NfvInsight: A Benchmark Suite for Network Function Virtualization Research

ABSTRACT

NFV is attracting more and more interest in both industry and academia. It is promising to reduce cost for network operators and accelerate innovation by replacing traditional hardware-based appliance with virtualized functions deployed in cloud computing environment. NFV testing environment deploying involves multiple layers in the software stack including VNF image packaging, virtualization management, network configuration, forwarding rule installation, and performance metric measurement. According to our survey, it takes averagely 1-2 human-month to set up a testing environment. Motivated by the time taking process of deployment, we decided to develop an easy-to-use NFV benchmark named NI. The benchmark has three key features: 1) Representative NFs 2) One-click running 3) Plenty metrics for measurement. NI includes 5 VNF and provides typical NF chaining policies. NI leverages Kubernetes and Docker to do infrastructure management, and OVS to control data flow. Our evaluation shows measurement results NI can provide.

Keywords

Network Function Virtualization;

1. INTRODUCTION

Network function virtualization (NFV) has become a hot topic, both in industry and academia. Since the publication of NFV Introductory White Paper [] of ETSI in 2012, a lot of works have been emerged in this field. Modification works of NFV were done on the whole software stack (or maybe both SW and HW stack?). There are de facto industrial NFV platform OPNFV [1] as well as advanced NF allocation frameworks like OpenNF [5], CoMb [8] and E2 [6]. Also, there are works like OpenBox [4] and Netbricks [7] to rewrite or modify the NFs.

However, a suite of easy-to-use NFV benchmark is not yet existed. It is unavoidable to experience a time consuming process finding both open source software and proper chaining policies. According to our observation, most NFs used in papers are different open source implementations linked in different kinds of NF chain policies. So that, there is not yet a general baseline for measurement and comparison. Furthermore, the workload generator and network traffic trace used are also different, traces need to be provided to test different NF chains.

We also did a survey among top conference researchers who have experiences setting up NFV environment. In our survey, we found that the deploying time varies much due to the scale. In average, it takes 1-2 months to build up a NF cluster having less than 10 VM instances. But when the scale of instances increase to over 50, the build up process can take 3-4 months or more. One of our respondent said that they were still keeping on iterating and improving their testbed constantly.

In this paper, we develop a suite of NF benchmarks, which is supposed to have the following characteristics: 1) Representing typical NFs 2) Easy to use 3) Plenty metrics for measurement.

According to our research and observation, the representative of an NFV benchmark should be satisfied in three aspects: **1) Representative NFs.** For the demand of representative NFs, we referred to the NFV Introductory White Paper [], which defined ten scenario of NFV use cases. In the first version of our benchmark, the open source implementation of NFs we collected covers half of the ten use cases. Table 2 lists the basic information of NFs used in our benchmark. **2) Representative NF chains.** However, not only single NFs should be typical, but also the NF chains. We referred to ETSI standard documents of SFC (Service Function Chaining) [3] for the typical use case of service chains in the scenarios of both enterprise user and datacenter. We also consulted our industrial partners for real world chaining policies. The typical NF chains our benchmark provides are listed in Table 3. **3) Proper workload generator.** Since each NF chain serves at different network level, only one workload generator is not enough to test all the scenarios. So we select different clients for each chain.

The goal of the 'easy to use' design is to achieve one-week setting up as well as one-click test running, no matter the scale of the testing environment. To finish a

test, users only need to touch one configuration file and execute one single command. In the end, the measurement report will be output to a file. To implement our design, we leverage Docker and Kubernets to pack NFs in docker, manage the images, and do allocation automatically. We use OVS to do switching and packet force forwarding. Pre-written scripts and Openflow rules are written to implement chaining and flow control.

In our benchmark, we provide the most concerned metrics for measurement, that is latency and throughput. We measure latency in the granularity of per-packet and per-NF, and output cumulative distribution of latency in the measurement report.

2. A SURVEY OF NFV TESTING ENVIRONMENT DEPLOYING

To get first-hand information about researchers' experience on NFV testing environment deploying, we did a survey among top conference authors dedicated in the field of NFV. We delivered the survey to sixteen people who are the first authors of their papers and we finally collected eight responses.

In the first place, we asked for the number of types of VNF they used, and the source of VNF, chaining policy and workload generator. Averagely, six types of VNF are used in one paper. Most of the VNF are open source implementations, though three of our respondents have written their own VNF. The chaining policies are all find in papers or public policies. And for the workload generator, half of them directly used open source software, and half of them wrote by themselves.

