https://doi.org/10.7840/kics.2016.41.12.1801

오픈데이라이트 컨트롤러 클러스터 성능 분석 및 최적 운영 방안

김 태 홍[◆°], 서 동 은[*], 백 상 헌[*], 김 명 섭[*], 임 창 규[**], 박 수 명[**]

Performance Evaluation and Optimal Operation Strategy of OpenDaylight Controller Cluster

Taehong Kim[◆°], Dongeun Suh[*], Sangheon Pack[*], Myung-Sup Kim[*], Chang-Gyu Lim[**], Soomyung Park[**]

요 약

본 논문에서는 최근 많은 관심과 집중을 받고 있는 SDN 분야 오픈 소스 프레임워크인 오픈데이라이트 컨트롤러를 중심으로 컨트롤러 클러스터 구조 및 동작방식에 대하여 분석한다. 오픈데이라이트 컨트롤러 클러스터에서는 컨트롤러 간 데이터 스토어의 동기화를 위하여 분산 샤드 구조 및 샤드 리더 선정을 위한 Raft 알고리즘을 적용하고 있다. 성능 분석에서는 컨트롤러 클러스터 크기, 샤드 역할, 샤드 정책에 따른 CRUD, Routed RPC 지연시간 및 리더 재선정 지연시간 등을 분석함으로써, 오픈데이라이트 컨트롤러 클러스터 운영 시의 최적 운영 방안을 논의한다.

Key Words : OpenDaylight, SDN, Controller, Cluster, Performance

ABSTRACT

The OpenDaylight controller has been receiving significant attention as one of the enabling open source framework for SDN, and this paper analyzes the architecture and procedure of OpenDaylight based controller cluster. The OpenDaylight controller cluster uses shard based distributed datastore and Raft algorithm to maintain consistency among controllers inside a cluster. The performance evaluation analyzes the leader re-election time as well as latencies of CRUD and Routed RPC according to cluster size, shard role, and sharding strategy, and we discuss the optimal operation strategy for OpenDaylight controller cluster.

Ⅰ. 서 론

소프트웨어 정의 네트워킹 (Software Defined Networking, SDN) 기술은 하드웨어 기반의 경직된 네트워크 구조를 소프트웨어 기반의 중앙집중적인 제어 계층과 개방형 API에 기반한 프로그래머빌리티를 통하여 네트워크 인프라의 구조적 유연성과 개방성을 제공할 수 있는 기술로써, 기존 네트워크 구조의 문제

※ 본 연구결과의 일부는 한국통신학회지 (정보와 통신)에 발표되었음.[13]
※ 본 연구는 미래창조과학부 및 정보통신기술연구진흥센터의 정보통신·방송 연구개발사업 [B0101-16-0233, 스마트 네트워킹 핵심 기술 개발] 및 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임. (No. 2016R1D1A1B03933007)
◆° First and Corresponding Author : Chungbuk National University School of Information and Communication Engineering, taehongkim@cbnu.ac.kr, 정회원
* Korea University, fever1989@korea.ac.kr, 학생회원, shpack@korea.ac.kr, 종신회원, tmskim@korea.ac.kr, 종신회원
** Electronics and Telecommunications Research Institute (ETRI), human@etri.re.kr, smpahk@etri.re.kr
논문번호 : KICS2016-08-211, Received August 28, 2016; Revised November 11, 2016; Accepted November 21, 2016

점을 해결할 수 있는 차세대 네트워킹 기술로써 주목받고 있다. [1, 2] 2011년 SDN 기술에 대한 표준 제정과 연구 촉진을 목적으로 비영리단체인 오픈 네트워킹 파운데이션 (Open Networking Foundation, ONF)이 설립되었으며, 미국, 일본, 한국 등을 포함한 전 세계 산업체, 학계, 연구 기관을 중심으로 Floodlight [3], NOX/POX [4], Beacon [5], Ryu [6], IRIS [7] 등 다양한 종류의 SDN 컨트롤러 기술들이 개발되었다.

특히 2013년 이후에는 SDN 컨트롤러의 고확장성 (High Scalability)/고가용성 (High Availability) 이슈와 함께 벤더 중심으로 개발되었던 SDN 컨트롤러 기술이 오픈데이라이트 (OpenDaylight, ODL)와 오노스 (Open Network Operating System, ONOS) 오픈소스 프레임워크 중심으로 시장이 재편되고 있으며, 두 오픈소스 프레임워크 모두 분산 컨트롤러 클러스터 구조 기반의 고확장성 및 고가용성 지원을 목표로 개발이 진행 중에 있다. 고확장성이란 SDN 컨트롤러에 집중적으로 유입되는 대용량의 트래픽을 처리할 수 있는 능력을 의미하며, 고가용성이란 서버의 장애 등으로 인한 끊김 없이 시스템 또는 서비스가 지속적으로 정상 운영되는 상태를 의미한다. 즉, 컨트롤러 클러스터 구조 방식을 통하여, 논리적으로는 중앙집중적인 제어 방식의 컨트롤러라는 패러다임을 유지하면서도 물리적으로는 복 수 개의 컨트롤러를 배치함으로써 트래픽의 분산처리, 컨트롤러 장애 처리 등의 컨트롤러의 고확장성 및 고가용성을 제공할 수 있게 된다.[8] 분산 컨트롤러의 필요성과 가능성에 대한 연구는 2010년 HyperFlow[9]에서 처음 제안되었으며, 이후 Elasticon[10], ONOS[11] 등에서 분산 컨트롤러 구조에 대한 연구 및 성능 검증이 이루어졌다. 또한, 분산 컨트롤러 환경에서는 컨트롤러 간 상태 정보 불일치로 인한 네트워크 동작 오류 및 성능 저하 문제가 발생할 수 있으므로, 분산 데이터의 일관성 (Consistency) 문제는 분산 컨트롤러의 구조 및 기능 설계 시 필수적으로 고려해야 하는 중요한 이슈이다. [12]

본 논문에서는 최근 많은 관심과 집중을 받고 있는 오픈 소스 프레임워크인 오픈데이라이트를 중심으로 컨트롤러 클러스터 구조 및 컨트롤러 간 상태 정보 공유 방식을 살펴보고, 오픈데이라이트 컨트롤러 클러스터의 성능 및 최적 운영 방안에 대하여 논의한다. 본 논문의 구성은 다음과 같다. 2장에서는 오픈데이라이트 컨트롤러 개요에 대하여 살펴보고, 3장에서는 컨트롤러 클러스터 구조, 관련 프로토콜 및 동작과정에 대하여 분석한다. 4장에서는 오픈데이라이트 컨트롤러의 성능 평가 및 분석을 통한 최적 운영 방안에 대하여 논의하고, 5장의 관련연구와 함께 6장에서 결론을 맺는다.

