ONOS: 통신사업자를 위한 오픈소스 SDN 분산 제어기

리건, 이도영, 정세연, 홍원기
포항공과대학교

요 약

Software-Defined Network (SDN) 기술이 산업계의 관심을 받게 되면서 SDN 핵심 기술인 제어기에 대한 연구 및 개발이 활발히 이루어져 왔고 이렇게 개발된 결과물은 오픈소스 형태로 많이 공개되었다. 하지만 현존하는 대부분 SDN 제어기는 SDN 기술이나 잠재성을 시험하고 시연하는데 초점을 맞추어 개발되었기 때문에 대규모 통신사업자 네트워크에 적용하여 사용하기가 힘들었다. 이런 문제를 해결하기 위하여 통신사업자를 위한 오픈소스 SDN 분산 제어기인ONOS가 개발되었다. 본 논문에서는 아키텍처, 분산 코어, 노스바운드, 사우스바운드 및 애플리케이션 등 다양한 측면에서 ONOS에 대하여 자세히 살펴보고자 한다.

Ⅰ. 서 론

SDN은 새로운 네트워크 패러다임으로 제어 평면과 데이터 평면의 분리를 통해 기존 IP 기반 프로토콜들로는 실현하기 어려웠거나 불가능했던 유연한 네트워크 관리를 가능하게 한다. 분리된 제어 평면 기능들은 중앙 집중화된 제어기(컨트롤러)에 위치하고 데이터 평면은 데이터 전송 기능만 수행한다. SDN에서 네트워크를 관제하는 기능이 제어 평면에 집중 되어 있기 때문에 SDN에서 가장 중요하게 연구 및 고려가 되는 기술은 제어 평면을 구성하고 있는 SDN 제어기이다. SDN 제어기는 SDN 기술이 출현된 초기부터 많은 연구 및 개발이 되었으며 NOX[1]는 최초의 오픈 소스 형태로 공개된 SDN 제어기이다. NOX 제어기 뒤를 이어 Python 언어로 구현된 POX, Ryu[2] 등의 제어기들이 있었고, Java 언어로 구현된 Floodlight[3], Beacon[4] 등의 제어기들이 있었다. 많은 제어기들이 오픈소스로 공개되었지만 이런 제어기들 대부분 SDN의 기술 가능성과 잠재성을 시험하고 연구하는데 초점을 맞추어 설계되어 학생 혹은 연구원들이 연구의 목적으로만 사용하여 왔고 사실상 대규모 네트워크를 운용하고있는 통신사업자의 네트워크 서비스나 상용 제품에 적용하기에는 많은 어려움이 있었다. 오픈소스로 공개된 제어기들은 완성도 측면에서도 문제가 있었지만 상용 제품에 적용하기 가장 어려웠던 이유는 대부분 제어기들이 확장성 (Scalability), 고가용성(High Availability) 및 고성능(High Performance) 등 요구사항을 만족시켜 주지 못하였기 때문이다. ONOS(Open Network Operating System)는 미국 비영리 연구소인 Open Networking Lab.(ON.Lab)에서 개발하여 2014년 12월 오픈소스로 공개한 SDN 제어기로 통신사업자의 다양한 요구사항을 만족시키기 위하여 설계 및 개발되었다. ONOS는 코드 모듈화 (Code Modularity), 설정 가능성 (Configurability), 관심사의 분리 (Separation of Concern) 및 프로토콜의 불가지론(Protocol Agnosticism) 같은 디자인 철학을 염두에 두고 설계되었다.

- **코드 모듈화**: ONOS는 각 모듈 사이의 의존성을 줄이는 것을 목표로 설계되었으며 따라서 전체 프로젝트를 여러 개의 하위 프로젝트들로 나누어 관리한다. 각각의 하위 프로젝트는 서로 의존 관계가 적으며 각 하위 프로젝트 별로 빌드 및 실행이 가능하다.
- **설정 가능성**: 새로운 네트워크 서비스 혹은 설정은 제어기가 운영 중에도 서비스 단절없이 실시간으로 설치, 수정 및 삭제가 되어야 하며 이러한 특징을 설정 가능성으로 지칭한다.
- **관심사의 분리**: ONOS는 역할에 따라 제어기를 구성하는 모듈들을 여러 개 계층(Tier)로 나누어 관리한다. 각 계층에 위치한 컴포넌트들은 해당 계층에서 정의한 인터페이스를 통하여 다른 계층에 위치한 컴포넌트들과 통신하며 따라서 개발자와 운영자는 특정 계층의 관심사를 분리하여 개발 및 관리할 수 있다.
- **프로토콜의 불가지론**: ONOS는 하나의 통합된 SDN 제어기로 다양한 SDN 프로토콜들을 사용하여 네트워크를 제어할 수 있어야 한다. 따라서 애플리케이션에서 내려진 명령들은 OpenFlow[5]와 같은 특정 제어 프로토콜에 편향되지 말아야하며 제어가 될 네트워크 장치 종류에 따라 그에 맞

는 형태로 변환된 후 장치에 전달되어 원하는 오퍼레이션을 수행할 수 있어야 한다.

Ⅱ. ONOS 개요

1. ONOS 시스템

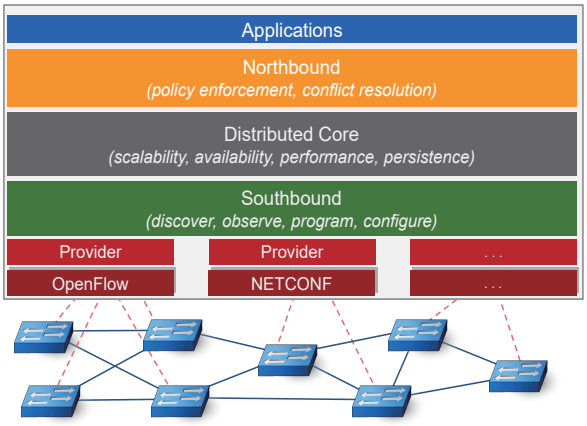

그림 1. ONOS 시스템 아키텍처

ONOS 애플리케이션의 구현 복잡도를 낮추고, 모듈 사이 느슨한 의존관계(Loosely Coupled)를 유지하고자 ONOS는 계층적 구조 (Layered Architecture)로 〈그림 1〉 설계가 되었으며 각 계층별 역할은 다음과 같다.

