Probius: Automated Approach for VNF and Service Chain Analysis in Software-Defined NFV

Jaehyun Nam
KAIST
namjh@kaist.ac.kr

Junsik Seo
KAIST
js0780@kaist.ac.kr

Seungwon Shin
KAIST
claude@kaist.ac.kr

ABSTRACT

As the complexity of modern networks increases, virtualization techniques, such as software-defined networking (SDN) and network function virtualization (NFV), get highlighted to achieve various network management and operating requirements. However, those virtualization techniques (specifically, NFV) have a critical issue that the performance of virtualized network functions (VNFs) is easily affected by diverse environmental factors (e.g., various workloads, resource contentions among VNFs), so resulting in unexpected performance degradations - *performance uncertainty*. Unfortunately, existing approaches mostly provide limited information about a single VNF or the underlying infrastructure (e.g., Xen, KVM), which is deficient in reasoning why the performance uncertainties occur. For such reasons, we first deeply investigate the behaviors of multiple VNFs along service chains in NFV environments, and define a set of critical performance features for each layer in the NFV hierarchical stack. Based on our investigations and findings, we introduce an automated analysis system, Probius, providing the comprehensive view of VNFs and their service chains on the basis of NFV architectural characteristics. Probius collects most possible NFV performance related features efficiently, analyzes the behaviors of NFV, and finally detects abnormal behaviors of NFV - possible reasons of performance uncertainties. To show the effectiveness of Probius, we have deployed 7 open-source VNFs and found 5 interesting performance issues caused by environmental factors.

CCS CONCEPTS

• **Networks → Network performance analysis**; *Network monitoring*; Network measurement;

KEYWORDS

Virtualized Network Function; Service Chain; Performance Analysis

ACM Reference Format:
Jaehyun Nam, Junsik Seo, and Seungwon Shin. 2018. Probius: Automated Approach for VNF and Service Chain Analysis in Software-Defined NFV. In *SOSR '18: SOSR '18: Symposium on SDN Research, March 28–29, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3185467.3185495

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SOSR '18, March 28–29, 2018, Los Angeles, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185495

1 INTRODUCTION

Modern enterprise networks are now in transforming into a new era of virtualization. By adopting new virtualization techniques, such as software-defined networking (SDN) and network function virtualization (NFV), enterprise networks are able to reduce their operational/management costs and complexities [55]. For example, CoMb has introduced software-defined middleboxes to replace existing hardware middleboxes [56] and we can also find real commercial products relying on these virtualization techniques [22].

However, as with great power comes great responsibilities, these techniques raise another critical side issue of *performance uncertainty*, meaning unexpected performance degradation or variation of NFV. Since those virtualization techniques are mostly operated with software functions (instead of hardware logics), it is hard to say their expected performance (compared with hardware dedicated ones). There are many variables, such as co-locating applications, OS scheduling methods, and I/O bottlenecks, causing performance variations and they make us hard to predict average, minimum, and maximum performance of an NFV system.

While this performance uncertainty is a known problem, most existing works have not touched this problem clearly. Most recent NFV related works have focused on its performance issue, technically improving the performance of NFV. They are trying to reduce network I/O overheads of NFV by employing high-performance packet I/O engines [30, 33, 45, 49–51], or to eliminate inter-VM communication overheads by introduced user-space packet processing stacks (e.g., E2 [46], NetVM [32, 61]). Indeed, those pioneering works are really helpful in letting people adopt NFV in real world environments. However, those works are still not able to solve the problem of performance uncertainty. They do NOT reveal where is the major bottleneck point of an NFV system, and they do NOT show why this performance uncertainty happens.

We may be able to employ the knowledge from other domains (e.g., system profiling) to solve this issue, and we notice that some early stage works have been proposed [39, 43, 52]. For example, VBaaS [52] focuses on the variations of vCPU usages and throughputs in VNFs to understand the behaviors of VNFs, and NFVPerf [43] presents the throughput and delay differences between inter-VNF communication paths to understand service chains across VNFs. In addition, there are some more systematic methods to profile virtualized environments [6, 31, 44]. Unfortunately, their outcomes are still uncertain and do not explain why a performance degradation happens although they may point out some conspicuous locations. Since VNFs are highly dependent on network workloads, the profile results could be different according to constantly changing workloads, and this situation is much worse, if VNFs are distributed, causing service chains. Moreover, the overall performance of VNFs could be significantly degraded due to the profiling overheads. In

sum, current analysis approaches are insufficient to thoroughly address the root causes of performance issues in NFV environments.

To understand the performance uncertainty of NFV (i.e., why and where it happens), we introduce a comprehensive and systematic NFV profiling approach. We first investigate the behaviors of VNFs based on the correlations with a hypervisor and other VNFs in a service chain, giving insight into the analysis methodology suitable for NFV environments. With the knowledge of NFV environments, we define performance features required to analyze each layer in the NFV architecture. Then, we design Probius, a novel analytic system that automatically collects the features for VNFs from each layer in the NFV hierarchical stack and analyzes performance problems through performance anomaly detection and graph-based behavior analysis. For the automated analysis of various service chains, Probius generates all possible service chains with the given VNFs and automatically builds up the VNFs of each service chain in NFV environments with different resource constraints and workloads.

The analysis part of Probius is conducted with three major steps: *feature collection, performance anomaly detection, and behavior analysis*. First, Probius extracts performance features from the entire NFV architecture (i.e., VMs containing VNFs, a virtual switch, a hypervisor, and even the underlying hardware). In addition, it draws the state transition graphs of VNFs to figure out the internal workflow of VNFs. Second, it sorts out service chains that may have some performance issues. For this, it employs regression analysis techniques to evaluate the influence of each service chain in all possible service chains. Lastly, it analyzes the behaviors of VNFs based on the criteria of our workflow classification. To point out the specific reasons of performance problems, it separates the state transitions of VNFs according to our criteria and figures out the most reliable reasons of the performance issues through the portion variations of each VNF. To show the effectiveness of Probius, we have deployed 7 open-source VNFs in our NFV environment with the KVM hypervisor. By analyzing those VNFs under various conditions, we have discovered 5 interesting performance issues caused by the environmental factors.

Our contributions are summarized as follows.

- We analyze the architectural characteristics of NFV environments based on the behaviors of VNFs along service chains and define performance features suitable for NFV environments.
- We develop Probius that automatically analyzes VNFs in various service chains and reasons out performance issues on the basis of the NFV architectural characteristics without human intervention.
- We provide the rigorous analysis results with 7 open-source VNFs and discuss possible performance issues according to how to deploy VNFs and how to make a service chain.
- We release the source code of Probius as an open-source project to let network operators analyze their NFV environments themselves.

2 BACKGROUND AND MOTIVATION

Here, we briefly present a basic NFV architecture and discuss its potential performance bottleneck points. Then, we introduce two example scenarios presenting performance uncertainty of NFV.

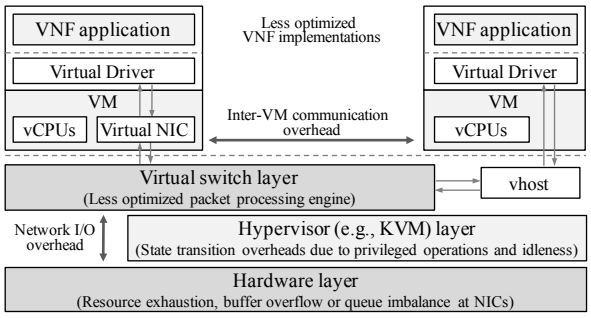

Figure 1: NFV architecture with major bottleneck points

2.1 NFV Architecture and Bottleneck Points

As shown in Figure 1, network function virtualization (NFV) is basically composed of four layers: virtualized network function (VNF), virtual switch, hypervisor, and hardware layers. Here, we describe the functionality and potential bottleneck points of each layer (from bottom to top).

Hardware layer: This layer basically provides hardware resources (e.g., CPU and memory) to NFV, and each hardware resource will affect the performance of NFV. Especially, according to the computing power and the number of CPUs, the packet processing performance would vary, resulting in the changes in the baseline performance of VNFs. Network interface cards (NICs) are also critical in terms of performance since they determine the maximum throughputs (e.g., 1Gbps, 10Gbps) of NFV environments. Besides them, disk performance can affect the overall throughputs as well.

In the hardware layer, most performance bottlenecks happen when system resource usages are reached at the hardware limitations. For example, when all CPU resources are exhausted, performance degradations can obviously arise. In the case of NICs, the buffer overflow or inefficient queue utilization can result in low performance. The disk I/O performance can incur longer I/O waiting time, unnecessarily making VNFs idle.

Hypervisor layer: Hypervisors (e.g., Xen [23] and KVM [34]) are used to virtualize the physical resources in the underlying hardware. Thus, when VNFs trigger I/O operations, hypervisors handle those operations instead of the VNFs since the target devices in the operations are also virtualized. To enhance the I/O performance of VNFs, in the case of KVM, it applies para-virtualized drivers (i.e., VirtIO [54]) that provide the fast and efficient I/O between VNFs and devices. Recently, by adopting the vhost-based networking model [37], a VNF can directly access network packets in the host kernel; thus, multiple packet copy steps across the hierarchical layers are no longer required.

