K-ONE 기술 문서 #12

ONOS 상 LISP Controller 설계 및 구현

Document No. K-ONE #12
Version 1.0
Date 2016-05-08
Author(s) 한윤선

K-ONE

ONK
OpenNetworkingKorea

■ 문서의 연혁

버전	날짜	작성자	내용
초안 - 0.1	2016. 03. 28	한윤선	초안 작성
0.2	2016. 04. 15	한윤선	설계 내용 추가
1.0	2016. 05. 08	한윤선	오탈자 수정

본 문서는 2015년도 정부(미래창조과학부)의 재원으로 정보통신 기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-15-2012, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. B0190-15-2012, Global SDN/NFV OpenSource Software Core Module/Function Development)

기술문서 요약

최근 소프트웨어 정의 네트워크 (Software-Defined Networking; SDN) 기술들이 네트워킹 관련 연구자들, 개발자들, 제조사, 표준화 기구, 대학들로부터 매우 높은 주목을 받고 있다. SDN의 기본 철학은 네트워크의 제어 평면과 데이터 평면을 분리하는 것고, 중앙집중화된 관리를 제공하는 일종의 Clean-State 접근법에 입각한 네트워크 구조에 관련된 새로운 패러다임이다. SDN이 가지는 장점으로는 네트워크의 관리 구조를 단순하게 구성할 수 있다는 점이며, 이와 더불어 programmability, 유연성, 탄력성들을 네트워크에 부여할 수 있게 된다. 이를 통해 네트워크 사업자는 기존의 비싼 네트워크 구성 장비들을 저렴하게 구입하고 유지함으로써 CAPEX/OPEX를 줄이는 결과를 가져올 것으로 기대하고 있다.

하지만, SDN 기술은 기존에 존재하는 IP 기반의 장치 및 네트워크 기반 구조에 점진적으로 배포가 가능한 구조를 지닌 것이 아니라, 전체 네트워크를 새롭게 구성하는 Clean State 접근 방식을 취하고 있다. 따라서 SDN을 구성하기 위해서는 OpenFlow를 지원하는 SDN 스위치와 컨트롤러가 필요하다. 하지만 현재 인터넷을 구성하고 있는 기존 네트워크 장비들은 OpenFlow를 지원하지 않으므로, OpenFlow를 사용하는 SDN으로 인터넷을 대체하는데 문제가 있다. 더불어, OpenFlow 또한 L2/L3의 영역에만 작동한 다는 한계를 지니고 있다.

따라서 기존 인터넷과 SDN이 공존하면서 SDN의 영역을 차츰차츰 넓혀감으로써 기존 인터넷을 SDN으로 전환하기 위한 기술이 필요하다. 따라서 현재 IP 기반의 네트워크 기술들이 가지고 있는 필수 서비스를 SDN 환경으로 넓히기 위해, MPLS, PCEP, BGP, VPN, Multicast와 같은 네트워크 서비스를 이전 할 수 있는 방법이 필요하다.

본 기술문서는 SDN 패러다임과 유사한 구조를 지니며, 현재 인터넷이 가지고 있는 확장성 문제를 극복하기 위해 제안된 LISP (Locator Identifier Separation Protocol)을 대표적인 SDN 컨트롤러인 ONOS (Open Networking Operating System) 상에서 LISP을 지원할 수 있도록 하는 설계 및 구현을 서술한다.

Contents

K-ONE #12. ONOS 상 LISP Provider 설계 및 구현

1. 서론 ····· 1
2. LISP 프로토콜 ····· 3
2.1. LISP 프로토콜 개요 ····· 3
2.2. LISP 용어 정리 ····· 4
2.3. LISP의 동작 원리 ····· 6
2.4. LISP 도입 효과 ····· 10
3. 소프트웨어 정의 네트워크 (Software Defined Network, SDN) ····· 13
3.1. SDN 등장 배경 ····· 13
3.2. SDN의 개념 ····· 14
3.3. SDN 도입 효과 ····· 16
3.4. SDN 핵심 구성 요소 ····· 17
4. ONOS (Open Network Operating System) ····· 21
4.1. ONOS 소개 ····· 21
4.2. ONOS의 설계 구조 ····· 22
4.3. ONOS SBI 확장의 필요성 ····· 24
4.4. ONOS 설치 방법 ····· 25
5. LISP Controller 설계 및 구현 ····· 32
5.1. LISP Controller 설계 ····· 32
5.2. LISP Controller 구현 ····· 34
6. 결론 ····· 39
7. Reference ····· 40

그림 목차

그림 1 DNS lookup과 LISP 매핑 시스템 비교 ····· 7
그림 2 LISP Map-Register 패킷 구조 (출처: RFC6830) ····· 8
그림 3 LISP Map-Request 패킷 구조 (출처: RFC6830) ····· 9
그림 4 LISP Map-Reply 패킷 구조 (출처: RFC6830) ····· 9
그림 5 LISP IPv4 헤더 (출처: CiscoLive2013 Presentation: "TECIPM-3191") ····· 10
그림 6 LISP 패킷 전달 과정 ····· 10
그림 7 SDN의 개념구조 ····· 15
그림 8 SDN 개방형 인터페이스의 뷴류 ····· 19
그림 9. ONOS의 분산 구조 ····· 21
그림 10. ONOS의 계층 구조 ····· 22
그림 11 다양한 ONOS의 서브시스템(서비스)들 ····· 23
그림 12 ONOS 서브시스템 구조 ····· 23
그림 13 LISP 컨트롤러의 개념 ····· 32
그림 14 LISP 컨트롤러의 설계 ····· 33
그림 15. LISP 프로바이더와 프로토콜 경로 ····· 35
그림 16 LISP contoller 응용 구조 ····· 36
그림 17 LISP 컨트롤러의 빌드 결과 ····· 36
그림 18 ONOS Controller의 설치 ····· 37
그림 19 ONOS Contoller 구동 ····· 37
그림 20 LISP controller의 동작 확인 ····· 37
그림 21. LISP 실행 시 로그 내용 ····· 38

K-ONE #12. ONOS 상 LISP Controller 설계 및 구현

1. 서론

최근 소프트웨어 정의 네트워크 (Software-Defined Networking; SDN) 기술들이 네트워킹 관련 연구자들, 개발자들, 제조사, 표준화 기구, 대학들로부터 매우 높은 주목을 받고 있다. SDN의 기본 철학은 네트워크의 제어 평면과 데이터 평면을 분리하는 것으로, 중앙 집중화된 관리를 제공하는 일종의 Clean-State 접근법에 입각한 네트워크 구조에 관련된 새로운 패러다임이다. SDN에서는 컨트롤러라고 불리는 중안 집중화된 제어 장치로부터 데이터 전달 계층에 위치한 스위치 또는 라우터와 같은 장치들을 원격으로 제어 할 수 있으며, 컨트롤러는 다양한 응용들을 개발하기 위한 API들을 제공할 수 있다. 이러한 구조는 기존의 IP 기반의 네트워크의 닫힌 구조에서는 불가능했던 방식으로써 개방형 구조로 네트워크를 구성함으로써 각 사용자가 원하는 응용들을 자유롭게 개발 할 수 있는 환경을 제공할 수 있게 되었다는데 가장 큰 의의를 가진다. 더불어, 이러한 구조와 응용에 기반하여 기존의 정적인 네트워크 관리 체계에 유연성 및 새로운 서비스의 빠른 개선 및 배포를 실현 할 수 있어 다양한 사업자들로부터 높은 관심을 받고 있다.

SDN이 가지는 장점으로는 네트워크의 관리 구조를 단순하게 구성할 수 있다는 점이며, 이와 더불어 programmability, 유연성, 탄력성들을 네트워크에 부여할 수 있게 된다. 이를 통해 네트워크 사업자는 기존의 비싼 네트워크 구성 장비들을 저렴하게 구입하고 유지함으로써 CAPEX/OPEX를 줄이는 결과를 가져올 것으로 기대하고 있다.

하지만, SDN 기술은 기존에 존재하는 IP 기반의 장치 및 네트워크 기반 구조에 점진적으로 배포가 가능한 구조를 지닌 것이 아니라, 전체 네트워크를 새롭게 구성하는 Clean State 접근 방식을 취하고 있다. 따라서 SDN을 구성하기 위해서는 OpenFlow를 지원하는 SDN 스위치와 컨트롤러가 필요하다. 하지만 현재 인터넷을 구성하고 있는 기존 네트워크 장비들은 OpenFlow를 지원하지 않으므로, OpenFlow를 사용하는 SDN으로 인터넷을 대체하는데 문제가 있다. 이러한 상황에서 각 인터넷 또는 네트워크 서비스 제공자는 기존의 네트워크에 배포되어 있는 각종 서비스와 IP 네트워크의 모든 설정들을 문제 없이 SDN 기반의 네트워크로 이전 할 수 있는 방법이 필요하다. 아직 까지 SDN은 연구수준 또는 데이터센터와 같은 닫힌 환경의 네트워크에 적용되고 있는 수준이며, 전체 네트워크 구조를 SDN기반으로 전환하기 위해서는 해결해야할 문제가 많이 있다. 더불어, 현재 대표적인 SDN 프로토콜로 각광 받고 있는 OpenFlow는 모든 IP 네트워크의 기능을 제공하는데 한계가 있다.

따라서 기존 인터넷과 SDN이 공존하면서 SDN의 영역을 차츰차츰 넓혀감으로써 기존 인터넷을 SDN으로 전환하기 위한 기술이 필요하다. 따라서 현재 IP 기반의 네트워크 기술들이 가지고 있는 필수 서비스를 SDN 환경으로 넓기히 위해, MPLS, PCEP, BGP, VPN, Multicast와 같은 네트워크 서비스를 이전 할 수 있는 방법이 필

요하다. 본 기술 문서에서는 SDN과 유사하게 제어평면과 데이터 평면이 분리된 구조를 가지고 있으며, 현재 인터넷이 가지고 있는 확장성 문제를 극복하기 위해 제안된 LISP을 SDN환경에 활용하기 위한 설계와 구현에 대해서 서술한다.

더불어 LISP이 가지고 있는 장점을 활용하여 새로운 형태의 응용들을 개발 할 수 있는 가능성을 높일 수 있다. LISP은 Vertical Handover, VM live migration, 인입 트래픽 제어, 재난 복구와 같은 다양한 네트워크 기반 서비스들을 제공할 수 있으며 그 효율성이 검증 되었다. 따라서 SDN 환경에서 이러한 LISP 기반의 서비스를 제공함으로써 SDN의 배포 속도를 가속화 할 수 있을 것으로 예상된다.

본 기술문서는 대표적인 SDN 컨트롤러인 ONOS (Open Networking Operating System) 상에서 LISP을 지원할 수 있도록 하는 설계 및 구현을 포함한다. 이후의 구성은 LISP 소개, SDN 소개, ONOS 소개, LISP 서브 시스템의 설계 및 구성으로 되어 있다.

2. LISP 프로토콜

본 절에서는 LISP(Locator Identifier Separation Protocol) 이 등장한 배경과 필요성, 용어, 동작 원리, 도입 효과에 대해 설명한다.

2.1. LISP 프로토콜 개요

국내외 인터넷 트래픽은 매년 40% 이상 증가하고 있다. 이러한 트래픽의 증가 추세는 스마트 디바이스의 보급, 클라우드 기반 서비스의 증가, 그리고 사물 인터넷(IoT: Internet of Things)의 기술 등장으로 인한 것으로 분석 할 수 있다. 모바일 트래픽의 급속한 증가, 멀티미디어 스트리밍과 IPTV와 같은 높은 대역폭을 필요로 하는 서비스의 확산, 클라우드 컴퓨팅을 기반으로 한 새로운 서비스들(IP-CCTV 감시 서비스 등)의 등장으로 인해 인터넷 트래픽 증가 추세는 더욱 가속될 것으로 예측된다. 하지만 증가하고 있는 트래픽 수용 능력과 관련하여 현재 인터넷 구조의 확장성에 대한 지속적인 문제 제기가 이루어지고 있다.

현재 인터넷의 구조가 당면하고 있는 가장 큰 문제는 라우팅 테이블 크기가 증가하는 것으로, 2014년 현재 백본 라우터 FIB(Forwarding Information Base)의 BGP 엔트리 개수가 50만개를 넘어섰으며, 기하급수적으로 증가하는 추세를 보이고 있다 [1]. FIB 크기의 증가는 라우터 하드웨어 규모와 투자비용의 증가를 초래한다는 근본적인 문제를 야기한다. 이 외에도 multi-homing, 도메인 간 트래픽 엔지니어링 등의 문제가 부각되고 있다. 이러한 문제의 큰 원인 중 하나는 IP주소만을 사용하는 현재 인터넷의 단일 주소 체계에 있다. IP 주소를 단말 간 연결에서의 단말 식별자로 사용함과 동시에 네트워크 라우팅에 사용함으로써 IP주소에 부여되는 의미가 이중적으로 사용되어 발생하는 문제이다.

미국과 유럽, 일본의 통신사업자와 벤더들은 미래 인터넷 연구 과제의 핵심기술의 하나로 구현된 LISP(Locator Identifier Separation Protocol) [2]를 이용하여, 현재 IP주소를 식별 및 위치 주소로 병용함으로써 발생하는 확장성 문제를 해결하고 기존의 네트워크 서비스를 보다 낮은 원가로 개발 및 운용할 수 있도록 하는 시도를 하고 있다. LISP는 기존 인터넷의IP 주소를 사용한 단일 주소 체계를 단말 식별자(EID: Endpoint Identifier)와 위치자(RLOC: Routing Locator)로 구분하여 확장성 문제를 해결하고자 하는 것으로서, LISP를 사용하면 새로운 서비스의 배포를 촉진할 수 있을 뿐만 아니라 지속적으로 증가할 것으로 예상되는 인터넷 트래픽을 비용 대비 효율적으로 수용할 수 있을 것으로 기대된다.

LISP 프로토콜이 기반하고 있는 식별자/위치자의 분리 패러다임은 인터넷에서 발생하는 암호화 같은 국지적인 문제를 해결하고자 등장하였다. 이러한 IP 주소의 역할 분리 패러다임을 구현 하는 방법에 관하여 많은 논의가 있었으며, 현재는 터널링 (Tunneling) 기법을 사용하여 구현하는 것으로 표준화가 제정되었다. 현재 널리 알려진 LISP 프로토콜은 2007년에 Cisco에 의해 제안된 프로토콜로, 인터넷의 라우팅 확장성 문제를 해결하는 것이 주요 목적이었다.