- **애플리케이션(Applications) 계층**: 사용자 애플리케이션들이 위치하는 계층이며 네트워크를 제어 및 설정하기 위한 다양한 로직들이 포함되어 있다. 애플리케이션들은 노스바운드에서 제공하는 추상화된 인터페이스를 이용하여 네트워크를 프로그래밍한다.
- **노스바운드(Northbound) 계층**: 정책기반으로 네트워크를 제어하기 위한 다양한 추상화 인터페이스를 제공한다. 뿐만 아니라 이 계층은 하위에서 발생하는 네트워크 이벤트들을 해당 이벤트를 구독(Subscribe)한 애플리케이션들에 전달해주는 역할도 한다.
- **분산 코어(Distributed Core) 계층**: 제어 평면의 확장성, 고가용성 및 부하 분산 (Load Balance) 기능을 제공하기 위하여 추가된 계층으로 분산 처리에 필요한 복잡한 동기화 과정을 수행한다.
- **사우스바운드(Southbound) 계층**: 네트워크를 제어하는데 사용되는 다양한 통신 프로토콜을 추상화해주는 인터페이스이다. Protocol-specific한 내용들을 숨김으로써 분산 코어와 protocol-neutral한 방식으로 소통한다.
- **프로바이더/프로토콜(Provider/Protocol)계층**: 통신 장치와 실제 통신하는 계층으로 통신 시 사용되는 제어 메시지의 serialization 및 deserialization을 수행하고 각 통신 장치 별로 채널 세션 관리를 하는 등 protocol-specific한 오퍼레이션을 수행한다.

2. ONOS 서브 시스템

ONOS는 제공하는 서비스 종류에 따라 여러 서브 시스템(Subsystem)으로 나뉘며 하나의 서브 시스템은 전체 ONOS를 수직적으로 슬라이스하여 얻은 하나의 기능 조각이다. 서브 시스템은 〈그림 2〉에서와 같이 애플리케이션(Application), 매니저(Manager) 및 프로바이더(Provider) 세 가지 컴포넌트들로 나뉜다.

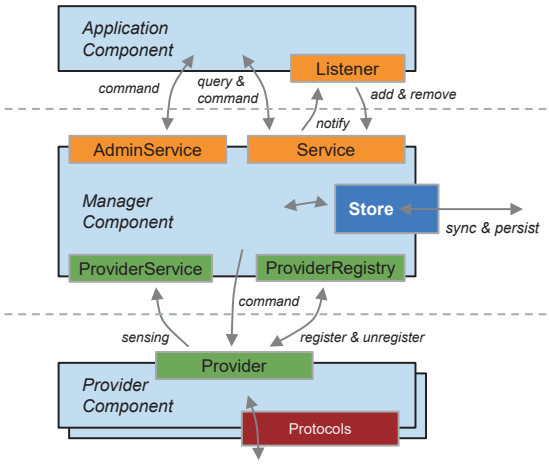

그림 2. ONOS 서브 시스템

- **애플리케이션 컴포넌트**: 애플리케이션은 노스 바운드에 정의된 AdminService와 Service 인터페이스를 통하여 매니저로부터 네트워크 이벤트들을 전달받고 전달받은 이벤트에 근거하여 네트워크를 제어하기 위한 명령(Command)을 매니저로 전달한다. 명령은 AdminService를 통하여 매니저에 전달되고, 네트워크 이벤트 수신은 하나의 Listener을 Service에 등록(Register) 하는 것을 통하여 쉽게 실현한다.
- **매니저 컴포넌트**: 매니저는 프로바이더에서 데이터를 받은 후 애플리케이션 또는 다른 서비스에 전달해 주는 역할을 한다. 매니저에는 다음과 같은 인터페이스들이 있다. Service는 애플리케이션 또는 다른 매니저 컴포넌트

들에 네트워크 상태를 전달해 준다. AdminService는 네트워크 설정 등 관리 목적인 명령을 네트워크 또는 시스템에 전달한다. ProviderRegistery는 해당 매니저와 통신하기 위한 프로바이더를 등록한다. ProviderService는 매니저와 프로바이더 간에 데이터 통신 작용을 한다. 스토어(Store)는 네트워크 오브젝트를 저장하고, 네트워크 이벤트를 다른 서브 시스템 혹은 주변 ONOS 노드들에게 전파(Propagation)해주는 역할을 한다. 프로바이더에서 제공하는 description 데이터를 네트워크 오브젝트로 변환하여 저장하고 관리자가 정의한 규칙에 의하여 알림을 생성한다. 또한, 다중 ONOS들로 형성된 클러스터 환경에서는 ONOS의 같은 서브 시스템 간의 데이터 동기화 작업을 진행한다.

- **프로바이더 컴포넌트**: 프로바이더는 네트 워크 요소들과 통신하여 장치를 제어하고 장치들로부터 전달 받은 네트워크 이벤트들을 매니저한테 전달하는 역할을 한다. 프로바이더는 네트워크 프로토콜에 정의된 라이브러리를 통하여 네트워크 장치 및 호스트와 통신하고, Provider Service를 통하여 매니저와 통신한다.

Ⅲ. ONOS 분산 코어

SDN 네트워크를 운용함에 있어 서비스의 고가용성(High Availability), 안정성(Stability), 신뢰성(Reliability) 및 확장성을 향상시키기 위하여 ONOS의 제어 평면은 논리적으로는 중앙 집중화 되어있지만 물리적으로는 분산되어 있는 아키텍처를 채택하고 있다. ONOS 제어 평면은 복수의 ONOS 제어기들로 구성되어 있으며 이런 제어기들은 단일 ONOS 클러스터를 형성하여 운용된다. 이와 같이 논리적으로 중앙 집중화된 제어 평면을 구현하기 위하여 ONOS 분산 코어는 제어기들 사이에 마스터쉽(Mastership)과 리더쉽(Leadership) 프로토콜을 정의하였고 이를 통하여 제어 평면의 장애 복구와 부하분산 기능을 실현하였다.