Ⅱ. 오픈데이라이트 (OpenDaylight) 개요

오픈데이라이트는 2013년 4월 리눅스 파운데이션 (Linux Foundation)에 의해 공식 출범한 오픈소스 프레임워크이며, 오픈소스 기반의 표준 SDN 프레임워크 개발 및 어플리케이션, 툴, 서비스 등을 포함하는 SDN 생태계 전반을 구축하는 것을 목표로 한다. 시스코를 비롯하여 IBM, 브로케이드, 빅스위치네트웍스, 주니퍼네트웍스, VMWARE, 마이크로소프트 등 글로벌 IT 업체 대부분이 참여한 범 개방형 네트워크 연합체이며, 약 50여 개 회원사와 300여 명이 넘는 개발자가 오픈소스 프레임워크 개발에 참여하고 있다. 오픈데이라이트는 원소기호를 코드명으로 사용하고 있으며, 첫 번째 버전인 Hydrogen을 2014년 2월에 소개한 이후, 두 번째 버전인 Helium은 2015년 3월, 세 번째 버전인 Lithium을 2015년 6월에 발표하였으며, 2016년 3월 네 번째 버전인 Beryllium을 발표하였다.

그림 1의 오픈데이라이트 컨트롤러 플랫폼[13]은 MD-SAL (Model Driven Service Abstraction Layer)이라 불리는 동적으로 장착 가능한 모듈 형태의 구조를 갖추고 있으며, 필요에 따라 네트워킹 기능들을 모듈 형태로 추가할 수 있다. 토폴로지 관리 기능, 플로우 관리 기능, 상태 관리자, 호스트 트래커 등이 기본적으로 제공되는 모듈이며, 응용 요구사항에 따라 웹 기반 UI 접근을 위한 DLUX UI, 고확장성/고가용성을 위한 클러스터링 모듈, 오픈플로우 스위치 모듈, 클라우드와의 연동을 위한 오픈스택 서비스, 멀티테넌트 가상 네트워크를 위한 VTN (Virtual Tenant Networking) 서비스, NFV를 위한 서비스 기능 체이

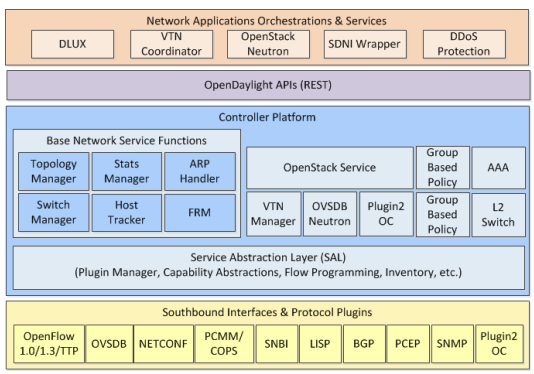

그림 1. 오픈데이라이트 컨트롤러 프레임워크 구조[13]
Fig. 1. Architecture of OpenDaylight controller framework[13]

닝 (Service Function Chaining) 서비스 등을 자유로이 확장할 수 있다.

오픈데이라이트는 응용 계층을 위해 OSGI 프레임워크 및 노스바운드 (Northbound) API를 지원하며, 응용 계층에서는 컨트롤러를 통하여 네트워크 정보를 수집/모니터링하거나 네트워크 전반에 대한 제어관리용 알고리즘 및 비즈니스 로직을 적용할 수 있다. 사우스바운드 (Southbound) 인터페이스에서는 오픈플로우 1.0/1.3, BGP-LS 등 다중 프로토콜 등을 지원한다. 또한, 각각의 네트워크 프로토콜은 서비스 추상화 계층 (Service Abstraction Layer, SAL)과 동적으로 연결되어, 컨트롤러와 네트워크 장치 사이의 통신 방식과 독립적으로 서비스 추상화 계층에서 요청 서비스에 대한 처리방법을 결정할 수 있다.

2015년 3월에 발표된 Helium 버전에서는 기존의 AD-SAL (API Driven Service Abstraction Layer) 구조 대신 MD-SAL 구조가 도입되었으며, 이 외에도 컨트롤러의 고가용성과 클러스터링, 보안 기능에 대한 개선이 이루어졌다. 또한, 2015년 6월에 발표된 Lithium 버전에서는 오픈스택의 뉴트론 (Neutron) 지원을 통한 데이터센터 네트워크 서비스, 시계열 데이터 저장소 (Time Series Data Repository, TSDR), 토폴로지 프로세싱 프레임워크 (Topology Processing Framework) 등 통신사업자 서비스에 필요한 기능들이 신규 추가되었으며, 2016년 3월에 발표된 Beryllium 버전에서는 데이터 핸들링 처리 향상, 네트워크 가상화 기능 개선 등 전반적으로 네트워크 서비스의 성능 향상 및 안정화에 중점을 맞추어 개선이 진행되었다. [14]

Ⅲ. 오픈데이라이트 컨트롤러 클러스터 구조

오픈데이라이트 컨트롤러 클러스터란 고확장성과 고가용성의 지원을 위하여 3대 이상의 컨트롤러로 구성되는 클러스터를 의미하며, MD-SAL 클러스터링 모듈에 구현되어 있다. 오픈데이라이트 클러스터링 서비스는 크게 분산 데이터 스토어 (Distributed Data Store)와 원격 RPC (Remote RPC) 서비스로 구성되어 있으며, 분산 데이터 스토어에서는 클러스터 내 모든 컨트롤러에서 사용자의 요청을 처리할 수 있도록 데이터 스토어의 동기화와 분산 처리 기법을 제공한다. 원격 RPC 컨넥터에서는 개별 컨트롤러에 구현되어 있는 RPC 서비스를 다른 컨트롤러에서 탐지하고 사용자 요청에 따라 원격으로 실행하기 위한 메커니즘을 제공한다. 오픈데이라이트의 데이터 스토어 및 원격 RPC 컨넥터 서비스는 오픈소스 기반의 분산 처리 프레임워크인 Akka 라이브러리 [15] 환경에서 구현되어 있으며, Akka Persistence, Akka Remoting, Akka Clustering 라이브러리를 이용한다.