LISP는 네트워크 기반의 솔루션으로, map-and-encap 을 통해 IP 주소의 역할을 위치자(RLOC)와 식별자(EID)로 구분한다. 이 중 RLOC만 BGP FIB에 등록되며, BGP 테이블 크기의 증가를 유발하는 EID (Edge network의 IP prefix)는 BGP에서 사용되지 않기 때문에 FIB 급증 문제를 해결할 수 있다.

발신 단말에서 송신된 패킷은 Edge 라우터(ITR: Ingress Tunnel Router)에서 LISP 헤더를 추가하여 encapsulation을 수행한다. 인터넷 백본에서는 LISP 헤더 정보만으로 라우팅이 수행되며 수신 측 Edge 라우터 (ETR: Egress Tunnel Router)에서 LISP 헤더를 제거한 후 EID를 사용하여 수신 단말로 패킷을 전달한다. EID의 RLOC을 알아내기 위해서는 EID와 RLOC 의 매핑 정보를 관리하는 시스템이 필요하다. 기존의 라우팅 시스템에서는 모든 라우팅 정보를 proactive하게 전달하는 반면, LISP에서는 DNS와 유사한 방법을 사용하여, 특정 EID에 대해 매핑 시스템으로 쿼리를 전송하면 해당 EID에 관련된 매핑을 전달하는 구조를 가진다. 매핑 시스템은 LISP-DDT(Delegated Database Tree)를 사용하여 Instance ID와 EID를 계위적인 트리 구조로 관리한다. 또한 매핑 시스템의 front-end인 Map-Server/Map-Resolver를 통해 data-plane(e.g., 패킷 encapsupation/decapsulation, 패킷 포워딩)과 control-plane(e.g., map registration, distribution)이 분리되어 있어 SDN과 유사한 구조를 보이고 있다.

현재 LISP의 표준화는 완료단계에 있지만 이를 이용한 애플리케이션 개발은 초기 단계에 머물러 있다. LISP 관련 연구를 통해 LISP 이 현재 인터넷 구조(장치 및 프로토콜)이 가지고 있는 확장성 문제를 극복하고, 더불어 Multi-homing, 사설망의 ISP 변경, IPv6 전환, 데인터 센터간의 VM Live Migration, 이동통신 장치의 Vertical Handover, 인입 트래픽 엔지니어링과 같은 유용한 유즈 케이스 및 응용을 개발하는데 활용할 수 있음을 보이고 있다. 실제로 LISP을 사용하여 Facebook, Cisco, IBM, Verisign, NTT와 같은 대형 글로벌 회사에서 자사의 네트워크에 부분적으로 LISP을 사용하여 IPv4네트워크와 IPv6 네트워크 사이의 연결성 확보, 호스트의 이동성 지원, 그리고 데이터 센터의 이전에 LISP 사용했다고 보고 하고 있다. 하지만, 아직까지 LISP의 실제 인터넷 환경에 배포는 요원하다.

LISP은 앞서 서술하였듯이 소프트웨어 정의 네트워크와 유사하게 제어평면과 데이터 평면을 분리하여 가지고 있다. 이러한 구조를 활용하여, SDN 기반 네트워크의 South Bound API (SBI)로써 LISP을 구현 할 수 있다. 본 문서는 LISP이 가지는 장점들을 SDN 환경 내에서 활용할 수 있도록 하며, LISP에 기반한 서비스들의 개발 및 배포를 촉진 시킬 수 있도록 LISP을 SDN의 SBI로써 작동 할 수 있도록 하기 위한 설계와 구현을 서술 한다.

2.2. LISP 용어 정리

- EID (Endpoint ID)

LISP 사이트(고객 사이트 또는 액세스 네트워크) 내에서 단말이 연결되는 네트워크 인터페이스 또는 단말에 할당되는 식별자이다.

- RLOC (Routing Locator)

LISP 터널 라우터에 할당된 주소로서 캡슐화된 LISP 패킷(고객 단말에서 송신한 패킷)을 목적지 EID를 가진 LISP 사이트의 LISP 터널 라우터까지 전달하기 위해 코어 네트워크 내에서 라우팅하는 데 사용된다.

- MS (Map-Server)

EID-RLOC 매핑을 관리하는 주체로, 특정 EID-prefix를 가지고 있는 ETR(Egress Tunnel Router)에 대한 정보를 관리한다. 각 ETR은 주기적으로 Map-register 메시지를 MS로 전송하여 ETR이 관리하는 EID에 대한 정보와 ETR이 가진 RLOC 목록을 MS에 등록하고 갱신한다. MS가 Map-Request 를 받으면 데이터베이스로부터 해당 EID-prefix를 관리하는 ETR을 찾아내어 해당 ETR로 Map-Request 메시지를 전달한다.

- MR (Map-Resolver)

ITR이 착신 측 EID에 대한 위치 정보(RLOC)를 MS에서 검색하는 데 사용되는 장치로서, ITR로부터 EID lookup request (Map-Request)를 받으면 매핑 데이터베이스를 참조하여 그 정보를 가진 MS에게 Map-Request를 전달한다.

- ITR (Ingress Tunnel Router)

LISP 사이트와 인터넷의 경계에 설치되며 호스트로부터 IP 패킷을 전달받아 encapsulation을 수행한 후 착신지로 전송하는 기능을 수행한다. 호스트로부터 전달받은 IP 패킷의 목적지 IP 주소를 착신 EID로 간주하고 MS로부터EID-RLOC 매핑을 찾는다. EID에 매핑된 RLOC을 찾으면 이를 목적지 IP 주소로 변경(encapsulation)하여 패킷을 전송한다.

- ETR (Egress Tunnel Router)

LISP 사이트와 인터넷의 경계에 설치되며 encapsulation된 LISP 패킷을 전송 받은 후 encapsulation된 LISP 헤더를 제거하여 원래의 IP 패킷을 복원 후, 내부 네트워크로 전달한다. 일반적으로 ITR과 ETR은 하나의 장치로 구현된다.

- PITR (Proxy ITR)

LISP 사이트와 non-LISP 사이트 간에 연결성을 제공해 준다. LISP EID 주소 공간을 인터넷 DFZ (default-free zone)로 advertise하여 non-LISP 트래픽을 LISP 사이트로 전달해 주는 기능을 수행한다.

- PETR (Proxy ETR)

LISP 사이트로부터 non-LISP 사이트로 패킷을 전달할 수 있도록 해 준다. LISP 사이트의 ITR은 PETR로 LISP encapsulated 패킷을 전달하고, PETR에서 decapsulation을 수행 후 non-LISP 사이트로 패킷을 전달한다.

- RTR (Re-encapsulating Tunnel Router)

RTR은 직접적인 연결이 불가능한 LISP 사이트를 연결하는데 사용되며, LISP 사이트들 사이에 고정점(Anchor point)역할을 수행한다. 주된 기능으로 들어오는 LISP을 받아 목적지에 맞도록 LISP헤더정보를 갱신하는 역할을 수행한다.

- xTR

패킷 데이터 흐름의 방향이 구분될 필요가 없는 경우, ITR과 ETR을 통칭하여 xTR이라 한다.

- PxTR

xTR과 마찬가지로, 패킷 데이터 흐름의 방향이 구분될 필요가 없는 경우 PITR과 PETR을 통칭하여 PxTR이라 한다.

2.3. LISP의 동작 원리

- Mapping System

LISP를 이용하여 구성된 네트워크의 구성요소들은 식별자(EID) 정보와 위치자(RLOC)의 매핑 정보를 필요로 한다. 이러한 정보 제공의 역할은 Mapping System이 담당한다.

ITR이 ETR로 LISP 패킷을 전송하기 위해서는 매핑 시스템으로부터 목적지 EID에 해당하는 RLOC 정보를 받아와야 한다. 매핑 시스템은 하나의 EID prefix에 대해 <RLOC, priority, weight>로 구성된 EID-RLOC 매핑 엔트리 리스트를 가지고 있다. Priority와 weight값은 각 RLOC마다 지정되며, ITR이 여러 개의 RLOC 중 priority 값이 가장 큰 RLOC을 선택한다. Priority 값이 같은 경우, weight값을 사용하여 priority가 같은 RLOC들로 부하분산(Load balancing)을 수행한다.

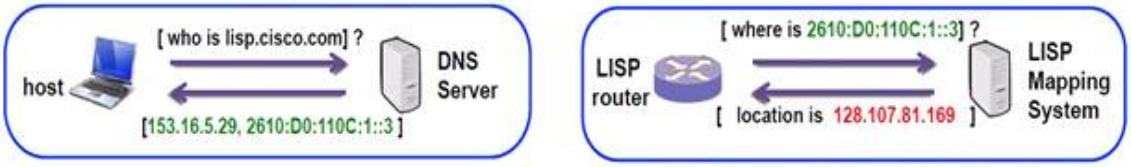

그림 1 DNS lookup과 LISP 매핑 시스템 비교

LISP 매핑 시스템의 동작 원리는 DNS와 유사하다 (그림 1). 호스트는 특정 도메인 네임에 대한 IP 주소를 알아내기 위해 DNS에 요청을 보낸다. 이와 유사하게 LISP xTR도 특정 EID에 대한 RLOC 정보를 알아내기 위해 LISP 매핑 시스템에 요청을 보내게 된다. LISP Mapping System의 구현으로 LISP ALT [3], LISP DDT [4] 등이 있다.

- Map Registration

LISP ETR은 Map-Register 메시지(그림 2)를 MS로 전송함으로써 로컬 네트워크의 EID prefix를 등록한다. Map-Register 메시지에는 인증 데이터가 포함되어야 하며, 이를 위해 사전에 ETR과 MS는 비밀 키 혹은 다른 인증 정보를 공유하고 있어야 한다. 또한, MS는 각 EID prefix를 등록 가능한 ETR의 목록을 가지고 있어, 해당 ETR로부터 받은 Map-Register 메시지만 수용한다. 이러한 절차를 통해 EID prefix 하이재킹 공격을 예방할 수 있다.

Map-Register 메시지는 ETR로부터 MS로 주기적으로 전송되며, 권장 주기는 1분이다. 3분간 Map-Register 메시지를 받지 못하면 MS는 등록된 해당 EID prefix를 제거한다.

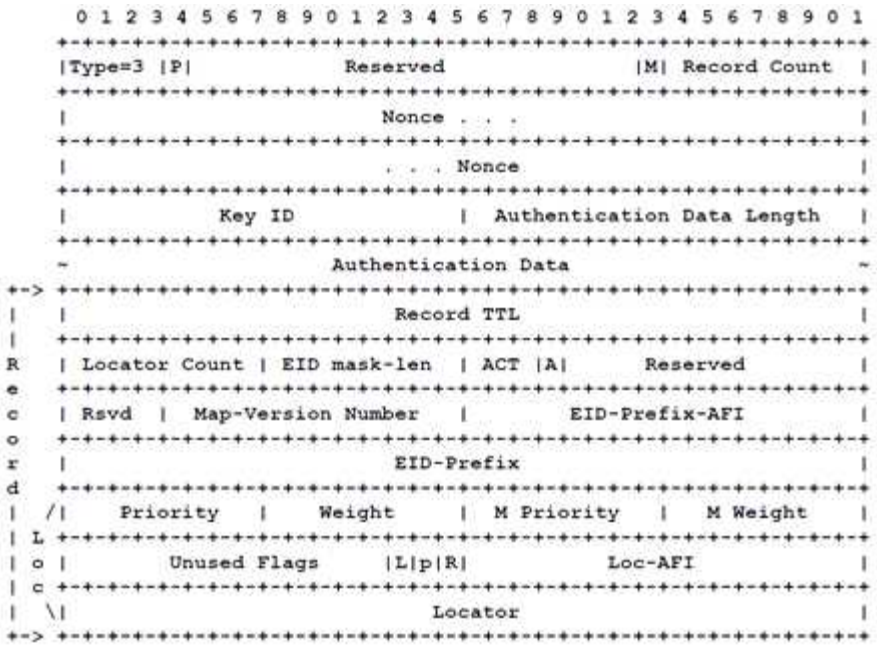

그림 2 LISP Map-Register 패킷 구조 (출처: RFC6830)

- Map Request

목적지 주소가 외부 EID인 패킷이 ITR로 전달되면, ITR은 우선 매핑 캐시를 참조한다. 매핑 캐시에 해당 EID에 대한 매핑 정보가 존재하지 않는 경우, MR로 해당 EID에 대한 매핑 정보를 요청하는 Map-Request 메시지(그림 7)를 전송한다. Map-Request 메시지는 MR에 저장된 등록 정보를 바탕으로 해당 EID를 관리하는 ETR로 전달된다. ETR에는 EID에 대한 정보가 매핑 데이터베이스에 저장되어 있기 때문에 ITR로 Map-Reply 메시지(그림 8)를 전송하여 해당 EID에 도달할 수 있는 RLOC 정보를 알려준다. 최종적으로 EID-RLOC 매핑이 ITR의 매핑 캐시에 저장된다.