In the hypervisor layer, frequent state transitions can cause the performance degradations of VNFs. Since VNFs necessarily require network I/O, they frequently trigger I/O interrupts. Unfortunately, when they trigger and receive interrupts, the state transitions of VNFs occur, generating high overheads to manage the context structures of VNFs. Moreover, even though a VNF is in an idle state, a hypervisor cannot know its state until CPU resources are allocated to the VNF. As a result, the idle state of a VNF can trigger frequent context switches, wasting the CPU resources.

Virtual switch layer: Virtual switches are a software layer that resides in the host. They provide the networking connectivity for VNFs. Similar to hardware-based switches, they also support various management interfaces and protocols (e.g., NetFlow, OpenFlow, GRE) for dynamic network management.

In general, virtual switches are often considered as performance bottlenecks in NFV environments since all network workloads pass through virtual switches and their complicated packet processing logics can cause longer delivery time than those in hardware-based switches. Thus, if a virtual switch is not accelerated or optimized for packet processing, it could become a bottleneck point. Due to this, previous studies [30, 32, 51] mostly focus on reducing the network I/O and inter-VM communication overheads.

VNF layer: Virtualized network functions (VNFs) are the implementations of network services formerly carried out by dedicated hardware, and they are operated on virtual machines (VMs) under control of a hypervisor. Since those are executed on x86 machines, they can be much more flexible than hardware-based appliances.

In the case of VNFs, most performance bottlenecks are found from less-optimized implementations for the given network environments. Before VNFs are released, developers usually optimize them for general cases. However, it is possible that the performance of VNFs is not maximized for certain cases.

2.2 Challenges in Existing Analysis Methods

As we noted above, there are many performance bottleneck points in each NFV layer, and thus it is very hard for network operators to specify which point(s) cause(s) performance bottleneck of NFV when they face performance uncertainty. Moreover, if multiple VNFs are correlated (i.e., service chaining), this situation is much worse. Although they try to inspect the root causes of the problems, existing analysis methods have limitation on pointing out the reasons due to the lack of understanding NFV environments. Here, we elaborate on why existing analysis methods are insufficient for the analysis of NFV environments and what makes difficult to find the fundamental reasons of performance uncertainty of NFV.

C1) Function-specific Analysis Methods: The traditional analysis methods can address performance issues within their local coverages. Thus, to analyze the behaviors of VNFs, network operators need a bunch of specialized analytical tools to extract various performance-related information. For example, network operators need multiple monitoring tools to collect host and VNF resource usages respectively. In depth, if they want to analyze VNFs in a systematic way (e.g., monitoring hardware-based performance counters or system events), they have to utilize system-wide profiling tools such as Perf [6] and Oprofile [38]. In terms of networking, they need network utility tools (e.g., netstat) to monitor the network statistics of each VNF. If they have the source code of VNF implementations, application profilers such as JProfiler [5] and GProf [3] should be used to inspect the code. Unfortunately, they cannot point out what is a fundamental performance bottleneck.

C2) Large Set of Unorganized Data: Even though network operators collect all required features from various sources, they need to analyze the collected data to figure out the faced problems. In this case, according to how much accurate reasons network operators want, the amount of the data could become huge. Especially, if they have the profile data from hardware and system events, the size of the entire data significantly increases. In addition, VNFs are highly related to network workloads; thus, they also need to collect the features according to various workloads, which means that the amount of the data would rapidly grow up and it could be beyond their abilities. In the end, unless they have some knowledge about how the features are correlated to each other in NFV environments, it is hard to deal with the huge amount of the data to find the fundamental performance issues.

C3) Hidden Correlations between VNFs: NFV environments are composed of multiple virtualization layers. Thus, there are lots of variable factors that internally influence on the performance of VNFs. For instance, the overheads from external factors (e.g., privileged instruction emulation overheads in a hypervisor, network I/O overheads in a virtual switch) are naturally involved in the processing times of VNFs. Moreover, due to the different processing and I/O waiting times of VNFs, hardware resources can be disproportionately allocated to VNFs. As a result, some of VNFs may waste their CPU resources for meaningless state transitions not packet processing. Besides them, not only VNFs but also the other NFV entities (e.g., virtual NICs at VNFs, a virtual switch, and even a hypervisor) utilize locks for multithreading. Unfortunately, their lock contentions significantly reduce the available processing time for each VNF. Likewise, without the knowledge of NFV environments, network operators can miss hidden issues and make a wrong guess. That is why we need to understand NFV environments to manifoldly analyze the root causes of performance issues.

2.3 Motivating Examples

The key benefit of NFV environments would be the flexibility of VNF deployments. Network operators can dynamically deploy VNFs for specific operating requirements. However, although they deploy VNFs with the consideration of diverse environmental conditions (e.g., available CPU and memory resources and network loads), it is possible that those VNFs behave unexpectedly. Here, we show some motivating examples supporting this claim. We deploy multiple VNFs (Suricata-IPS, NAT, Tcpdump, Firewall, Netsniffer-ng) with different chains (will be presented in each Figure, and each legend represents the sequence of a service chain).

Performance issue when adding more VNFs: Figure 2 illustrates the throughput variations when additional VNFs are attached to a service chain. The throughput of the service chain {Suricata-IPS, NAT} is saturated near by the maximum throughput of Suricata-IPS. However, when Tcpdump is added into the service chain, the throughputs of the new service chain {Tcpdump, Suricata-IPS, NAT} suddenly decreases. More surprisingly, the Tcpdump is a passive VNF, which means that it does not directly affect the throughput of a service chain because a virtual switch simply delivers incoming packets to the Suricata-IPS right after copying the packets to the Tcpdump. In turn, it is difficult to say that the Tcpdump is a performance bottleneck with the only reason that the performance of the service chain is dropped as soon as the Tcpdump is deployed. In addition, it is unclear that either the Suricata-IPS or the NAT is a bottleneck because the service chain with these VNFs previously shows reasonable throughputs. As a result, network operators

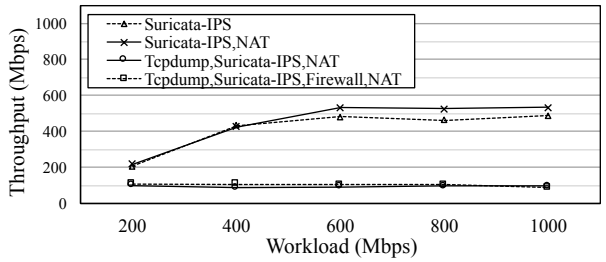

Figure 2: Throughputs with the different number of VNFs

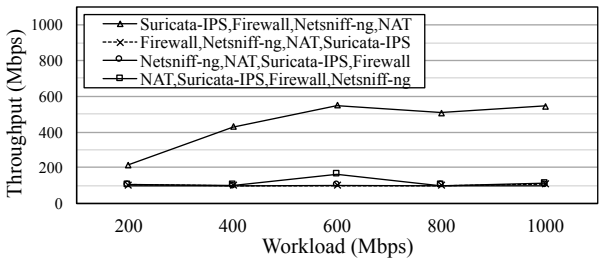

Figure 3: Throughputs with the different sequences of VNFs

cannot simply determine which VNF causes the performance degradation, and they eventually need the comprehensive analysis for NFV environments to find the actual reasons.

Performance issue when changing VNF sequences: Figure 3 shows the throughput variations in different service chains with the same VNFs. In this case, the throughputs of the service chain {Suricata-IPS, Firewall, Netsniff-ng, NAT} are similar to those of Suricata-IPS. However, the throughputs of the other chains significantly decrease although the only change is the sequence of VNFs. From this fact, we may think that when the NAT is followed by the Suricata-IPS, the overall throughputs might be affected. However, if a network operator only sees the throughputs of one of the three service chains, he cannot infer that the NAT or the Suricata-IPS might be a bottleneck. Furthermore, if he knows the previous performance issue, he cannot determine which VNF (i.e., NAT or the Suricata-IPS) may have a problem. Due to those reasons, to figure out whether some of VNFs affect performance degradations or the underlying infrastructure does, we need to analyze the entire NFV environments rather than focusing on a specific case.

2.4 Our Approach

Our strategy to analyze performance issues in NFV environments is four folds. First, we need to understand NFV environments as well as the behaviors of VNFs in service chains. This gives insight into a performance analysis methodology suitable for NFV environments. Second, we need to build the various kinds of service chains with the given VNFs. As shown in the previous examples, a simple sequence change of a chain causes serious performance impacts. To consider all possible cases in operation, we need to figure out exceptional cases before VNFs are deployed. Third, we need comprehensive monitoring and tracing mechanisms for both VNFs and the host. More specifically, monitoring system resources at both VNF and host sides allows to understand where system resources are mostly used in the NFV hierarchical stack. Tracing the behaviors of VNFs finds grounds to support the explanation of performance issues. Lastly, we need a system that automatically analyzes the collected data based on the NFV architectural characteristics and extrapolates the root causes of performance issues with reliable evidences. We explain the details of our system in the next section.