Akka 라이브러리는 타입세이프 (Typesafe)에서 개발된 액터 모델 (Actor model) 기반의 플랫폼으로써 대규모 분산 시스템에서의 확장성과 고장 감내 기능을 제공한다. 액터 모델이란 기존의 분산/병렬 시스템을 위하여 개발된 모델로써, 액터는 자신만의 메시지 큐를 가지고 다른 액터와 메시지를 주고 받거나 해당 메시지에 대응하는 동작을 수행할 수 있는 객체이다. 액터 간의 통신은 고속처리를 위하여 비동기 방식의 동시처리 (Concurrent processing) 및 병렬처리 (Parallel processing) 방식을 지원하며, 기존 방식과 달리 데드락이나 기아상태 (Starvation) 등 분산 환경에서 발생하는 문제에도 자유로울 수 있도록 설계되었다. 또한, 하나의 액터 객체가 처리할 수 있는 데이터의 단위가 클 경우에는 재귀적인 방식으로 작은 단위의 데이터를 갖는 액터로 분할함으로써 데이터 처리 성능을 향상시킬 수 있다.

오픈데이라이트의 분산 데이터 스토어는 샤드 (Shard)라 불리는 Akka 라이브러리의 액터 객체 형태로 구현된다. 즉, 샤드는 하나의 트리 형태로 구현된 데이터 저장소이며, 동시에 다른 컨트롤러 내 샤드와 메시지 교환을 통하여 데이터의 복제 및 분산 처리를 수행할 수 있는 객체이다. 오픈데이라이트에서 기본적으로 제공되는 디폴트 (default) 샤드는 하위 서브 트리로 토폴로지 (topology), 인벤토리 (inventory), 토스터 (toaster) 등의 샤드 등을 가지고 있으며, 사용자 설정에 따라 단일 샤드 또는 사용자가 지정한 샤드들의 분산 관리 방식을 선택할 수 있다. 즉, 사용자가 디폴트 샤드만을 선택할 경우 모든 데이터 스토어가 단일 샤드를 통해 관리되는 반면, 분산 샤드 방식을 선택할 경우 개별 샤드 단위의 분산된 형태로 복제/관리할 수 있게 된다. 또한, 사용자는 설정을 통해 각각의 샤드를 복제하는 컨트롤러의 리스트를 지정할 수 있으며, 복제하는 컨트롤러 수가 증가할수록 컨트롤러 장애 발생 시의 복구 가능성이 높아지는 반면 데이터 스토어의 동기화로 인한 지연시간은 증가하게 된다.

오픈데이라이트에서는 분산 데이터의 동기화 및 일관성 유지를 위하여 엄격한 일관성 (Strong consistency) 정책을 적용하고 있으며, 각 샤드별 읽기 및 쓰기 등의 접근을 샤드 리더 (Leader)만을 통하도록 제한하고 있다. 예를 들어, 사용자 요청이 해당 샤드의 리더로 선정된 컨트롤러에게 전달될 경우, 컨트

롤러는 자신의 샤드를 변경한 후 샤드 트랜잭션을 통하여 나머지 팔로어 (Follower)로 선정된 컨트롤러들에게 동기화를 수행하게 된다. 만약, 사용자의 요청이 해당 샤드의 팔로어로 선정된 컨트롤러에게 전달될 경우, 컨트롤러는 해당 샤드의 리더로 선정된 컨트롤러에게 샤드 변경 요청을 전달하고 그 결과만을 사용자에게 응답하게 된다. 즉, 모든 샤드의 삽입/삭제/수정 등의 작업은 샤드의 리더에게로 전달되어 동기화가 이루어지게 된다. 이 때, 각 샤드 리더의 선정은 미국 스탠포드대에서 개발한 Raft 알고리즘[16]을 이용한다. Raft 알고리즘에서는 Follower, Candidate, Leader 등의 상태를 정의하고 있으며, 타이머 만료 및 투표 (Voting) 절차에 따라 분산 방식으로 하나의 리더를 선출하게 된다. 또한, 팔로어는 리더에게 주기적으로 하트비트 (Heartbeat) 메시지를 전송하여 리더의 상태를 확인하며, 만약 리더로 선정된 서버에 장애가 발생할 경우 나머지 팔로어들 중에서 새로운 리더를 선정할 수 있게 된다. [17]

오픈데이라이트의 MD-SAL에서는 RPC를 제공하는 플러그인 또는 서비스를 제공자 (Provider), 반대로 RPC를 호출하는 플러그인 또는 서비스를 사용자 (Consumer)로 명명하며, 각 제공자의 RPC를 RPC Registry에 등록하고 관리한다. 컨트롤러 클러스터 환경에서의 원격 RPC 서비스는 클러스터 내 하나의 컨트롤러가 다른 컨트롤러에서 제공하는 RPC를 호출하고자 할 때 사용되며, 이를 위해 RPC Registry와 RPC Broker 기능이 구현되어 있다. RPC Registry는 클러스터 내 개별 컨트롤러가 제공하는 RPC 리스트를 <Address, Bucket>의 형태로 관리하며, 이 때 주소 (Address)란 개별 컨트롤러의 주소를 의미하며, 버킷 (Bucket) 내부에는 RPC 버전 관리를 위한 시간 정보 (Timestamp)와 RPC 리스트가 저장되어 있다. 클러스터 내 컨트롤러들은 RPC 리스트를 교환하고 RPC Registry를 최신 버전으로 유지하기 위해 Gossip 프로토콜을 이용한다. 즉, 각 컨트롤러는 자신의 버킷 정보를 GossipStatus 메시지에 담아 전송하며, GossipStatus 메시지를 수신한 컨트롤러는 버킷 내 버전 정보를 바탕으로 자신의 버전 정보가 높을 경우 GossipEnvelope 메시지에 자신의 버킷정보를 담아 재전송함으로써 상대방의 RPC Registry를 업데이트하도록 한다. 만약, GossipStatus 메시지의 버킷 내 버전 정보가 자신의 버전 정보보다 낮을 경우에는 상대방에게 자신의 GossipStatus 메시지를 전송함으로써 상대방이 GossipEnvelope 메시지로 응답할 수 있도록 요청한다. 이에 따라 클러스터 내 각 컨트롤러는 자신이 제공하고 있는 RPC 리스트를 교환하고 자신의 RPC Registry에 유지할 수 있으며, RPC 서비스의 탐색 및 호출에 활용하게 된다. 만약 필요한 RPC가 클러스터 내 다른 컨트롤러에서 제공할 경우, 해당 컨트롤러의 RPC Broker에 RPC 실행 요청이 전달되며, RPC Broker는 내부의 RPC 제공자에게 전달함으로써 원격 RPC 서비스를 요청하게 된다.

