DBpia

논문 18-43-10-17 The Journal of Korean Institute of Communications and Information Sciences '18-10 Vol.43 No.10
https://doi.org/10.7840/kics.2018.43.10.1676

NFV 관리 및 오케스트레이션을 위한 경량 시뮬레이터

응웬트리하이◆, 유 명 식°

Lightweight Simulator for NFV Management and Orchestration

Tri-Hai Nguyen◆, Myungsik Yoo°

요 약

네트워크 기능 가상화 (NFV)는 전용 하드웨어의 네트워크 기능을 분리하고 기존 상용서버에 가상 네트워크 기능 (VNF)을 통해 서비스를 제공함으로써 오늘날의 네트워크에서 네트워크 관리 및 서비스 제공을 용이하게 하는 유망한 패러다임이다. NFV 관련 연구를 가속화하기 위해서는 실험을 수행 할 NFV 환경을 제공 할 수 있는 경량 시뮬레이터가 필요하다. 본 논문에서는 잘 알려진 네트워크 에뮬레이터인 Mininet 기반의 경량 NFV 시뮬레이터를 개발하였다. Python 프로그래밍 언어로 작성되었으며 Linux 환경에서 실행된다. 제안 NFV 시뮬레이터는 VNF 및 서비스 기능 체이닝 (SFC)을 평가하는 데 필수적인 환경을 제공 할 수 있다.

Key Words : NFV, Simulator, VNF, VNFFG, Mininet

ABSTRACT

Network Function Virtualization (NFV) is a promising paradigm which facilitates network management and service provision on today's network by decoupling network functions from the underlying dedicated hardware and offering them through Virtual Network Functions (VNFs) on the commercial off-the-shelf servers. To accelerate the related research on NFV, there is a need for an inexpensive simulator that can provide the NFV environment to conduct the experiments. In this paper, we develop a lightweight NFV simulator based on Mininet, which is a well-known network emulator. It is written by Python programming language and runs on the Linux environment. This NFV simulator can create the essential environment to evaluate the VNF and Service Function Chaining.

Ⅰ. Introduction

Currently, launching a new network service needs to deploy a variety of proprietary hardware appliances and accommodating these hardware appliances is becoming more and more difficult. Moreover, with the ever-increasing and various service requirements, service providers must scale up their physical infrastructure periodically, which directly leads to high capital expenditure (CAPEX) and operation expenses (OPEX). Network Function Virtualization (NFV)[1] utilizes standard IT

※ This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2018-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Promotion).

◆ First Author : (ORCID:0000-0002-2132-2290)Department of ICMC Convergence Technology, Soongsil University, Seoul, Republic of Korea, nguyentrihai@soongsil.ac.kr, 학생회원

° Corresponding Author : (ORCID:0000-0002-5578-6931)School of Electronic Engineering, Soongsil University, Seoul, Republic of Korea, myoo@ssu.ac.kr, 종신회원

논문번호 : 201810-299-D-RN, Received October 1, 2018; Revised October 4, 2018; Accepted October 4, 2018

www.dbpia.co.kr

virtualization technologies to separate the network function from hardware. Therefore, it can effectively reduce the CAPEX and OPEX for service providers.

In principle, all network functions and other network elements can be considered for virtualization. These virtualized instances are referred to as Virtual Network Functions (VNFs) in the context of NFV, which provide the same functionalities as the corresponding physical instances. Besides, VNFs can be instantiated, executed and deployed by service providers in the NFV Infrastructure (NFVI) environment which provides the required resources such as compute, storage, and network. To manage and orchestrate these VNFs in NFVI, there is an element called NFV Management and Orchestration (MANO). NFV MANO can be further divided into three entities, that is, Virtualized Infrastructure Manager (VIM), VNF Manager (VNFM) and NFV Orchestrator (NFVO). The NFV architecture is depicted in Figure 1. NFV MANO, NFVI, and VNFs communicate each other through application programming interfaces (APIs) as shown by lines in Figure 1.