3 SYSTEM DESIGN

In this Section, we present Probius, an automated analysis system to find out the reasons of performance uncertainties in NFV environments. The goals of Probius are to comprehensively extract performance features from the NFV hierarchical stack and to extrapolate the root causes of performance uncertainties. To achieve those goals, we first investigate the behaviors of VNFs along their service chains in NFV environments (Section 4). With the knowledge of NFV environments, Probius collects a bunch of performance features from VNFs and the other NFV entities (e.g., a virtual switch and a hypervisor) by monitoring the system resources of VNFs and tracing the behaviors of VNFs (Section 5). Then, Probius discovers performance issues and their reasons through the regression analysis on the resource usages of VNFs and the behavior analysis on VNF state transition graphs (Section 6). To make all analysis procedures automated, we design an analysis framework that automatically builds all possible service chains with the given VNFs and emulates possible network workloads based on operator-defined configurations.

3.1 Probius Architecture

Figure 4 illustrates the overall architecture of Probius. It is composed of 6 main components (a service chain builder, a VNF manager, a workload manager, a monitor, a tracer, and a performance analyzer). In addition, Probius requires 3 types of configurations (global, VNF, and service chaining policies) for automation.

Service chain builder: As a first step, an operator provides Probius configuration - global, VNF, and service chaining policies (please see Section 3.2) - to define working environments of a target NFV. Then, the service chain builder generates all possible service chains based on this configuration, considering service chaining policies (if generated service chains violate the provided service chaining policies, they will be ignored). After the generation of service chains, the Probius system starts to analyze each service chain case (i.e., one of the generated service chain) by iterating the following procedures.

VNF manager: During the iteration for each service chain, the VNF manager generates all possible resource constraints of VNFs based on the provided global and VNF configurations. Then, it creates VNFs with each set of resource constraints for them. While most NFV platforms have their own VNF management functionalities, the underlying operational mechanisms are quite similar to each other. They commonly adopt libvirt APIs [25] to manage VNFs, and thus Probius also employs these APIs (of course, another VNF management functions can be added to Probius).

After all resources of VNFs are configured, it creates a service chain (i.e., selected service chain for this iteration). Each service chain will be created by enforcing flow rules to network devices

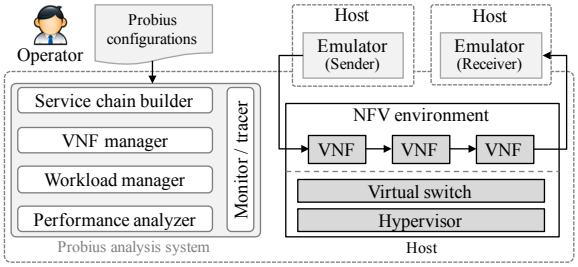

Figure 4: Probius architecture overview

(including virtual switches), and Probius employs a network device management protocol (i.e., OpenFlow [41]) to flexibly create service chains. In the case of inline VNFs, the inbound and outbound interfaces of consecutive VNFs are simply chained. Between them, passive VNFs are additionally attached to the last chaining point when the passive VNFs are chained. More specifically, it additionally put one or more output actions to duplicate incoming packets for passive VNFs in the flow rule for forwarding the packets to the next inline VNF.

Workload manager: The performance of VNFs will be highly related to network workloads. Thus, the workload manager emulates various kinds of workloads based on the given configurations. For this, it manages external workload emulators (i.e., a sender and a receiver). In terms of workloads, it generates various workload combinations with the different type of protocols (e.g., TCP, UDP) and the different amount of network volumes (e.g., 100Mbps, 1Gbps). In the case of TCP, it also configures the number of sessions to concurrently maintain. Under various network workload specifications, it controls the workload emulators to generate diverse network workloads flexibly.

Monitor, tracer, and performance analyzer: Using the above components, Probius builds up the basis for the performance analysis of a service chain. Then, the monitor and the tracer start to collect performance features (we describe the performance features in Section 5). With those collected features, the performance analyzer produces the explanations of the discovered performance issues during analysis. These modules will be explained more in the following sections.

3.2 Probius Configurations

For an automated analysis of NFV environments, Probius requires three kinds of configurations (i.e., global and VNF configurations and service chaining policies). Here, we describe those configurations.

Global configuration: Probius requires the range of resource constraints, and workload configurations. Resource constraints indicate the pairs of the number of vCPUs and memory sizes. In modern NFV platforms (e.g., OpenStack [10]), those pairs are called as flavors. In the case of workload configurations, there are three kinds of sub-configurations. First, it requires inbound and outbound network interfaces to monitor the end-to-end throughput variations of NFV environments. Then, it needs the IP addresses of workload generators for management and data planes to control workloads between them. Lastly, the characteristics of workloads (i.e., the types of network protocols, the number of sessions, and the range of network volumes) are necessary to emulate network workloads.

VNF configuration: Each VNF is defined with four parameters: type, inbound and outbound interfaces, CPU and memory constraints, management IP address and scripts. First, the type of a VNF is determined as either inline (e.g., IPS, NAT) or passive (e.g., IDS). Second, inbound and outbound interfaces are used to define service chaining points to receive and forward network traffic. Thus, inline services require both inbound and outbound interfaces while passive services only require an inbound interface. If network operators want to restrict the range of resource constraints against the global constraints, it is possible to refine them per VNF. Unless the applications of VNFs are automatically executed once their VMs are turned on, Probius manages VNFs through the given management IP address and scripts. In the case of passive VNFs, since they do not directly affect the performance of the following VNFs, it is hard to measure their actual performance. Thus, the stats script retrieves the statistics of passive VNFs from their logs instead of simply monitoring their network statistics.

Service chaining policy: While network operators can statically make a service chain with VNFs, it is possible that they only define the partial sequence of VNFs and the rest of VNFs could be freely chained. Thus, Probius provides three types of policy operators to let them define basic requirements for service chaining. Then Probius generates all possible service chains that satisfy the given service chaining policies. Here is the description of each operator.

- **AND (A & B):** This operator indicates that VNF A and B should be activated together. For example, if a network operator defines a policy {Firewall & IPS}, Probius generates possible service chains that both VNFs are always together.
- **OR (A | B):** This operator is used to activate either VNF A or VNF B. If a network operator has two kinds of IDSs and wants to deploy one of them since those IDSs have the same functionality, he can use this operator and let Probius analyze service chains with VNF A and VNF B respectively.
- **FOLLOW (A > B):** This operator defines a policy that VNF B should be chained after VNF A. If a network operator wants to deploy a firewall before a virtual router in the same service chain, he can define a policy {Firewall > vRouter}.

4 EXAMINATION OF NFV ENVIRONMENTS

The most important step of the performance analysis in NFV environments is understanding the working behavior of the NFV architecture. For such reason, here, we examine the behaviors of VNFs based on the correlations with a hypervisor and other VNFs in a service chain respectively.

4.1 VNF Workflow with a Hypervisor

In general, hypervisors (e.g., Xen, KVM) provide a number of trace points at key locations inside of them. With those trace points, we can get a picture of how VNFs work. While there are a bunch of trace points (e.g., 52 trace points for KVM [12]), with our careful analysis, we realize that we can determine the core workflow of a VNF with six trace points (i.e., scheduled in and out, load and store the virtual machine context structure (VMCS) of a VM, enter a VM,

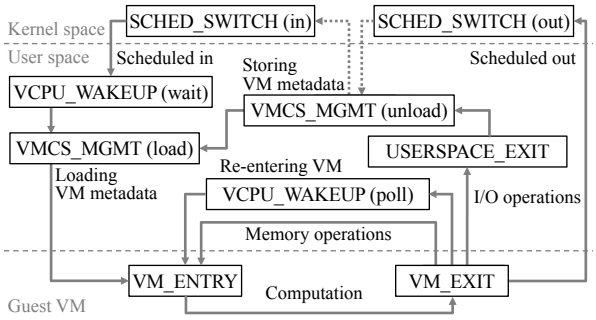

Figure 5: The overall workflow of a VM

exit a VM, move to the host kernel) since those trace points are necessarily passed when the state of a VM is changed.

Figure 5 illustrates the overall workflow of a VM. As CPU resources are allocated to a VM, the vCPUs of the VM start to run (VCPU_WAKEUP). Then, whenever a VM is activated or deactivated, its context structure (VMCS) is loaded or unloaded respectively (VMCS_MGMT). In terms of performance, the management of the VMCS causes noticeable overheads. Then, the VM executes the functionality of its VNF (VM_ENTRY). The time when a VM is in this state is solely considered as the given time to the VNF. If the VM requires some operations that need the access of memory or I/O devices, the control flow of the VM is first moved to the user-space of a hypervisor (VM_EXIT). Through the context of VM_EXIT trace points, we can figure out the reasons of the exits (e.g., EXTERNAL_INTERRUPT, IO_INSTRUCTION, and HLT). In the case of memory operations, the control flow of the VM is mostly returned back right after the operations are done. However, if the required operations need privileged permissions (e.g., the access of I/O devices), the control flow of the VM is further moved to the kernel-space of the hypervisor (VM_USERSPACE_EXIT). Then, the hypervisor handles the given operations for the VM. Likewise, whenever a VM requires some privileged operations, the operations should be passed through a long path (i.e., from a guest VM to the kernel-space of the hypervisor), causing significant overheads due to multiple state transitions only for a single operation. In turn, frequent privileged operations reduce the available processing time of the VM in the given quantum time. Once the given quantum time to the VM is consumed, the current VMCS of the VM is stored and the CPU resources are reallocated to another VM.