Ⅳ. 오픈데이라이트 컨트롤러 클러스터 성능 분석

본 장에서는 오픈데이라이트 컨트롤러 클러스터를 구축하고 다양한 측면에서의 컨트롤러 클러스터의 성능을 분석한다. 먼저, 컨트롤러 클러스터 환경에서 리더/팔로어 등 샤드 역할에 따른 CRUD 성능을 비교, 분석함으로써, 컨트롤러 클러스터의 동작 방식에 대하여 논의한다. 또한, 컨트롤러 클러스터 크기 및 단일 샤드/분산 샤드 등의 샤드 운영 정책에 따른 CRUD 성능, 컨트롤러 클러스터 환경에서의 Routed RPC 성능과 Raft 알고리즘의 설정 값에 따른 리더 재선정 지연시간을 비교함으로써, 컨트롤러 클러스터 운영 시의 최적 운영 방안에 대하여 논의한다.

4.1 컨트롤러 클러스터 시험 환경 설정 및 성능 분석 방안

오픈데이라이트 컨트롤러 클러스터 구축을 위하여 오라클 버추얼박스 (Virtualbox) 가상머신 (Virtual Machine, VM)을 사용하였으며, Intel i5 코어 및 32GB RAM을 갖춘 호스트 PC에서 컨트롤러 클러스터 운용을 위한 가상머신들을 구동한다. 오픈데이라이트 컨트롤러용 가상 머신은 Ubuntu 14.04.2 LTS에서 동작하며 2 CPU 코어와 4GB RAM에서 동작하도록 설정하였으며, CRUD/Routed RPC 요청 및 성능분석을 위한 가상머신은 Ubuntu 14.04.2 LTS 및 2 CPU 코어와 6GM RAM에서 동작하도록 설정하였다. 가상머신 간의 네트워크 인터페이스는 Intel PRO/1000를 사용하며, 각각의 가상머신에 192.168.56.X 대역의 사설 IP를 할당하여 가상머신 간 내부 네트워킹을 설정하였다.

오픈데이라이트 컨트롤러는 웹사이트 https://www.opendaylight.org/downloads에서 다운받을 수 있으며, 클러스터로 운영하기 위해서 오픈데이라이트 매뉴얼[22]에 따라 클러스터링 관련 모듈을 설치하고 환경설정 파일 (akka.conf, modules.conf, module-shard.conf)에서 클러스터 정보를 입력한다. 본 논문에서는 CRUD (Create, Read, Update, Delete)의 성능

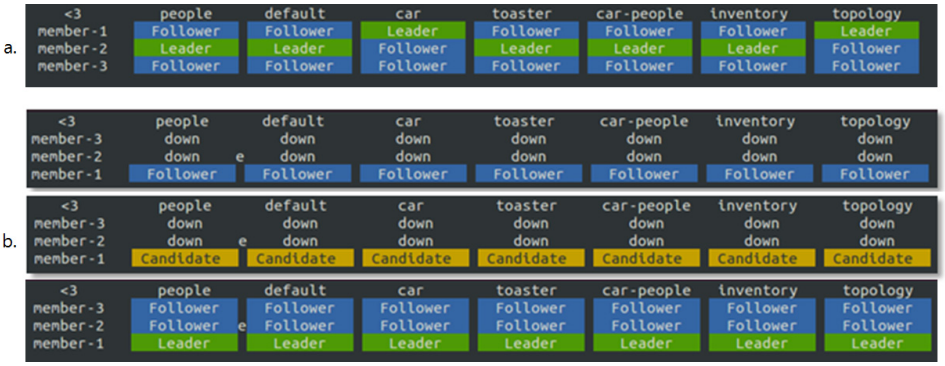

그림 2. 분산 샤드 환경에서 샤드 리더 선정 예 (a. 샤드 리더가 클러스터 내 컨트롤러들로 분산되는 경우, b. 하나의 컨트롤러가 모든 샤드의 리더로 선정되는 경우)
Fig. 2. Example of shard leader election in distributed shard strategy (a. shard leaders are distributed to controllers in the cluster, b. all shard leaders are assigned to one controller)

평가 테스트자동화 모듈을 활용하였으며, 이를 위하여 데이터 스토어는 default, topology, inventory, toaster 등의 샤드 이 외에 car, people, car-people 샤드를 추가하였다. 또한, CRUD 및 Routed RPC 성능 분석 툴은 오픈 소스 사이트 (https://github.com/opendaylight/integra tion/tools)의 CrudLibrary.py를 활용하였다.

그림 2는 3대의 오픈데이라이트 컨트롤러를 이용하여 분산 샤드 기반의 컨트롤러를 구성했을 때의 예이며, 그림 2. a에서는 각 샤드 별로 리더가 독립적으로 선정되는 것을 확인할 수 있다. 즉, 각 샤드 별 리더의 설정이 독립적으로 동작하므로, 그림 2.a처럼 각 샤드별 리더가 클러스터의 컨트롤러들에게 분산되어 설정될 수 있다. 만약, 컨트롤러를 순차적으로 동작시킬 경우에는 그림 2. b에서 보는 것처럼 첫 번째 전원이 인가되는 컨트롤러가 모든 샤드의 리더로 선정될 수 있다.

한편, 클러스터 환경 설정에서 데이터 스토어를 default 샤드로만 설정할 경우에는, topology, inventory, car, people, car-people 등의 샤드가 default 샤드에 포함되어 단일 샤드로 동작하게 된다. 이 때, default 샤드의 리더로 선정된 컨트롤러가 단일 샤드 전체의 리더로 동작하게 된다. 또한, 설정에서는 각 샤드별 복제/유지하는 컨트롤러 리스트를 지정할 수 있으며, 본 실험에서는 오픈데이라이트의 기본 설정과 동일하게 클러스터 내 모든 컨트롤러가 모든 샤드를 복제/유지하도록 설정하였다.