1. 마스터쉽 관리

제어 평면의 고가용성을 실현하기 위하여 단일 네트워크 장치는 복수의 ONOS 제어기들과 연결된다. 장치에 대한 제어 권한 수준에 따라 제어기의 역할을 아래와 같이 구분 지을 수 있다.

- NONE: 이 역할을 가진 제어기는 해당 네트워크 장치를 인지하지못하며 해당 장치와 상호 연동되지 않는다.
- STANDBY: SLAVE 역할로도 불리며 이 역할을 가진 제어기는 네트워크 장치를 인지하고 해당 장치의 상태를 읽어 올 수 있지만 Flow Rule을 설치하는 등 장치에 대한 관제 기능은 수행하지 못한다. 단일 네트워크 장치는 복수의 SLAVE 제어기와 연결될 수 있다.
- MASTER: 이 역할을 부여 받은 제어기는 네트워크 장치를 관제하기 위한 최고의 권한을 행사할 수 있으며 장치의 상태를 읽을 수 있을 뿐만 아니라, 장치의 상태를 변경할 수도 있다. 단일 네트워크 장치는 오로지 하나의 MASTER 제어기와 연결될 수 있으며 그 외에 연결된 제어기는 모두 STANDBY 역할을 부여 받아야만 한다.

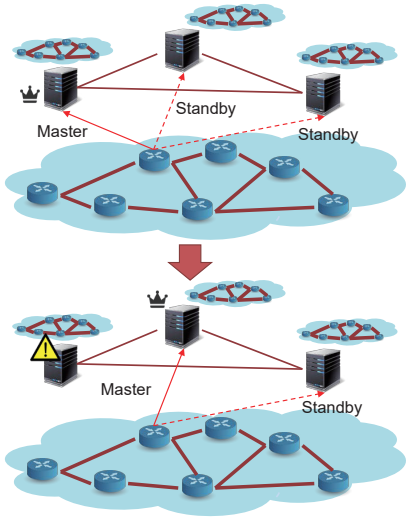

그림 3. 마스터쉽을 이용한 장애 복구 과정

제어기에 대한 마스터쉽 할당은 1) 제어 평면과 데이터 평면의 합의 및 2) 제어 평면에 위치한 제어기들 사이의 합의에 의하여 이루어진다.

우선 제어 평면과 데이터 평면의 합의에 의하여 마스터쉽이 부여되는 과정을 살펴보겠다. 제어기가 처음으로 구동이 되면 NONE 역할을 부여 받는다. 제어기가 네트워크 장치와 핸드쉐이킹(Hand-Shaking) 과정을 거쳐 연결이 되면 해당 장치의 마스터쉽을 확인하여 만약 장치에 MASTER 제어기가 연결되지 않은 상태이면 자기 자신을 해당 장치의 MASTER로 정하고 해당 장치에게 자신이 MASTER임을 알리는 역할 부여 확인 메시지를 보낸다. 만약 연결된 장치에 이미 MASTER 제어기가 연결되어 있으면 자신의 역할을 NONE으로부터 SLAVE로 바꾼다.

마스터쉽 할당은 제어 평면에 위치한 제어기들 사이의 합의에 의하여 이루어질 수도 있다. 아래에 제어 평면의 장애 복구 과정을 통하여 두번째 마스터쉽 할당 방법을 살펴보겠다. 〈그림 3〉은

ONOS를 이용한 장애 복구 과정을 도식화 하고 있다. 네트워크 장치에 연결된 MASTER 제어기에 장애가 발생하면 주변 제어기들은 DeviceService 스토어를 통하여 장애를 인지하고 장애가 발생한 제어기에 연결된 네트워크 장치를 위한 새로운 MASTER 제어기를 선출한다. 기본 선출 방법은 First-Come-First-Served (FCFS) 기준을 따르고 이 기준은 네트워크 관리자에 의하여 바뀔 수 있다. 이렇게 MASTER로 선출된 새로운 제어기는 해당 장치에 대하여 MASTER 역할을 부여 받게 되고 MASTER임을 알리는 역할 부여 확인 메시지를 장치에 전송한다.

ONOS는 제어 평면 장애 복구를 염두에 두고 설계되었기 때문에 모든 제어기들에서 같은 네트워크 상태(제어 상태 스위치에 설치된 Flow Rule등)를 유지하고 있다. 따라서 특정 네트워크 장치의 마스터쉽이 바뀌어도 새로 MASTER 역할을 부여 받은 제어기가 장애가 발생한 제어기와 같은 네트워크 상태를 가지고 네트워크 장치를 지속적으로 관리할 수 있는 것이다.

현재 ONOS는 naïve한 부하분산 알고리즘을 사용하여 스위치와 제어기의 마스터쉽을 관리하고 있고 알고리즘에 따르면 각 제어기는 같은 개수의 스위치들로부터 마스터쉽을 할당 받아 제어 평면의 워크로드를 부하 분산시킨다. 하지만 각 스위치에서 발생하는 제어 메시지 수가 다름에 따라 이런 방법을 사용하면 제어 평면 워크로드가 완벽히 각 제어기에 부하 분산되지 않는다. 따라서 제어 평면의 워크로드를 모니터링하는 방법[7]과 이렇게 모니터링된 결과를 바탕으로 워크로드를 균일하게 부하분산하는 방법[8][9] 들이 제안되었다.

2. 네트워크 상태 동기화 합의 방법

복수개의 제어기들 사이에서 같은 네트워크 상태를 유지하기 위하여 상태 정보의 일관성을 유지하는 합의(Consensus) 방법이 필요하다. ONOS는 Onix[6]와 흡사하게 엄격한 일관성(Strong Consistency)과 결과의 일관성(Eventual Consistency)을 보장 하기 위한 두 가지 합의 방법을 사용한다. Onix는 Nicira에서 상용화한 분산 SDN 제어기이며 오픈소스로 공개되지는 않았다.