Through the workflow of a VM, we can extract four important features. First, we can see how much time a VNF consumes its CPU resources only for packet processing since there are many additional steps to handle privileged operations and to manage a VM itself. Second, we can know the reasons of VM state transitions. This allows to understand how frequently specific operations are triggered and how much time those operations consume. Third, the total time that a VM consumes indicates how much resources are allocated to a VM compared to those of other VMs. Besides them, we can infer the processing delay of a hypervisor for specific operations and the performance of the hardware in the host. In sum, those features allow to understand the behaviors of both VNFs and the underlying infrastructure in detail.

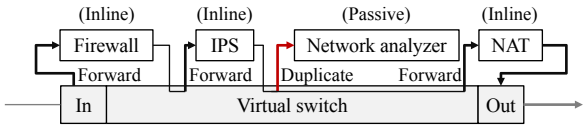

Figure 6: The example of a service chain

4.2 VNF Workflow in a Service Chain

In a service chain, the performance of a VNF is significantly important to the other VNFs since it can limit their maximum throughputs. For example, if it takes a long time to process packets in a VNF, the following VNFs could be idle during the packet processing time of the previous VNF. Likewise, VNFs are highly dependent on each other. However, VNFs are not the only ones that influence on their performance. Under those VNFs, a virtual switch is actually in charge of all packet delivery among VNFs. In turn, the performance of a virtual switch can also influence on the performance of VNFs.

Figure 6 shows the example of a service chain with four VNFs. From this example, we can see three possible performance problems due to the pipelined processing. First, as we discussed before, if the packet processing delay from the IPS is long, the following VNFs (the Network analyzer and NAT) can be frequently in an idle state. As a result, the overall performance of the service chain would be saturated under that of the IPS. Second, in the view of a virtual switch, due to the low performance of the IPS, the packets received from the Firewall should be kept in the queues inside of the virtual switch. In this case, it is possible that the switch consumes more CPU resources to manage its queues than to deliver packets to VNFs. In the worst case, if the queues in the virtual switch are overflowed due to the slow rate of packet consumptions by VNFs, new incoming packets are inevitably dropped, causing the incorrect operational behaviors of VNFs. In addition, the similar situation can happen in the VNF that shows the low performance. When the packet buffer of the VNF becomes full, the incoming packets have to be dropped, resulting in the same problem with the case of the virtual switch. Third, in the case of passive VNFs, a virtual switch duplicates incoming packets and delivers them to both passive VNFs and the next inline VNF. Since a virtual switch is a software implementation, the packet copy overheads (especially, for large packets) affect the performance of the switch [29]. In turn, the performance degradation of the switch can result in the entire performance degradation of VNFs.

Through those possible performance problems, we can get three meaningful features. First, the performance difference between adjacent VNFs can be useful to narrow down the possible points of performance issues. Second, the length of queues in a virtual switch can be used to infer how fast each VNF processes packets and which VNF increases the overall processing delay in a service chain. Lastly, we can infer the packet processing load of a virtual switch based on its CPU utilization.

5 PERFORMANCE FEATURE COLLECTION

On the basis of understanding NFV environments, we define a set of performance features that are useful for reasoning unexpected performance issues. Then, we describe the comprehensive monitoring and tracing mechanisms of the Probius system.

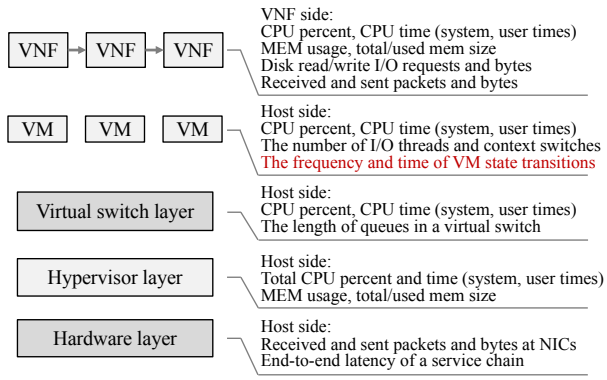

Figure 7: Performance features

5.1 Performance Features

From the analysis of NFV environments, we realize the importance of the state transitions of VNFs and the packet processing behavior of a virtual switch. Here, with the features discovered in Section 4, we define a set of performance features for the NFV architecture.

Figure 7 illustrates the feature set of each layer in NFV environments. Among those features, most layers interestingly have CPU-related metrics. However, the CPU usage of each layer has a different meaning. For example, at the VNF side, it indicates how much vCPU resources a VNF consumes. On the other hand, at the VM side, it indicates how much time a VM spends for state transitions (user time) and waits for privileged operations (system time). Theoretically, the CPU time of a VM is considered as the sum of the user time spent by the user space of a hypervisor (e.g., QEMU), the system time spent by the kernel space of a hypervisor (e.g., the KVM module), and the guest time spent by a VNF. By extracting CPU usages in various sources, we can thus narrow down the possible locations of performance problems. In terms of the visibility of performance issues, the network statistics of VNFs are the most distinguishable factors that we can easily get close to performance issues. For example, if the throughput variation of a hop between certain VNFs in a service chain is bigger than those of the other hops, we can regard that the service chain may have some performance issues and we can first look into those VNFs at the suspicious hop during performance analysis.

Besides them, we can use several features for specific purposes. In terms of disk performance problems, disk I/O usages at the VNF side and the number of I/O threads at the VM side can be used as the clues for reasoning the issues. The number of context switches at the VM side can be used to infer the frequency of the idleness in a VM. The total CPU usages at the host side can used to understand how much CPU resources are used for the entire VNFs. From these, we can make sure whether some performance degradations happen due to the lack of system resources or not. Likewise, the features for each layer are useful to address the possibility of performance issues against specific NFV entities. While most features can be used to narrow down the possible locations of performance issues, the frequency and time of VM state transitions can be used to closely point out the root causes of the issues by investigating how much time VNFs are in certain states and what cause state transitions.

Table 1: The form of raw data classifications

			Performance feature i	
Service chain	VNF	Proto/BW	Value	Stdev
VNF1(4), VNF2(2),VNF3(2)	VNF1	TCP / x	y	σ

5.2 Monitoring and tracing VNFs

To monitor NFV entities including VNFs, Probius has three types of monitors: an internal VNF monitor, an external VNF monitor, and a host monitor. First, the internal VNF monitor collects the features (e.g., vCPU, memory, disk, and network usages) at the VNF side. In general, if we can monitor those features inside of a VM, more accurate data can be retrieved. However, there are a bunch of VNF packages [53, 58] that have their own customized operating systems in real environments; thus, we cannot apply general analysis tools to monitor the features. For such reasons, Probius utilizes virtualization APIs [25] that allow to extract the features in the middle of VNFs and a hypervisor. Second, the external VNF monitor keeps track of the features of VNFs, but it monitors them at the host side. In the view of a host, each VNF is considered as a single process. With this fact, it retrieves the features on the VM processes corresponding to VNFs. Lastly, the host monitor covers the other features for the virtual switch, hypervisor, and hardware layers. Similar to the external VNF monitor, it also extracts the features for those layers from the system information of the host.

While multiple monitors give an insight into the global view of the NFV infrastructure, the VNF tracer reduces the semantic gap between the collected features and the actual workflows of VNFs. During the feature collection by the monitors, the VNF tracer keeps track of the trace points of a hypervisor for each VNF. Then, it maintains those trace points in the form of a graph. Each vertex presents a trace point and its data, and each edge presents the pair of {previous trace point, its data} and {current trace point, its data} with two properties: a counter and a consuming time. In the end, the generated graphs for VNFs are utilized in the performance analysis discussed in the next section.

6 ANALYSIS OF PERFORMANCE ISSUES

To extract the correlations among performance features and to provide the explanations of performance issues, the performance analyzer conducts three major steps: prerequisite, performance anomaly detection, and graph-based behavior analysis.

6.1 Prerequisite for Analysis

The prerequisite step is to convert raw data to the organized forms for the performance analysis. For this, the analyzer first removes measurement errors from the raw data and then classifies the refined data according to performance features.

In terms of measurement errors, it is possible that a measured value could be suddenly lower or higher than the other values in an instant when a monitor measures the value. Since that kind of measurement errors can mislead performance issues, the analyzer filters out them by using interquartile range (IQR), which is a simple method to detect outliers in a dataset. After the analyzer excludes outliers, it takes the average of the remaining values and their standard deviation as representative values for the corresponding

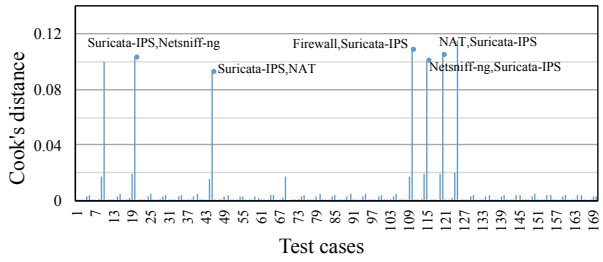

Figure 8: Cook's distances with 2-VNF service chains

features. As shown in Table 1, those values are classified according to performance features for each service chain.