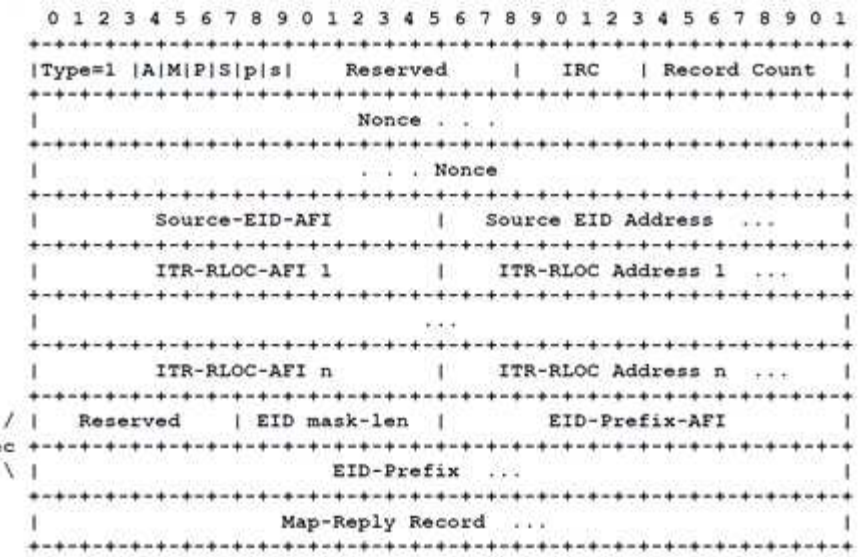

그림 3 LISP Map-Request 패킷 구조 (출처: RFC6830)

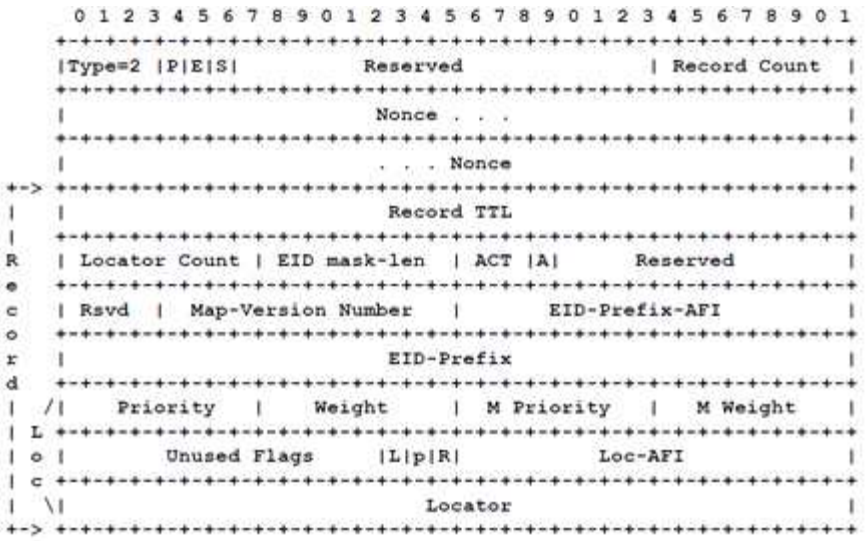

그림 4 LISP Map-Reply 패킷 구조 (출처: RFC6830)

- Dynamic Encapsulation

그림 5는 ITR에서 encapsulation된 IPv4 패킷의 LISP IPv4 헤더를 나타낸 것이다. Outer header의 송신지 IP주소와 목적지 IP 주소에는 각각 ITR의 RLOC과 ETR의 RLOC이 사용된다. UDP 헤더의 소스 포트는 원본 패킷의 소스 포트를 그대로 사용하며, 목적지 포트는 well-known LISP UDP 포트인 4341을 사용한다.

Encapsulation 과정을 통해 각 패킷마다 36 바이트 (IPv4 헤더 20 바이트 + UDP 헤더 8 바이트 + LISP 헤더 8 바이트)가 추가된다.

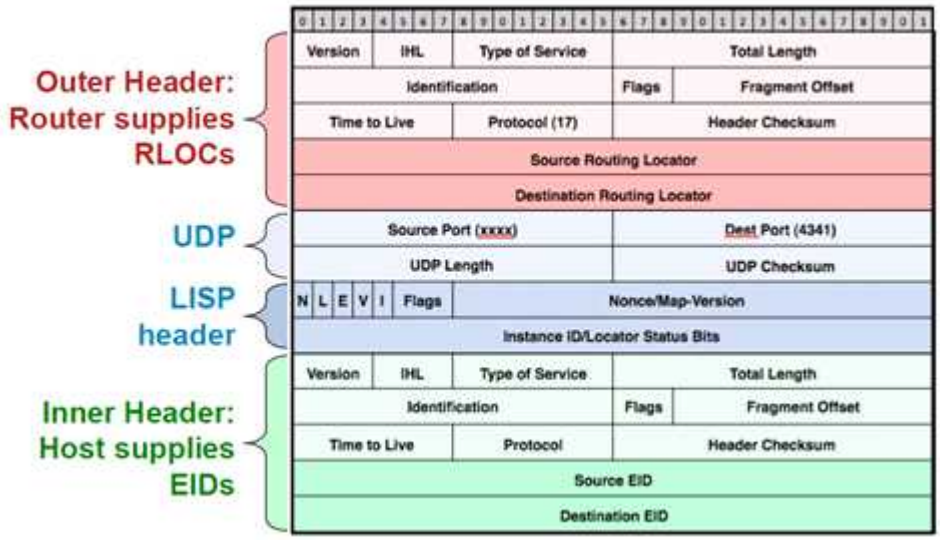

그림 5 LISP IPv4 헤더
(출처: CiscoLive2013 Presentation: "TECIPM-3191")

그림 6은 LISP의 앞서 설명한 encapsulation 을 사용하여 서로 다른 LISP 사이트에 있는 두 호스트 S와 D 간에 데이터 패킷을 전달하는 과정을 나타낸 그림이다. S에서 D로 보내는 패킷의 송신자 IP 주소는 S의 EID이며, 수신자 주소는 D의 EID를 사용한다. 패킷이 ITR S2로 도달하면, S2는 매핑 엔트리로부터 D의 EID에 도달할 수 있는 RLOC정보를 확인하고 ETR D1의 RLOC을 사용하여 encapsulation을 수행한다. Encapsulation된 LISP 패킷은 인터넷을 거쳐 D1까지 도달하며, D1에서는 LISP 헤더를 제거한 후 (decapsulation) D의 EID를 사용하여 패킷을 전달한다.

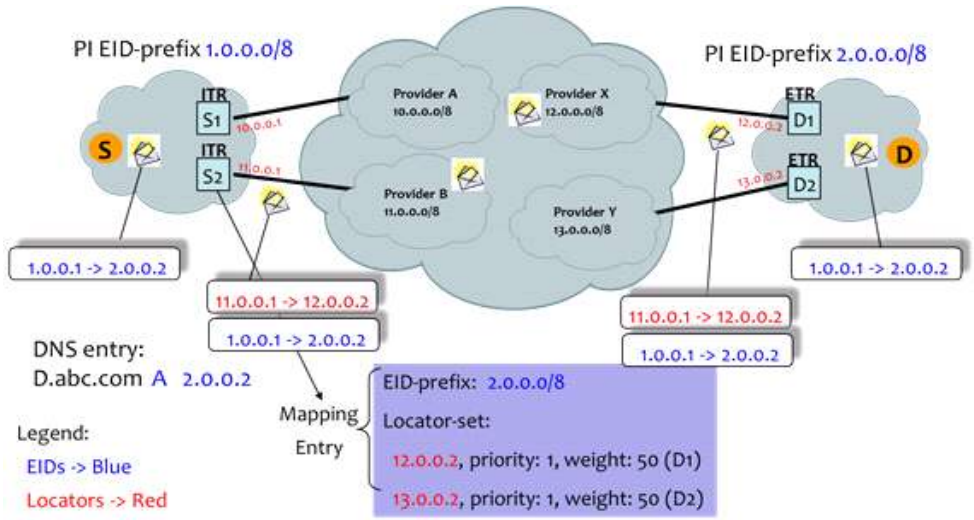

그림 6 LISP 패킷 전달 과정

2.4. LISP 도입 효과

LISP의 가장 큰 장점 중 하나는 도입 과정에서 점진적인 확산이 가능하다는 장점이 있다. IPv6, Multicast, MPLS 등의 프로토콜을 도입하기 위해서는 백본 네트워크 전체를 동시에 변경하여야 하지만, LISP를 도입하기 위해서는 LISP 네트워크를 제어하는 매핑 시스템과 LISP를 도입하고자 하는 액세스 네트워크의 Edge 라우터만 변경하면 된다. LISP 도입을 위해 기존 네트워크를 구성하는 코어 라우터나 사용자 단말을 변경할 필요가 없기 때문에 네트워크 변경을 최소화할 수 있으며 도입 비용도 크게 줄일 수 있다.

LISP를 도입할 경우, 다음과 같은 도입 효과를 기대할 수 있다.

- 코어 라우터의 RIB 크기 감소

LISP를 사용함으로 인해 다양한 EID 주소 블록은 코어 라우터로 유입되지 않으며, RLOC 주소공간만 코어에서 사용하게 된다. 따라서 다양한 IP 주소 블록의 사용으로 인해 집적화되지 않은 주소가 RIB를 점유하는 문제를 해결하고 토폴로지 기반의 집적을 통해 라우팅 테이블 엔트리 증가를 줄일 수 있다.

- 원격지 데이터센터 간의 VM 마이그레이션 지원

LISP를 지원하는 스위치를 사용하면 데이터센터 내부에서뿐만 아니라 데이터센터 간의 자유로운 VM 이동이 가능해진다. VM 이동이 탐지되면 해당 VM으로 전송되던 패킷은 수정 없이 이동된 VM으로 전달된다. 그리고 triangle routing을 방지하기 위해 새로운 VM 위치 정보가 매핑 시스템과 xTR및 PxTR로 전달된다. LISP에서는 항상 패킷을 encapsulation하기 때문에 VM에서는 IP를 변경할 필요가 없고, 마이그레이션 전후에 끊김 없는 전송이 가능해진다.

- Mobile-IP (vertical handover) 지원

무선 노드가 WiFi – 3G/LTE 네트워크 사이를 이동하는 경우, 무선 노드는 새로운 네트워크에서 새로운 연결성을 다시 얻게 되고, 무선 노드는 새로운 RLOC 정보를 얻어 EID-to-RLOC 바인딩(binding)을 매핑 서버에게 알려준다. 무선 노드는 Solicit-Map-Request (SMR)이라는 특별한 시그널링 메시지를 보냄으로써 통신하고 있는 라우터 또는 노드들의 매핑 캐시에 저장된 모든 바인딩을 업데이트하게 된다. SMR 메시지의 수신은 바인딩을 새로 갱신하기 위해 Map-Request를 트리거하게 된다. 이 후, 무선 노드로 향하는 패킷들은 새롭게 갱신된 캐시 정보를 이용하여 버티컬 핸드오버가 일어나더라도 목적지까지 정확하게 도달할 수 있게 된다.

- 효율적인 Disaster Recovery

특정 Application 서버(웹, 스트리밍 등)에 장애가 발생한 경우 관리자는 이를 검출하여 즉시 트래픽의 경로를 다른 지역에 설치된 백업 서버로 redirection 가능하

다.

- Traffic Engineering 실현

LISP를 이용하면 Best effort 네트워크에서 트래픽의 부하 분산이 가능해진다. 즉, Ingress Router에서 백본 네트워크를 통해 전달되는 트래픽을 관리자가 지정하는 임의의 비율로 나누어 전달할 수 있다.

- Dual stack을 사용하지 않는 IPv6 전환 및 연동

LISP에서는 EID에서 사용하는 주소 체계와 RLOC에서 사용하는 주소 체계가 독립적으로 구성되므로, 이러한 특성을 네트워크의 IPv6전환에 활용할 수 있다. 예를 들어, EID가 IPv6, RLOC을 IPv4를 사용하도록 LISP 네트워크를 구성하는 경우, IPv6를 사용하는 ISP에 가입하지 않아도 IPv6 단말 간 연결이 가능하다.

- 네트워크 프로그래밍 지원 (Network Programmability)

LISP를 지원하는 스위치나 라우터 등의 하드웨어는 LISP의 control-plane인 매핑 시스템에 의해 관리되며, 매핑 시스템을 통해 programmability가 지원된다. 따라서 LISP를 SDN/NFV 솔루션으로 사용할 수 있는 가능성을 지니고 있다. 또한 매핑 시스템이 표준화된 인터페이스를 지원하기 때문에 3rd party 벤더들이 개발에 참여할 수 있다.

- LISP 기반 Simple/Low-cost 네트워크 구축을 통한 CAPEX 절감

LISP를 이용하여 현재의 인터넷이 가지고 있는 근본적인 문제를 해결하고, 기존의 네트워크 서비스를 보다 낮은 원가로 개발하여 제공할 수 있다. 이를 통해 CAPEX를 최소화할 수 있다.

3. 소프트웨어 정의 네트워크 (Software Defined Network, SDN)

본 절에서는 SDN이 등장한 배경, 필요성, 개념, 도입효과, 용어에 대해 설명한다.

3.1. SDN 등장 배경

소프트웨어로 네트워크를 제어하는 기술은 1980년대에 등장한 지능망(IN) 기술에 최초 도입된 이후, 점차 진화하여 현재의 인터넷 기반 SDN으로 발전한 것으로 인식되고 있다. 1980년 대에 AT&T에서 No.7 프로토콜을 이용한 800 서비스를 개발하면서 No.7 신호교환기가 고객의 위치 정보 DB를 조회하여 800 서비스 호의 경로를 제어하도록 한 것에서 비롯된다. 즉, 신호교환기가 제어평면 기능을 수행하고 전화교환기는 데이터평면 역할을 한 것이다 [5]. 이후 90년 대에는 ATM 기술이 붐을 이루면서 Programmable Network에 관한 연구가 많이 수행되었다. 대표적인 것으로 케임브리지 대학의 Tempest 프로젝트가 있다 [6]. Tempest는 ATM 스위치의 자원을 논리적으로 분할하여 외부에서 동적으로 입력하는 프로그램에 의해 서비스가 정의되도록 하는 것이었다. 프로그램에 의한 서비스 제어에 제어평면과 데이터(포워딩 또는 전달) 평면의 분리는 필수적이었다.

이러한 연구를 바탕으로 2003년에 IETF에 forCES(forwarding and Control Element Separation)라는 워킹 그룹이 만들어졌다. 이는 NGN 환경에서 이종망간의 확장성을 제공하기 위해 전달 평면과 제어 평면을 논리적으로 분리하여, 독립적으로 표준화 및 기술 개발이 진행되도록 하면서 이들 간의 연동 구조 및 시그널링 방법에 대한 표준화를 추진하기 위한 것이다. forCES는 IP 제어평면과 포워딩 평면의 분리를 위한 RFC 3654(requirement)를 발표한 이후 현재까지 IP 포워딩, IntServ, DiffServ QoS를 제어대상으로 표준화를 추진하고 있다. forCES도 궁극적으로는 SDN의 실현을 목표로 하고 있다.

ITU-T는 미래 패킷 기반 네트워크 환경에서 향상된 품질의 멀티미디어 서비스를 제공하기 위해 2008년 9월 SG13에 워킹 그룹을 만들고 전송 계층과 제어 계층을 분리한 iSCP(independent Scalable Control Plane) 구조를 제안하고 있다. ITU-T는 논리적인 분할을 위해 IEFT의 forCES 프로토콜을 채택하고 있으며 iSCP의 구조는 Y.2622로 발표되어 있다[7].