엄격한 일관성은 한 노드에서 발생한 상태의 변화가 즉시 다른 노드들에 전달됨으로써 클러스터를 구성하는 모든 노드들이 단일 트랜잭션내에서 같은 상태 정보를 접근할 수 있도록 보장하는 특성이다. 엄격한 일관성은 일관성 보장 수준이 가장 높아 모든 노드들이 항상 동일한 상태 정보를 접근할 수 있다는 장점이 있지만 반면 클러스터의 규모가 커짐에 따라 각 노드들을 접근하여 상태를 업데이트 해주는데 드는 오버헤드가 크기 때문에 확장성에 따른 제약이 있다. 따라서 ONOS는 모든 네트워크 상태 정보들에 대하여 엄격한 일관성을 보장하지는 않고 스위치와 제어기 사이의 매핑 정보 등 즉시 상태를 일치시켜야 하는 정보에 대해서만 엄격한 일관성을 보장하고 있다. ONOS는 엄격한 일관성을 실현하기 위하여 간편하면서도 효과적인 RAFT 합의 알고리즘[10]을 사용하고 있다 〈그림 4〉.

ONOS는 RAFT를 사용하여 전체 네트워크 데이터를 여러 개의 파티션으로 나누고 각 파티션의 복사본(replica)를 만들어 복수 개(보통 3개)의 제어기에 저장한다. 그 중 한 제어기는 리더(Leader)로 선출(Election)되고 리더는 복사본에 업데이트가 발생하였을 때 다른 Follower들에 업데이트된 내용을 전달한다. RAFT는 투표(voting)와 랜덤화한 타임아웃 (randomized timeout) 두 가지 방법을 사용하여 리더를 선출한다. 선출된 리더는 주기적으로 하트비트(heart beat)신호를 주변 Follower들에 전송하며 리더로부터 전달되는 신호가 일정시간 동안 없으면 Follower들의 합의하에 새로운 리더가 선출된다. ONOS는 리더쉽(Leadership) 서비스를 이용하여 RAFT의 리더쉽을 관리한다.

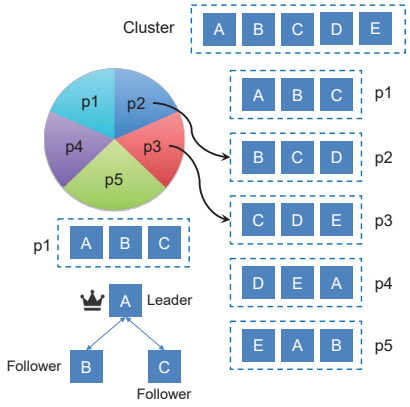

그림 4. RAFT 합의 알고리즘 동작 방식

결과의 일관성은 엄격한 일관성과는 달리 상태의 변화를 즉시 모든 노드들에 전달하지 않고 순차적으로 주변 노드들에 전달함으로써 결과적으로 일정 시간이 지난 후 모든 노드들에게 업데이트된 상태를 알리는 특성을 가진다. 결과의 일관성은 점진적으로 상태의 변화를 알리기 때문에 전체 클러스터에 가져다주는 오버헤드가 적어 제어 평면의 확장성을 보장한다. 따라서 상태를 즉시 일치시킬 필요가 없는 경우에 많이 활용된다. 토폴로지 상태 갱신은 일부 동기화 지연을 허용하기 때문에 ONOS는 이 작업에 결과의 일관성만 보장해준다.

결과의 일관성을 실현하는 방법은 다양하며 ONOS는 Gossip 프로토콜을 사용하여 결과의 일관성을 실현하였다. Gossip 프로토콜은 상태 갱신이 발생한 노드가 인접한 주변 노드들에 갱

신된 정보를 브로드캐스트(Broadcast) 방식으로 전달하고 정보를 수신 받은 노드들이 다시 주변 노드들에 갱신된 정보를 전달하는 단순한 방식으로 구현되었다.

IV. ONOS 노스바운드

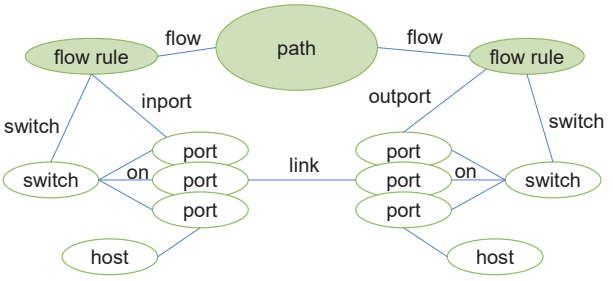

그림 5. ONOS 데이터 모델

1. 사용자 인터페이스

ONOS는 다섯 가지 사용자 인터페이스를 제공하며 그것들로는 각각 웹 인터페이스, REST API, gRPC API, Command Line Interface (CLI) 및 Native Service Interface이다. 다섯 가지 인터페이스들 중 웹 인터페이스와 CLI는 사용자 지향적 인터페이스로 사용자는 해당 인터페이스를 통하여 ONOS와 연결된 네트워크 장치들을 직접적으로 제어 및 관리할 수 있다. REST API, gRPC API및 Service Interface는 ONOS에서 제공하는 세 가지 프로그래밍 인터페이스로 개발자는 해당 인터페이스에서 정의한 메서드를 호출하여 ONOS의 다양한 서비스들을 사용할 수 있다. gRPC API는 HTTP/2기반 양방향 스트리밍을 사용하여 API 호출 시 발생하는 지연을 대폭 줄였고 Protocol Buffer (ProtoBuf)[11] 바이너리 포맷으로 인코딩 된 데이터를 송수신 함으로써 네트워크 트래픽을 대폭 줄여 시스템 리소스를 절약하고 성능을 높일 수 있다.