6.2 Performance Anomaly Detection

In general, if there is no performance anomaly, the results of most service chains are similar to each other. With this in mind, we adopt a regression analysis technique (i.e., Cook's distance [27]) to find out service chains that connote performance problems. The basic idea is that if the collected data of a specific service chain (i.e., an abnormal case) are added into the dataset for all service chains, the change in slope between the original regression line and the new regression line noticeably occurs. Thus, we can figure out the most influential service chains (i.e., the suspects of performance issues) among the entire service chains.

To sort out suspicious service chains, Probius calculates Cook's distance D of performance features for each service chain as follows.

$$\hat{y}_i = \bar{y} + \left(\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)(x_i - \bar{x}) \quad (1)$$

where n is the number of the dataset, x_i is the ith observation point and y_i is the ith observed value. $\bar{x}$ and $\bar{y}$ are the means of the observation points and values respectively.

$$r_i = y_i - \hat{y}_i \quad (2)$$

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{n}(x_j - \bar{x})^2} \quad (3)$$

where r_i and h_i is the redidual and leverage of the ith observation point.

$$MSE = \frac{\sum_{i=1}^{n}(r_i - \bar{r})^2}{(n - k - 1)} \quad (4)$$

where MSE indicates the mean squared error for the given data. k is the number of independent variables in a model; thus, $k = 1$ in this model.

$$D_i = \frac{\sum_{j=1}^{n}(\hat{y}_j - \hat{y}_{j(i)})^2}{(k+1)MSE} = \frac{r_i^2}{(k+1)MSE}\left(\frac{h_i}{(1-h_i)^2}\right) \quad (5)$$

where $\hat{y}_{j(i)}$ is $\hat{y}_j$ that does not include the ith observations.

Figure 8 shows the distances (y) of the network volumes (x) according to the final throughputs of each service chain in the 2-VNF cases. From the distances, we can easily figure out which service chain shows different throughputs compared to those of the other service chains. In the same way, Probius also computes the distances for the other features of each service chain. In the end, Probius sorts all service chains according to the average of the computed distances and regards the service chains whose distances exceed a specific threshold as suspects. Here, we use $4/n$ as a threshold [24].

Table 2: The criteria for workflow classification

Category	State transitions / VM exit reasons
VM initialization	vCPU preemption - vCPU wakeup - VMCS load - VM entry
Hard switch (to the kernel)	vCPU preemption, but VMCS unload Userspace exit - VMCS unload
Soft switch (to the user-space)	vCPU preemption - halt poll, vCPU wakeup - VM entry
Computation	VM entry - VM exit
Memory access	VM exit(msr) - apic, msr - VM entry, emulate insn VM exit(ept misconfig) - emulate insn - MMIO - userspace exit, VMCS (un)load, VM entry VM exit(ept violation) - page fault - VM entry Exit reasons: MSR/APIC read, MSR/APIC write EPT violation, EPT misconfig
Interrupt	VM exit - VMCS unload, vCPU preemption Asynchronous trace points (e.g., IRQ) Exit reasons: Pause instruction, External interrupt, pending interrupt Exception NMI, EOI induced
I/O operation	VM exit - emulate insn - userspace exit, VM entry VM exit - PIO - userspace exit, VMCS (un)load, VM entry Exit reasons: IO instruction
Idleness	VM exit - VMCS unload, vCPU wakeup, VM entry Exit reasons: HLT
N/A (not assigned)	VMCS unload - vCPU preemption

6.3 Behavior Analysis of Suspicious VNFs

From the previous step, Probius statistically excludes most reasonable cases and leaves a small number of service chains that need to be deeply inspected. In this step, Probius focuses on the feature variations and working behaviors of VNFs in each service chain. To point out specific reasons among all possibilities, Probius takes the results of single VNF cases as ground truths.

In terms of feature variations, when a service chain is considered as a suspicious one, Probius compares and analyzes the data of each VNF in the service chain with those in the ground truths. For example, as shown in Figure 8, the service chain {Tcpdump, Suricata-IPS} is regarded as a suspicious one. Thus, Probius first compares the throughput of each VNF in the service chain with the throughput of the VNF in the ground truths. Then, it sequentially compares the values corresponding to performance features (e.g., CPU usage, CPU time, and disk usage) with those in the ground truths, and chooses suspicious features by checking if those features are within the error boundaries ($y \pm 3\sigma$ where y is the representative value and σ is its standard deviation in single VNF cases). Through the comparative analysis, Probius extracts which performance features are mostly changed compared to the ground truths.

As the last step, the goal of behavior analysis is to specifically point out the reasons of performance issues. With the state transition graphs for each VNF, Probius computes how frequently each state transition is triggered and analyzes why the state transition happens. To narrow down the range of possible performance issues, it categorizes the state transitions of VNFs according to the functionalities of a hypervisor. Table 2 presents the criteria for VNF workflow classification. In terms of VNF-related operations, Probius divides them into four categories: computation, memory access, interrupt, and I/O operation. Those categories determine the characteristics of VNFs. For example, if there are a bunch of VM

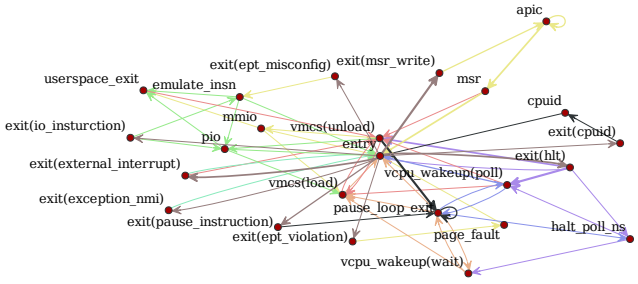

Figure 9: The behavior classification of a VNF

exits due to 'Pause instruction', it can infer that a VNF internally suffers severe lock contentions. If the computation time of a VNF is close to the given time to the VNF by a hypervisor, it can infer that the VNF is highly compute-intensive. In terms of hypervisor-related operations, unlike a general process running on a host, there are additionally required steps (e.g., loading the context of a VM, switching the execution mode of a VM) to execute the functionality of a VNF. Thus, the other categories are used to express the behaviors of a hypervisor. Since VNFs share all CPU times with each other, Probius separately maintains the time spent for other VMs by marking 'N/A' that means system resources are not assigned to a VNF. After categorizing the state transitions for each VNF as shown in Figure 9, Probius evaluates the portion variations for each VNF compared to those in the ground truths. In the long run, Probius summarizes the whole results (i.e., suspicious performance features and their variations, the portion variations in terms of the behaviors of VNFs) and lets network operators understand their performance issues.

7 IMPLEMENTATION

The Probius system is implemented with 4.2KLoC in Python. In terms of monitoring resource usages, we utilize libvirt [25] and psutil APIs [13] to monitor them at the VNF side and the host side respectively. In order to keep track of KVM events, we use Ftrace [2] that is a tracing framework for the Linux kernel. Specifically, we take the event traces (i.e., the workflows of VNFs) through Trace-cmd [20] which is a front-end tool for Ftrace. For the visibility of VNF workflow, the traced data are maintained as the forms of a vertex and an edge by using Graph-tool [28]. Then, the collected data from the monitor and tracer are stored in SQLite3 [16], which is a lightweight and self-contained database engine. While Probius uses iPerf3 [4] as the default workload generator, any traffic generators can be replaced. In order to analyze the data, we use Numpy [9], Pandas [11], and statsmodels [17] to calculate Cook's distances for performance features. The source code of the Probius project will be released at https://github.com/sdx4u/probius.

8 USE CASES

Now, we demonstrate how Probius reasons out performance issues with various service chains. Then, we discuss the reasons of the performance issues found from the Probius analysis.

Test environments: We use an experimental environment comprising of 3 machines to evaluate VNFs with the Probius prototype implementation. Two systems are used for workload generation. Each of these systems is equipped with an Intel Xeon E5-1650 CPU processor and 16GB of RAM. The last system is used for a NFV environment with the KVM hypervisor, running an Intel Xeon E5-2630 processor with 64GB of RAM.

VNF configurations: We deploy 7 open-source VNFs widely used for network and security services (i.e., Tcpdump [19] and Netsniff-ng [8] traffic analyzers, iptables-based firewall and NAT [7], a Snort IDS [14], Suricata IDS and IPS [18]). As a virtual switch managing the connectivity among VNFs, we use Open vSwitch v2.0.2 [47]. In terms of configurations, we basically use their default configurations with minor changes. For the firewall case, we insert 100 rules (randomly generated 99 unmatched rules and a matched rule for workloads sequentially). We set a NAT environment using the MASQUERADE function. In the case of the Snort IDS, we use the official rules for Snort 2.9 [15] (4K TCP rules). For the Suricata IDS and IPS, we use the Emerging Threats Open rulesets [1] (4K TCP rules) and execute them as AF_PACKET run-modes. Lastly, we use default configurations for Tcpdump and Netsniff-ng.