To speed up NFV related research, tools that can provide a testbed for experiments with NFV systems is required. Currently, there are some related NFV simulation tools. Devstack[3] enabled Tacker[4] can bring up a complete OpenStack environment with NFV MANO service. Unfortunately, it is quite heavy and takes a lot of time for bringing complete NFV environment. Mininet[5] is a well-known network emulator but it only supports the traditional network and the software-defined network. For these reasons, a lightweight NFV simulator is proposed and implemented in this paper.

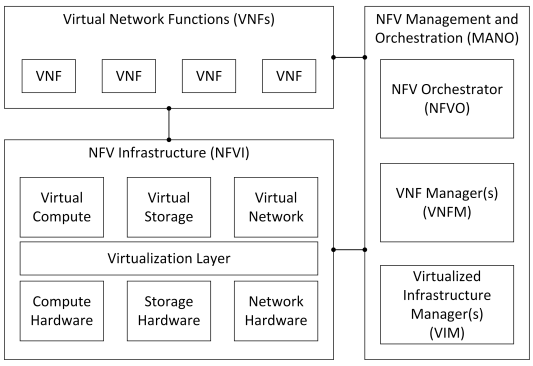

Fig. 1. NFV architecture.

The rest of this paper is organized as follows. Section II introduces the proposed NFV simulator and its features. The implementation of the NFV simulator is shown in Section III. Finally, the conclusion of this paper is presented in Section IV.

Ⅱ. A Lightweight NFV Simulator

In the OpenStack, Tacker[4] is the project that implements a generic NFV MANO. DevStack[3] tool along with Tacker service can simplify the deployment of an OpenStack cloud with NFV orchestration and management. However, deploying, configuring and managing NFV-based OpenStack environments is still a time-consuming process with a considerable learning curve. On the other hand, Mininet[5] has shown itself as a great tool for agile network/SDN experiment. However, it does not support NFV. The goal of the proposed NFV simulator tool is to alleviate the developer and researchers' tedious task of setting up a complete NFV environment. As a result, they can focus on other works such as developing a VNF, prototyping, implementing an orchestration algorithm or a customized traffic steering.

The proposed NFV simulator is depicted in Figure 2. It is written in Python programming language and based on the Mininet API library. By using Mininet API, the simulator can the essential networking environment, which includes the controller, switches, links, nodes, network topology. We extend the specific NFV scenario that can run and evaluate the VNF and VNF Forwarding Graph (VNFFG) or Service Function Chain (SFC). A VNFFG or SFC is a set of ordered list of VNFs. NFV simulator allows to define the VNF and VNFFG into Mininet by using Topology and Orchestration Specification for Cloud Applications (TOSCA) templates as same as Tacker.

NFV simulator includes several characteristics such as NFV catalog, VNF Manager, and NFV Orchestrator. In particular, NFV catalog includes VNF descriptors, VNFFG descriptors. VNF Manager

www.dbpia.co.kr

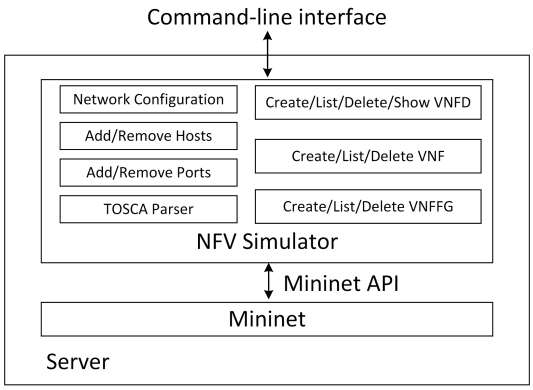

Fig. 2. NFV simulator architecture.

can do the basic life-cycle of VNF such as create, update, delete VNF and it also can facilitate initial configuration of VNF. NFV Orchestrator can connect VNFs using a Service Function Chain (SFC) that described in a VNFFG Descriptor, support VNF placement policy that ensures efficient placement of VNFs, and also support symmetrical and asymmetrical traffic from and to the VNFs. Other features such as network definition via Virtual Link (VL), IP/MAC definition via Connection Point (CP), emulation of flavor properties though Mininet's CPULimitedHost, and cloud-init scripts are also supported in NFV simulator.