2. 데이터 모델 추상화

ONOS는 directed cyclic 그래프를 이용하여 네트워크 장치, 링크, 호스트들과 같은 네트워크 요소들을 모델링하였고 이것을 자체 데이터 모델로 사용하고 있다 〈그림 5〉. 데이터 모델링을 통하여 protocol-specific한 네트워크 요소들이 protocol-agnostic한 요소로 추상화 되며 ONOS는 이렇게 추상화된 개개의 요소를 Model Object라고 지칭한다. 예를 들어, OpenFlow Device는 Device Model Object로 추상화 되고, Packet-In 제어 메시지는 inbound packet으로 추상화 된다. 이렇게 추상화 된 Model Object는 애플리케이션에서 사용할 수 있도록 노스바운드 를 통하여 서비스 API형태로 제공된다.

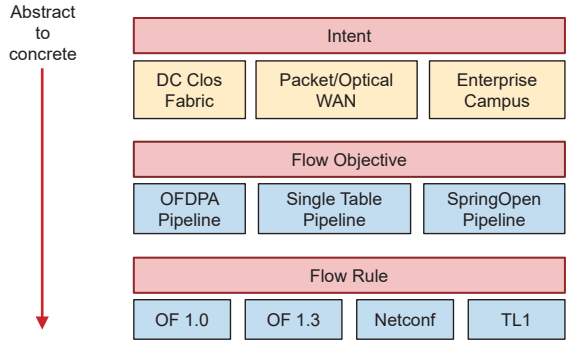

그림 6. ONOS 노스바운드 프로그래밍 모델

3. 프로그래밍 모델 추상화

ONOS는 데이터 모델뿐만 아니라 프로그래밍 모델을 이용하여 네트워크 제어에 대한 추상화를 제공한다. 추상화 수준에 따라 프로그래밍 모델을 크게 세 가지로 나눌 수 있으며 그것들로는 각각 Flow Rule, Flow Objective 및 Intent이다. 추상화 수준은 〈그림 6〉에서 도식화한 것과 같이 위에서부터 아래로 내려오면서 점차 낮아지며 Intent가 가장 높은 수준의 추상화 인터페이스를 제공한다.

Flow Rule은 추상화 수준이 가장 낮은 프로그래밍 모델로 해당 모델에서 정의한 인터페이스를 통하여 네트워크 장치에 직접 Flow Rule 프로그래밍을 할 수 있다. 이렇게 프로그래밍 된 Flow Rule은 Flow Rule 서브 시스템에 의하여 Flow Rule 스토어에 저장될 뿐만 아니라 사우스바운드 인터페이스를 통하여 관련된 네트워크 장치에 설치된다. Flow Rule이 처음 스토어에 저장되면 PENDING_ADD 상태로 표시되고 장치에서 정상적으로 설치되었다는 확인 이벤트를 받게 되면 ADDED 상태로 바뀐다.

네트워크 장치는 패킷을 처리함에 있어 서로 다른 파이프라인을 사용한다. 예를 들어 제조사 A의 스위치에서는 첫번째 Flow Table을 MAC 매치를 위한 목적으로, 두번째 Flow Table을 Port와 VLAN 매치를 위한 목적으로 사용하는 반면 제조사 B의 스위치에서는 첫번째 Flow Table을 Port 매치를 위한 목적으로 두번째 Flow Table을 MAC 매치를 위한 목적으로 사용할 수 있다. 제조사 별로 Flow Table 사용용도가 다름에 따라 Flow Rule을 설치할 때에도 이런 점을 고려하여 각 Flow Table에 적합한 Flow Rule들을 프로그래밍하여 내려야한다.

Flow Rule 프로그래밍 모델의 가장 큰 문제점은 장치 파이프라인에 대한 추상화를 제공하고 있지 않기 때문에 장치의 파이프라인 구조를 애플리케이션 개발자가 미리 인지하고 Flow Rule을 프로그래밍 해야한다는 것이다. Flow Objective는 Flow Rule이 가지고 있는 파이프라인 문제를 해결하기 위하여 제안된 프로그래밍 모델로 제조사별 파이프라인 구조를 사우스바운드에 위치한 드라이버(Driver)에 정의하고 노스바운드에는 Device-Neutral한 추상화 인터페이스를 제공하여 개발자로 하여금 파이프라인 독립적인(Pipeline-independent) Flow Rule 프로그래밍을 가능하게 한다.

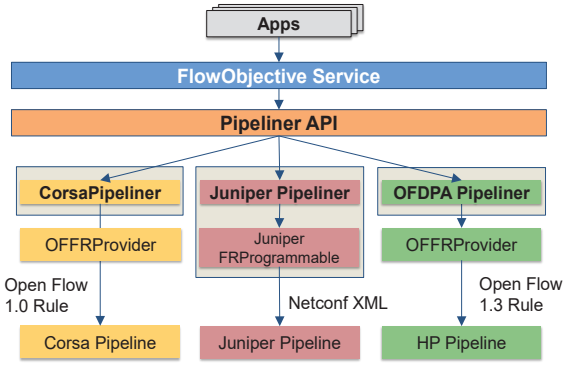

그림 7. ONOS Flow Objective 프로그래밍 모델

〈그림 7〉에서 Flow Objective를 이용한 Rule 프로그래밍 과정을 도식화 하고 있다. 애플리케이션에서 Flow Objective 서비스 인터페이스를 호출하여 Objective를 내리게 되면 해당 Objective는 파이프라인 API를 거쳐 특정 제조사 파이프라이너(Pipeliner)에 전달되고 파이프라이너에서 미리 정의한 파이프라인 모델에 따라 관련된 Flow Rule로 decompose 된 후 장치에 설치된다. 파이프라인이 어떤 SDN 프로토콜을 구현 하였는지에 따라 Objective는 NETCONF XML 혹은 OpenFlow 제어 메시지로 변환되어 장치에 전달된다. Flow Objective는 사용목적에 따라 Filtering Objective, Forwarding Objective 및 Next Objective로 나뉜다.