2006년에는 FIND(Future INternet Design) 프로그램의 일부로 스탠포드와 버클리 대학에서 SANE(clean-slate Security Architecture for Enterprise Network)과 Ethane 프로젝트가 수행되었다. SANE은 현재의 IP 망에서 필수 요소가 되어버린 Box 형태의 VLAN, ACL, 방화벽, NAT 등이 수행하는 네트워크 보호 기능을 전체 망의 차원에서 집중화하여 처리하는 기술이다. 이를 위해 DC(Domain Controller)라는 분리된 보안 장치를 두고, 호스트들이 우선적으로 DC에 접속하도록 IP 헤더와 Ethernet 헤더 사이에 SANE 헤더를 추가하였다. DC는 접속인증, ACL 및 DNS 처리 등을 수행한다. Ethane은 SANE의 후속 과제로서 SANE 호스트에 설치되어야 하는

proxy를 제거하여 단순화 시킨 것이다. SANE과 Ethane의 성공은 2007년 스탠포드 대학의 OpenFlow 프로젝트로 이어졌다[8]. 이후 OpenFlow 프로젝트에 벤더들이 참여하면서 시제품이 만들어졌고, 미래인터넷 연구(GENI, FIND 등)에 참여하는 대학을 중심으로 연구개발이 확산되었으며. 2008년에는 Nicira의 OpenFlow Controller인 NOX가 오픈 소스로 공개되었고 2012년 1월에는 BigSwitch의 Floodlight가 역시 오픈소스로 공개되었다.

2011년 3월에는 OpenFlow 기술의 상용화를 촉진시키기 위해서 표준화 단체인 ONF(Open Networking foundation)가 만들어 졌다. ONF는 Deutsche Telekom, Facebook, Google, Microsoft, Verizon, Yahoo! 에 의해 설립되었으며, OpenFlow 기술을 SDN 기술로 확장하여 표준화를 추진하고 있다 [9].

3.2. SDN의 개념

SDN은 제어평면과 전달평면을 분리하는 개념이다. 일반적으로 SDN과 OpenFlow [10]가 밀접한 관계인 것으로 알려져 있지만 SDN은 그 하부 기술로 OpenFlow만을 한정하지는 않는다. SDN은 훨씬 더 큰 개념으로 네트워크 구조 혹은 새로운 패러다임이며, OpenFlow는 SDN을 위한 인터페이스 기술의 하나이다.

현재 SDN 표준화를 추진하고 있는 ONF가 SDN을 바라보는 관점은 크게 두 가지의 기본적인 원칙을 바탕으로 하고 있다. 첫째로, SDN은 소프트웨어 정의 포워딩(Software Defined Forwarding)을 해야 한다. 이것은 스위치/라우터에서 하드웨어가 처리하는 데이터 포워딩 기능은 반드시 개방형 인터페이스와 소프트웨어를 통해서 제어되어야만 한다는 것을 의미한다. 둘째는 SDN이 글로벌 관리 추상화(Global Management Abstraction)를 목표로 한다는 것이다. SDN은 추상화를 통해 보다 진보된 네트워크 관리 툴이 개발될 수 있도록 해야 한다. 예를 들면 이런 추상화 도구들은 전체 네트워크의 상태를 보면서 이벤트(토폴로지 변화나 새로운 플로우 입력 등)에 따른 반응, 그리고 네트워크 요소를 제어할 수 있는 기능 등을 포함할 수 있다.

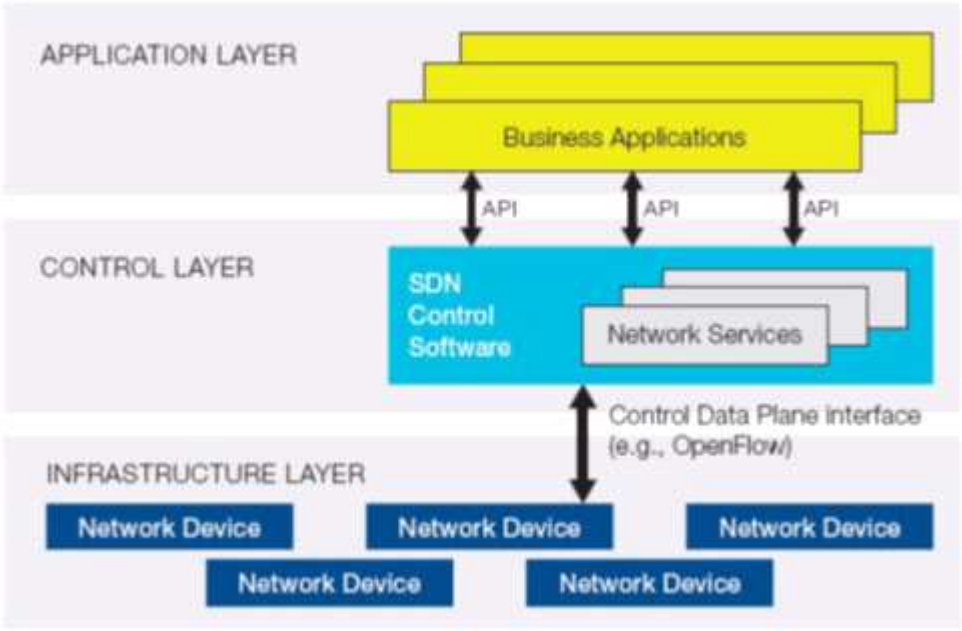

그림 7 SDN의 개념구조

그림 1은 SDN의 기본 구조를 도시한 것이다. 네트워크의 지능은 SDN 컨트롤러에 집중화되어 네트워크 전체를 관리하며 네트워크는 하나의 논리적인 스위치로 간주된다. 관리자(또는 NMS)는 표준화된 인터페이스를 통해 전체 네트워크를 벤더에 의존하지 않고 제어할 수 있고, 네트워크 설계와 운용을 보다 단순화할 수 있게 된다. 또한 SDN은 네트워크 장치를 단순화할 수도 있다. 즉, 장치를 설계할 때에 수백~수천 개의 프로토콜(RFC) 처리를 고려하지 않아도 되며, 단순히 SDN 컨트롤러로부터의 명령을 받아 데이터를 처리하는 기능만 갖추도록 하면 되기 때문이다.

네트워크 관리자는 예를 들어 수천 개의 장치에 분산된 수만 라인의 구성정보(configuration)를 수작업으로 관리할 필요가 없고 단순히 추상화된 네트워크를 프로그램만으로 구성하는 것으로 네트워크를 제어할 수 있다. 네트워크 추상화와 함께 SDN은 SDN 컨트롤 계층과 애플리케이션 계층간에 일련의 API를 제공한다. API를 이용하여 공통적인 네트워크 서비스를 구현할 수 있고 비즈니스 목표에 맞는 라우팅, 접근제어, 트래픽 엔지니어링, QoS 관리, 전력제어 등 모든 형태의 정책 관리가 가능하다.

그림 7은 SDN의 인프라스트럭처, 제어, 애플리케이션의 3-계층 구조를 나타내며 ONF에서 정의한 SDN의 개념을 잘 보여주고 있다. 인프라스트럭처 계층에는 물리적/논리적 네트워크 장치가 포함되며 실제 데이터 전송이 이뤄지는 전송 평면을 담당한다. 제어 계층은 이름 그대로 제어 평면을 담당하며 소프트웨어로 정의된 컨트롤러가 인프라스트럭처 계층에 존재하는 네트워크 장비들을 관리 및 제어한다. 컨트롤러가 네트워크 장비와 통신위해 사용하는 프로토콜들은 컨트롤러를 기준으로 아래쪽에 존재한다 하여 South-Bound Interface (SBI)라 부르며 일반적으로

OpenFlow를 많이 사용하고 있다. 컨트롤러가 애플리케이션 개발을 위해 제공하는 API를 컨트롤러를 기준으로 위쪽이라 하여 North-Bound Interface (NBI)라고 하며 NBI를 이용하여 개발된 라우팅, 접근제어, QoS관리 등과 같은 SDN 애플리케이션들이 애플리케이션 계층에 해당된다.

SDN에서는 이러한 표준화된 인터페이스를 통해서 장비 제조사에 의존하지 않고 네트워크 장비를 제어할 수 있어 보다 유연하게 네트워크를 운용할 수 있다. 또한, SDN에서 네트워크 장비는 컨트롤러의 명령을 수행하는 단순한 구조를 가져 장비의 단가가 낮아지게 된다. 이러한 SDN의 특징으로 보다 손쉽게 네트워크를 구축 및 운용할 수 있다.

3.3. SDN 도입 효과

기업이나 통신사업자에게 SDN은 네트워크를 불가피한 비용 영역에서 경쟁력을 높일 수 있는 차별화 영역으로 변화시킨다. OpenFlow 기반 SDN 기술은 IT 부서가 높은 대역폭과 최신 애플리케이션의 역동성 등에 대응하고, 계속 변하는 비즈니스 수요(needs)에 적응할 수 있게 해주며, 운영 및 관리상의 복잡성을 줄여 준다. 기업이나 통신 사업자가 OpenFlow 기반 SDN를 도입함에 따라 얻을 수 있는 이점은 다음과 같다.

첫째, 다수의 장비공급자가 존재하는 환경에서 중앙 집중화된 제어가 가능하다. SDN 컨트롤 소프트웨어는 스위치, 라우터, 가상 스위치 등 모든 벤더의 OpenFlow 지원 네트워크 장비를 제어할 수 있다. IT 부서는 각 벤더 별 장비들을 개별적으로 관리하는 것이 아니라, SDN 기반의 조정(orchestration) 및 관리도구를 이용해서 신속하게 전체 네트워크에 걸쳐 장비들을 배치하고, 구성(configure) 및 업데이트를 할 수 있다.

둘째, 자동화를 통해 복잡성이 감소한다. OpenFlow 기반 SDN은 유연성 있는 관리 프레임워크를 제공해서 아직까지 수작업으로 수행하고 있는 많은 일들을 자동화할 수 있게 해준다. 이러한 자동화 도구들은 운용비용(OPEX)을 줄여 주고, 운용자 오류에 의한 네트워크 장애를 줄일 수 있으며, 최근 등장한 IT-as-a-Service(IaaS)와 셀프 서비스 프로비저닝 모델을 지원할 수 있다. 이외에도 SDN을 이용하여 복잡한 클라우드 기반 애플리케이션들을 지능적으로 조정(intelligent orchestration)하고 관리할 수 있어 OPEX를 줄일뿐 아니라 비즈니스에 대한 민첩성을 높일 수 있다.

셋째, 기업의 혁신을 촉진시킨다. SDN의 채택은 IT 네트워크 운영자들이 특정 비즈니스 수요 또는 특정 사용자 요구에 맞춰 실시간으로 네트워크를 정확하게 프로그램하고, 재사용할 수 있으므로 비즈니스 혁신을 가속화한다. SDN과 OpenFlow는 IT 부서에게 빠른 시간 내에 네트워크 문제를 해결하거나, 새로운 서비스 및 새로운 네트워크 역량을 도입할 수 있는 능력을 제공한다.

넷째, 네트워크 신뢰성과 보안성의 증가이다. SDN은 IT 부서가 OpenFlow를 통해 인프라를 변화시킬 수 있는 높은 수준의 설정과 전략을 수립할 수 있게 해준다. OpenFlow 기반 SDN 아키텍처는 서비스, 애플리케이션을 추가 및 이동하거나, 또는 정책이 바뀔 때마다 매번 네트워크 장비를 각각 설정해야 할 필요를 없애 주므로 구성이나 정책의 일관성 부족(inconsistency)으로 네트워크가 다운될 가능성을 크게 줄여 준다. 이것은 SDN 컨트롤러들이 네트워크의 가시성을 제공하면서 네트워크를 제어할 수 있기 때문에 액세스 제어, 트래픽 엔지니어링, 서비스 품질 관리, 보안 및 각종 비즈니스 정책적용이 가능해져, 지사/지점, 캠퍼스, 데이터센터 등을 포함한 유/무선 네트워크 인프라에 걸쳐 다양한 정책들을 일관성 있게 적용할 수 있기 때문이다. 특히 기업이나 통신사업자들은 이를 통해 운용비용을 줄이고, 더욱 역동적인 설정 능력을 실현할 수 있고, 오류를 줄이면서 일관성 있는 설정과 전략 실행이 가능해진다.

다섯째, 좀 더 정교한 네트워크 제어가 가능하다. OpenFlow 기반 제어 모델은 IT 부서가 정책을 세션, 사용자, 도구, 애플리케이션 등의 수준에서 더욱 정교하고 때로는 매우 추상적이며 자동화된 방식으로 적용할 수 있게 해준다. 이 제어기능은 고객들이 같은 인프라를 공유할 때, 클라우드 서비스 운용자에게 트래픽의 격리(isolation), 보안, 탄력적인 자원 관리를 할 수 있도록 하는 한편 multi-tenancy를 지원할 수 있도록 한다.

마지막으로 사용자는 좀 더 좋은 경험을 할 수 있다. SDN 인프라는 네트워크 컨트롤을 중앙 집중화시키고, 네트워크 상태 정보를 더 높은 수준의 애플리케이션에서 이용할 수 있게 해줌으로써 사용자 요구에 더 역동적으로 부응할 수 있다. 예를 들면, 현재 비디오 스트리밍 서비스 사용자들은 네트워크의 지원 여부를 모르는 상태에서 해상도를 선택해야 한다. 결과적으로 이로 인해 발생되는 지연이나 일시적인 서비스 중단은 사용자 만족도를 저하시킨다. 하지만 OpenFlow 기반 SDN에서는 비디오 애플리케이션이 네트워크에서 사용할 수 있는 대역폭을 실시간으로 탐지해서 비디오 해상도를 자동 조절하도록 할 수 있으므로 사용자 만족도를 높일 수 있다.

3.4. SDN 핵심 구성 요소

- SDN 컨트롤러 (SDN controller)

SDN 컨트롤러는 네트워크 상태에 대한 글로벌 뷰를 기반으로 포워딩 제어, 토폴로지 및 자원의 상태 관리, 라우팅 제어 등 중앙 집중적인 망 제어를 위한 기본 기능을 수행하며, 상위의 응용이나 정책 요구에 따라 차별화된 포워딩 및 패킷 처리 룰(Rule) 들을 결정하여 하위의 SDN 스위치 박스들에게 포워딩 룰(Forwarding Rules)을 내려줌으로써 네트워크를 소프트웨어적으로 매우 유연하게 제어 운용한다.

SDN 기술에서 컨트롤러는 비즈니스 민첩성(Business Agility)을 지원하고 효율적인 개방형 인프라를 지향하는 데에 핵심 요소이며, SDN 컨트롤러의 기능 범위는 인프라를 제어하는 기본 기능뿐만 아니라 다양한 응용 기능의 포함 여부에 따라 다르나, 일반적으로 SDN 컨트롤러는 스위치와의 통신 및 이벤트 처리를 담당하는 컨트롤러 코어, 디바이스 관리, 링크 및 토폴로지 관리 등 응용에서 공통으로 필요한 기능을 제공하는 공통 컴포넌트 모듈, 응용 지원을 위한 라이브러리 및 API 등으로 구성된다.

