논문 18-43-09-09 The Journal of Korean Institute of Communications and Information Sciences '18-09 Vol.43 No.09
https://doi.org/10.7840/kics.2018.43.9.1476

NFV에서 동적 Service Function Chain 업데이트를 위한 메커니즘

응웬쭝당찐[◆], 유 명 식[°]

Mechanism for Dynamically Updating Service Function Chains in NFV

Trinh Nguyen[◆], Myungsik Yoo[°]

요 약

네트워크 트래픽 복잡성 증가와 SFC (Service Function Chaining)이 설계된 방식은 DoS 공격과 같이 비정상 상황이 발생할 경우 현재 진행되는 SFC를 재 설정하는 것을 어렵게 만들었다. 네트워크 관리자는 성능 저하된 SFC를 수동적으로 새로운 트래픽 분류자를 통하여 새로운 가상 서비스 셋을 정의하고, 기존의 SFC를 제거하고, 마지막으로 새로운 SFC를 생성하게 된다. 이러한 과정은 긴 단절 시간으로 인하여 사용자들의 불만을 초래한다. 본 논문에서는 연속적 모니터링을 통하여 동적인 SFC 업데이트가 가능한 방법을 제안하는데, 제안 방식으로 인하여 단절 시간 단축 및 결함 관리를 용이하게 할 수 있다. 사용사례 구현을 통하여 제안 방식의 타당성을 검증하였다.

Key Words : Network Function Virtualization, Continuous Monitoring, Service Function Chaining, Self-healing, Dynamic Update

ABSTRACT

The increasing complexity of the network traffic and the way service function chains (SFC) were architect make it difficult to readjust an on-going SFC when unusual events occur such as traffic overhead or Denial-of-Service (DoS). Normally, network administrators have to modify the degraded SFC manually by defining a new set of virtual services with new traffic classifiers, then deactivating the current set of virtual services and finally activate the new set. That process often causes dissatisfaction from customers because of prolonged downtime. This paper proposes a novel mechanism that supports dynamic update of SFC(s) using a continuous monitoring technique based on predefined policies which will help reduce the chance of disconnection and improve the effectiveness of fault management. An use case is also discussed to show the feasibility of proposed mechanism.

※ 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:0000-0001-6885-5935)Department of ICMC Convergence Technology, Soongsil University, Seoul, Republic of Korea, dangtrinhnt@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, 정회원

논문번호 : 201809-274-D-RN, Received September 6, 2018; Revised September 15, 2018; Accepted September 17, 2018

Ⅰ. Introduction

Network Function Virtualization (NFV) introduces a new way to architect, instantiate and administrate networking services. NFV separates network functions, some of which are firewalls, proxy cache, and domain name service etc. from proprietary hardwares so they can run in software. It is created to unify and supply the networking elements needed to support a comprehensively virtualized infrastructure including virtual storages, servers, and or multiple networks. NFV virtualizes network functions by making use of standard virtualization technologies that run on high volume servers and switches. Similar to traditional networking, NFV provides end-to-end services by defining ordered lists of network functions except NFV uses software-defined components to perform the networking features instead of relying on middleboxes. These Virtual Network Functions (VNFs) are then virtually attached together in the network to create one or more service chains. In addition, classifiers are added to SFC as a mean to match traffic flows against the rule for subsequent application of the specified set of network service functions[3]. Those traffic classifiers and VNF ordering constraints add complexity to the designing SFC part of Network Service (NS) deployment process in NFV based networks.

State-of-the-art deployment strategies for NS are often closely linked to physical infrastructure and network topology, consequently leading to comparatively strict and fixed deployments. The static nature of such deploying plans largely shortens and tightens the ability of operators to introduce new or modify existing services. Besides that, readjusting elements of an SFC usually impacts other units in the chain and or the network elements used to build the chain. This drawback is specifically critical in elastic service environments that need comparatively speedy creation, deletion, or changing of VNF or network components. Moreover, the conversion to virtual infrastructure needs a quick service modification model that supports elastic and grainy NS provision, and the migration of VNF and application workloads in the existing network. Therefore, It is necessary to develop SFC dynamic update methodologies to align with ETSI' NFV resiliency requirements[2].