ONOS는 Flow 기반 서비스 API 이외에도 정책 기반 프로그래밍 모델인 Intent도 지원한다. Intent는 애플리케이션 개발자로 하여금 무엇을 실현 할 것인지에 대하여 정의만 내려주면 해당 기능을 어떻게 실현 할 것인지에 대한 구체적인 내용을 실질적으로 구현하지 않아도 정의된 Intent로부터 해당 기능을 실현하는 자동화된 프레임워크 (Framework)를 제공한다. 그림 8은 애플리케이션에서 HostToHostIntent를 이용하여 두 호스트 사이 종단간 네트워크 연결 설정을 위한 Flow Rule을 내려주는 예시를 도식화 하였다. 애플리케이션에서 HostToHostIntent를 정의 후 Intent 서비스 API를 통하여 Intent 프레임워크에 전달하면 프레임워크는 1) HostToHostIntent를 근거로 패킷 송신과 수신을 위한 두개의 Path Intent들을 생성하며(컴파일 단계), 2) Path Intent는 최종 Flow Rule로 변환되어 경로에 위치한 각 스위치들에 설치된다. 네트워크 장애 등 이유로 토폴로지에 변화가 발생할 경우 Intent 프레임워크는 이런 장애를 자동으로 감지하고 새로운 Path Intent를 생성 후 새로 계산된 경로에 위치한 스위치들에 설치한다. Intent의 가장 큰 장점은 애플리케이션 개발자가 세부적인 네트워크의 동작 과정을 이해하거나 복잡한 프로그래밍 코드를 작성하지 않고도 원하는 기능을 쉽게 개발할 수 있다는 점이다. Intent는 학계에서도 많은 관심을 가지고 다루고 있는 주제로 논문[12][13]에서는 Intent를 이용한 가상 네트워크 프로비저닝 과정을 보이고 있다.

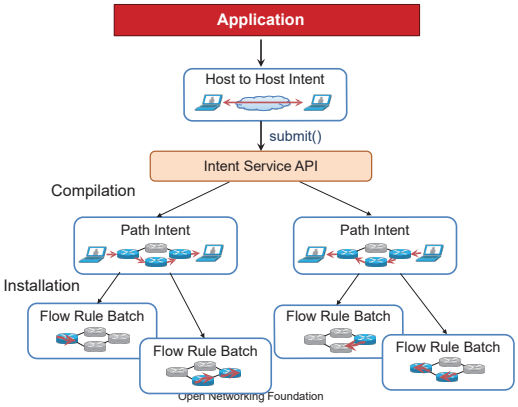

그림 8. ONOS Intent 프로그래밍 모델 예시

V. ONOS 사우스바운드

ONOS 사우스바운드는 분산 코어가 실질적인 네트워크 디바이스 종류나 제어 프로토콜 등 세부적인 내용을 알 필요 없이 네트워크 요소들을 관리할 수 있도록 추상화 해준다. 이런 추상화의 장점은 1) 분산 코어가 프로토콜 혹은 장치에 특화된 데이터를 저장할 필요가 없고, 2) ONOS만을 이용하여 다양한 SDN 네트워크를 관리할 수 있으며 3) 새로운 SDN 프로토콜을 지원할 경우 해당 프로토콜을 플러그인 형태로 추가하는 것만으로 쉽게 실현이 가능하여 특정 프로토콜이나 장치에 대한 의존성을 제거할 수 있다. ONOS 사우스바운드는 기능에 따라 크게 세 가지 컴포넌트들로 나눌 수 있으며 각 컴포넌트는 〈그림 9〉에 도식화 되어있다.

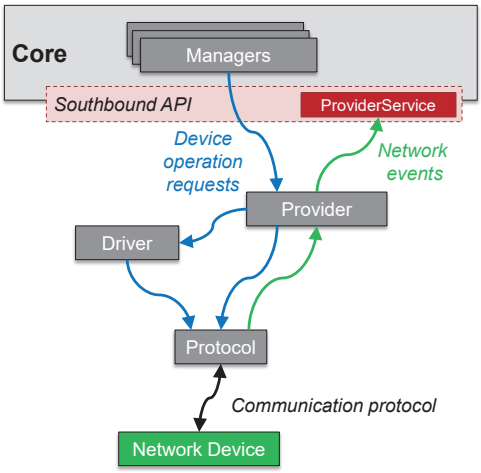

그림 9. ONOS 사우스바운드 아키텍처

1. 프로토콜(Protocol)

프로토콜 컴포넌트는 사우스바운드의 가장 하위에 위치하며 SDN 런타임 제어 혹은 관리 프로토콜을 이용하여 네트워크 장치와 직접적으로 통신한다. 본 논문 작성 시점 기준 최신 ONOS stable 버전은 v1.11이고 이 버전은 총 13가지 사우스바운드 프로토콜들을 지원하며 그것들로는 OpenFlow(v1.0, v1.3)[5], OVSDB[14], NETCONF/ YANG[15], SNMP[16], BMv2[17], BGPLS, ISIS, OSPF, LISP[18], PCEP[19], REST, RESTCONF, TL1및 gRPC이다. 이런 프로토콜들은 통신사업자 네트워크 장치 관리를 위하여 많이 사용되는 프로토콜로 Internet Engineering Task Force (IETF), Open Networking Foundation(ONF) 등 표준화 기관에 의하여 표준화 및 상용화가 이루어진 상태이다. 프로토콜들을 구현함에서 제어 혹은 관리 메시지의 serializing 및 de-serializing부분은 openflowj, snmp4j, netty등과 같은 third-party 라이브러리를 사용하여 실현하였다. 프로토콜은 일반적으로 모듈 사이의 의존성을 낮추고 코드의 간결성 향상 등 이유로 API와 CTL 서브 모듈로 나뉜다. API 서브 모듈은 일반적으로 네트워크 장치 제어에 사용되는 오퍼레이션과 메시지를 추상화하는 일련의 인터페이스들을 포함하고 있다. OpenFlow를 예로 들어보면, Flow Table 조작(Manipulation)에 필요한 FlowMods와 Group Table 조작에 필요한 GroupMods 오퍼레이션들을 API에 정의하고 있다. CTL 서브 모듈에는 API에서 정의한 인터페이스의 복수개의 구현체들이 위치하고 있다. OpenFlow CTL 서브 모듈은 OpenFlow 스위치와 통신할 때 필요한 핸드쉐이킹, 장치와의 연결 수립 및 유지 기능을 구현하고 있다. ONOS에서 프로토콜은 플러그인 형태로 구현되어 필요에 따라서 활성화 혹은 비활성화 할 수 있다.