More specifically, the NFV simulator will create three default networks including net_mgmt (192.168.120.0/24), net0 (10.10.0.0/24) and net1 (10.10.1.0/24). It will assign random IPs within the defined networks for hosts. In the TOSCA-based template, we can also manually define the networks in the VL definition section and assign the IP/MAC address for a VNF in the CP definition section. The NFV simulator emulates VNF resource configuration defined via num_cpus properties or through flavor in the TOSCA-based template. Currently, NFV simulator supports VNF flavors such as small (1 CPU), medium (2 CPUs), large (4 CPUs), and xlarge (8 CPUs). The NFV simulator also supports VNFs configuration through user-data. The NFV simulator can be run with the standalone mode or with the external network controller. With the external network controller mode, an external controller (e.g., POX, OpenDaylight, Floodlight controller) needs to be run on another terminal in the Linux operating system.

Ⅲ. Experiment

The source code of the NFV simulator is available online[6]. The NFV simulator is written in Python, running on Linux (i.e., Ubuntu 16.04) environment. The NFV simulator only takes a few minutes and small storage (a few megabits) to completely setup the NFV environment. It is more lightweight than Devstack, which takes an hour and some gigabytes of storage to fully install the NFV environment.

The main source code of NFV simulator is contained in "nfv-simulator.py" file. The samples of VNF descriptor, VNFFG descriptor, and functional testing files are included in the "samples" folder. The following commands are used to run the NFV simulator and executed by command-line interface (CLI) in Ubuntu 16.04. The NFV simulator is required some dependent libraries and packages such as mininet, openvswitch-testcontroller, python-netaddr, python-yaml which will be installed by the "install-prequirements.sh" script in the source code. Then, NFV simulator can be run in standalone mode as shown in Figure 3 by the following command.

```
$ sudo python ./nfv-simulator.py --standalone
```

The user can type "help" in "nfv-simulator>" to show the commands can be used. There are some NFV related commands such as vnf_create, vnf_delete, vnf_list, vnfd_create, vnfd_delete, vnfd_list, vnfd_template_show, vnffg_create, vnffg_delete, vnffg_list. To know how to use the command, the user can type the command to CLI without other parameters. For example, to show the usage of vnf_create command, the user only needs to type "vnf_create" into nfv-simulator CLI as illustrated in Figure 4.

The "samples/test" folder contains some functional testing files. For example, there is a testing file named "vnffg_test" for evaluating the VNFFG as shown in Table. 1. It creates a "vnfUD"

www.dbpia.co.kr

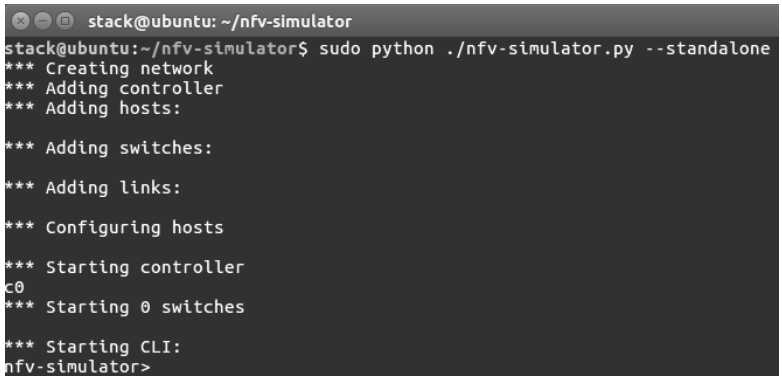

Fig. 3. NFV simulator is started.