Although dynamic SFC has been discussed oftentimes in literature, there is still missing of such simple and cost effective modification techniques for SFC in NFV environment. Most of the existing solutions embeds dynamic SFC functionalities into NFV which will not only increase workload of the system itself while dealing with extra computational tasks, but also encounters challenges to add new features because of the inflexible nature of such embedded softwares. Liu et al.[6] tried to optimize the readjusting on-going users' SFC process by acquiring an integer linear programming model along with a column generation model. That method consumes an amount of computing resource and may add latency to services provisioning. Zave et al.[9] developed Dysco, a session protocol that dynamically reconfigures on-going SFC sessions. Despite Dysco does not require any changes to end-host and VNF applications, it only supports TCP-based services[4], including but not limited to load-balancer, proxy caching etc. Hence, Domain Name Service (DNS), Dynamic Host Configuration Protocol (DHCP) or other UDP-based VNF applications[5] are left unsupported. Cunha et al.[10] achieved dynamic SFC using a Policy Enforcer and Application Programming Interface (API) to reconfigure VNFs. Owing to relying on a set of VNFs including a Deep Packet Inspection (DPI), a Content Filter (CF), a Firewall (FW), a Traffic Shaper (TShap), and a Traffic Accounter (TAcc) to provide SFC functionalities, that approach creates a dependency problem for other VNFs. Mechtri et al.[7], different from two previous papers, took a broader view when trying to design a new NFV orchestration framework that focuses on tackling some of the SFC challenges. One of the issues that framework attempts to solve is supporting SFC in NS descriptors by extending the TOSCA data model. However, new release of TOSCA will break its current implementation.

On the contrary to those before-mentioned works, this paper takes a more feasible approach to the dynamic SFC readjustment problem that is continuously monitor system components using external client-server based monitoring service plus predefined backup or remedy policies based on existing knowledge of events that may occur. As a result of leaving the NFV Orchestrator (NFVO) untouched, the proposed system introduces plug-and-play and extensible capabilities. That mechanism will be explained thoroughly in Section II of this paper. Additionally, a specific use case is explored in Section III. Finally, the concluding remarks and future works are provided in Section IV.

Ⅱ. Dynamic Update Mechanism df Service Function Chains

At a high level point of view, the designed system achieves dynamic update of SFC capability by leveraging a continuous monitoring technique that is periodically gathering metric-based information, e.g. CPU, workload, memory etc. sent from monitored targets such as VNF(s), the NFVO and so on. With each piece of information collected, monitoring server then matches with the predefined policies to see whether a certain service request is normal or abnormal (a DoS attack, or some VNF(s) down and replacement etc.). When a traffic behavior-policy match is found, the monitoring server will trigger a backup or remedy action for that particular SFC. The actions are also planned before hand by network administrators based on knowledge of incidents. Some example actions are deploying a new traffic classifier that blocks the Internet Control Message Protocol (ICMP) when high load detected, or replacing a web VNF that is down. High level work flow of the proposed mechanism can be seen in Figure 1.

Instead of hard-coding into NFV for realizing the dynamic updating SFC features which requires lots of development effort and platform specific oriented, the proposed system takes advantages of existing solutions that is client-server based monitoring software and just focuses on the actual

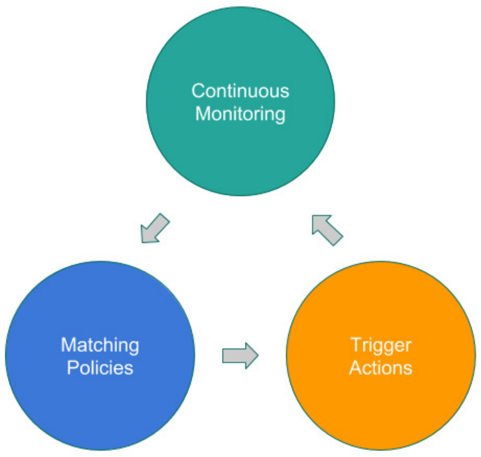


Fig. 1. High level work flow of the proposed system

functionalities it desired. Thus, monitoring features and trigger logics are decoupled from the NFV Management and Orchestration (NFV MANO) that leaves rooms for further extensions and future maintenance processes. Additionally, the current monitoring features of well-known NFV MANO implementations, e.g. Tacker, can only handle a limited number of system properties for instance, Ceilometer only supports hardware infrastructure resource-related parameters (e.g. CPU or memory usage). Other drivers like ping or http_ping support basic checking for VNFs reachability. Therefore, to guarantee the availability and stability of network services provided by VNFs, a more sophisticated monitoring tool is required. By using an external tool that has the capability to monitor many aspects of a NFV system, the challenge of dynamically modifying the SFCs becomes solvable.