8.1 VNF analysis

To get the ground truths for further analysis, Probius first analyzes each VNF. Here, we describe the noticeable results in single VNFs.

The characteristics of VNFs: With Probius, we can easily draw the overall resource usages of VNFs, and determine their characteristics. As shown in Figure 10-12, most VNFs show their throughputs as high as the hardware limitation (i.e., 1Gbps) with 2 vCPUs while Snort-IDS and Suricata-IDS show their maximum throughputs (763.2, 753.4 Mbps respectively) with 4 vCPUs. Unlike those VNFs, the throughput of Suricata-IPS is saturated at 489.2 Mbps even though the Suricata-IDS and the Suricata-IPS have the almost same functionalities (we discuss this reason later). In terms of memory usages, NAT and Firewall consume 8.74%, and Snort-IDS and Suricata-IDS/IPS utilize 12.92% of the given memory (i.e., 4GB). On the other hand, the memory usages of Tcpdump and Netsniff-ng keep increasing (up to 72.2% for Netsniff-ng, 44.6% for Tcpdump) as incoming network volumes increase. Their disk I/O usages also have similar trends while the other VNFs barely trigger disk I/O.

From the results, we can conclude that Tcpdump and Netsniff-ng have disk I/O intensive tasks, and Snort and Suricata show CPU intensive characteristics. Firewall and NAT are network I/O intensive VNFs.

The hidden CPU consumptions of VNFs: In general, most monitoring systems focus on VNFs themselves. However, with our careful analysis, we realize that there are several processes (e.g., vhosts) that support VNFs outsides of them. For example, vhosts are created as many as the number of network interfaces in VNFs. However, none of current monitoring systems cumulate the CPU usages of those processes when they monitor VNFs. That is because they monitor the resource usages of VNFs by observing those of VM processes or collecting them through virtualization APIs. Figure 13 shows CPU usages and extra CPU usages for each VNF. In the case of inline VNFs, they have inbound and outbound interfaces; thus, they mostly have doubled extra CPU usages compared to passive VNFs. While the extra CPU usages of each VNF might be small compared to the original CPU usages, the cumulated extra

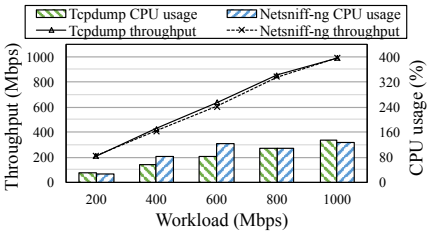

Figure 10: Tcpdump vs. Netsniff-ng

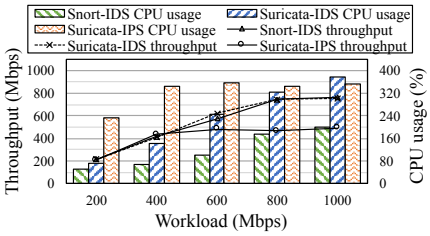

Figure 11: Snort-IDS vs. Suricata-IDS/IPS

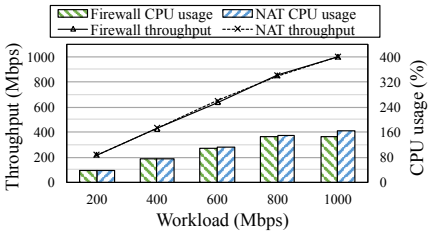

Figure 12: Firewall vs. NAT

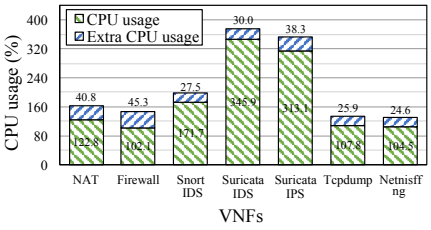

Figure 13: CPU and Extra CPU usages for each VNF

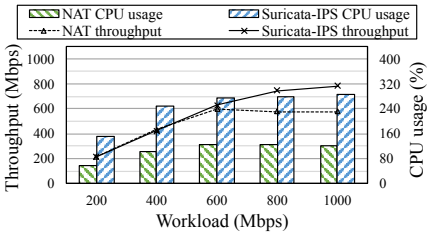

Figure 14: Throughputs of Suricata-IPS and NAT with Intel E1000

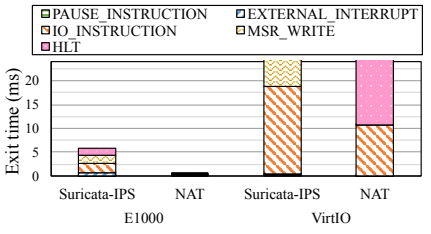

Figure 15: Exit reasons of Suricata-IPS and NAT with different NICs

CPU usages for all VNFs could be higher than that of a single VNF. Thus, those extra CPU usages could be a critical factor when the resource usages of VNFs are used as the guideline for VNF resource allocation or auto scaling.

The performance inversions with different NICs: In NFV environments, the VirtIO driver is used as the default network driver since it can directly access network packets in the host kernel. Thus, with VirtIO NICs, most VNFs show high throughputs. However, Suricata-IPS shows much lower performance than the others.

To see the reasons of the performance degradation, we look into the state transitions and VM exit reasons of Suricata-IPS and NAT for comparison. In terms of time consumption, Suricata-IPS (0.078/s for computation, 0.29/s in total) has slightly more time than NAT (0.032/s for computation, 0.22/s in total). They mostly consume their time for state transitions due to idleness (0.11/s in Suricata-IPS, 0.16/s in NAT). However, as shown in Figure 15, Suricata-IPS triggers doubled I/O instructions (0.019/s) compared to those (0.001/s) of NAT and those I/O instructions eventually cause the hard switches of the Suricata-IPS. On the other hand, when we change their drivers to Intel E1000, the throughput of Suricata-IPS highly increases up to 780.3 Mbps, but the throughput of NAT suddenly decreases under 575.4 Mbps. In terms of time consumption, Suricata-IPS (0.76/s) has much more time than the case with the VirtIO driver although NAT has a similar time (0.21/s) with the previous case. With the new driver, the time for hard switches in Suricata-IPS decreases (-65%), resulting in the throughput improvement, but the time for soft switches in NAT increases (+43%) while the time for hard switches is almost the same (+1.6%).

In sum, the reason of the performance degradation in Suricata-IPS is that Suricata-IPS consumes the large amount of CPU resources for I/O operations to send out packets rather than processing packets. In terms of performance inversions, the hard switch overheads of Suricata-IPS are significantly reduced by switching virtual NIC drivers; thus, Suricata-IPS can process more packets. In contrast, NAT with the E1000 driver uses more CPU resources than that with the VirtIO driver for state transitions. As a result, those state transitions unfortunately result in performance loss.

8.2 Service chain analysis

Even though single VNFs show high throughputs, it is possible that some of VNFs in the same service chain can cause resource contentions with each other. In addition, the operations of VNFs could influence on the throughputs of the subsequent VNFs. Here, we confirm those possibilities with actual cases.

The context switch overheads of VNFs: In NFV environments, context switches require a large amount of time due to loading and unloading the context structures of VMs. Thus, when the control flows of VNFs are moved to the host and moved back in VNFs, unnecessary CPU resources are consumed. Unfortunately, the idleness of a VNF triggers a bunch of context switches. Figure 16 shows the CPU usages of Tcpdump in two different service chains: {Tcpdump, NAT} and {Tcpdump, Suricata-IPS}. Unlike the former case, even though the input volumes (i.e., 200 and 400 Mbps) are low, the CPU usages of Tcpdump in the latter case are similar to those with the higher network volumes. By looking the analysis results (Figure 17), we realize that when Tcpdump runs with Suricata-IPS, it frequently becomes in an idle state; thus, unnecessary CPU consumption significantly increases its CPU usages.

