JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 19, No. 4, pp. 809-814, Apr. 2018

마이크로 서비스 아키텍쳐 기반 가상 인프라 매니저 설계 및 구현

나 태 흠 · 박 평 구 · 류 호 용*
한국전자통신연구원(ETRI) 네트워크연구본부

Design and Implementation of virtualized infrastructure manager based on Micro Service Architecture

TaeHeum Na · PyungKoo Park · HoYong Ryu*
Network Research Division, Electronics and Telecommunication Research Institute (ETRI)

[요 약]

클라우드 컴퓨팅 기반 인프라가 확산됨에 따라, 서비스 프로바이더는 온-디맨드 방식의 서비스 배포가 가능해졌다. 최근 클라우드형 인프라의 자원 확장성 효율을 극대화하기 위해 마이크로 서비스 구조가 주목받고 있다. 모든 서비스 기능을 하나의 소프트웨어로 구현하는 대신 필요한 서비스를 효율적으로 설계된 Application Programming Interface (API)를 통해 연동함으로써 쉽고 자율적으로 구현할 수 있고, 기능의 요구사항에 맞는 프로그래밍 언어, 소프트웨어, 기능구조를 자유로이 정할 수 있다. 본 논문에서는 마이크로 서비스 구조를 기반으로 가상 인프라 매니저 서비스를 설계하고 제안된 구조가 부하에 따라 효율적으로 스케일링이 가능함을 실험을 통해 검증한다.

[Abstract]

With the proliferation of cloud computing infrastructures, service providers are able to deploy services in on-demand manner. Recently, microservice architecture has been attracting attention in order to maximize the efficiency of resource expansion in cloud infrastructure. Instead of implementing all of the service functions in a single software, service providers can easily and autonomously implement the necessary services by interconnecting the necessary services through an efficiently designed application programming interface (API). Moreover service developer can freely choice programming languages and define software, and functional structures to meet their functional requirements. In this paper, we propose virtual infrastructure manager service based on microservice architecture and evaluates its performance in scalability perspective

색인어 : 클라우드 컴퓨팅, 마이크로 서비스, 가상머신, 컨테이너, 확장성

Key word : Cloud computing, Microservice, VM, Container, Scalability

http://dx.doi.org/10.9728/dcs.2018.19.4.809

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-CommercialLicense(http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 13 April 2018; **Revised** 24 April 2018
Accepted 25 April 2018

***Corresponding Author;** **HoYong Ryu**

Tel: +82-42-860-5364
E-mail: hyryu@etri.re.kr

Ⅰ. 서 론

클라우드 컴퓨팅 [1] 기반 데이터 센터가 확산됨에 따라, 서비스 프로바이더는 온-디맨드 방식의 서비스 배포가 가능해졌다. 클라우드 이전의 서비스의 경우 피크-시간 기반으로 서비스를 구동하는 서버의 자원을 준비하였고 이로 인해 피크-시간 외에는 낮은 자원 사용률을 보였다. 전술한 서비스의 낮은 자원 효율성을 향상시키기 위해 클라우드형 서비스로 배포를 함으로써 자원 효율성을 높일 수 있다.

최근 마이크로 서비스는 클라우드형 스케일링[1]-[3]을 가장 효율적으로 활용할 수 있는 구조로 주목받고 있다. 모든 서비스 기능을 하나의 소프트웨어로 구현하는 대신 필요한 서비스를 효율적으로 설계된 Application Programming Interface (API)를 통해 연동함으로써 쉽고 자율적으로 구현할 수 있고, 기능의 요구사항에 맞는 프로그래밍 언어, 소프트웨어, 기능구조를 자유로이 정할 수 있다. 기능별로 구분 된 마이크로 서비스 구조의 특징은 클라우드형 스케일-아웃에 매우 적합한 구조를 가진다. 기능별로 워크로드를 구분 지어 모니터링이 가능하고, 이를 기반으로 확장이 필요한 기능에 대해서만 (높은 부하의 기능) 스케일-아웃을 할 수 있다.