Another question we concerned most is the time they spent on deploying a NFV testing environment. To more precisely measure the labor time, we use the metric of man-month which indicates the number of months used if the work is done by one person. We also asked for the scale of physical servers and VM instances, as well as the virtualization technology they used. The result is shown in Table 1.

From the responses, we can see the average time to deploy a testing environment for NFV is around 1-2 months for a cluster not in very large scale. It can be quite fast for a experienced person, as the responser No.1, who used less than half a month setting up a cluster including 4 physical servers and 11-20 VMs. However, for the others, the deploying process can be time taking and painful. Take respondent No.6 and No.7 as examples, they built their test environment in quite large scale, and it results in much longer setting up time.

NFV setting up involves multi-layers in the whole software stack, which includes virtualization, cluster management, network virtualization, and NF configuration, so that any point can become a bottleneck, especially for a green hand. Take respondent No.8 as an example, Most problems they encountered are related to the hypervisor they used. And respondent No.7 said they changed from Xen to KVM with Openstack and things got much better. We asked the authors for their most painful experiences, and we summarize the reasons making the setting up time so long as the following:

	Man-Month Used	# of Servers	# of VMs	Virtualization Technology
1	<0.5	4	11-20	KVM
2	0.5-1	2	1-5	Hyper-V
3	0.5-1	4	1-7	Container
4	0.5-1	4	6-10	KVM
5	1-2	4	6-10	Container, KVM and other
6	4	10	100	KVM
7	3	24	72	KVM
8	Constantly Iterating	4	1-5	Xen

Table 1: The result of our survey reflects the relationship between labor consuming and the scale of the testing environment.

- Automating the process of setting up the testbed.
- Setting up the datapath (interfaces, DPDK, routing tables, etc).
- Installing proper rules in OpenFlow-enabled switches to enforce chaining.
- Figure out appropriate workload to test different types of VNFs.
- Figuring out how to determine that NFs were done starting up and ready to forward packets.

One of our responser pointed out that:

> If there is a system that will take as input key parameters (e.g., number of nodes, topology, VM images, choice of hypervisor) and then automatically generates a ready to go set up, that will be of a huge value!

This suggestion firms our intention to publish an NFV benchmark which provides typical NF services and workload generator, automatically run the test according to user configurations, and reports the measurement results in the end.

3. BENCHMARK DESCRIPTION

3.1 Overview

According to our research and survey, we found that a suite of easy-to-use NFV benchmark is indeed. First, for all these researches, a universally used baseline is

lacked for comparison experiments. So, a selected suite of representative VNF with widely accepted chaining policy and proper workload generator is needed. Second, a well-designed benchmark with a bunch of pre-written scripts to automate the deployment process can effectively save time. Since NFV involves multi-layers in the whole software stack, the automation process includes VNF image packaging, NF allocation, networking config, forwarding rules installation, connection testing, and metrics measurement.

The goal of our work is to provide a benchmark suite with the following three characteristics: 1) Representing typical NFs 2) Easy to use 3) Plenty metrics for measurement.

3.2 VNF

To select representative VNF, We reviewed papers on NFV and find out all the open source implementations has been used. We categorized them into several kinds according to their function, and selected the VNF which needs no modification of the kernel. NI provides DOCKERFILE to packet VNF into docker images, and provides pre-wirtten config files to config each VNF working in the way as demanded. Table 2 lists information of the VNF and the workload generator used. Because iptables, Haproxy and Squid need real tcp requests, packet trace replay cannot make them work properly, we use Httperf to generate requests and Apache Server to respond the requests. The following are the brief introduction of the VNF:

- Clearwater [2] is an open source implementation of IMS (the IP Multimedia Subsystem) designed from the ground up for massively scalable deployment in the Cloud to provide voice, video and messaging services to millions of users.
- Snort [] is an open source intrusion prevention system capable of real-time traffic analysis and packet logging. We config it working under intrusion detection model.
- Iptables [] We set particular rule to use iptables as DNAT and SNAT.
- Haproxy [] is a reliable, high performance TCP or HTTP load balancer. We config it working under TCP mode to do L4 load balancing.
- Squid [] is a caching proxy for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages.

Category	NF	Workload Generator
IMS	Clearwater	SipP
IDS	Snort	Httperf
NAT	iptables	
L4 Load Balancer	Haproxy	
L7 Cache	Squid	

Table 2: NFs included in our benchmark.