2. 프로바이더(Provider)

프로바이더는 프로토콜과 분산 코어 사이에 위치하여 코어로부터 오는 명령을 특정 장치에 맞는 명령으로 구체화하고 반대로 특정 장치로부터 오는 네트워크 이벤트를 장치 중립적인 이벤트로 추상화하여 분산 코어에 전달한다. 분산 코어는 여러 프로바이더들을 사용하여 네트워크 장치와 소통한다. 분산 코어가 장치에 맞는 프로바이더를 선택함에 있어 장치의 식별자(Device ID)를 사용한다. ONOS에서 장치 식별자는 고유한 이름 공간 (Name Space)을 prefix로 사용하고 있다. 예를 들어 OpenFlow를 지원하는 장치는 "of"를 장치 식별자 prefix로 사용하고, NETCONF를 지원하는 장치는 "netconf"를 장치 식별자 prefix로 사용한다.

```
<driver name="default" manufacturer="ON.Lab"
    hwVersion="0.0.1" swVersion="0.0.1">
        <behaviour api=Tracking
                impl=TrackingImpl />
</driver>

<driver name="ovs" extends="default"
        manufacturer="Nicira, Inc\."
        hwVersion="Open vSwitch" swVersion="2\..*">
        <behaviour api="Tracking"
                impl="TrackingImpl" />
</driver>
```

그림 10. 드라이버에서 OVS ConnTrack 정의 예제

3. 드라이버(Driver)

드라이버는 일련의 장치 행위(Behavior)들의 집합으로 필요에 따라서 선택적으로 활성화하여 사용할 수 있다. 드라이버는 장치에 특화된 기능들을 포함하고 있으며 이런 기능들은 보통 장치 제조사들에서 자사 장치를 차별화하기 위하여 추가한 기능들로 표준 프로토콜 명세서는 정의되어 있지 않다. 예를 들어 Nicira는 상태 기반 방화벽을 쉽게 실현하기 위하여 자사 소프트웨어 스위치 OpenvSwitch(OVS)[20]에 TCP 세션 상태를 추적할 수 있는 ConnTrack 모듈을 추가하였다. ConnTrack은 OpenFlow 표준 명세서에는 정의되지 않은 기능으로 이 기능을 ONOS에서 사용하려면 그림 10에서와 같이 Tracking 행위를 드라이버 설정 파일에 정의하고 ConnTrack을 사용하기 위한 TrackingImpl을 구현하여야 한다.

VI. ONOS 애플리케이션

ONOS v1.11 기준 총 80개 이상의 빌트인(Built-in) 애플리케이션들이 개발되어 오픈소스로 공개되고 있다. ONOS 애플리케이션은 단일 .oar (ONOS Application aRchive)에 패키징되어 실행 중인 ONOS에 배포 및 설치된다. .oar은 zip형식의 아카이브 파일로 애플리케이션 구동에 필요한 모든 artifacts들을 내장하고 있다. Artifacts에는 OSGi 번들, feature 및 애플리케이션 메타 파일 등이 포함된다.

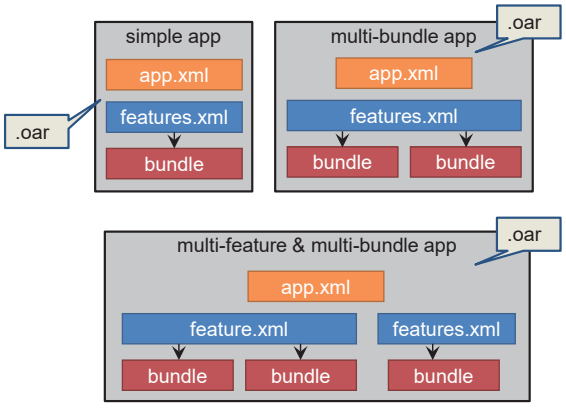

그림 11. ONOS 애플리케이션 예시

애플리케이션 메타 파일에는 애플리케이션 버전, 제조사, 설명 및 의존 애플리케이션 목록 등 정보들이 포함되고 feature 메타 파일에는 애플리케이션 실행에 필요한 의존 라이브러리 및 OSGi 번들 목록들이 포함된다. 필요에 따라 단일 ONOS 애플리케이션은 단일 혹은 다중 OSGi 번들 파일을 포함할 수 있고 단일 혹은 다중 feature 메타 정보 파일을 포함할 수 있다 〈그림 11〉. ONOS가 통신사업자를 위한 SDN 제어기인만큼, ONOS 애플리케이션들도 통신사업자가 필요로 하는 애플리케이션들로 많이 개발되었다. 대표적인 애플리케이션으로는 광통신장비를 제어하기 위한 vOLT 애플리케이션, 유선망 가입자 관리를 위한 AAA 애플리케이션, BGP 라우터와 통신을 위한 SDN-IP 애플리케이션, 데이터센터 네트워크를 이루는 스위치들을 관제하기 위한 Trellis 애플리케이션 및 데이터센터 네트워크 가상화를 위한 SONA[21] 등 애플리케이션들이 존재한다.

VII. 결 론

본 논문에서는 통신사업자를 위한 오픈소스 분산 SDN 제어기인 ONOS에 대하여 자세히 알아보았다. ONOS는 2015년 Linux Foundation 산하 글로벌 오픈소스 프로젝트로 선정이 되었고 현재 ONF에 의하여 노스바운드 및 사우스바운드에 대한 표준화 및 개발이 이루어지고 있다. 이뿐만 아니라 여러 통신사업자, 제조사 및 연구기관들에서도 ONOS 커뮤니티에 적극 참여하여 많은 기여[22]를 하고 있으며 국내에서도 ONOS기반 SDN 솔루션들에 대한 상용화가 이루어지고 있다. ONOS는 오픈소스로 공개된 이래 총 11번의 판올림을 통하여 많은 기능들이 추가되었고 성능 향상과 코드 안정화가 적용되었다. 끝으로 ONOS가 SDN 산업계의 발전을 이끄는 표준 제어기로 거듭나기를 기대해본다.