본 논문에서는 마이크로 서비스 구조에 기반을 둔 Network Function Virtualization (NFV)-Virtualzed Infrastructure Manager (VIM)의 구조를 제안한다. 제안된 구조가 부하에 따라 효율적으로 스케일링이 가능한지 여부를 실험을 통해 검증한다. 이를 활용하면 인프라를 관리함에 있어, VIM API의 호출 수에 따라 자원 효율적인 서비스 운용이 가능해진다.

Ⅱ. 마이크로서비스 기반 가상 인프라 매니저

마이크로 서비스의 핵심은 각 기능을 담당하는 소프트웨어를 자율적으로 구성/구현함으로써, 신속한 서비스 구현이 가능하게 하고, 더불어 분리되어 구동되는 특징으로 자원 효율적인 스케일 아웃이 가능하다. NFV MANO [4-5]는 국제 표준화 단체인 European Telecommunications Standards Institute (ETSI)에서 제안한 구조로, NFV를 관리/제어하기 위한 구조를 제안한다. Management and orchestration (MANO)의 구성 요소인 VIM의 전체적인 구조는 표준화 문서를 기반으로 설계하였고, 실제 Microservice Architecture (MSA)를 기반으로 구현함과 동시에 필요에 의해 기능을 추가하였다. 그림1은 VIM의 기능 구조를 보여준다.

설계된 VIM은 RESTful API [6-7] 를 동기/비동기 적으로 제공하며, API를 통해 호출된 기능은 내부의 인프라 커넥터의 기능을 호출한다. 비동기 처리를 위해 메시지 큐를 포함하고, 데이터베이스도 포함된다. 인프라 커넥터는 클라우드 운영체제인 OpenStack의 요소별 API를 호출하는 구조를 갖는다.

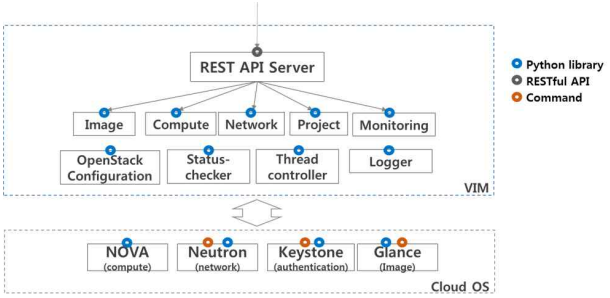

그림 1. 가상 인프라 매니저의 기능 구조
Fig. 1. Functional Architecture of VIM

2-1 VIM의 소프트웨어 구조

전술한 VIM의 기능 구조를 기반으로 실제적으로 소프트웨어 구조를 설계할 필요가 있다. 실제로 MSA 형식의 소프트웨어는 기능별로 분리된 구조를 설계하고, API를 통해 연동될 수 있어야 한다.

스케일 아웃을 지원하기 위한 소프트웨어 블록은 다음과 같다. VIM전체의 front-end를 담당하는 REST API 서버, 인프라 커넥터를 포함하는 VIM 내부 Logic, 비동기 태스크를 처리하는 Async Worker, 데이터베이스, Message Queue로 나누고, 각 소프트웨어 블록이 논리적으로 분리된 컨테이너 [8] 상에 구동된다. 그림2는 소프트웨어 구조를 나타낸다. 5가지 소프트웨어 블록은 각각 확장이 [9] 가능하고, 컨테이너 기반 가상화를 활용하여 확장에 필요한 소요시간과 확장 용이성을 향상시킬 수 있다.