- SDN 디바이스

SDN 디바이스는 SDN 컨트롤러의 지시에 따라 패킷 전송을 수행하는 네트워크 장치를 통칭하며, 제어 계층으로부터 주어지는 가변 단위의 플로우 별 패킷에 대한 포워딩 룰을 기반으로 패킷 전송을 수행한다. SDN 디바이스의 핵심 기술로는 고속의 플로우 테이블 매칭, 네트워크 가상화를 위한 디바이스 자원 가상화, 기존 장비의 SDN화 지원을 위한 플러그인 에이전트, QoS 및 DPI 기술 등이 있다.

- SDN 가상 스위치

가상 스위치는 하이퍼바이저 상에 생성되는 가상 머신들 간의 통신을 지원하거나 또는 물리 NIC에 연결되어 가상 머신과 물리 네트워크 간의 통신을 지원하는 소프트웨어 스위치로서 트래픽에 대한 가시성, 테넌트 간 트래픽 분리, 트래픽에 대한 세밀한 제어 등의 기능 제공하는 소프트웨어 기반 스위치를 지칭한다.

SDN에서 가상스위치 기술이 중요한 이유는 클라우드 데이터센터에서 서버 가상화 기술에 기반하여 다수의 가상 머신들이 생성되고 이들 간의 통신이 가상 스위치를 통해 빈번히 일어나는 환경에서는 가상 스위치 제어에 SDN 개념을 적용하지 않고 물리 네트워크를 사용하는 것만으로는 가상 머신 간의 트래픽을 효과적으로 제어할 수 없기 때문이다.

가상 스위치 모듈은 일반적으로 가상화 계층을 지원하는 하이퍼바이저에 통합되어 있거나 서버의 하드웨어에 펌웨어 형태로 제공될 수 있으며, 각 물리 서버에 분산되어 있는 가상 스위치들은 SDN 컨트롤러에 의해 중앙 집중식으로 통합 제어됨으로써 하이퍼바이저 상의 모든 가상 머신들이 물리적 위치에 상관없이 논리적으로 하나의 가상 스위치에 연결되는 하부 물리 구조에 대한 투명성을 제공한다

일반적으로 SDN 가상스위치는 다음과 같은 요구 사항을 갖고 있다.

• 이동성 (Mobility): 가상 머신의 마이그레이션에 따른 보안 및 네트워크 속성의 이동
• 확장성 (Scalability): 분산형 가상 스위치 간의 협력을 통한 수 만개 이상의 가상 머신 간 통신 지원
• 격리성 (Isolation): Multi-tenant 관점에서 다수의 논리적으로 분리된 가상 네트워

크 지원
• 가시성 (Visibility): 가상 스위치의 상태 및 통계 정보에 대한 제공
• 고정밀 제어 (Fine-grained Control): 오픈 인터페이스를 통한 다계층 (L2~L4) 스위칭 지원

현재 Open vSwitch를 비롯하여 많은 종류의 가상 스위치가 공개되어 있으나[11], 성능 측면에서 하드웨어 스위치에 비해 많이 떨어지는 것이 사실이다.

- 개방형 인터페이스 기술

SDN에서는 SDN을 구성하는 전송 계층, 제어 계층, 응용 계층 간을 연계하는 표준 인터페이스를 제어 계층을 중심으로 그림 4와 같이 세 종류의 인터페이스를 정의한다.

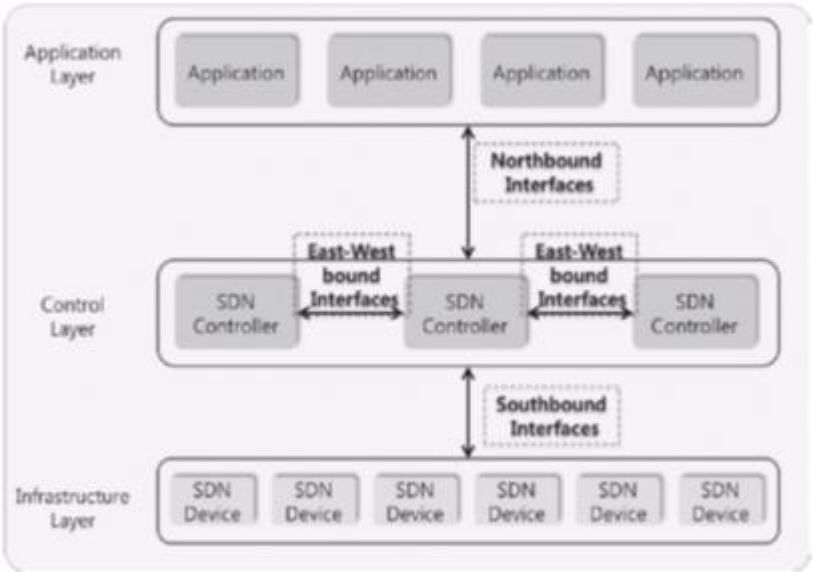

그림 8 SDN 개방형 인터페이스의 분류

• 사우스 바운드 인터페이스 (South Bound Interface; SBI): 전송 계층에 대한 포워딩 제어 및 정보 수집을 하기 위한 인터페이스로서 ONF에서 정의하고 있는 OpenFlow Protocol과 OF-Config 가 대표적이다. 각 장비 벤더 입장에서는 자신들의 장비가 추가적으로 지원하는 벤더 확장 기능에 대한 인터페이스를 공개하거나 기존 장비에 OpenFlow를 비롯한 자체 솔루션을 동시에 지원하는 하이브리드 접근 방법을 통해 다른 장비와의 차별성을 추구할 것으로 예상된다.

• 노스바운드 인터페이스 (North Bound Interface; NBI): 응용 계층에 제공되는 인터페이스로서 제어 계층에 대한 직접적인 제어와 전송 계층에 대한 간접 제어를 수행할 수 있다. 아직 표준은 정의되어 있지 않으나 자체 API를 중심으로 새로운 생

태계를 구축하려는 여러 벤더들의 치열한 경쟁을 통해 de-facto 표준이 만들어질 가능성이 크다. 특히 노스바운드 인터페이스는 SDN을 통한 비즈니스 창출 기회가 있는 부분으로 많은 업체들이 자체 응용에 맞는 인터페이스 개발에 집중하는 양상을 보이고 있다.

• 이스트-웨스트바운드 인터페이스 (East-West Bound Interface): 제어 계층 간의 인터페이스로서 서로 다른 SDN 도메인 간 또는 동일 도메인 내의 서로 다른 SDN 컨트롤러 간의 인터페이스를 말한다. SDN 도메인을 나누고 서로 다른 도메인 간의 메시지 교환 필요성에 대한 논의는 활발하지 않으나 대규모 엔터프라이즈 네트워크나 캐리어 네트워크에 적용하기 위해서는 필요한 인터페이스이다.

4. ONOS (Open Network Operating System)

4.1. ONOS 소개

기존 네트워크 장비에서 소프트웨어로 구현된 기능들을 SDN에서는 컨트롤러가 수행하며 컨트롤러가 다수의 네트워크 장비를 제어하기 때문에 컨트롤러에 대한 중요도가 매우 높다. 2014년도 이전까지 공개되거나 출시된 컨트롤러는 약 15종에 달하며 대부분 OpenFlow 기반의 공개 소스이다. 상용으로는 NEC의 PF6800 [12], IBM의 PNC[13], CPlane의 OpenTransit[14], Big Switch의 Big Network Controller[15] 등이 존재했었으며, 연구망 혹은 데이터센터 내 통신용이나 소규모 엔터프라이즈 적용을 1차 목표 시장으로 하고 있어 플로우 처리의 확장성과 안정성 및 응용 기능의 확장성에 한계가 있었다. 이러한 컨트롤러들은 SDN을 아래와 같은 기술적 문제로 인하여 캠퍼스나 작은 기업체와 같은 중규모의 네트워크에 적용이 가능한 기술로 평가되었다.

- 중앙 집중 구조에서 컨트롤러가 bottleneck이 되어 네트워크의 성능이 감소
- 다수의 컨트롤러를 사용하는 SDN에서 컨트롤러가 네트워크의 global view를 알지 못함

하지만 최근에는 ONOS[16]와 OpenDayLight (ODL)[17]가 대규모의 네트워크에 적용 가능한 컨트롤러로써 많은 관심을 받고 있다. 본 보고서에서는 ONOS를 기반으로 하여 LISP 프로토콜을 Openflow와 함께 SDN 컨트롤러의 SBI로써 제공하기 위한 설계 및 구현을 서술한다.

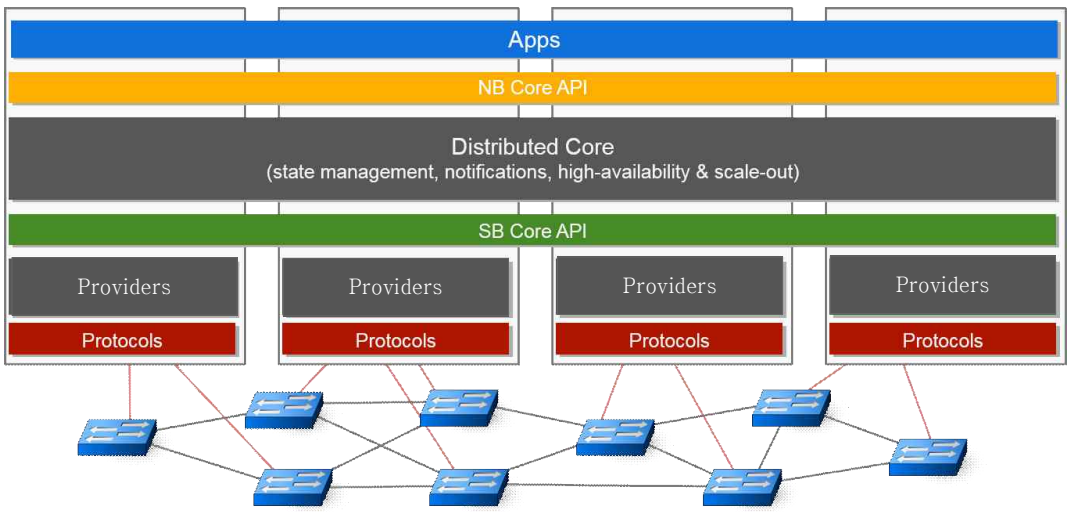

그림 9. ONOS의 분산 구조

ONOS에서는 분산 구조를 이용하여 위의 문제를 해결하여 기존 컨트롤러보다 높은 네트워크 성능을 제공한다. 그림 9는 ONOS의 분산 구조를 나타내고 있다. ONOS는 다수의 ONOS 인스턴스(virtual machine)로 구성되며 각 인스턴스는 ONOS 컨트롤러로써 네트워크 장비를 제어한다. ONOS 인스턴스는 분산 저장소(distributed store)을 이용하여 네트워크 정보를 공유하여 모든 ONOS 인스턴스가 global view를 볼 수 있게 도와준다. ONOS는 분산 구조를 이용하여 컨트롤러에 몰리는 제어 메시지를 분산시키면서도 네트워크의 global view를 가지고 네트워크 장치들을 제어할 수 있다.

4.2. ONOS의 설계 구조

ONOS는 계층을 가지는 구조로 설계되어 있으며 각 계층의 역할은 다음 과 같다 (그림 10 참조).

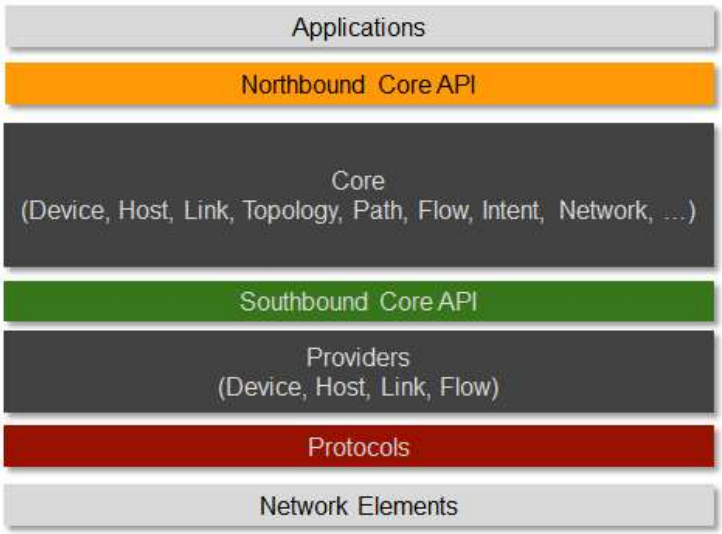

그림 10. ONOS의 계층 구조

ONOS에서 정의하는 서비스는 그림 10에서 보이는 계층구조에 수직적으로 위치하는 스택 구조를 이루는 복수의 컴넌트들의 집합을 통해 구현되는 기능들을 지칭한다. 특정 서비스 또는 기능을 제공하기 위해 구성되는 컴포넌트들의 집합을 ONOS에서는 서브시스템 (Subsystem)이라고 지칭한다. 따라서 ONOS에서 서브시스템과 서비스는 거의 동일한 의미를 가지는 단어로써 사용된다. 현재 ONOS에서 제공하고 있는 서브시스템들은 그림 11에 나타나 있다. 대표적인 서브시스템으로써,

인프라스트럭처에 위치하고 있는 각 장치들을 관리하는 Device Subsystem, 현재 구성된 링크들을 관리하는 Link Subsystem, 현재 상황에서 구성되어 있는 토플로지를 그래프 형태로 관리하는 Topology Subsystem, 각 장치들 사이에 존재 하는 경로를 탐색 하고 연결하여 주는 Path System 등이 있다.