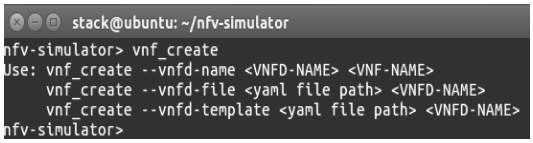

Fig. 4. vnf_create command usage

VNF with TOSCA-based template named "tosca-vnf-userdata.yaml". Then, it creates a client host, a server host, and a switch. The server is installed as an HTTP server by the "python -m SimpleHTTPServer 80 &" command. Finally, a VNFFG or service chain is created with TOSCA-based template named "tosca-vnffgd-hello-world.yaml" and "vnfUD" VNF is a node that is mapping the client and server hosts.

To run this testing file, the user type "source samples/test/vnffg_test" in nfv-simulator CLI. Figure 5 depicts the results of vnffg_test testing. The final line in the functional test shows the chain of service which includes the IP source, port source, IP

Table 1. The content of vnffg_test

py "*** Starting vnf <tosca-vnfd-userdata.yaml> ..."
vnf_create --vnfd-file samples/vnfd/tosca-vnfd-userdata.yaml vnfUD
py "\n*** Creating http client ..."
add_host client 192.168.120.1/24
py "*** Creating http server ..."
add_host server 192.168.120.2/24
py "*** Starting the switch ..."
switch s192.168.1 start
py "*** Starting HTTP server ..."
py server.cmdPrint('python -m SimpleHTTPServer 80 &')
py "*** Starting vnffg <tosca-vnfd-hello-world.yaml> ..."
vnffg_create --vnffgd-template samples/vnffgd/tosca-vnffgd-hello-world.yaml --vnf-mapping vnfd-hello-world:'vnfUD' --symmetrical false vnffg-sample

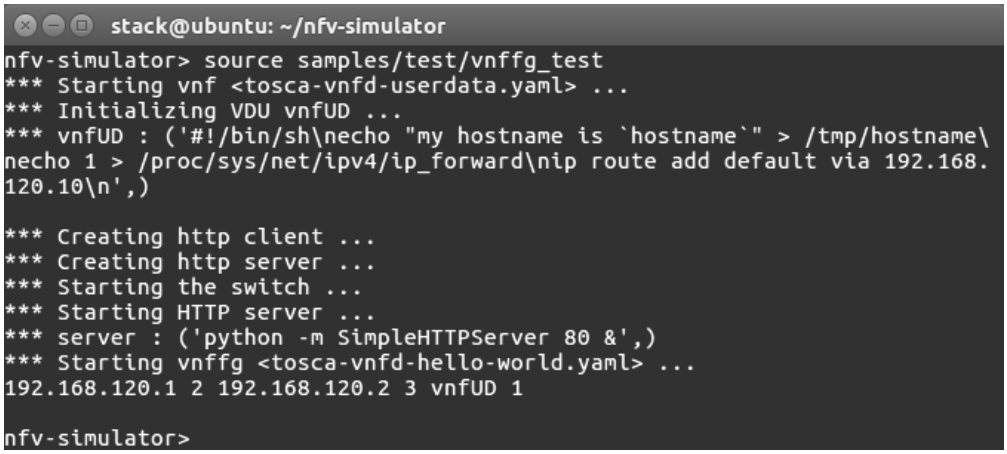

Fig. 5. The Results Of Vnffg_Test Testing.