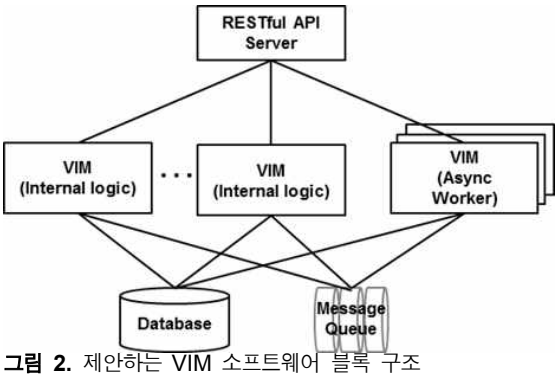

그림 2. 제안하는 VIM 소프트웨어 블록 구조
Fig. 2. Software Architecture of proposed VIM

요청을 처리하는 과정은 동기/비동기로 나뉘는데, 비동기 처리를 위한 메시지 순서도는 그림3과 같다. 기본적으로 Async Worker는 설정에 따라 다수의 스레드로 동작한다. 우선 API서버에서 외부 컴포넌트로부터 비동기 요청을 받으면, 해당 Task를 메시지큐에 넣는다. 메시지 큐를 주기적으로 polling하고 있던 Async worker는 idle상태일 경우 메시지 큐에서 task를 할당받아 요청된 작업을 수행한다. VIM의 경우 South-bound API는 클라우드 운영체제가 (e.g., 오픈스택= NFVI) 된다. 본 논문에

서 활용되는 NFVI (i.e., 오픈스택)는 대부분 비동기 API를 제공하기 때문에, task를 처리하는 API를 호출 한 후, NFVI에서 처리중인 task의 상태를 주기적으로 확인하는 방식으로 동작한다. 처리가 완료된 것을 확인하면 상위 블록으로 메시지를 전달함으로써 비동기 처리가 마무리 된다.

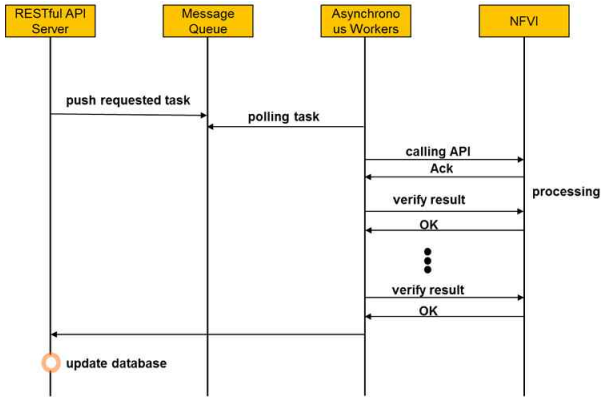

그림 3. 비동기 처리 절차의 메시지 순서도
Fig. 3. The Message sequence chart for processing asynchronous task

2-2 서비스 프로파일링을 통한 스케일링 메커니즘

마이크로 서비스는 특정 소프트웨어 블록을 스케일 아웃 함으로써 해당 서비스의 수요에 따른 성능을 충족시킨다. 그러므로 서비스 성능과 분리되어 구동하는 기능 블록 간의 관계를 파악하는 것이 매우 중요하다. 서비스의 수요 대비 성능을 늘려야 하는 시점에 어떤 기능블록을 스케일 아웃 할지는 서비스 특성에 매우 종속적이기 때문이다. 그림 2의 VIM서비스를 구성하는 다섯 가지 기능 블록 중 특정 블록의 성능저하가 전체 서비스의 성능을 저하시키는지 구분 지을 필요가 있다. 최근에는 이를 모니터링하기 위한 도구들이 존재한다. 하지만 이러한 도구들은 서비스의 성능과 직접적인 관계를 나타내기에는 한계가 있다.

실제로 구현한 MANO 시스템에서는 전체 시스템의 최 상위 블록인 UI/UX에서 네트워크 서비스를 불러오는 경우에는 하위 블록으로 내려갈수록 호출되는 API의 수가 지수 적으로 증가한다. 네트워크 서비스에 n개의 VNF와 m개의 가상링크가 있을 경우, 이를 포함하는 서비스를 로딩할 때 VIM-NFVI사이의 API호출 수는 식 1과 같다. 두 서비스 사이에는 대다수의 API가 비 동기식으로 호출되기 때문이다. 최초 요청하는 API를 호출하고, 요청된 VNF 또는 링크가 생성이 완료되는 시점까지 주기적인 API호출을 통해 상태확인을 하기 때문이다.

$$the\ \nu mber\ of\ called\ API(VIM, NFVI, n, m) = 2n^2 + 4m^2 + \alpha \quad (1)$$

그러므로 제안하는 NFV MANO 서비스는 상위에서 하위로 갈수록 API의 수가 늘어나기 때문에 단순히 호출되는 API수를 기반으로 스케일링 하기는 어렵다.