Figure 2 describes the main architecture of designed mechanism to readjust SFC dynamically. By employing a client-server scheme, the proposed system requires to install agent software in each of the VNFs and the NFV MANO itself. Monitoring agents for VNFs will be deployed at the time of VNF instantiation. The agents installed inside VNFs only take care of the VNFs while at NFV MANO level, agents will be assigned manually by network administrators and handle the SFC modification. Those agents then extract critical information of

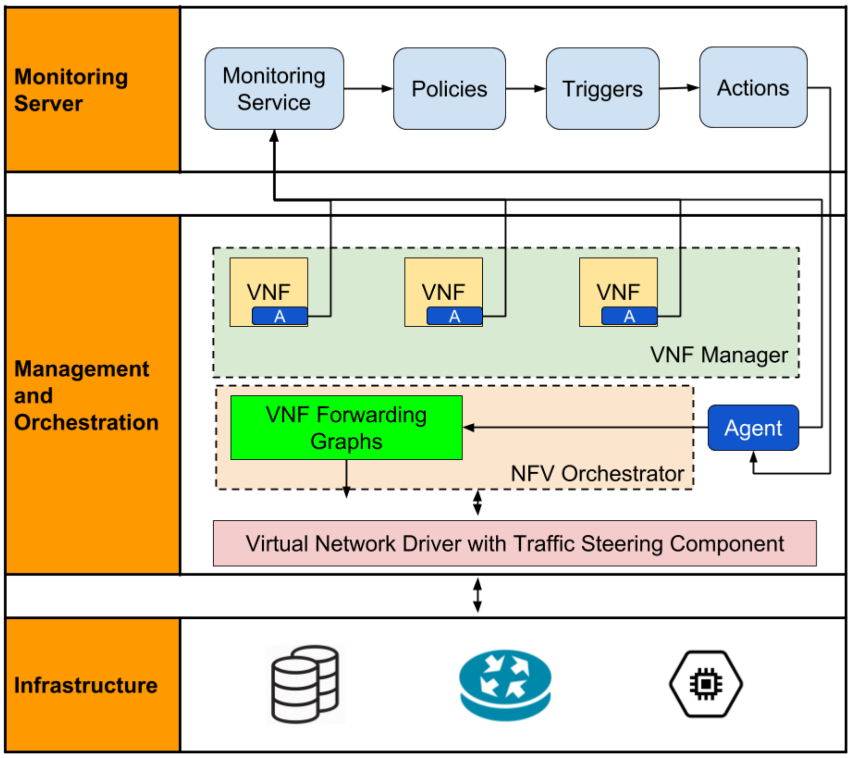


Fig. 2. Dynamic update for SFC architecture

monitored targets and send to the monitoring server. Critical information including CPU utilization, memory usage of the VNFs, packets that go through the system and so on will be filtered by the server based on predefined policies. There are rules that matches traffic behavior and status of monitored targets with known faults that may lead to failures. If a match is found, a trigger will be executed to produce some actions that try to recover the degraded SFC. Examples of policies are monitoring server will trigger the SFC readjustment procedure that adds new traffic classifier to drop ICMP packages when the number of ICMP packages coming through is too high (e.g. greater than 2000 bytes per second), or the server will trigger the SFC update to replace an unresponsive web VNF. This policy-driven nature along with extensible characteristic of the proposed mechanism help the NFV system accomplish the desired dynamic SFC readjustment.

Ⅲ. Experiment

The experiment introduces a specific use case in which open source software solutions including OpenStack cloud platform and Zabbix[11] monitoring server are leveraged to realize the desired dynamic SFC update feature. In this experiment we chose to monitor the ICMP traffic as it is a basic indicator for abnormal activities when the number of the ICMP packages is too high.