표 1에서는 본 논문에서 수행하는 시험항목별 시험방법과 비교조건을 요약하였으며, 샤드 역할, 클러스터 크기, 샤드 운영 정책 등에 따른 CRUD 성능과 Routed RPC 및 리더 재선정 지연시간 등의 성능을

표 1. 오픈데이라이트 컨트롤러 클러스터 성능분석 방법 요약
Table 1. Summary of performance evaluation methods of OpenDaylight controller cluster

No	시험항목	시험방법	비교조건 요약
1	샤드 역할에 따른 CRUD 성능 분석	Northbound에서 CRUD를 주기적으로 요청하면서 지연시간을 측정	- 단일 컨트롤러 환경 - 컨트롤러 클러스터/단일 샤드 환경에서 샤드 리더/팔로어 상태에 따른 CRUD 지연시간
2	컨트롤러 클러스터 크기 및 샤드 운영 정책에 따른 CRUD 성능 분석	Northbound에서 CRUD를 주기적으로 요청하면서 지연시간을 측정	- 컨트롤러 클러스터 3대/5대 환경에서의 CRUD 지연시간 - 샤드 운영 정책 (단일 샤드/분산 샤드)에 따른 CRUD 지연시간
3	컨트롤러 클러스터 환경에서의 Routed RPC 성능 분석	Northbound에서 Routed RPC를 주기적으로 요청하면서 지연시간을 측정	- 컨트롤러 클러스터 3대 환경에서 단일샤드/분산샤드 및 샤드 리더/팔로어 상태에 따른 CRUD 지연시간
4	리더 재선정 지연시간 측정	Raft 알고리즘의 타임아웃 설정을 변경함으로써, 컨트롤러 장애/삭제에 따른 지연시간 측정	- 컨트롤러 클러스터/단일 샤드 환경에서 Raft 타임아웃 설정별 리더 재선정 지연시간

분석한다.

4.2 성능 분석 결과 및 최적 운영 방안

4.2.1 샤드 역할에 따른 CRUD 성능 분석

본 실험에서는 단일 컨트롤러 환경과 컨트롤러 클러스터 환경에서 샤드 역할에 따른 CRUD 요청에 대한 성능을 비교 분석한다. 컨트롤러 클러스터 환경에서는 샤드 전체가 하나로 관리되는 단일 샤드 환경으로 설정하였으며, 실험에서는 car 샤드의 car entry에 대한 CRUD 요청을 500회씩 5번 수행하여 각 요청당 최소/평균/최대 지연시간 (msec/request)을 도출하였다.

표 2는 단일컨트롤러 및 컨트롤러 클러스터의 CRUD 실험 결과이며, 단일 컨트롤러 대비 컨트롤러 클러스터 환경에서 CRUD 명령 처리에 따른 지연시간이 높게 나타남을 확인할 수 있다. 이는 컨트롤러 클러스터의 고가용성을 제공하기 위하여 분산 데이터 스토어의 일관성을 유지하기 때문이며, 샤드 리더의 데이터에 추가/삭제/변경 등이 발생할 경우 클러스터 내 다른 컨트롤러들로의 복제 및 검증과정을 수반하게 된다. 즉, CRUD 요청에 대해, 단일 컨트롤러 환경에서는 하나의 컨트롤러 내 데이터 스토어만 변경하면 되는 반면, 컨트롤러 클러스터 환경에서는 클러스터 내 데이터 스토어의 변경사항 복제 및 데이터 스토어 일관성 검증을 위한 제어 메시지 교환이 추가로 필요하며, 제어 메시지의 교환과정에서의 지연시간이 데이터 처리 지연시간 증가로 나타나게 된다. 이는 표 3의 컨트롤러 크기에 따른 CRUD 성능 분석 결과와 일치한다. 현재 오픈데이라이트 커뮤니티에서는 각 컨트롤러별로 관리하는 네트워크의 분할 (Partitioning)을 통한 성능 개선을 고려하고 있다. [18]

표 2. 샤드 역할에 따른 CRUD 지연시간 (최소/평균/최대) 분석 (Lx: x 샤드의 리더, Fx: x 샤드의 팔로어)
Table 2. CRUD Latency (min/avg/max) according to shard role (Lx: Leader of shard x, Fx: Follower of shard x)

	평균 요청 당 지연시간 (msec/req)		
구성	단일 컨트롤러	컨트롤러 클러스터 /단일(default) 샤드	
CRUD 타겟 (항목)	N/A (car-entry)	L_{car} (car-entry)	F_{car} (car-entry)
Create	4.11/5.12/6.88	8.18/12.6/17.11	13.58/16.7/21.28
Read	1.42/1.56/1.9	1.26/1.48/1.78	4.21/4.65/5.46
Update	4.2/4.66/5.73	9.31/12.43/17.33	10.60/15.5/25.35
Delete	2.85/3.26/4.01	6.94/8.98/12.84	7.99/10.21/11.39

또한, 컨트롤러 클러스터 환경에서는 CRUD 수행 시 팔로어에게 요청할 때보다 리더에게 요청할 때 지연시간이 작게 소요됨을 확인할 수 있다. 이는 오픈데이라이트의 분산 데이터의 일관성을 유지하는 데에 강한 일관성 (Strong consistency) 정책을 적용하기 때문이며, 강한 일관성 정책에서는 모든 데이터의 접근 및 수정이 샤드의 리더를 통하여 수행하도록 설계되어 있다. 즉, 사용자의 요청이 리더에게 전달될 경우, 리더는 데이터 스토어의 변경사항을 나머지 팔로어들에게 복제하고 과반 이상의 팔로어들에게 응답받은 후에 데이터의 변경사항을 최종적으로 반영 (commit)하게 된다. 데이터의 변경이 수반되지 않는 읽기 (Read) 명령의 경우에는 데이터 스토어의 복제 및 검증 작업이 생략된다. 만약, 사용자의 요청이 팔로어에게 전달되는 경우, 팔로어는 CreateTransaction을 통하여 리더에게 수신한 명령을 전달함으로써 리더가 데이터의 접근 또는 수정 명령을 실행하도록 하고, 샤드 리더의 수행 결과를 전달받아 다시 사용자에게 응답하게 된다. 따라서 CRUD 명령이 팔로어를 통해 전달되는 경우에는 팔로어가 리더에게 해당 요청을 전달하고 결과를 다시 전달받는 과정에서 추가적인 지연시간이 발생하게 된다. 이에 따라 실제 운영 시 특정 샤드의 데이터에 자주 접근해야 하는 경우, 해당 샤드의 리더를 확인한 후에 리더를 통해 사용자의 CRUD 요청을 전달하는 것이 지연시간을 줄일 수 있는 방법이다.