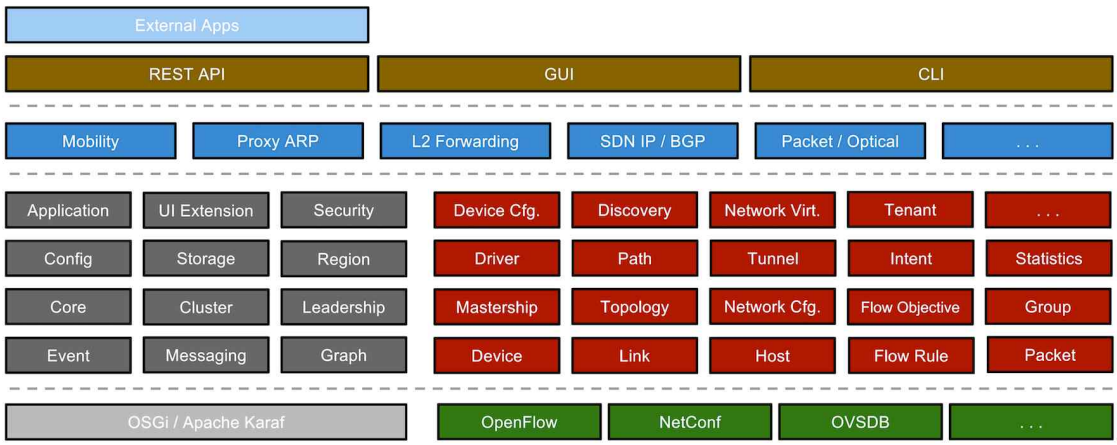

그림 11 다양한 ONOS의 서브시스템(서비스)들

ONOS상에서 특정 서비스를 추가적으로 제공하기 위해서는, 네트워크 스택을 전체적으로 포함하는 새로운 서브시스템을 설계할 필요가 있다. 각 서브시스템을 구성하는 컴포넌트들은 App, Manager, Provider 컴포넌트 중의 하나로 분류되며, JAVA에서 제공하는 인터페이스를 통해 구별될 수 있다. ONOS의 서브시스템 구조는 그림 12와 같이 도시할 수 있다.

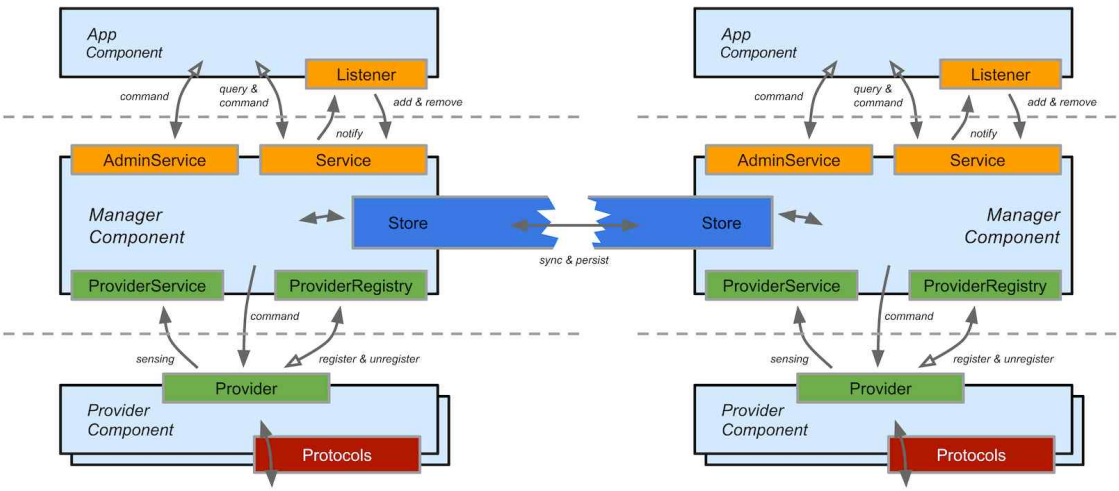

그림 12 ONOS 서브시스템 구조

- **프로바이더 (Provider) 계층 컴포넌트**

ONOS 계층 구조의 가장 하위에 위치하는 컴포넌트이며, 특정 프로토콜 (e.g. OpenFlow)들을 지원하기 위한 프로토콜 라이브러리를 포함하며, ProvierService 인터페이스를 통해 상위에 위치하고 있는 코어 계층과 통신한다. 특정 프로토콜을 인지 있는 프로바이더는 다양한 제어 및 설정을 가능하게 하는 프로콜을 사용할 수 있으며, 특정 서비스에 종속되는 데이터들을 코어 계층으로 전달하는 역할을 한다. 프로바이더는 단순히 데이터 평면에 위치하는 특정 장치들과의 통신 및 인터페이싱만을 담당하는 것이 아니라, ONOS 상에서 구동되고 있는 서비스 (또는 서브시스템)들로부터 데이터를 수집하여 특정 서비스를 만족시키기 위한 프로바이더를 설계할 수도 있다.

- **매니저 계층 컴포넌트**

이 분류의 컴포넌트들은 ONOS의 코어 영역에 위치하며, 프로바이더 계층에 위치하고 있는 컴포넌트들로부터 전달 받은 정보들을 응용 또는 다른 서비스들로 전달 하는 역할을 수행한다. 이 계층에 위치하고 있는 컴포넌트들은 다수의 인터페이스를 가지고 있다. Northbound Service 서비스 인터페이스는 특정 네트워크 상태에 관심을 가지고 있는 응용 또는 다른 코어 컴포넌트들이 사용하는 인터페이스이다. Admin Service 인터페이스는 관리자의 명령을 받아 들여 네트워크 또는 시스템에 적용을 위한 인터페이스이다. Southbound ProviderRegistry 인터페이스는 프로바이더들이 매니저에 등록을 가능하게 하여 다른 응용 또는 서비스들과 상호 작용 할 수 있도록 한다. Southbound ProviderService 인터페이스는 매니저에 등록된 프로바이더가 정보를 보내고 수신하기 이ㅜ해 사용되는 인터페이스다. 마지막으로 매니저의 Consumer 인터페이스는 서비스로부터 동기화된 질의를 받거나 Event Listener를 통해 비도기적으로 전달되는 이번데트들 수신하기 위한 인터페이스를 제공한다.

- **응용 계층 컴포넌트**

응용 계층에 위치하는 컴포넌트들은 매니저로부터 수집된 정보들을 사용하거나 가공하는 역할을 수행한다. 응용들은 매우 넓은 영역에 해당하는 기능을 제공하며, 시각화, 경로 설정 등과 같은 로직들을 구현한다.

4.3. ONOS SBI 확장의 필요성

SDN 기술은 기존에 존재하는 IP 기반의 장치 및 네트워크 기반 구조에 점진적으로 배포가 가능한 구조를 지닌 것이 아니라, 전체 네트워크를 새롭게 구성하는 Clean State 접근 방식을 취하고 있다. 따라서, SDN을 구성하기 위해서는 OpenFlow를 지원하는 SDN 스위치와 컨트롤러가 필요하다. 하지만 현재 인터넷을

구성하고 있는 기존 네트워크 장비들은 OpenFlow를 지원하지 않으므로, OpenFlow를 사용하는 SDN으로 인터넷을 대체하는데 문제가 있다. 이러한 상황에서 각 인터넷 또는 네트워크 서비스 제공자는 기존의 네트워크에 배포되어 있는 각종 서비스와 IP 네트워크의 모든 설정들을 문제 없이 SDN 기반의 네트워크로 이전 할 수 있는 방법이 필요하다. 아직 까지 SDN은 연구수준 또는 데이터센터와 같은 닫힌 환경의 네트워크에 적용되고 있는 수준이며, 전체 네트워크 구조를 SDN기반으로 전환 하기 위해선는 해결해야할 문제가 많이 있다. 더불어, 현재 대표적인 SDN 프로토콜로 각광 받고 있는 OpenFlow는 모든 IP 네트워크의 기능을 제공하는데 한계가 있다. 따라서, 기존 인터넷과 SDN이 공존하면서 SDN의 영역을 차츰차츰 넓혀감으로써 기존 인터넷을 SDN으로 전환하기 위한 기술이 필요하다. 따라서 현재 IP 기반의 네트워크 기술들이 가지고 있는 필수 서비스를 SDN 환경으로 넓기히 위해, MPLS, PCEP, BGP, VPN, Multicast와 같은 네트워크 서비스를 이전 할 수 있는 방법이 필요하다.

OpenFlow가 SDN을 위해서 디자인된 통신 프로토콜이긴 하지만 L2/L3 스위치에서 패킷 처리하는 것을 목표로 하고 있기 때문에 다양한 요구사항을 지원하기 위해서는 OpenFlow만으로는 한계가 있다. 이러한 문제를 해결하기 위해서 ODL은 SBI에 여러 프로토콜을 지원하고 있다. 일반적으로 SDN 컨트롤러는 OpenFlow를 유일한 SBI로 사용하는데 반해서 ODL은 OVSDB, NetConf, CAPWAP, BGP, PCEP 등 총 15종의 프로토콜을 지원함으로써 더 많은 네트워크 장비를 제어 할 수 있다. 이에 반해, ONOS에서는 OpenFlow를 포함해 5 종의 프로토콜들만을 SBI로 제공하고 있다. ONOS도 네트워크의 다양한 장비들과 협업을 통해서 네트워크의 global view를 가질 필요가 있으므로 더 많은 프로토콜을 ONOS의 SBI로 확장할 필요성이 존재한다.

더불어 LISP이 가지고 있는 장점을 활용하여 새로운 형태의 응용들을 개발 할 수 있는 가능성을 높일 수 있다. LISP은 1장에서 서술한 바와 같이 Vertical Handover, VM livemigration, 인입 트레픽 제어, 재난 복구와 같은 다양한 네트워크 기반 서비스들을 제공할 수 있으며 그 효율성이 검증 되었다. 따라서, SDN 환경에서 이러한 LISP 기반의 서비스를 제공함으로써 SDN의 배포 속도를 가속화 할 수 있을 것으로 예상된다.

4.4. ONOS 설치 방법

본 절에서는 VirtualBox에서 생성된 가상머신을 상에서 ONOS를 설치 하는 방법을 서술 한다. 가상 머신의 운영체제는 Ubuntu를 사용하였다. 만약 Mac의 OSX에서 로컬로 돌린다면 15번까지만 하고, 17번을 이용해서 실행 할 수 있다. ONOS의 공식 설치 튜토리얼 문서는 다음 URL을 통해 접근할 수 있다.

https://wiki.onosproject.org/display/ONOS/ONOS+from+Scratch#ONOSfromScratch-OntheBuildmachine

1. VirtualBox 설치, VirtualBox에 VM을 만들고 Ubuntu 설치
VirtualBox 환경설정에서, VM에 2개의 Network interface 생성 (NAT, Host Only)

각각의 인터페이스는 eth0, eth1에 해당.
eth0 - 외부 인터넷을 위해 사용
eth1 - 각 VM의 연결을 위해 사용

2. 네트워크 인터페이스 설정
NAT로 설정된 eth0는 자동으로 설정되지만, eth1은 수동으로 설정해줘야함.
/etc/network/interfaces 파일에 다음 내용 추가.

auto eth1
iface eth1 inet static
address 192.168.56.101
gateway 192.168.56.1
netmask 255.255.255.0
network 192.168.56.0
broadcast 192.168.56.255

네트워크 서비스 재시작
sudo /etc/init.d/networking restart

Ifconfig를 통해 eth0과 eth1이 정상적으로 작동하는지 확인

기존 package들 업데이트
sudo apt-update

2. 생성된 VM 사용자 계정생성
sudo adduser
ID/passwd : sdn/rocks
id와user를 다르게 세팅 시, 나중에 환경변수에서 수정해줘야 한다.

3. 생성된 사용자에게 sudo 권한 주기
sudo visudo

sdn 유저 권한 추가를 위해 아래 구문 삽입
sdn ALL=(ALL) NOPASSWD:ALL

4. SSH 로그인시 passwd 필요 없도록 하기 (선택사항)
이 과정은 생략가능, 이후에 ONOS 설치 후 onos-push-keys 명령어를 통해 동일하게 설정가능. (14번 항목 참조)

rsa key 생성
ssh-keygen -t rsa

rsa key 복사
ssh-copy-id sdn@192.168.56.101

5. Git 설치 (이후 내용 모두 sdn 계정으로 접속하여 진행)
sudo apt-get install git-core

6. Karaf 3.0.3과 Maven 3.3.9 설치
cd; mkdir Downloads Applications
cd Downloads
wget http://archive.apache.org/dist/karaf/3.0.3/apache-karaf-3.0.3.tar.gz
wget http://archive.apache.org/dist/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz
tar -zxvf apache-karaf-3.0.3.tar.gz -C ../Applications/
tar -zxvf apache-maven-3.3.9-bin.tar.gz -C ../Applications/
바이너리 파일이므로 추가적인 설치 과정 불필요.

7. JAVA8 설치
sudo apt-get install software-properties-common -y
sudo add-apt-repository ppa:webupd8team/java -y
sudo apt-get update
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y

8. ONOS 소스 다운 받기
git clone https://gerrit.onosproject.org/onos
sdn 사용자의 홈폴더에서 진행 할것.

현재 최신 버전으로 나와있는 1.5.0은 제대로 설치가 안되는것 같아. 1.3.0을 사용하기로 함.

```
cd ~/onos
git checkout 1.3.0
```

9. 환경변수 설정

환경변수는 위의 설치 위치가 다르다면 ~/onos/tools/dev/bash_profile를 수정할것. sdn 로그인시 마다, 자동으로 환경변수들을 추가하기 위해 ~/.profile에 아래내용 추가. (.bashrc, .bash_aliases도 가능)

```
. ~/onos/tools/dev/bash_profile
```

export 명령 시 다음과 같이 ONOS관련 환경 변수, karaf 위치, JAVA_HOME등이 정상적으로 설정되었는지 확인할 것.

```
declare -x MAVEN="/home/sdn/Applications/apache-maven-3.3.9"
declare -x OC1="192.168.56.101"
declare -x OC2="192.168.56.102"
declare -x OCI="192.168.56.101"
declare -x OCN="192.168.56.103"
declare -x OLDPWD
declare -x ONOS_APPS="drivers,openflow,fwd,proxyarp,mobility"
declare -x ONOS_CELL="local"
declare -x ONOS_INSTANCES="192.168.56.101
192.168.56.102"
declare -x ONOS_NIC="192.168.56.*"
declare -x ONOS_ROOT="/home/sdn/onos"
declare -x ONOS_SCENARIOS="/home/sdn/onos/tools/test/scenarios"
declare -x ONOS_USER="sdn"
declare -x ONOS_WEB_PASS="rocks"
declare -x ONOS_WEB_USER="onos"
declare -x
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/sdn/onos/tools/dev/bin:/home/sdn/onos/tools/test/bin:/home/sdn/onos/tools/test/scenarios/bin:/home/sdn/onos/tools/build:/home/sdn/Applications/apache-maven-3.3.9/bin:/home/sdn/Applications/apache-karaf-3.0.3/bin"
```

10. Apache Karaf 설정하기
이과정은 onos-karaf 명령을 치면 자동으로 된다. 그냥 넘어가길.

Karaf에서 ONOS feature들을 인식하게 하기 위해 필요.
~/Applications/apache-karaf-3.0.3/etc/org.apache.karaf.features.cfg 파일의 featuresRepositories에 아래 내용 추가.
mvn:org.onosproject/onos-features/1.5.0-SNAPSHOT/xml/features