3.3 VNF Chain

To find representative NF chaining policies, we referred to IETF Service Function Chaining Use Cases In Data Centers [3]. The standard provides several sample service chains, as listed in Table 3, the second column. The sample chains are connected by four network function concepts. Definitions of each function concepts are given in the standard file, and are listed in the following part of the paper. The definitions are generalized, actual implementations may vary and have functions overlapped. We form the chains in our benchmark using the open sources implementations in Table 2 according to the sample chains. The chains in NI are listed in Table 3, the right column.

- Edge Firewall (EdgeFW): Service functions such as VPN, DHCP, NAT, IP-Audit, Protocol Inspection, DPI etc., with policies primarily focusing on threats external to the data center.
- Application Delivery Controller (ADC): Service Function that distributes traffic across a pool of servers (applications) for efficient resource utilization, application scaling as well as to provide high availability among others.
- Application Firewall (AppFW): Service Function that isolates traffic within a segment or protects from application specific threats. This falls into the same class as DPI but deployed much closer to the applications. It is an intra-segment firewall.
- Web Optimization Control (WOC): Service functions to optimize the use of WAN link bandwidth, improve effective user throughput and latencies leading to overall improved user experience. WOC includes various optimizers such as compression, de-duplication, congestion control, application specific optimizers, etc.

3.4 Benchmark

To achieve the demand of easy to use, users are supposed to run only one command to start the test, and touch only two files, one configuration file and one measurement report.

The architecture of our benchmark is shown in Figure 1. It has three main parts, NFVI, NF service and NI. NFVI (NFV Infrastructure) is consisted of hardware resources including compute, storage and network. NF services are deployed upon NFVI, forming a service

	Sample Service Function Chains	Chains in NI
SFC1	EdgeFW	Httperf → iptables → Apache
SFC2	EdgeFW → ADC	Httperf → iptables → Haproxy → Apache
SFC3	EdgeFW → ADC → AppFW	Httperf → iptables → Haproxy → Snort → Apache
SFC4	WOC → EdgeFW → ADC → AppFW	Httperf → Squid → iptables → Haproxy → Snort → Apache

Table 3: Sample VNF service function chains and chains provided by our benchmark.

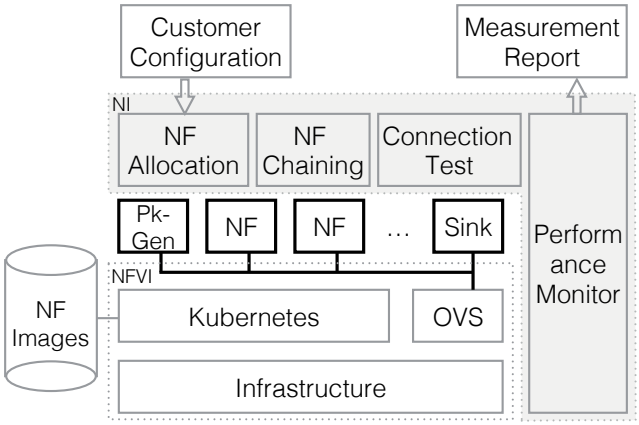

Figure 1: An overview of our benchmark design.

chain. NI is the key component of our benchmark. It does NF allocation and chaining automatically leveraging NFVI management framework. Furthermore, it does connection test and performance monitoring during test running. The process to use the benchmark includes five steps:

1. **Pre-install** In the first step, users need to install Kubernetes, OVS and Openflow controller for their physical server cluster. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. To force forwarding packets cross nodes, OVS bridge and Openflow global control is needed.

2. **Build images** We provide DOCKERFILE to build images for each VNF listed in Table 2, A private registry is needed to manage the images, so that Kubernetes can create containers distributely using local images.

3. **Write config file** This config file is the only file needs user edit. It includes choice of testing chain, mapping between physical server and docker, resource configuration of each docker, host network and other NF specific configurations. To config each NF working in the demanded mode, we provide pre-written config files for each NF.

4. **One click running** Users can run the test by executing only one command. NI does a series automate processes. First it reads the configuration file edited by users and rewrites the related scripts, which including container deployment scripts and NF config files. Then the NF Allocation module deploys containers according to user configuration. When containers are started and their status turn to 'ready', NF Chaining module adds virtual NIC for each container, connects them to OVS bridge, and installs Openflow forwarding rules according to the chaining policy. To check whether the chain works, the Connection Test module send a TCP request from the packet generator and check if it gets response from Apache server. If the connection test is passed, the test is run and metric measurement begins.

5. **Get the report** The Performance Monitor module does metric measurement and output a report.