서비스 성능 저하와 가장 관련도가 높은 마이크로 서비스를 찾기 위해서는 각 블록 컴포넌트의 자원 사용률과 서비스 성능 변화량과의 유사도를 고려할 필요가 있다. 시간 기준으로 측정된 서비스 성능, 각 소프트웨어 블록의 자원 사용량을 기준으로 유사도에 기반을 두어 전체 서비스 성능과 각 마이크로 서비스의 자원 사용률과의 관계를 도출한다.

$$similarity(x_t, y_t) = \sum_{t=0}^{n} \sqrt{(-x_t' - y_t')^2} \quad (2)$$

수식 2에서 x_t'는 서비스 성능의 변화량을 의미하며, y_t'는 소프트웨어 블록의 자원 사용률을 의미한다. 수식 1의 결과가 최솟값을 나타내는 소프트웨어 블록이 스케일 아웃의 대상이 된다.

Ⅲ. 구현 및 평가

제안한 MSA형 VIM의 성능을 평가하기 위해 구축한 시험환경은 다음과 같다. 두 대의 가상머신에 Docker를 설치하고, 두 가상머신을 Docker 오케스트레이션 도구인 Swarm을 활용하여 클러스터로 연동하였다. 호스트 노드는 Intel i7-6700k@4.2GHz, 32GB DDR4 메모리를 보유하고 있고, 각 가상머신은 2코어 vCPU, 4GB 메모리의 자원을 할당하였다. 가상머신은 표 1의 소프트웨어 버전으로 시험환경을 구축하였다.

표 1. 시험환경 구축에 활용된 VM의 소프트웨어 버전
Table 1. Software version of VM for building testbed

Software	Version
OS	Ubuntu-16.04.4LTS
Kernel	Linux 4.4.0-116-generic
Docker	18.03.0-ce, build 0520e24

기존 설계된 소프트웨어는 5개의 기능 요소들(API, logic, 비동기 워커, DB, 메시지 큐)로 구성되어 있다. 시험은 DB와 연계된 동기식 API를 호출하는 방식으로 수행되었다. 비동기 API를 처리하기 위한 비동기 워커와 메시지 큐는 스케일링의 대상에서 제외하였다. API를 호출하기 위한 RESTful API Client는 12 threads를 기반으로 400 connections 을 동시에 요청한다.

그림 4은 해당 클라이언트를 기반으로 내부 로직 컨테이너 수에 따른 마이크로 서비스로 구성된 VIM의 단위 시간당 처리량을 보여준다. DB컨테이너의 수는 하나로 고정한 채 시험을 진행하였다.

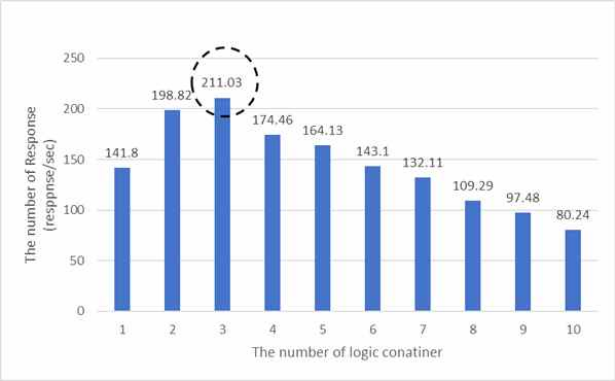

그림 4. Logic컨테이너의 스케일 아웃에 따른 응답 수
Fig. 4. The number of responses according to scale out of the logic container

내부 로직 컨테이너의 수가 1에서 3으로 증가함에 따라 처리량도 늘어난 반면, 4개 이후로는 처리량이 줄어드는 모습을 나타낸다. 이러한 현상의 원인으로는 단일 DB에 Access하려는 내부 로직의 수가 증가함에 DB Access 제어에 혼잡이 발생한 것으로 판단된다. 단일 DB컨테이너에 대해 유효한 로직 컨테이너는 3인 것을 알 수 있다. 그림 4에 원으로 표시한 시점에서 수식 3이 충족된다. 스케일링이 필요한 시점마다 내부 구성 요소들의 자원 사용량과 서비스 성능과의 유사도가 계산 되어야 한다. 이를 기반으로 스케일링의 대상을 적절히 선택할 수 있다. 서비스 성능에 따른 마이크로서비스와의 관계를 실제 실험을 통해 파악할 수도 있지만, 서비스를 운용함에 있어 시간 소모적이며, 오랜 시간 시행착오를 야기한다. 2-2장에서 제시한 유사도 값을 주기적으로 계산하여, 적절한 시점에 스케일 아웃을 수행해야 한다.