Acknowledgement

본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구결과로 수행되었음" (IITP-2017-2017-0-01633)

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2015-0-00575, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

참 고 문 헌

[1] N. Gude et al., "NOX: Towards an operating system for networks", Computer Communications. Review, vol. 38, no. 3, pp. 105-110, 2008.

[2] Ryu controller. Website. https://osrg.github.io/ryu/

[3] Floodlight controller. Website. http://www.projectfloodlight.org/floodlight/

[4] D. Erickson, "The Beacon OpenFlow controller", Proc. 2nd ACM SIGCOMM Workshop Hot Topics on Software-Defined Networking, pp. 13-18, 2013.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "Openflow: enabling innovation in campus networks", ACM SIGCOMM Computer Communications Review, vol. 38, no. 2, pp. 69-74, 2008.

[6] T. Koponen et al., "Onix: A distributed control platform for large-scale production networks", Proc. 9th USENIX Conference on Operating Systems Design and Implementation, pp. 1-6, 2010.

[7] W. Kim, J. Li, J. W.-K. Hong, Y.-J. Suh, "OFMon: Openflow monitoring system in ONOS controllers", 2016 IEEE NetSoft Conference and Workshops (NetSoft 2016), pp. 397-402, 2016.

[8] J. Li, J.-H. Yoo, and J. W.-K. Hong, "Dynamic control plane management for software-defined networks," International Journal of Network Management, vol. 26, no. 2, pp. 111 - 130, 2016.

[9] W. Kim, J. Li, Y.-J. Suh and J. W.-K. Hong, "HeS-CoP: Heuristic Switch-controller Placement Scheme for Distributed SDN Controllers in Data Center Networks," International Journal of Network Management, vol. 13, 2017.

[10] D. Ongaro, J. Ousterhout, "In search of an understandable consensus algorithm", 2014 USENIX Annual Technical Conference (USENIX ATC '14), Jun. 2014.

[11] Protocol Buffer. Website. https://developers.google.com/protocol-buffers/

[12] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong, "An intent-based network virtualization platform for SDN," in Proceedings of the 2016 12th International Conference on Network and Service Management (CNSM 2016), pp. 353 - 358, Québec, Canada, October 2016.

[13] Y. Han, T. Vachuska, A. Al-shabibi, J. Li, H. Huang, W. Snow and J. W.-K. Hong, "ONVisor: Towards a scalable and flexible SDN-based network virtualization platform on ONOS," International Journal of Network Management, vol. 12, 2017.

[14] B. Pfaff, B. Davie, The Open vSwitch database management protocol, IETF RFC 7047, Dec. 2013.

[15] R. Enns, NETCONF configuration protocol, Feb. 2006, [online] Available: http://www.ietf.org/internet-drafts/draft-ietf-netconf-prot-12.txt.

[16] J. Case, M. Fedor, M. Schoffstall, J. Davln, A Simple Network Management Protocol (SNMP), May 1990, [online] Available: http://www.ietf.org/rfc/.

[17] Barefoot Networks. P4 behavioral model. Website. https://github.com/p4lang/behavioral-model.

[18] D. Farinacci, V. Fuller, D. Oran, D. Meyer, "Locator/ID Separation Protocol (LISP)", Apr. 2008.

[19] J. Vasseur, J. L. Roux, Path computation element (PCE) communication protocol (PCEP), Mar. 2009.

[20] B. Pfaff et al., "The design and implementation of Open vSwitch", Proc. USENIX Symp. NSDI, pp. 117-130, 2015.

[21] SONA, [online] Available: https://wiki.onosproject.org/display/ONOS/SONA%3A+DC+Network+Virtualization

[22] S. Jeong, D. Lee, J. Choi, J. Li, J. W.-K. Hong, "Application-aware Traffic Management for OpenFlow Networks", 18th Asia-Pacific Network Operations and Management Symposium (APNOMS 2016), Kanazawa, Japan, Oct. 5-7, 2016.

약 력

리 건

2007년 연변과학기술대학교 전자통신공학과 학사
2012년 포항공과대학교 컴퓨터공학과 석사
2012년~2016년 포항공과대학교 정보전자융합공학부 박사
2016년~2017년 포항공과대학교 컴퓨터공학과 박사 후 연구원
2017년~현재 포항공과대학교 정보통신대학원 연구 조교수
관심분야: SDN, OpenFlow, 클라우드 컴퓨팅, 모바일 장치 관리, 스마트 그리드

이 도 영

2015년 건국대학교 컴퓨터공학부 학사
2015년~현재 포항공과대학교 컴퓨터공학과 통합과정
관심분야: SDN, OpenFlow, 네트워크 트래픽 모니터링

정 세 연

2015년 경북대학교 컴퓨터공학부 학사
2015년~현재 포항공과대학교 컴퓨터공학과 석사과정
관심분야: SDN, OpenFlow, 네트워크 관리

홍 원 기

1983년 Univ. of Western Ontario, BSc in Computer Science
1985년 Univ. of Western Ontario, MS in Computer Science
1985년~1986년 Univ. of Western Ontario, Lecturer
1986년~1991년 Univ. of Waterloo, PhD in Computer Science
1991년~1992년 Univ. of Waterloo, Post-Doc Fellow
1992년~1995년 Univ. of Western Ontario 연구교수
1995년~현재 포항공과대학교 컴퓨터공학과 교수
2007년~2010년 포항공과대학교 정보통신연구소 연구소장
2008년~2010년 포항공과대학교 컴퓨터공학과 주임교수
2008년~2012년 포항공과대학교 정보전자융합공학부장
2012년~2014년 KT 종합기술원 원장
2007년~2011년, 2015년~현재 포항공과대학교 정보통신대학원장
관심분야: 네트워크 트래픽 모니터링, 네트워크 및 시스템 관리, SDN/NFV, IoT, 무크기반 교육