튜토리얼에는 나와있지 않지만, onos-branding을 karaf에 추가해야 로고가 보인다.
cp ~/onos/tools/package/branding/target/onos-branding-1.5.0-SNAPSHOT.jar ~/Applications/apache-karaf-3.0.3/lib/

제대로 설정 되었다면 아래와 같은 형태가 될것임.
featuresRepositories=mvn:org.apache.karaf.features/standard/3.0.3/xml/features,mvn:org.apache.karaf.features/enterprise/3.0.3/xml/features,mvn:org.ops4j.pax.web/pax-web-features/3.1.4/xml/features,mvn:org.apache.karaf.features/spring/3.0.3/xml/features,mvn:org.onosproject/onos-features/1.5.0-SNAPSHOT/xml/features

11. ONOS 빌드하기
mvn clean install 대신 mci명령어로 대체 가능.
이 과정은 처음에 수행할 경우, 의존성이 있는 패키지들을 다운받아야 하므로 시간이 오래 걸림.
cd ~/onos
mvn clean install

빌드 과정 중, java.lang.OutOfMemoryError: Java heap space 발생할 경우.
~/onos/tools/dev/bash_profile에 아래 변수 추가.
export MAVEN_OPTS="-Xmx512m -XX:MaxPermSize=128m"

업데이트된 환경변수는 자동적용되지 않으므로, shell에 다시 접속하거나 아래 명령으로 환경변수 갱신.
source ~/.profile

12. ONOS 설정하기
ONOS에서 내부적으로 사용되는 변수들의 집합을 cell으로 표현함.
Cell의 샘플들은 ${ONOS_ROOT}/tools/test/cells/ 에서 확인 할 수 있음.
tutorial이라는 파일을 생성 후, 아래 내용 작성.

```
# ONOS from Scratch tutorial cell

# the address of the VM to install the package onto
export OC1="192.168.56.101"

# the default address used by ONOS utilities when none are supplied
export OCI="192.168.56.101"

# the ONOS apps to load at startup
export ONOS_APPS="drivers,openflow,fwd,proxyarp,mobility"

# the Mininet VM (if you have one)
export OCN="192.168.56.102"

# pattern to specify which address to use for inter-ONOS node communication
(not used with single-instance core)
export ONOS_NIC="192.168.56.*"
```

13. Cell 적용하기
cell은 ONOS에서 제공하는 명령임. 12에서 작성한 tutorial 파일을 읽어서 적용함.
cell tutorial

14. SSH 환경 설정
password 없이 ssh를 통한 VM 접속.
rsa key 생성.
ssh-keygen -t rsa

비밀번호는 sdn 계정의 비밀번호와 동일.
onos-push-keys 192.168.56.101

15. ONOS package 생성
onos-package
생성된 onos-package는 /tmp에 위치한다.

16. ONOS 배포
싱글 머신에서 사용할 경우 15과정까지만 하면, 된다. 16번 이후는 분산 컨트롤러

또는 원격에서 배포하는 과정임.
12번 항목에서 작성한 OC1 노드에 onos를 설치.
MAC의 경우 -fn 옵션을 사용할 것.
onos-install -f $OC1

sudo: start: command not found 에러 발생시, upstart가 없는 것이므로 설치해준다.
sudo apt-get install upstart
sudo apt-get install upstart-sysv
sudo dpkg-divert --local --rename --add /sbin/initctl
ln -s /bin/true /sbin/initctl

17. ONOS CLI 환경 접속하기
onos-karaf clean
CLI에서 나가는 방법은 Ctrl+D임.
web의 경우 아이디와 비밀번호 기본은 karaf/karaf로 되어 있다.

5. LISP Controller 설계 및 구현

5.1. LISP Controller 설계

기존의 LISP은 SDN과 유사하게 제어평면과 데이터평면으로 분리된 구조를 가지고 있었지만 중앙집중화 된 관리 체계의 부재 및 EID-RLOC 정보의 긴 수렴시간이라는 단점들을 가지고 있었다. LISP 컨트롤러 설계는 이러한 단점들을 극복하는 하고 SDN의 SBI로써 LISP을 제공하는 것으로써 다음과 같은 목적을 가진다. 1) LISP 컨트롤러로 불리는 중앙 집중화된 단일의 개체를 통해 LISP과 관련된 제어들을 중앙집중화하는 것, 2) OpenFlow와 같이 SDN의 SBI로써 LISP을 지원 할 것, 3) LISP에 기반한 응용들을 개발하기 위한 추상화와 API를 제공할 것으로 요약할 수 있다. 이러한 이러한 LISP 컨트롤러의 개념은 그림 13에 도식화 되어 있다.

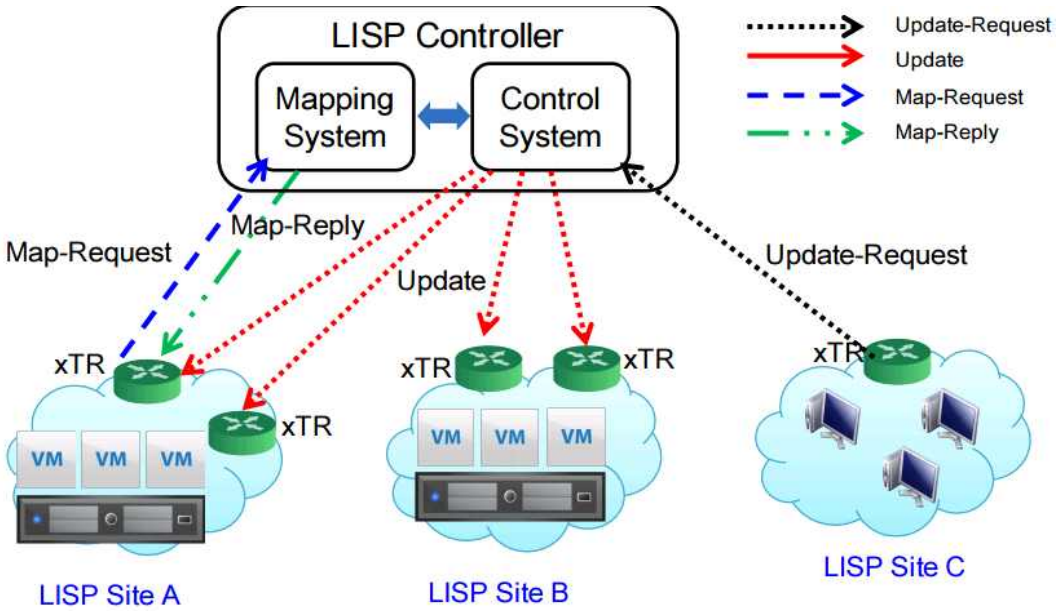

그림 13 LISP 컨트롤러의 개념

이러한 LISP 컨트롤러를 설계하고 구현하기 위해 다음과 같이 LISP 컨트롤러가 가져야 하는 요구사항들을 도출 하였다.

- xTR 상태 감시 및 정보 저장
- RTR 위치 저장 및 전달
- Map 정보 저장 및 전달
- MAC-SHA 1 인증 구현
- 관리를 위한 외부 API
- LISP 장치 중앙 관리 메커니즘 지원

이러한 요구사항을 반영하기 위해서 그림 14와 같이 ONOS상에서 LISP을 SBI로 지원하기 위한 서브시스템을 설계 하였다.

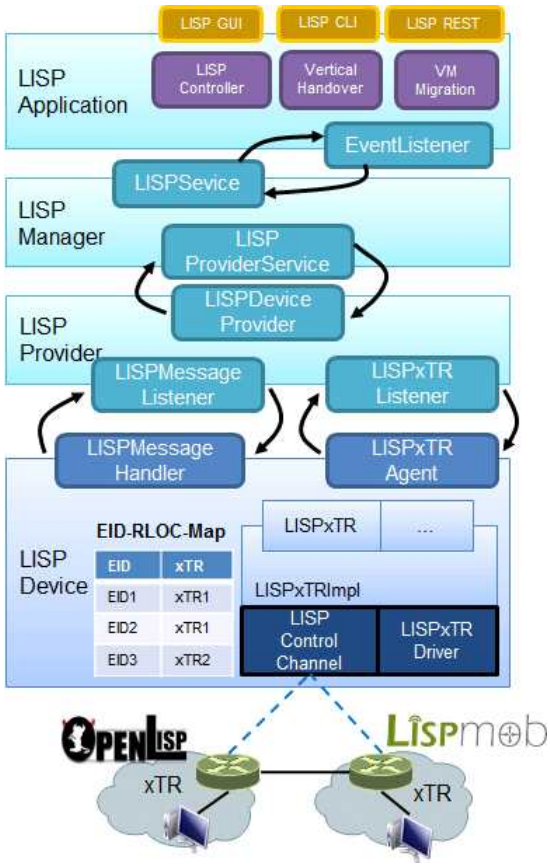

그림 14 LISP 컨트롤러의 설계

- LISP 프로바이더 컴포넌트

LISP 프로바이더 컴포넌트는 그림 14에서 LISP provider와 LISP device 영역에 위치하고 있는 컴포넌트들을 지칭하는 것으로써, 앞서 설명한 것과 같이 ONOS의 프로토콜 계층은 네트워크 장비와 통신하는 실제적인 역할을 하고 프로바이더 계층은 코어와 프로토콜 사이에서 중계자 역할을 한다. 이러한 구조로 인해 ONOS에서는 프로바이더 계층이 프로토콜 계층을 포함하여 SBI를 담당하는 것으로 나타낸다.

- LISP 매니저 컨포넌트

이 계층에 위치한 컴포넌트는 LISP provider로부터 수신된 메시지를 상위 응용 계층층으로 전달해주는 역할을 수행한다. 현재 (16년 5월 기준) 작성된 매니저는 ONOS에서 제공하는 추상화 기능과 통합을 하지는 않은 상황이며, 단순히 LISP을 이용하는 응용 계층에 위치하는 컴포넌트에게 정보를 전달하는 역할 만을 수행하고

있다.

- LISP 응용 컴포넌트

LISP 응용 컴포넌트 계층에는 LISP Mapping system과 중앙 집중화된 관리 기능을 담당하는 LISP Contoller 응용과 LISP에 기반 한 새로운 서비스 (예: Vertical handover, VM migration)과 같은 응용들이 위치하게 된다. 이 계층에서 가장 중요한 컴포넌트는 LISP Contoller 응용이다. 이 응용은 기존의 LISP Mapping 시스템이 제공하고 있는 Map-server와 Map resolver 기능을 제공하기 위해, EID-RLOC 매핑 기능을 수행한다. 추가적인 응용으로써 LISP에 기반한 외부 응용 또는 관리자가 사용학 위한 LISP 제어 관리 인터페이스를 GUI, CLI, 그리고 REST API 형태로 제공하는 역할을 수행한다.

5.2. LISP Controller 구현

본 기술문서에서는 ONOS 버전 1.4를 기준으로 LISP 프로바이더와 프로토콜을 구현하였다. 구현을 위해서 ONOS의 Provider tutorial Wiki[8]과 ONOS에 구현되어 있는 다른 SBI 프로토콜 소스코드를 참고하여 LISP Contoller를 구현하였다.

LISP 서브시스템을 구성하기 위해서는 4.1에서 서술한 3가지 분류에 해당하는 컴포넌트를 개발 하여야한다. Provider의 경우 ONOS의 하위에 위치하는 계층으로써 새로운 응용의 형태로써 ONOS에 존재하는 것이 아니라, ONOS가 구성될 시에 구동 될 수 있도로 기존의 ONOS 최상위 폴더 아래에 protocols와 providers라는 디렉토리에 구현을 해주어야 한다.

Provider 계층에 위치하는 컴포넌트인 LISP provider와 LISP protocol을 그림 8과 같이 해당 디렉토리에 LISP을 위한 디렉토리를 생성한다. 이때 각 디렉토리에 있는 pom파일에 작성하여 LISP 프로토콜과 프로바이더가 사용됨을 명시해야 한다. LISP 프로바이더를 예를들면 providers 디렉토리의 pom 파일에 lisp 프로바이더가 추가되었음을 명시한다. lisp 디렉토리에 구현하고자 하는 프로바이더들을 pom파일로 명시한다. 마지막으로 구현하고자 하는 LISP 프로바이더의 디렉토리에서 pom파일 작성하여 준다. 유사한 방법으로 프로토콜을 위한 pom 파일을 작성한다.

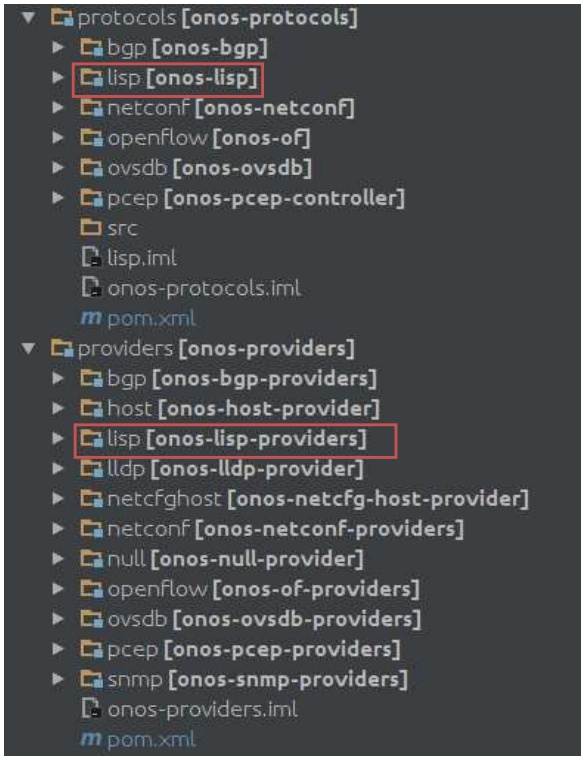

그림 15. LISP 프로바이더와 프로토콜 경로

프로바이더 계층의 컴포넌트에서 가장 핵심적인 기능을 수행하는 것은 LISP 프로토콜으로써, LISP에서 사용하는 메시지들을 수신하여 해석하고 정보를 응용으로 전달 하는 역할과, 응용에서 내려온 처리 결과를 LISP 메시지 형대로 다시 인코딩하여 각 LISP xTR들에게 전달하는 역할을 수행한다. LISP Provider는 LISP 프로토콜을 활성화 시키고, 응용들이 LISP을 사용할 수 있도록 하는 LISP 서비스를 관리하는 역할을 수행한다. 따라서 LISP Provider service와 LISP Protocol은 필수적으로 시에 수행되어야 한다.