Network operators can use other types of traffic and parameters such as FTP, SSH requests, or CPU usage of the VNFs etc. as monitoring metrics. HTTP requests are also used in this experiment to show that only the ICMP packages are blocked when the monitoring server executes the SFC update command to add the ICMP classifier. The experimental results are presented to demonstrate that the proposed mechanism can be plugged into an NFV MANO system with less effort and provides dynamic SFC readjustment. Table 1 is the

Table 1. Specifications of the experiment

Server	Details
Devstack Server	CPU Intel Xeon E3 1240 3.4 GHz, 4 x 4GB DDR3 1333MHz, 1TB HDD Ubuntu 16.04 LTS x64
Zabbix Server	CPU Intel Core i5 7600 3.5 GHz, 8GB DDR4 2400MT/s, 1TB HDD Ubuntu 16.04 LTS x64

specifications of this experiment.

As shown in the experimental topology (Figure 3), Devstack[12] is used to quickly create an OpenStack development environment with Tacker[16], a NFV MANO implementation, and other dependency services (e.g. Heat, Networking-SFC, Barbican, Mistral, Ceilometer, Aodh etc.). The Devstack server has two main virtual networks, one for management (net_mgmt) and the other for the internal network (net1) that connects the VMs. Besides, it is required that the Zabbix server is connected to the same network with Devstack' external interface to monitor VM instances through the floating IP.

Monitoring server is configured through several steps. Firstly, we register the OpenStack machine, then create a trigger with a name which contains the keyword BpS (e.g. BpS in the eth0 is too high), and add to that trigger a matching condition where the matching expression is anda-os:net.if.in[enth0].avg (2)>2000. We then create an action that will update the classifier if a trigger with the BpS name will be generated by Zabbix server. The action command can be found in [13].

After monitoring server has been set up properly, Zabbix agent is installed natively in the OpenStack server. That client software is used to send data back to the monitoring server and execute the