4.2.2 클러스터 크기 및 샤드 운영 정책에 따른 CRUD 성능 분석

본 실험에서는 컨트롤러 클러스터의 크기 및 샤드 운영 조건에 따른 CRUD 성능을 비교/분석한다. 먼저 컨트롤러 클러스터의 크기에 따른 성능을 비교하기 위하여 각각 3대, 5대의 클러스터를 구성하며, 샤드 운영 조건에 따른 성능 비교를 위하여 샤드 전체가 하나로 관리되는 단일 샤드 환경과 topology, inventory, car, people, car-people 등의 작은 단위의 샤드가 분산되어 관리되는 분산 샤드 환경에서 분석을 수행한다. 실험에서는 car 샤드의 car entry에 대한 CRUD 요청을 500회씩 5번 수행하여 각 요청당 최소/평균/최대 지연시간 (msec/ request)을 도출하였다.

먼저, 표 3의 컨트롤러 클러스터 크기에 따른 실험 결과를 비교해보면, 클러스터 내 컨트롤러의 수가 증가함에 따라 삽입 (Create)/수정 (Update)/삭제 (Delete) 요청의 지연시간이 증가하는 것을 확인할 수 있다. 한편, 데이터의 변화가 없는 읽기 (Read)요청의

표 3. 클러스터 크기 및 샤드 운영 정책에 따른 CRUD 지연시간 (최소/평균/최대) 분석 (Lx: x 샤드의 리더, Fx: x 샤드의 팔로어)
Table 3. CRUD Latency (min/avg/max) according to cluster size and shard strategy (Lx: Leader of shard x, Fx: Follower of shard x)

	평균 요청 당 지연시간 (msec/req)					
구성	컨트롤러 클러스터 3대/단일 샤드		컨트롤러 클러스터 3대/분산 샤드		컨트롤러 클러스터 5대/분산 샤드	
CRUD 타겟 (CRUD 항목)	L_{car} (car-entry)	F_{car} (car-entry)	L_{car} (car-entry)	F_{car} (car-entry)	L_{car} (car-entry)	F_{car} (car-entry)
Create	8.18/12.6/17.11	13.58/16.73/21.28	7.59/8.85/10.99	11.4/13.21/16.32	11.93/13.67/59.57	14.17/16.02/19.47
Read	1.26/1.48/1.78	4.21/4.65/5.46	1.23/1/42/1.64	3.88/4.38/5.26	1.38/1.48/4.67	4.15/4.81/5.33
Update	9.31/12.43/17.33	10.60/15.52/25.35	7.52/8.53/10.25	10.06/10.72/11.67	11.99/12.25/69.68	12.61/17.24/24.20
Delete	6.94/8.98/12.84	7.99/10.21/11.39	7.02/7.95/9.77	7.52/8.36/11.19	11.63/12.37/55.01	11.99/14.71/15.01

경우 클러스터의 크기가 증가하여도 지연시간에 영향을 끼치지 않는 것을 확인할 수 있다. 이는 샤드 리더가 데이터의 변경사항을 팔로어들의 샤드에 동기화하기 위한 정책 때문이며, 샤드 리더는 샤드 팔로어들에게 데이터의 변경사항을 복제 (Replication)하고 클러스터에 속한 과반수 이상의 팔로어들에게 동의를 받은 후에야 데이터의 변경사항을 최종적으로 반영(Commit)하게 된다. 따라서 컨트롤러 클러스터의 크기가 커질수록 동의를 받아야 하는 팔로어들의 수가 증가하게 되며, 이에 따라 데이터 변경 요청에 대한 지연시간이 증가하게 된다. 반면, 컨트롤러 클러스터의 크기가 증가할수록 컨트롤러의 장애/삭제에 따른 고가용성은 증가하게 된다. 예를 들어, 3대의 클러스터 환경에서는 최대 1대까지의 컨트롤러 장애/삭제를 허용하는 반면, 5대의 클러스터 환경에서는 최대 2대까지의 장애/삭제를 허용할 수 있다. 따라서, 삽입/수정/삭제 등 데이터 변경의 요청 빈도가 높은 응용의 경우, 고가용성 요구사항 및 클러스터 크기에 따른 데이터 동기화 오버헤드를 고려하여 최적의 클러스터 크기를 선정하는 것이 중요하다.

또한, 컨트롤러 클러스터 3대 환경에서 단일 샤드와 분산 샤드 환경의 비교 시에는 샤드를 여러 개로 분산하여 구성할 때에 CRUD 지연시간이 작게 나타남을 확인할 수 있다. 이는 샤드 리더의 데이터 변경이 발생할 때마다 수행되는 샤드 복제 및 검증 작업에 소요되는 시간이 샤드의 크기에 비례하기 때문이며, 이와 같은 이유로 데이터의 복제 및 검증 작업을 수반하지 않는 읽기 명령의 경우 단일 샤드와 분산 샤드 환경에서 차이가 나타나지 않는다. 따라서 특정 샤드 상에서 데이터의 추가/삭제/수정이 빈번이 발생하는 응용 환경에서는 분산 샤드 환경으로 설정함으로써 샤드 복제 및 동기화의 단위를 제한하는 것이 성능을 개선할 수 있는 방안이라 할 수 있다.

4.2.3 컨트롤러 클러스터 환경에서의 Routed RPC 성능 분석

본 실험에서는 3대의 컨트롤러 클러스터 환경에서 단일 샤드/분산 샤드 조건별 RPC 요청에 대한 서비스 지연시간 (msec/request)을 측정하였으며, 실험에서는 people 샤드의 add-person RPC 요청을 500회씩 5번 수행하여 각 요청당 평균 지연시간을 측정하였다.

표 4는 컨트롤러 클러스터 환경에서의 Routed RPC 지연시간의 실험결과를 나타내며, 두 가지 샤드 운영 정책 모두 리더로의 RPC 요청에 대한 지연시간이 팔로어로의 RPC 요청에 대한 지연시간보다 낮을 것을 확인할 수 있다. 이는 CRUD 성능 분석 실험과 유사하게, 팔로어에게 RPC를 요청하는 경우 리더를 통해 해당 요청을 수행한 뒤 결과를 다시 전달받는 과정에서 추가적인 지연 시간이 발생하기 때문이다. 따라서 특정 RPC에 대해 빈번한 요청이 필요한 컨트롤러 내부 모듈의 경우, 해당 RPC를 포함하는 샤드의 리더와 같은 컨트롤러 상에 위치시키는 것이 지연시간을 줄이는 데에 효과적일 수 있다. 또한, 샤드 운영 정책에 따른 RPC 지연시간에는 큰 차이가 나타나지 않았으며, 분산 샤드를 운영하는 경우에도 Routed RPC 실행에 따른 성능 저하가 발생하지 않는다는 점을 확인할 수 있다.