The adverse effect between VNFs: From Figure 2 in Section 2.3, we find that the throughput of the service chain {Tcpdump, Suricata-IPS, NAT} suddenly decreases compared to that of the service chain {Suricata-IPS, NAT}. To figure out the reasons of this performance degradation, we see the state transitions and VM exit reasons of both service chains. In terms of time consumption, a hypervisor fairly distributes time to all VNFs. However, in terms of their computation times, the times for Suricata-IPS and NAT

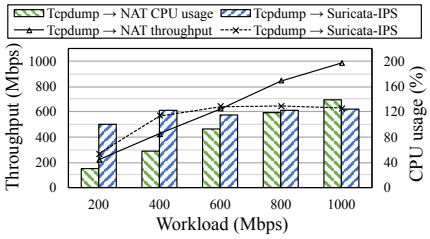

Figure 16: High CPU usages of Tcpdump due to context switches

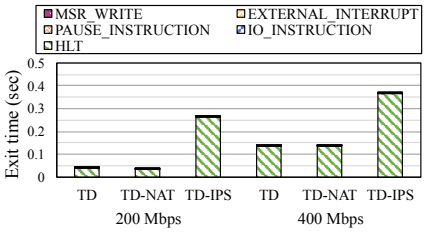

Figure 17: Exit reasons of Tcpdump in different service chains

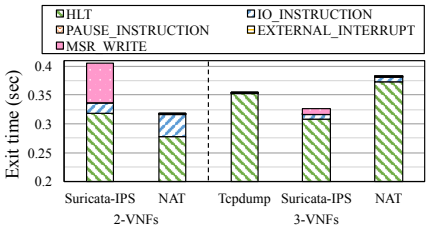

Figure 18: Exit reasons of VNFs in different service chains

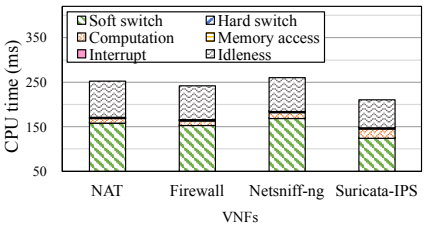

Figure 19: Behavior classification of VNFs in a normal service chain

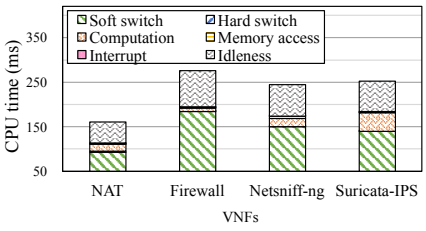

Figure 20: Behavior classification of VNFs in a problematic service chain

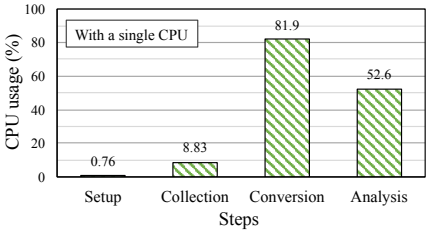

Figure 21: CPU usages for each processing step in Probius

decrease by 61% and 28.1% respectively when the Tcpdump is added into the service chain. In addition, when we see the VM exit reasons as shown in Figure 18, the portion of I/O instructions and MSR operations in Suricata-IPS (3-VNFs) are reduced by 55.5% and 85.6% respectively compared to those in the Suricata-IPS (2-VNFs). The portion of I/O instructions in NAT (3-VNFs) is also reduced by 82.2%, but the idleness of the NAT (3-VNFs) increases by 134%.

Based on those facts, as soon as the Tcpdump is added to the service chain, the other VNFs face I/O contentions and those contentions significantly drops the overall throughputs of the new service chain. However, it does not mean that one of the VNFs is a bottleneck since the Tcpdump triggers some overheads to the other VNFs. As a result, the performance degradation comes from the I/O bottleneck in the underlying infrastructure.

The importance of chaining orders: As shown in Figure 3, we find that the throughputs of service chains can be changed according to the sequence of VNFs. Here, we identify the reasons of the performance variations with two service chains: {Suricata-IPS,Firewall,Netsniff-ng,NAT} as a normal case and {Netsniff-ng,NAT, Suricata-IPS,Firewall} as a problematic case.

Figure 19 and 20 present the behavior classification results of the normal and problematic service chains respectively. In the normal case, all VNFs have similar time (0.24/s on average) to execute their functionalities. In contrast, the NAT in the problematic service chain only has 0.16/s and the other VNFs have a little bit more time (0.26/s on average) than the NAT. The reason that the NAT has less time than the others is due to frequent hard switches for I/O operations. The NAT triggers hard switches for 18.6 times in the normal case but 785.9 times in the problematic case. While the time to handle hard switches is relatively smaller than the times for the other categories, they significantly reduce the available time for the NAT. Similarly, the hard switches of the Netsniff-ng increase from 19.4 to 900.7 times and this situation could be found in its system time (0.77 in the normal case, but 1.45 in the problematic case).

In turn, the resource contentions between the Netsniff-ng and the NAT trigger the overall performance degradation. Here, the reason that the normal service chain shows higher throughputs is due to the low throughput of the Suricata-IPS. Since the Suricata-IPS sends out incoming packets at the beginning with a low speed, it could naturally control the I/O contentions of the following VNFs. This trend could be found in the other service chains as well.

9 SYSTEM EVALUATION

In this section, we evaluate the overheads caused by the Probius system. For this, we divide the whole analysis processes into four steps: VNF setup, feature collection, raw data conversion, and analysis. The VNF setup includes VNF resource allocations, service chain creations, and workload generations. The feature collection indicates the step to monitor and trace VNFs. In terms of performance analysis, we measure the CPU usages for the prerequisite step (i.e., raw data conversion) and the actual analysis step separately.

Figure 21 shows how much CPU resources each step consumes. Among the major steps, the CPU usage for the VNF setup is negligible (0.76% on average). Similar to the VNF setup, the collection step consumes 8.83% of CPU resources on average that is only a small portion against the entire CPU resources (2,000% in our test environment with 20 cores). While Probius monitors the resource usages of VNFs per second, the measured CPU consumptions are higher than what we expected. With our careful analysis, we realize that Probius monitors the resource usages of VNFs by creating two monitoring threads (one for the VNF-side and the other for the host-side) of each VNF for every second. Thus, Probius consumes CPU resources due to managing the monitoring threads rather than monitoring VNFs. The raw data conversion shows the highest CPU

consumption (81.9% on average). This is because the amount of the raw data is huge and the jobs to convert the raw data to our classified forms for analysis are highly CPU-intensive. The last step (i.e., the VNF analysis step) shows 52.6% CPU consumption on average. During this step, most CPU consumptions come from the query overheads to access the Probius database. In turn, the raw data conversion and analysis steps show relatively higher CPU usages than the others. However, those steps could be done outside of the host. Therefore, each step requires less than 100% CPU resources, which means a single CPU is enough to handle the Probius system.

10 RELATED WORK

NFV Performance Optimization: There are a number of studies tackling the performance problems in NFV environments. In terms of inter-VM communications, NetVM [32, 61] provides zero-copy packet delivery across a chain of VNFs through its shared memory architecture. Similarly, xOMB [21], CoMb [56] focus on flexible management of networks and high-speed packet forwarding. In the case of ClickOS [40], it is designed for a high-performance virtualized middlebox platform while reducing VM instantiation times on Click modular router [36]. FreeFlow [60] improves the network performance of containers by making a hole (i.e., shared memory, RDMA) between trusted containers. Besides the optimization of NFV platforms, high-performance network I/O engines such as Netmap [50], DPDK [33], and PF_RING [45] are widely used to boost up NFV performance by providing direct access to the ring buffers in a NIC, using custom network drivers. While those studies highly improve the NFV performance, traditional hypervisors (e.g., KVM, Xen) and software switches (e.g., Open vSwitch) are still widely used in real NFV environments; thus, network operators unfortunately suffer various performance issues.

System-wide VNF analysis: Xenoprof [31] is a system-wide profiler targeting virtual machines for the Xen hypervisor [23]. To extract the profiling events in guest VMs, it deploys the modified Oprofile [38] into the guest VMs and takes the event samples through hypercalls. Perfctr-Xen [44] is another profiler for the Xen hypervisor. The key idea of Perfctr-Xen is to directly access hardware performance counters in guest VMs by installing a hypervisor driver and a guest kernel driver. Linux perf [6] is recently extended to profile VMs running in the KVM hypervisor by extracting event samples from guest VMs through their virtual CPU states. Although those tools allow network operators to investigate their VNFs in depth, they severely cause the high CPU overheads to extract hardware events. For such reasons, Probius utilizes kernel-level trace points, which is much lighter than hardware-based performance counters. As a result, Probius can analyze VNFs without additional performance degradations of VNFs.

Inter-VM Performance Interference: In terms of the performance interference analysis between VMs, Koh et al. [35] study the effects of inter-VM interference by collecting the runtime performance characteristics of the different types of Linux applications. In contrast, Pu et al. [48] focus on a dedicated performance interference on network I/O running in separate VMs. Similarly, Mei et al. [42] show the performance study of network I/O workloads in a virtualized cloud environment. While previous studies mostly concentrate on the phenomenon of VMs in terms of performance, they could not look into the reasons of the phenomenon. In fact, our work covers the previous studies and even discovers the reasons of performance issues in depth.

Network-wide VNF analysis: So far, there are a few studies [39, 43, 52] aiming at analyzing VNFs to figure out performance issues in NFV environments. In the case of NFVPerf [43], it monitors the network statistics among VNFs and finds a bottleneck point in a service chain. For this, it sniffs packets on all inter-VM communication paths and computes per-hop throughputs and delays. M. Peuster et al. [39] analyze the performance behavior of VNFs by monitoring the throughput changes according to different resource restrictions. Unfortunately, those studies only detect some abnormal behaviors, they cannot explain the reasons of the behaviors. In terms of scaling and resource allocations, several studies (e.g., NFV-Vital [26], Sandpiper [59], CloudScale [57]) specifically monitor the utilization of multiple hardware resources. However, their views are highly limited to application-related metrics. On the other hand, the goal of the Probius system is to provide the comprehensive view of VNFs and service chains with the NFV architectural characteristics. Thus, not only can Probius point out the suspects of performance issues, but also it can explain the reasons of the performance issues in the comprehensive view of VNFs and their service chains on the basis of NFV architectural characteristics.