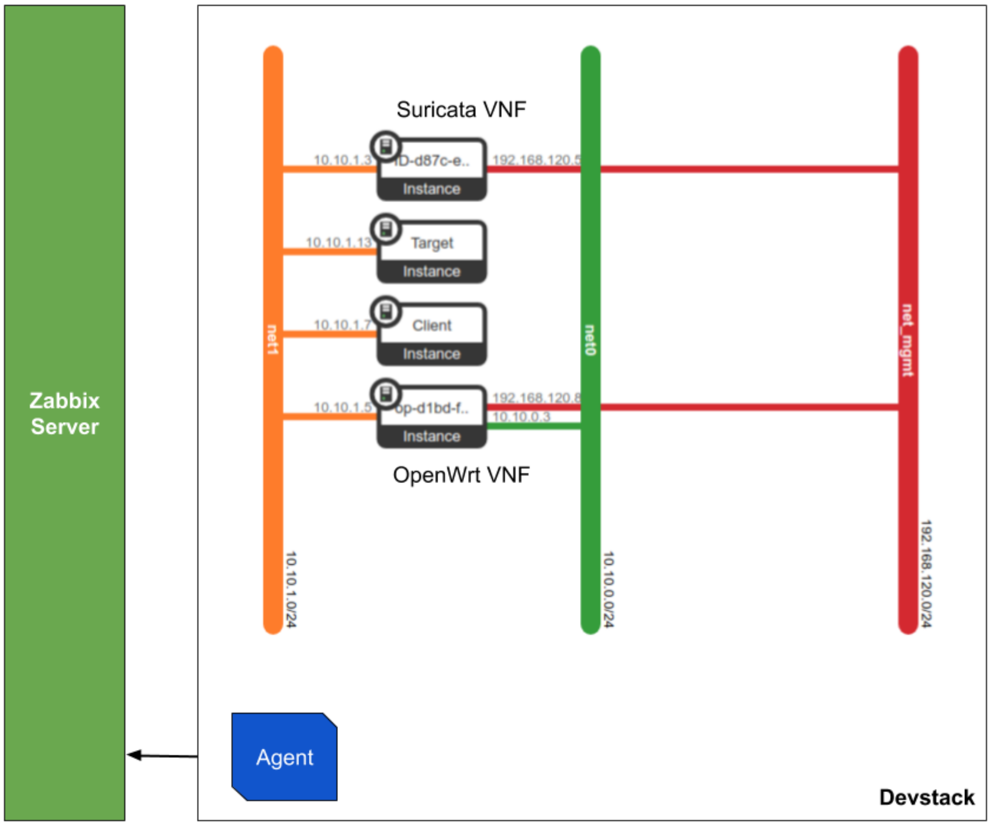


Fig. 3. Experimental Topology

VNFFG update command to modify the classifier of the chain. Thereafter, two Service Functions (SFs) via two VNFs are deployed. The first VNF is Suricata (IDS) which will be instantiated using this VNF Descriptor (VNFD) template[14]. The second one is OpenWrt (firewall) with the same configuration as it is described in the Tacker's documentations[15].

Initially, we create a VNFFG with a chain (IDS, firewall) and no classifier. We then generate ICMP traffic using PING towards the Devstack with a floating IP address. When that traffic reaches a threshold, a specific event is sent to the Zabbix server and Zabbix server tells its client to execute the VNFFG update command to add an ICMP classifier to the existing VNFFG. That workflow is illustrated in Figure 4.

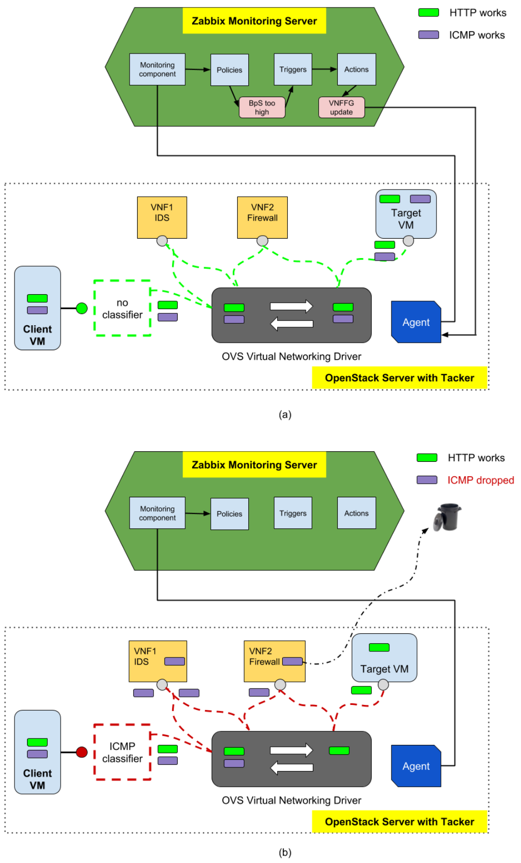
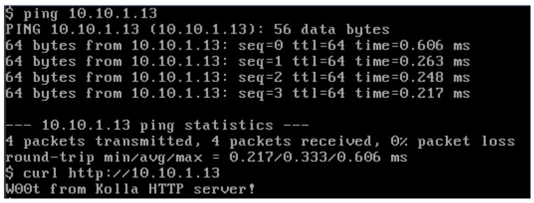


Fig. 4. Experimental Work Flow: (a) System under normal condition, no traffic classifier, the monitoring server was setup to capture high load event with a trigger to execute the VNFFG update action. (b) "BpS too high" event detected, ICMP traffic classifier added via VNFFG update.

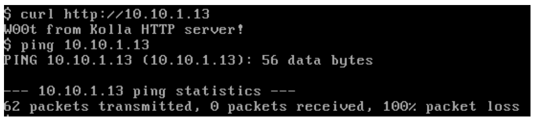
The screen snapshots of the experiment are shown in Figure 5 and Figure 6. Figure 5 presents the normal state of the system when the number of ICMP packages sent to Devstack is less than 2000. In this stage of the experiment, there is no traffic classifier, both PING and HTTP requests are processed successfully. Figure 6 shows the ICMP requests are blocked when the number of ICMP packages is more than 2000, which is the threshold we defined in the Zabbix server's trigger. The Zabbix server has updated the VNFFG to add an ICMP classifier when the number of ICMP packages reaches the threshold. In this stage of the experiment, only HTTP requests are passed.



```
$ ping 10.10.1.13
PING 10.10.1.13 (10.10.1.13): 56 data bytes
64 bytes from 10.10.1.13: seq=0 ttl=64 time=0.606 ms
64 bytes from 10.10.1.13: seq=1 ttl=64 time=0.263 ms
64 bytes from 10.10.1.13: seq=2 ttl=64 time=0.248 ms
64 bytes from 10.10.1.13: seq=3 ttl=64 time=0.217 ms

--- 10.10.1.13 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.217/0.333/0.606 ms
$ curl http://10.10.1.13
W00t from Kolla HTTP server!
```