4. EVALUATION

We have implemented a demo version of our benchmark. It now can work on a single node, and measure classic network metrics such as throughput and latency. In this section, we evaluate two chains in NFVInsight. The testbed we used for our evaluation builds on an edge server (Intel Xeon E5V3-2658 2.2GHz, 12 cores, 32GB, 2×1Gbps NICs) that runs CentOS 7 system, with Kubernetes 1.4 and Open vSwitch 2.5.1. We encapsulates NFs with Docker 1.12.5 running Ubuntu 14.14.

4.1 Throughput

We measure throughput of each container using Docker API. The transmit bytes and receive bytes are recorded two times at the begin and end of test running respectively. Throughput of each NF is calculated by doing subtraction of transmit bytes and divid it by the testing time.

Figure 2 shows the topology of two chains and the throughput labeled on the arrows. The throughput is divided by request throughput and respond throughput.

4.2 Latency

We modified the workload generator Httperf to output the latency of each request. Figure 3 shows the end to end latency of SFC1 and SFC2. Figure 4 shows the per-packet latency of NAT in SFC1.

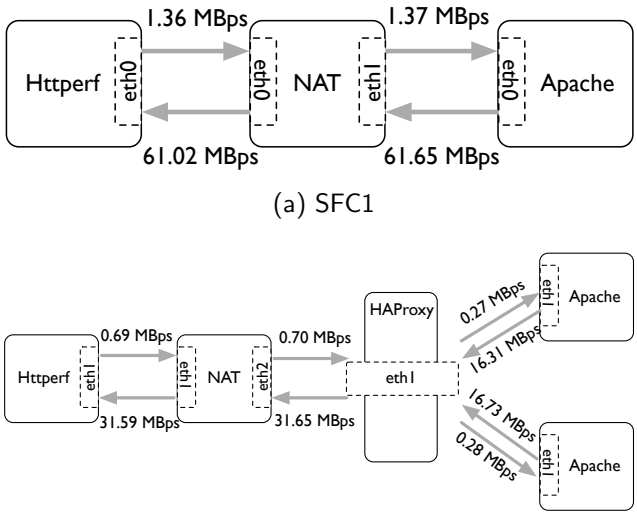

Figure 2: Throughput of each NF

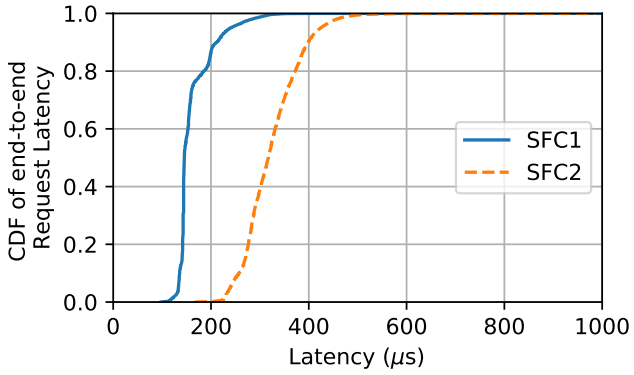

Figure 3: End-to-end request latency of SFC1 and SFC2.

5. CONCLUSIONS

6. REFERENCES

[1] Opnfv. `https://www.opnfv.org/`.

[2] Project Clearwater. `http://www.projectclearwater.org/`.

[3] Service function chaining use cases in data centers. `https://datatracker.ietf.org/doc/draft-ietf-sfc-dc-use-cases/?include_text=1`.

[4] A. Bremler-Barr, Y. Harchol, and D. Hay. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 511–524, New York, NY, USA, 2016. ACM.

[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling Innovation in Network Function Control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 163–174, New York, NY, USA, 2014. ACM.

[6] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A Framework for NFV Applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 121–136, New York, NY, USA, 2015. ACM.

[7] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. Netbricks: Taking the v out of nfv. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 203–216, GA, 2016. USENIX Association.

[8] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, San Jose, CA, 2012. USENIX.

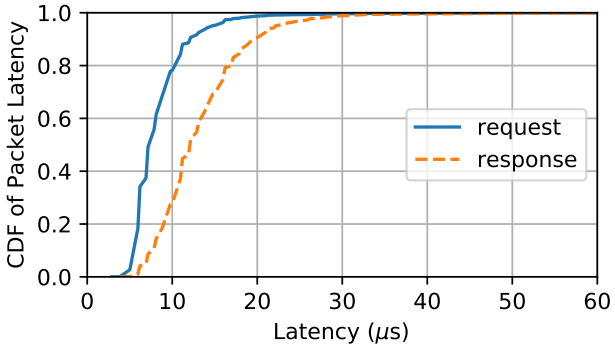

Figure 4: Per-packet latency of NAT in SFC1