표 4. 컨트롤러 클러스터 환경에서의 Routed RPC 성능 분석 (Lx: x 샤드의 리더, Fx: x 샤드의 팔로어)
Table 4. Latency of Routed RPC in the controller cluster (Lx: Leader of shard x, Fx: Follower of shard x)

	평균 요청 당 지연시간 (msec/req)			
구성	컨트롤러 클러스터 3대/단일 샤드		컨트롤러 클러스터 3대/분산 샤드	
RPC 타겟	$L_{default}$	$F_{default}$	L_{people}	F_{people}
add-person RPC	10.06	11.39	9.96	11.84

4.2.4 리더 재선정 지연시간 측정

오픈데이라이트 컨트롤러 클러스터의 샤드 리더에 선정되는 Raft 알고리즘에서는 팔로어가 주기적으로 리더에게 하트비트 메시지를 보내 리더의 상태를 파악하게 된다. 만약, Election Timeout (리더 선정 만료 시간) 동안 리더로부터 응답을 받지 못할 경우 리더에 장애가 발생하였다고 판단하고, 자신이 후보자가 되어 나머지 팔로어들에게 투표 (Vote)를 요청하여 과반 이상의 투표를 받을 경우 리더로 재선정되는 절차를 거친다. 만약, 리더 재선정 과정에 걸리는 시간이 Election Timeout보다 클 경우, 현재 단계에서의 리더 재선정 과정을 포기하고 새로운 리더 선출 과정을 거치게 된다. 따라서 Election Timeout 값은 리더 재선정 지연시간에 영향을 미치는 파라미터이며, 본 실험에서는 Election Timeout 값에 따른 리더 재선정 지연시간을 측정하였다. 이 때, 리더 재선정 지연시간은 기존에 리더로 동작하는 컨트롤러를 다운시킨 시점부터 새로운 리더가 재선정 될 때까지의 시간으로 정의하였다.

표 5에서는 Election Timeout에 따른 리더 재선정 지연시간의 최소/평균/최대값을 분석하였다. Election Timeout 값은 오픈데이라이트 컨트롤러에서 기본으로 설정되어 있는 10초 및 5초, 1초, 0.1초로 설정하여 수행하였다. Election Timeout 값이 10초, 5초, 1초일 경우에는 각각 리더 재선정 지연시간은 평균 10.78초, 5.28초, 1.18초로 Election Timeout 값과 비례함을 확인할 수 있다.

한편, Election Timeout 값을 0.1초로 설정할 경우 리더 재선정 지연시간이 3.8초로 증가하였음을 확인할 수 있다. 이는 후보자가 팔로어들로부터 표를 얻는데 걸리는 지연 시간이 Election Timeout 값을 초과하는 경우가 빈번하게 발생하여, 새로운 리더 재선출 과정에 반복되기 때문이다. 따라서 리더 재선정 시간을 최소화하여 컨트롤러 클러스터의 가용성을 높이기 위해서는 Election Timeout 값을 컨트롤러간의 통신 지연시간을 고려하여 최적값을 설정하는 것이 중요하다.

표 5. 리더 재선정 지연시간 (최소/평균/최대)
Table 5. Leader re-election latency (min/avg/max)

Shard 구성	분산컨트롤러/단일(default) 샤드			
Election Timeout (s)	10	5	1	0.1
리더 재선정 시간 (s)	10.04/ 10.78/ 31.52	4.85/ 5.28/ 5.80	0.63/ 1.18/ 1.76	1.19 /3.80 /6.24

Ⅴ. 관련연구

현재까지 발표된 오픈데이라이트 컨트롤러의 성능 분석에 관한 연구 결과는 많지 않다. Zuhran 등의 연구[19]에서는 단일 오픈데이라이트 컨트롤러를 이용하여 플로우 처리량 및 지연시간 등을 분석하였으며, 기존의 단일 컨트롤러[3-7]의 성능 분석방법과 동일하게 오픈플로우 패킷 생성 툴인 Cbench [20]를 이용하였다. 논문[21] 에서는 오픈데이라이트 컨트롤러 클러스터를 구축하고, 클러스터 내 컨트롤러의 수에 따른 패킷 처리량 (Throughput) 및 컨트롤러 장애에 따른 복구 지연시간 (Recovery time) 등 고확장성 및 고가용성에 초점을 맞추어 성능을 분석하였다. 지금까지의 연구들과는 달리, 본 논문에서는 컨트롤러 클러스터 환경에서의 분산 데이터 스토어 구조에 초점을 맞추어 성능을 분석하였으며, 리더/팔로어 등의 샤드 역할, 컨트롤러 클러스터 크기, 단일 샤드/분산 샤드 등의 샤드 운영 정책, 샤드 리더 재선정 지연시간 등에 대한 성능 분석을 수행하였다. 본 논문에서의 컨트롤러 클러스터 구축 및 성능 분석 환경은 오픈데이라이트 매뉴얼[22] 및 오픈데이라이트 오픈소스[23]를 활용하였으므로, 관련문헌을 참고하여 오픈데이라이트 컨트롤러 클러스터 테스트베드 및 성능 분석 환경을 재연할 수 있다.

Ⅵ. 결 론

본 논문에서는 SDN 분야의 대표적인 오픈 소스 프레임워크인 오픈데이라이트 컨트롤러 클러스터의 구조 및 동작 방법에 대하여 분석하였으며, 컨트롤러 클러스터의 성능 분석 결과를 바탕으로 컨트롤러 클러스터의 최적 운영 방안에 대하여 논의하였다. 성능 분석 결과에서 살펴본 것처럼 컨트롤러 클러스터의 크기가 증가할수록 클러스터의 고가용성은 높아지는 반면, 컨트롤러 간 데이터 스토어의 일관성을 유지해야 하는 제약사항으로 인하여 데이터 스토어의 변경 시에 지연시간이 증가하는 것을 확인하였다. 특히, 데이터 스토어의 강한 일관성 유지 정책으로 인하여 사용자의 CRUD 및 Routed RPC 요청은 샤드 리더로 선정된 컨트롤러에게 요청하는 것이 효율성을 높이는 방안이라는 점을 확인하였으며, Raft 알고리즘의 Election Timeout 값 설정에 따라 샤드 리더 재선정 시간을 조정함으로써 컨트롤러의 고가용성 수준을 결정할 수 있다는 점을 분석하였다. 본 논문에서 도출한 오픈데이라이트 컨트롤러 클러스터 성능 분석 및 최적 운영 방안이 오픈데이라이트 기반의 다양한 응용

환경에서 활용될 수 있기를 기대한다.