$$similarity(QoS, usage_{DB}) < similarity(QoS, usage_{Logic}) \quad (3)$$

그림 5는 3개의 DB컨테이너와 연동된 상황에서 로직 컨테이너의 증가에 따른 처리량을 나타낸다. 그림 4에서 확인해본 결과와는 다른 상태를 보여준다. Logic컨테이너가 늘어날수록 처리량이 저하되는데, 전술한 DB Access제어 문제와 더불어 언더레이 네트워크의 구조에 따른 성능변화로 판단된다. Docker 플랫폼 자체에서 로드밸런싱을 지원하지만, 해당 방법은 사용/구현 편의성을 중점적으로 설계되었기 때문에, 실제 Logic-DB간의 논리 가상링크의 경로에 대한 최적화가 미흡한 상황이다. 그림 5, 그림 6과 같이 스케일 아웃에 따른 선형적 성능 향상을 저해하는 요소가 된다.

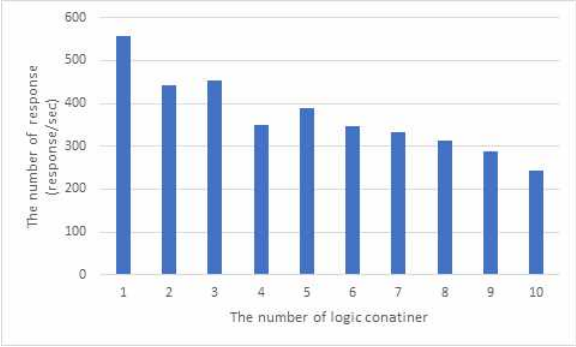

그림 5. Logic 컨테이너의 스케일 아웃에 따른 응답 수 (DB=3)
Fig. 5. The number of responses according to scale out of the logic container (DB=3)

그림 6은 DB컨테이너가 5개일 때 스케일링에 따른 성능치를 나타낸다. 그림 5와 유사한 양상을 보여준다. 이러한 현상은 자원 사용량과 성능 저하도의 유사도 외에도 언더레이 네트워크의 모니터링 정보가 필요함을 나타낸다. 컨테이너 플랫폼에 따른 가상 네트워크를 구성하는 방식과 멀티 호스트 간에 사용되는 실제적 매커니즘도 성능관 높은 연관성을 가진다. 그러나 전술한 내용은 컨테이너형 데이터 센터 운용자 관점에서 스케줄링 최적화 문제를 풀어야 할 내용이며, 사용자에게는 해당 정보를 조회할 권한은 주어지지 않는다. 특정 시점에 다중 호스트의 상태를 고정 또는 예측하기 어렵기 때문에 대다수의 스케줄링 최적화에서는 본 논문에서 제안한 바와 유사하게 휴리스틱한 접근을 통해 문제 해결을 시도한다.

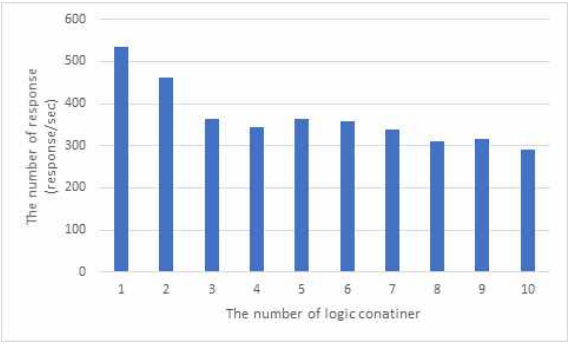

그림 6. Logic 컨테이너의 스케일 아웃에 따른 응답 수 (DB=5)
Fig. 6. The number of responses according to scale out of the logic container (DB=5)

내부 로직 컨테이너의 수를 고정한 경우, 서비스 성능은 DB 컨테이너의 수와 비례했다. 로직 컨테이너가 5개인 경우, DB컨테이너가 6개로 스케일 아웃 된 시점가지는 성능이 증가하다가 이후로는 유지 또는 감소되는 것을 확인했다. 반면 로직 컨테이너가 10개인 경우는 DB컨테이너가 10개인 경우 역시 꾸준히

성능이 향상되는 것을 확인할 수 있다.