11 CONCLUSION

While previous studies have significantly improved the performance of NFV environments, network operators still face unexpected performance issues - *performance uncertainty*. In this work, we introduce Probius, a performance analysis system that automatically extracts all meaningful performance features from the NFV architecture, and discovers the reasons of performance problems. For this, Probius collects specific features from all NFV entities and sorts out performance anomaly through statistical regression approaches. To point out the root causes of the problems, it newly adopts the behavior analysis of VNFs on the basis of the NFV architectural characteristics. To show its practicality, we utilize Probius to analyze the various kinds of service chains with 7 open-source VNFs. From the analysis results, we could find several interesting performance issues caused by the NFV environmental factors according to what kinds of VNFs are deployed and how VNFs are chained. We believe that the Probius system would be greatly helpful for network operators to understand performance issues in real NFV environments.

ACKNOWLEDGEMENT

This work has been supported by the Future Combat System Network Technology Research Center program of Defense Acquisition Program Administration and Agency for Defense Development.(UD160070BD)

REFERENCES

[1] Emerging Threats Open Rulesets. https://rules.emergingthreats.net. (online, 2018. 02.).
[2] Ftrace: Linux Kernel Internal Tracer. https://www.kernel.org/doc/Documentation/trace/ftrace.txt. (online, 2018. 02.).
[3] GNU Profiler. https://ftp.gnu.org/old-gnu/Manuals/gprof-2.9.1/html_mono/gprof.html. (online, 2018. 02.).

[4] iPerf: The TCP, UDP and SCTP Network Bandwidth Measurement Tool. https://iperf.fr. (online, 2018. 02.).

[5] Java Profiler. https://www.ej-technologies.com/products/jprofiler/overview.html. (online, 2018. 02.).

[6] Linux Perf. https://perf.wiki.kernel.org/index.php/Main_Page. (online, 2018. 02.).

[7] Netfilter/iptables Project. https://www.netfilter.org. (online, 2018. 02.).

[8] Netsniff-ng: A Free Linux Networking Toolkit. http://netsniff-ng.org. (online, 2018. 02.).

[9] NumPy. http://www.numpy.org. (online, 2018. 02.).

[10] OpenStack. https://www.openstack.org. (online, 2018. 02.).

[11] pandas: Python Data Analysis Library. http://pandas.pydata.org. (online, 2018. 02.).

[12] Performance Events in the KVM Kernel Module. http://www.linux-kvm.org/page/Perf_events. (online, 2018. 02.).

[13] PSUtil: A Cross-platform Library for Process and System Utilities. https://github.com/giampaolo/psutil. (online, 2018. 02.).

[14] Snort: Network Intrusion Detection & Prevention System. https://www.snort.org. (online, 2018. 02.).

[15] Snort v2.9 Rules for Registered Users. https://www.snort.org/downloads/#rule-downloads. (online, 2018. 02.).

[16] SQLite. https://www.sqlite.org. (online, 2018. 02.).

[17] StatsModels: Statistics in Python. http://www.statsmodels.org/stable/index.html. (online, 2018. 02.).

[18] Suricata: Open Source IDS / IPS / NSM Engine. https://suricata-ids.org. (online, 2018. 02.).

[19] TCPDUMP: A Packet Analyzer. http://www.tcpdump.org. (online, 2018. 02.).

[20] Trace-cmd: A User-space Front-end Command-line Tool for Ftrace. https://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git. (online, 2018. 02.).

[21] James W Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. 2012. xOMB: Extensible Open Middleboxes with Commodity Servers. In *Proceedings of the ACM/IEEE symposium on Architectures for networking and communications systems*. ACM.

[22] ATTO Research. Athene: Software-defined Elastic NFV Platform. http://www.atto-research.com/en/solutions/athene. (online, 2018. 02.).

[23] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the Art of Virtualization. In *ACM SIGOPS Operating Systems Review*. ACM.

[24] Kenneth A Bollen and Robert W Jackman. 1985. Regression Diagnostics: An Expository Treatment of Outliers and Influential Cases. *Sociological Methods & Research* (1985).

[25] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster, and André Brinkmann. 2010. Non-intrusive Virtualization Management using libvirt. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association.

[26] Lianjie Cao, Puneet Sharma, Sonia Fahmy, and Vinay Saxena. 2015. NFV-Vital: A Framework for Characterizing the Performance of Virtual Network Functions. In *IEEE Conference on Network Function Virtualization and Software Defined Network*. IEEE.

[27] R Dennis Cook. 1979. Influential Observations in Linear Regression. *J. Amer. Statist. Assoc.* (1979).

[28] Tiago de Paula Peixoto. graph-tool: Efficient Network Analysis. https://graph-tool.skewed.de. (online, 2018. 02.).

[29] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2014. Performance Characteristics of Virtual Switching. In *IEEE International Conference on Cloud Networking*. IEEE.

[30] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. 2015. mSwitch: A Highly-Scalable, Modular Software Switch. In *Proceedings of the ACM Symposium on Software Defined Networking Research*. ACM.

[31] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K Panda. 2006. A Case for High Performance Computing with Virtual Machines. In *Proceedings of the Annual International Conference on Supercomputing*. ACM.

[32] Jinho Hwang, KK Ramakrishnan, and Timothy Wood. 2014. NetVM: High Performance and Flexible Networking using Virtualization on Commodity Platforms. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. USENIX Association.

[33] Intel. DPDK. http://dpdk.org. (online, 2018. 02.).

[34] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. kvm: the Linux Virtual Machine Monitor. In *Linux Symposium*.

[35] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An Analysis of Performance Interference Effects in Virtual Environments. In *IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE.

[36] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. 2000. The Click Modular Router. *ACM Transactions on Computer Systems* (2000).

[37] KVM. VhostNet. https://www.linux-kvm.org/page/UsingVhost. (online, 2018. 02.).

[38] John Levon and Philippe Elie. Oprofile: A System Profiler for Linux. (2004).

[39] Holger Karl Manuel Peuster. 2016. Understand Your Chains: Towards Performance Profile-based Network Service Management. In *European Workshop on Software Defined Networks*. IEEE.

[40] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. USENIX Association.

[41] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review* (2008).

[42] Yiduo Mei, Ling Liu, Xing Pu, Sankaran Sivathanu, and Xiaoshe Dong. 2013. Performance Analysis of Network I/O Workloads in Virtualized Data Centers. *IEEE Transactions on Services Computing* (2013).

[43] Priyanka Naik, Dilip Kumar Shaw, and Mythili Vutukuru. [n. d.]. NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV. ([n. d.]).

[44] Ruslan Nikolaev and Godmar Back. 2011. Perfctr-xen: A Framework for Performance Counter Virtualization. In *ACM SIGPLAN Notices*. ACM.

[45] NTOP. PF_RING. http://www.ntop.org/products/packet-capture/pf_ring. (online, 2017. 2018. 02.).

[46] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: A Framework for NFV Applications. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM.

[47] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*.

[48] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao. 2013. Who is your neighbor: Net I/O Performance Interference in Virtualized Clouds. *IEEE Transactions on Services Computing* (2013).

[49] Kaushik Kumar Ram, Alan L Cox, Mehul Chadha, Scott Rixner, and TW Barr. 2013. Hyper-Switch: A Scalable Software Virtual Switching Architecture. In *USENIX Annual Technical Conference*. USENIX Association.

[50] Luigi Rizzo. 2012. Netmap: A Novel Framework for Fast Packet I/O. In *USENIX Security Symposium*. USENIX Association.

[51] Luigi Rizzo and Giuseppe Lettieri. 2012. Vale: A Switched Ethernet for Virtual Machines. In *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*. ACM.

[52] Raphael Vicente Rosa, Christian Esteve Rothenberg, and Robert Szabo. 2015. VBaaS: VNF Benchmark-as-a-service. In *European Workshop on Software Defined Networks*. IEEE.

[53] Rubicon Communications, LLC. pfSense: Open Source Firewall. https://www.pfsense.org. (online, 2018. 02.).

[54] Rusty Russell. 2008. virtio: Towards A De-facto Standard for Virtual I/O Devices. *ACM SIGOPS Operating Systems Review* (2008).

[55] SDxCentral. SDN and NFV Market Size and Forecast Report 2015. https://www.sdxcentral.com/reports/sdn-nfv-market-size-forecast-report-2015. (online, 2018. 02.).

[56] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. 2012. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the USENIX conference on Networked Systems Design and Implementation*. USENIX Association.

[57] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. Cloudscale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM.

[58] VyOS Project. VyOS: An Open Source Router Operating System. https://vyos.io. (online, 2018. 02.).

[59] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2009. Sandpiper: Black-box and Gray-box Resource Management for Virtual Machines. *Computer Networks* (2009).

[60] Tianlong Yu, Shadi Abdollahian Noghabi, Shachar Raindel, Hongqiang Harry Liu, Jitu Padhye, and Vyas Sekar. 2016. FreeFlow: High Performance Container Networking. In *Proceedings of the workshop on Hot topics in Networks*.

[61] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, KK Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A Platform for High Performance Network Service Chains. In *Proceedings of the workshop on Hot topics in Middleboxes and Network Function Virtualization*. ACM.