References

[1] G. Lee, I. Jang, W. Kim, S. Joo, M. Kim, S. Pack, and C. Kang, “SDN-Based middlebox management framework in integrated wired and wireless networks,” *J. KICS*, vol. 39B, no. 6, pp. 379-386, Jun. 2014.

[2] J. Jo, S. Lee, J. Kong, and J. Kim, “A centralized network policy controller for SDN-Based service overlay networking,” *J. KICS*, vol. 38, no. 4, pp. 266-278, Apr. 2013.

[3] *Floodlight Project*, Retrieved Aug. 26, 2016, from www.projectfloodlight.org/floodlight

[4] *NOX OpenFlow controller*, Retrieved Aug. 26, 2016, from www.noxrepo.org

[5] *Beacon controller*, Retrieved Sept. 30, 2014, from https://openflow.standford.edu/display/Beacon/Home

[6] *Ryu controller*, Retrieved Aug. 26, 2016, from osrg.github.io/ryu

[7] *IRIS: The Recursive SDN OpenFlow Controller by ETRI*, Retrieved Aug. 26, 2016, from openiris.etri.re.kr

[8] Y. Kyung, K. Hong, S. Park, and J. Park, “Load distribution method over multiple controllers in SDN,” *J. KICS*, vol. 40, no. 06, pp. 1114-1116, Jun. 2015.

[9] A. Tootoonchian and Y. Ganjali, “HyperFlow: A distributed control plane for openflow,” *INM/WREN’10*, San Jose, CA, Apr. 2010.

[10] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, “Towards an elastic distributed SDN controller,” *HotSDN’13*, pp. 7-12, Hong Kong, China, Aug. 2013.

[11] P. Berde, et al., “ONOS: towards an open, distributed SDN OS,” *HotSDN’14*, pp. 1-6, Chicago, Illinois, Aug. 2014.

[12] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized? State distribution trade-offs in software defined networks,” *HotSDN’12*, pp. 1-6, Helsinki, Finland, Aug. 2012.

[13] B. Yoon, T. Kim, Y. Kim, and S. Yang, “Technology trends on on transport SDN for high scalability and high availability,” *KICS Inf. & Commun. Mag.*, vol. 32, no. 7, pp. 9-16, Jun. 2015.

[14] *OpenDaylight project*, Retrieved Aug. 26, 2016, from www.opendaylight.org

[15] *Akka*, Retrieved Aug. 26, 2016, from www.akka.io

[16] *Raft algorithm*, Retrieved Aug. 26, 2016, from https://raftconsensus.github.io

[17] Moiz Raja, odl-mdsal-clustering, *OpenDaylight Silicon Valley Meetup*, Apr. 2015.

[18] C. Dixon, “Consistency Trade-offs for SDN Controllers,” *OpenDaylight Summit*, Santa Clara, USA, Feb. 2014.

[19] Z. Khattak, M. Awais, and A. Iqbal, “Performance evaluation of OpenDaylight SDN controller,” *ICPADS 2014*, Hsinchu, Taiwan, Dec. 2014.

[20] *Cbench: an OpenFlow Controller Benchmarker*, Retrieved Nov. 10, 2016, from http://www.openflow.org/wk/index.php/Oflops

[21] D. Suh, S. Jang, S. Han, S. Pack, T. Kim, and J. Kwak, “On performance of OpenDaylight clustering,” *NetSoft 2016*, Seoul, Korea, Jun. 2016.

[22] *Setting Up Clustering*, Retrieved Nov. 10, 2016, from http://docs.opendaylight.org/en/stable-boron/getting-started-guide/common-features/clustering.html

[23] *OpenDaylight Controller/Clustering*, Retrieved Nov. 10, 2016, from https://wiki.opendaylight.org/view/OpenDaylight_Controller/Clustering

김 태 홍 (Taehong Kim)

2005년 : 아주대학교 정보및컴퓨터공학부 학사
2007년 : KAIST 정보통신공학 석사
2012년 : KAIST 전산학 박사
2012년~2014년 : 삼성전자 책임연구원
2014년~2016년 : 한국전자통신연구원 선임연구원
2016년~현재 : 충북대학교 정보통신공학부 조교수
<관심분야> 무선센서네트워크, IoT, SDN/NFV

김 명 섭 (Myung-Sup Kim)

1998년 : 포항공과대학교 전자계산학과 학사
2000년 : 포항공과대학교 전자계산학과 석사
2004년 : 포항공과대학교 전자계산학과 박사
2006년 : Dept. of ECS, Univ of Toronto Canada
2006년~현재 : 고려대학교 컴퓨터정보학과 교수
<관심분야> 네트워크 관리 및 보안, 트래픽 모니터링 및 분석, 멀티미디어 네트워크

서 동 은 (Dongeun Suh)

2012년 : 고려대학교 전자공학과 학사
2012년~현재 : 고려대학교 IT융합학과 석·박사통합과정
<관심분야> SDN/NFV, 모바일 멀티미디어 스트리밍

임 창 규 (Chang-Gyu Lim)

2000년 : 한국과학기술원 전기전자공학과 학사
2002년 : 한국과학기술원 전기전자공학과 석사
2002년~현재 : 한국전자통신연구원 선임연구원
<관심분야> SDN, 트랜스포트 네트워크, 미래 인터넷

백 상 헌 (Sangheon Pack)

2000년 : 서울대학교 컴퓨터공학부 학사
2005년 : 서울대학교 전기컴퓨터공학부 박사
2007년~현재 : 고려대학교 전기전자공학부 교수
<관심분야> SDN/NFV, 미래 인터넷

박 수 명 (Soomyung Park)

1990년 : 단국대학교 컴퓨터공학 학사
1992년 : 건국대학교 컴퓨터공학 석사
1999년 : 건국대학교 컴퓨터공학 박사
2000년~현재 : 한국전자통신연구원 책임연구원
<관심분야> 트랜스포트 네트워크, SDN/NFV