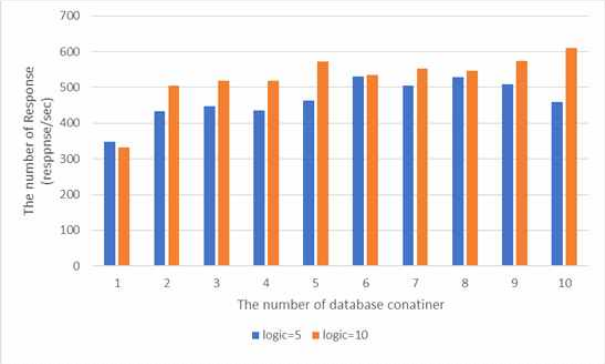

그림 7. DB컨테이너의 스케일 아웃에 따른 응답 수

Fig. 7. The number of responses according to scale out of the DB container

Ⅳ. 결 론

본 논문에서는 마이크로 서비스 구조에 기반을 둔 NFV-VIM 의 서비스 구조를 제안하였다. 제안된 구조에서 서비스 부하에 따라 효율적인 스케일링을 위한 방법을 제시했다. 서비스 성능과 컨테이너 기반 기능 소프트웨어 사이의 유사도를 비교함으로써 실제 성능과 직접적인 연관을 갖는 서비스를 구분하였고, 이를 활용하면 서비스 확장이 필요한 시점에 실제 확장이 필요한 기능 소프트웨어를 구분 지을 수 있음을 실험을 통해 검증했다.

감사의글

본 연구는 과학기술정통부의 R&D 사업의 [12221-14-1001, 차세대 네트워크 컴퓨팅 플랫폼]일환으로 수행하였습니다.

참고문헌

[1] Villamizar, Mario, et al. "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud." Computing Colombian Conference (10CCC), 2015 10th. IEEE, 2015.

[2] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239 (2014): 2.

[3] Fink, John. "Docker: a software as a service, operating system-level virtualization framework." Code4Lib Journal 25 (2014): 29.

[4] Ersue, Mehmet. "ETSI NFV management and orchestration-An overview." Proc. of 88th IETF meeting. 2013.

[5] Matias, Jon, et al. "Toward an SDN-enabled NFV architecture." IEEE Communications Magazine 53.4 (2015): 187-193.

[6] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

[7] Pautasso, C., Zimmermann, O., & Leymann, F. (2008, April). Restful web services vs. big'web services: making the right architectural decision. In Proceedings of the 17th international conference on World Wide Web (pp. 805-814). ACM.

[8] Docker container virtualization platform. Available: https://www.docker.com/

[9] Petering, Matthew EH. "Decision support for yard capacity, fleet composition, truck substitutability, and scalability issues at seaport container terminals." Transportation Research Part E: Logistics and Transportation Review 47.1 (2011): 85-103.

나태흠(TaeHeum Na)

2013년 : 충북대학교 정보통신공학 (공학사)
2015년 : 광주과학기술원 (GIST) (공학석사)

2015년～현 재: 한국전자통신연구원 (ETRI) 네트워크연구본부
※관심분야 : SDN, NFV, SDDC, Cloud, Intelligent Infra Management

박평구(PyungKoo Park)

2001년 : 고려대학교 대학원 (공학석사)
2005년 : 고려대학교 대학원 (공학박사)

2000년～현 재: 한국전자통신연구원 (ETRI) 네트워크연구본부
※관심분야 : 인터넷 보안, SDN/NFV, Netconf, 네트워크자동화

류호용(HoYong Ryu)

1995년 : 광운대학교 대학원 (공학석사)
1999년 : 광운대학교 대학원 (공학박사)

1999년～현 재: 한국전자통신연구원 (ETRI) 네트워크연구본부
※관심분야 : 인터넷 보안, 인터넷 라우팅, SDN/NFV, Netconf, 네트워크 운영체제