www.dbpia.co.kr

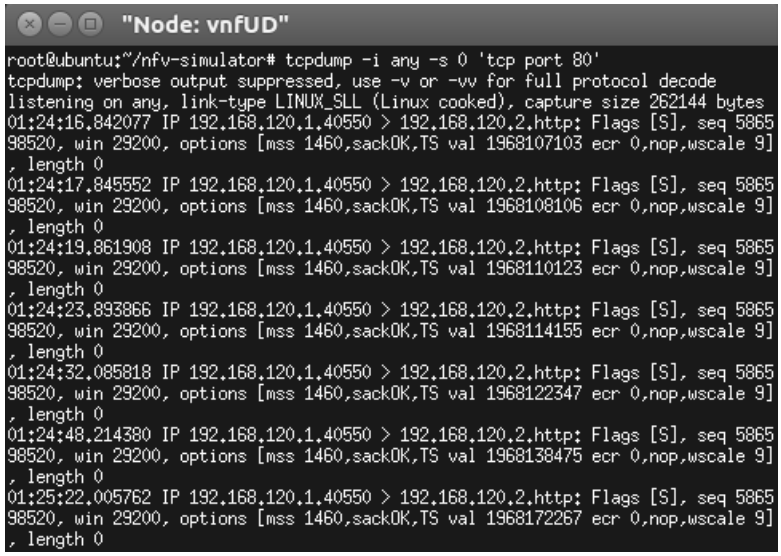

Fig. 6. The Http Traffic Goes Through Vnf Interface.

destination, port destination, VNF forwarder, and port of VNF forwarder.

To show if the VNF forwarder – vnfUD works, the xterm window of the vnfUD is called by "xterm vnfUD" in the NFV simulator CLI, and then the client host gets HTTP content of server host by "client curl server" command. In the xterm windows of the vnfUD, its network interface is sniffed by "tcpdump" tool so that the HTTP traffic of the curl command through the VNF interface can be seen as shown in Figure 6.

Ⅳ. Conclusion

In this paper, a lightweight NFV simulator is proposed and implemented. It based on Mininet and has the workflow the same as the Tacker NFV project. The completed installation of NFV simulator only takes a few minutes and it only occupied a small number of the hardware resource. By using this NFV simulator, the developer and researcher can offload the tedious task of setting up a whole NFV environment and help them to more focus on the developing of VNF, implementing the algorithm or customizing traffic steering.

References

[1] ETSI NFV, *Network function virtualisation: An introduction, benefits, enablers, challenges & call for action*, Introductory White Paper, Issue 1, SDN & OpenFlow World Congr., Darmstadt, Germany, Oct. 2012.

[2] T. H. Nguyen, T. Nguyen, and M. Yoo, "Analysis of deployment approaches for virtual customer premises equipment," in *Proc. 32nd ICOIN 2018*, Chiang Mai, Thai Lan, Jan. 2018.

[3] The OpenStack Foundation: Devstack. [Online]. Available: https://docs.openstack.org/devstack/.

[4] The OpenStack Foundation: OpenStack Tacker. [Online]. Available: https://docs.openstack.org/tacker/.

[5] Mininet. [Online]. Available: http://mininet.org/.

[6] NFV Simulator. [Online]. Available: https://anda.ssu.ac.kr/nfv-simulator/.

www.dbpia.co.kr

응웬트리하이 (Tri-Hai Nguyen)

Tri-Hai Nguyen received his B.S. degree in computer science from the University of Information Technology, Ho Chi Minh City, Vietnam, in 2015 and M.Eng. degree in information and communication technology from Soongsil University, Republic of Korea, in 2017. He is currently pursuing the Ph.D. degree in information and communication technology at Soongsil University, Seoul, Republic of Korea. His research focuses on network function virtualization and cloud computing.

유 명 식 (Myungsik Yoo)

Myungsik Yoo received his B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, Republic of Korea, in 1989 and 1991, and his Ph.D. in electrical engineering from State University of New York at Buffalo, New York, USA in 2000. He was a senior research engineer at Nokia Research Center, Burlington, Massachusetts. He is currently a professor in the school of electronic engineering, Soongsil University, Seoul, Republic of Korea. His research interests include visible light communications, sensor networks, Internet protocols, control, and management issues.

www.dbpia.co.kr