Online]. Available at https://www.iovisor.org/technology/xdp.