ONOS의 소스코드를 다운받으면 protocols와 providers라는 디렉토리가 존재하는데 그림 17과 같이 해당 디렉토리에 LISP을 위한 디렉토리를 생성한다. 이때 각 디렉토리에 있는 pom파일에 작성하여 LISP 프로토콜과 프로바이더가 사용됨을 명시해야 한다. ONOS Wiki를 보면 프로바이더를 위한 pom 파일 작성과 함께 프로바이적 작성을 위한 가이드가 존재한다[8]. LISP 프로바이더를 예를들면 providers 디렉토리의 pom 파일에 lisp 프로바이더가 추가되었음을 명시한다. lisp 디렉토리에 구현하고자 하는 프로바이더들을 pom파일로 명시한다. 마지막으로 구현하고자 하는 LISP 프로바이더의 디렉토리에서 pom파일 작성하여 준다. 유사한 방법으로 프로토콜을 위한 pom 파일을 작성한다. 이후 각 역할에 맞는 기능을 할 수 있도록 JAVA Class 파일을 생성하여, 구현 한다.

LISP의 서브시스템 중 응용 계층에 해당하는 컴포넌트는 ONOS의 기존 빌드 소스 코드로 구현하는 것이 아니라, 별도의 응용으로써 구현이 가능하다. ONOS에서는 “onos-create-app”이라는 명령을 통하여 ONOS 상에서 구동되는 응용이 가져야 하는 필수적인 구조 및 필요 파일들을 자동으로 생성하여 준다. 이러한 과정을 통해 LISP 컨트롤러 응용을 그림 18과 같이 구성 할 수 있다. 생성된 응용 패키지에 필요로 하는 LISP Controller에 구현을 완료 한다. 이후 구현이 완료되면, 응용이 위치하고 있는 패키지의 최상위 디렉토리에서 “mvn clean install” 또는 “mci” 명령을 통해 개발된 응용을 빌드 할 수 있다. 개발된 응용이 제대로 빌드가 된다면, 그림 19와 같은 메시지를 볼 수 있다.

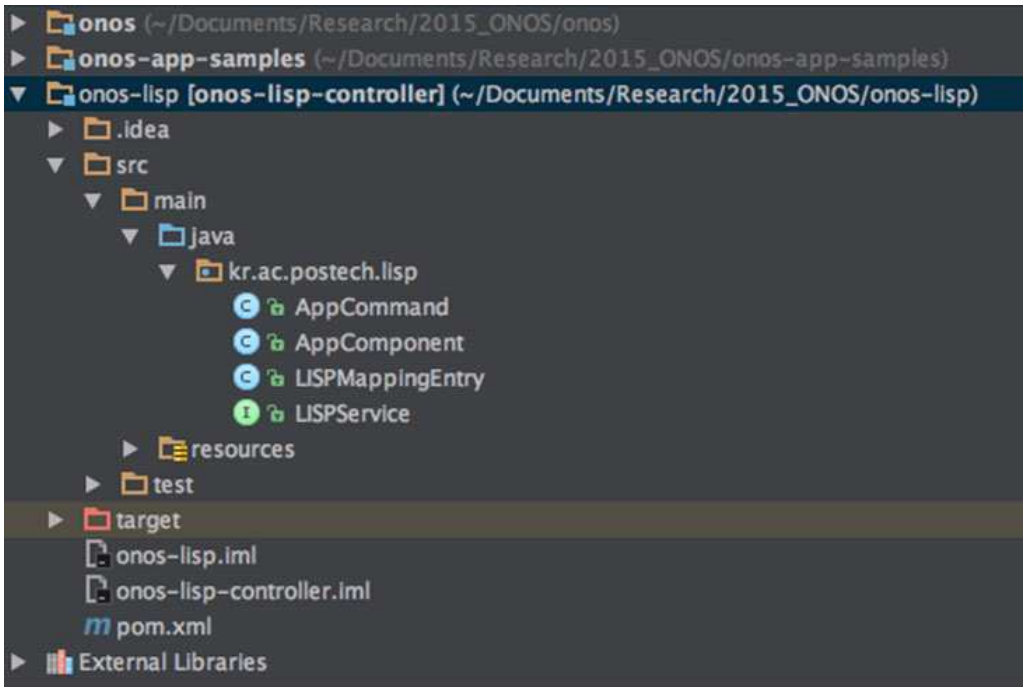

그림 16 LISP contoller 응용 구조

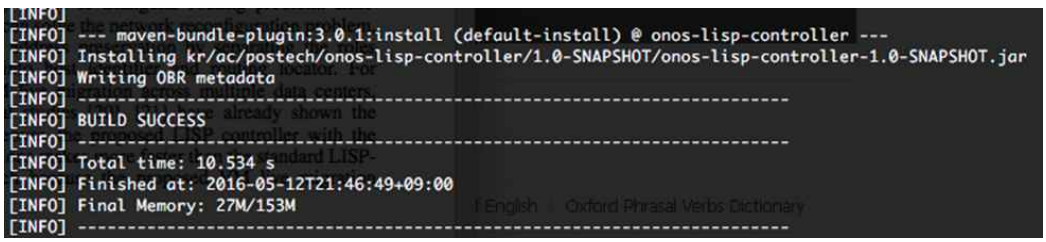

그림 17 LISP 컨트롤러의 빌드 결과

빌드된 LISP contoller 응용 및 프로바이더 계층의 컴포넌트를 실행하기 위해서는, 개발된 응용을 설치 할 필요가 있다. 프로바이더 계층의 컴포넌트는 이미 ONOS의 코어 패키지 상에 위치하고 있으며 ONOS를 빌드하는 과정에서 함께 빌드가 되지만, 응용 계층의 컴포넌트들을 수동으로 설치를 해주어야 한다. 이 과정은 ONOS에서 제공하는 “onos-app” 명령을 통해 수행 할 수 있다. onos-app 명령은

응용들의 설치(install), 재설치(reinstall), 제거(remove)등의 기능을 재공한다. 개발된 응용을 재설치 하는 경우 그림 20 과 같이 명령을 통해 재설치가 가능하다.

그림 18 ONOS controller의 설치

만약 LISP Conroller가 아닌 ONOS 프로바이더 계층에 위치한 컴포넌트드만을 구동가 히고 위해서는, ONOS의 CLI에서 “feature:install onos-lisp-app” 명령으로 LISP 프로바이더를 실행할 수 있다.

LISP contoller 응용 컴포넌트는 LISP provider 컴포넌트들에 대하 의존성을 가지고 있으므로, LISP contoller 수행 시 프로바이더들은 별도의 명령이 필요 없이 동시에 작동하게 된다. LISP contoller를 작동시키기 위한 명령은 ONOS CLI 상에서 “activate” 명령을 이용하면 된다 (그림 21). 제대로 LISP contoller가 구동되었다면 ONOS CLI 상에서 “list” 명령을 이용하여 lisp contoller가 작동중인 확인 할 수 있다 (그림 22).

그림 19 ONOS Contoller 구동

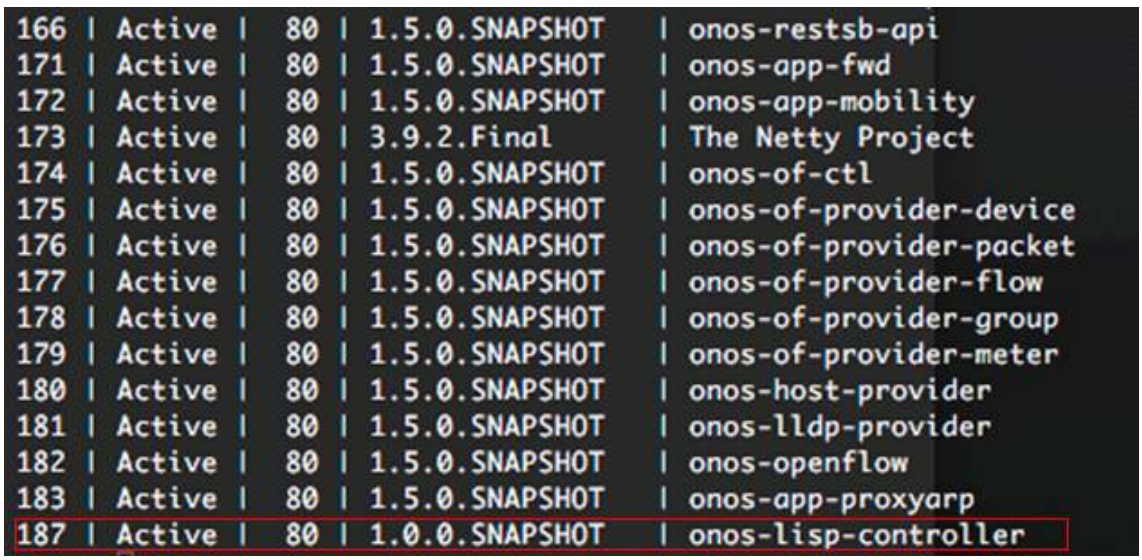

그림 20 LISP controller의 동작 확인

ONOS에서는 Karaf[9]의 로그 시스템을 이용하고 있으며 tl 툴을 이용하면 로그를 확인할 수 있다. 그림 21은 LISP 프로바이더 실행 후 로그 내용으로 LISP 프로바이더가 ONOS에 설치가 되었고 동작하고 있다는 것을 나타내고 있다. 로그 시스템을

이용하여 ONOS 개발에 필요한 정보를 확인 할 수 있다.

```
2016-05-08 17:01:43,225 | INFO  | l for user karaf | Controller
      | 183 - org.onosproject.onos-lisp-ctl - 1.4.1.SNAPSHOT | Started
2016-05-08 17:01:43,225 | INFO  | l for user karaf | Controller
      | 183 - org.onosproject.onos-lisp-ctl - 1.4.1.SNAPSHOT | init
2016-05-08 17:01:43,240 | INFO  | l for user karaf | Controller
      | 183 - org.onosproject.onos-lisp-ctl - 1.4.1.SNAPSHOT | Listening port 43
42
2016-05-08 17:01:43,240 | INFO  | l for user karaf | LispControllerImpl
      | 183 - org.onosproject.onos-lisp-ctl - 1.4.1.SNAPSHOT | Started
2016-05-08 17:01:43,243 | INFO  | l for user karaf | LispDeviceProvider
      | 184 - org.onosproject.onos-lisp-device-provider - 1.4.1.SNAPSHOT | LISP
provider started
```

그림 21. LISP 실행 시 로그 내용

6. 결론

SDN에 대한 관심이 높아짐과 더불어 기존 네트워크를 SDN으로 교체하기 위한 노력이 계속되고 있다. ONOS는 다른 컨트롤러들에 비해서 높은 가용성과 네트워크 속도를 보장하여 캐리어 레벨의 네트워크를 SDN으로 운용할 수 있을 것으로 기대되고 있다. 하지만 기존 인터넷을 SDN으로 대체하기 위해서는 오랜 시간이 소요될 것으로 보이며 현재 설치되어 있는 네트워크 장비를 모두 SDN 장비로 교체하기에는 무리가 따른다. 따라서 기존 인터넷과 SDN이 공존하면서 차츰차츰 SDN의 영역을 넓혀가면서 대체해 가야 할 것으로 보인다.

장비 제조사에서는 OpenFlow와 기존 IP 라우팅을 지원하는 장비를 선보이거나 펌웨어 업데이트를 통해서 OpenFlow를 지원하고 있는 추세이다. 이러한 인프라스트럭쳐 영역에서의 노력뿐만 아니라 SDN 컨트롤러에서도 기존 인터넷에서 사용하는 제어 프로토콜을 지원하여 SDN의 지원 범위를 넓혀 가고 있다.

본 기술문서에서는 상대적으로 적은 수의 프로토콜을 SBI로 지원하고 있는 ONOS에 LISP을 추가 하는 것을 서술 하였다. 보다 자세한 LISP에 대한 구현은 기술 문서 #13 “LISP Proviier 설계 및 구현”을 참고할 수 있다. 현재 개발된 ONOS 상 LISP은 LISP이 가지고 있는 모든 기능와 응용들을 수용할 수 있는 지에 대한 검증 여부와 보다 많은 메시지 종류의 지원이 필요로 한다. 더불어 ONOS의 코어 계층에서는 보다 높은 수준의 추상화를 지원하여 OpenFlow와 LISP의 연동 및 추가적인 프로토콜들과의 무리 없는 연동이 가능하도록 개발을 계속 진행 중에 있다.

References

[1] BGP Analysis Report, http://bgp.potaroo.net

[2] D. Farinacci, V. Fuller, D. Meyer, D. Lewis, "Locator/ID Separation Protocol", RFC6830, January 2013.

[3] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis, "Locator/ID separation protocol alternative logical topology (LISP+ALT)," Internet Engineering Task Force, RFC 6836, January 2013.

[4] V. Fuller, D. Lewis, and A. Ermagan, V. Jain, "LISP Delegated Database Tree," Internet Engineering Task Force, Internet Draft, Work in Progress draft-ietf-lisp-ddt-01.txt, March 2013.

[5] SS7, http://www.corp.att.com/cpetesting/ss7.html

[6] Van der Merwe, J.E., Rooney, S., Leslie, I.M. and Crosby, S.A., "The Tempest - A Practical Framework for Network Programmability", IEEE Network, November 1997.

[7] 김현종 외, "전달/제어 평면간의 연동구조 표준 동향", 전자통신동향분석 제25권 제1호 2010년 2월, pp.148~155.

[8] Zheng Cai, "Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane", Rice Univ. Ph.D thesis, Aug. 2011.

[9] 윤빈영, 이범철, Dan Pitt, "미래 네트워킹 기술 SDN (Future Networking Technology of SDN)", 전자통신동향분석, Vol. 27, No. 2, Apr. 2012, pp. 129-136.

[10] OpenFlow switch specification version 1.0, https:// https://www.opennetworking.org

[11] http://openvswitch.org/

[12] http://www.necam.com/SDN/doc.cfm?t=PFlowController

[13] http://www-03.ibm.com/systems/networking/software/pnc/index.html

[14] http://www.cplane.net/sdn_platform.php

[15] http://www.bigswitch.com/products/SDN-Controller

[16] ONOS, http://onosproject.org/

[17] ODL, https://www.opendaylight.org/

K-ONE 기술 문서

- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.

- 본 문서와 관련된 문의 사항은 아래의 정보를 참조하시길 바랍니다.
  (Homepage: http://opennetworking.kr/projects/k-one-collaboration-project/wiki, E-mail: k1@opennetworking.kr)

작성기관: K-ONE Consortium
작성년월: 2016/05