Fig. 5. No traffic classifier: Client VM can ping and request HTTP packages from the Target VM



```
$ curl http://10.10.1.13
W00t from Kolla HTTP server!
$ ping 10.10.1.13
PING 10.10.1.13 (10.10.1.13): 56 data bytes

--- 10.10.1.13 ping statistics ---
62 packets transmitted, 0 packets received, 100% packet loss
```

Fig. 6. ICMP blocked: Client VM cannot ping the Target VM, only HTTP works

Ⅳ. Conclusion

This article proposes a feasible and extensible mechanism for dynamically updating the SFCs. Contrarily to other works, this mechanism utilizes an external monitoring tool and existing capabilities of NFV implementations to update the desired SFCs. This approach not only releases the burden of extra development and maintenance effort but also reduces the opportunity of overhead caused by additional built-in components of NFV system. Future works will take into account the proactive SFC generation process based on multiple criteria.

References

[1] European Telecommunications Standards Institute (ETSI), *Network Functions Virtualisation Assurance*; Report on Active Monitoring and Failure Detection, Apr. 2016.

[2] European Telecommunications Standards Institute (ETSI), *Network Function Virtualisation Resiliency Requirements*, Jan. 2015.

[3] Internet Engineering Task Force (IETF), *RFC 7665 Service Function Chaining (SFC) Architecture*, https://www.ietf.org/rfc/rfc7665.txt, Oct. 2015.

[4] Internet Engineering Task Force (IETF), *RFC 793 Transmission Control Protocol*, https://tools.ietf.org/rfc/rfc793.txt, Sept. 1981.

[5] Internet Engineering Task Force (IETF), *RFC 768 User Datagram Protocol*, https://www.ietf.org/rfc/rfc768.txt, August 1980.

[6] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. and Serv. Management*, vol. 14, no. 3, pp. 543-553, Sept. 2017.

[7] M. Mechtri, C. Ghribi, O. Soualah, and D. Zeghlache, "NFV orchestration framework addressing SFC challenges," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 16-23, Jun. 2017.

[8] Organization for the Advancement of Structured Information Standards (OASIS), *TOSCA Simple Profile for Network Functions Virtualization* (NFV), Version 1.0, May 2017.

[9] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, J. Rexford, "Dynamic service chaining with Dysco," *SIGCOMM'17*, pp. 57-70, Aug. 2017.

[10] Vitor A. Cunha, Igor D. Cardoso, João P. Barraca, and Rui L. Aguiar, "Policy-driven vCPE through dynamic network service function chaining," *IEEE NetSoft*, pp. 156-160, Jun. 2016.

[11] Zabbix. [Online], Available: https://www.zabbix.com

[12] The OpenStack Foundation: DevStack. [Online], Available: https://bit.ly/2xi77KG

[13] VNFFG update command. [Online], Available: https://bit.ly/2QufsDV

[14] Suricata VNFD. [Online], Available: https://bit.ly/2MwZupm

[15] The OpenStack Foundation: Deploying OpenWRT as VNF. [Online], Available: *https://bit.ly/2NGTSNX*

[16] The OpenStack Foundation: OpenStack Tacker. [Online], Available: *https://bit.ly/2QsSBIS*

응웬쫑당찐 (Trinh Nguyen)

Trinh Nguyen received his B.Eng. degree in Computer Networking from University of Information Technology, VNU-HCM, Ho Chi Minh City, Vietnam, in 2012. He has been pursuing the Master's degree in ICT at Soongsil University since Fall 2017. His research interests include Software-Defined Networking, 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.