K-ONE 기술 문서 #34

RSP Construction and Monitoring Method in Multisite

Document No. K-ONE #34
Version 1.0
Date 2018-12-27
Author(s) 이재욱, 이호찬

K-ONE

ONK
OpenNetworkingKorea

■ 문서의 연혁

버전	날짜	작성자	내용
0.1	2018.11.01	이재욱	목차 작성
0.3	2018.11.21	이재욱,이호찬	1,2 장 작성
0.4	2018.12.02	이호찬	3장 작성
0.5	2018.12.04	이재욱	4장 작성
0.7	2018.12.13	이재욱,이호찬	초안 작성
1.0	2018.12.27	이재욱,이호찬	검토 및 수정

본 문서는 2018년도 정부(미래창조과학부)의 재원으로 정보통신 기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-16-2012, 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. B0190-16-2012, Global SDN/NFV OpenSource Software Core Module/Function Development)

기술문서 요약

본 고는 Multisite 네트워크 환경에서 RSP 구성 및 모니터링 기술 개발에 대한 기술문서이다. 본 고는 다음과 같이 구성된다. 1장은 OpenDaylight 및 OPNFV SFC에 대한 개요에 대해 기술한다. 2장은 Multisite 네트워크에 관한 설명과 in network 모니터링 기법에 대해 기술한다. 3장은 Multisite 네트워크에서의 SFC 개요 및 필요성과 ODL SFC를 기반으로 한 Multisite RSP 구성 방안을 기술한다. 4장은 ODL SFC에서의 RSP Monitoring Framework 개발 내용에 대해 기술한다.

Contents

K-ONE #34. RSP Construction and Monitoring Method in Multisite

그림 목차

K-ONE #34. RSP Construction and Monitoring Method in Multisite

1. OpenDaylight / OPNFV SFC 개요

최근 네트워크가 다양한 서비스를 제공하면서 보안 유지나 성능 향상 등을 위해 Firewall, Deep Packet Inspection (DPI), Network Address Translation (NAT) 등과 같은 미들박스 (Middlebox), 즉 서비스 기능에 대한 의존성이 높아지고 있다. 또한, 특정 플로우 (flow)가 여러 네트워크 서비스 기능을 필요로 하는 경우에는, 서비스들이 해당 플로우에 대하여 논리적인 순서에 따라 적용될 수 있는 합리적인 방법이 필요하다. 즉, 여러 요구사항에 따라 통신 사업자의 목적과 정책에 맞추어 서비스 기능들을 하나의 논리적인 연결로 순서화하는 서비스 기능 체이닝 기술이 주목 받고 있으며, 이에 대한 구조 및 프로토콜 표준화, 실제 구현에 관련된 다양한 연구가 학계, 산업계를 불문하고 활발하게 이루어지고 있다.

이러한 서비스 기능 체이닝은 Software Defined Networking (SDN) 기술 및 Network Function Virtualization (NFV) 기술을 통하여 더욱 효과적이고 유연하게 제공될 수 있다. SDN은 기존 네트워크 장비의 제어 평면과 데이터 평면을 분리시켜 논리적으로 중앙 집중된 컨트롤러를 통해 네트워크 제어 기능을 제공한다. 따라서, SDN 기술을 통해 전체 네트워크 뷰를 유지하며, 요구 사항 및 가변적인 네트워크 상황에 따라 동적으로 서비스 기능 체인을 구성할 수 있다. 한편, NFV 기술이란 고가의 네트워크 장비에 대한 투자비용 (CAPEX) 및 운용비용 (OPEX) 문제를 해결하기 위하여, 다양한 네트워크 서비스 기능들을 범용의 표준 서버 하드웨어에서 실행될 수 있는 소프트웨어로 구현하여 운용하는 것을 의미한다. 관련 표준화 노력의 일환으로, 통신 분야 표준화 단체인 ETSI (European Telecommunications Standards Institute)는 전 세계 주요 통신사업자들과 NFV ISG (Industry Specification Group)을 출범하고, 활발한 표준화 활동을 진행하고 있다.

현재 오픈소스 커뮤니티 기반의 SDN 컨트롤러 개발 프로젝트인 OpenDaylight [1]에서는 서비스 기능 체이닝을 제공하기 위하여, 하위 프로젝트로 SFC 프로젝트를 유지하며 활발한 개발을 진행 중에 있다. OpenDaylight SFC 프로젝트는 IETF SFC [2] 작업 그룹에서 제안한 SFC 구조를 기반으로 개발을 진행중에 있다. 또한, ETSI NFV ISG [3] 에서 제안한 NFV 구조를 기반으로 NFV 레퍼런스 플랫폼을 개발하고 있는 프로젝트인 OPNFV (Open Platform for NFV) [4] 에서도 하위 프로젝트로 SFC 프로젝트가 존재하며 서비스 체인이 기술 개발을 진행하고 있다. 한편, 기존의 OpenDaylight SFC 프로젝트에서는 OPNFV 플랫폼 상에서의 가상화된 네트워크 기능들 (즉, Virtual Network Function (VNF))에 대한 서비스 기능 체이닝을 고려하지 않고 있기 때문에, OPNFV SFC 프로젝트에서는 OpenDaylight SFC 프로젝트를 업스트림 프로젝트로 하여 추가적으로 필요한 인터페이스 및 기능들에 대한 정의 및 개발을 진행 중이다. 다음으로 1.1과 1.2에서는 OpenDaylight 와 OPNFV에서 개발되고 있는 각각의 SFC 프로젝트에 대해 분석한다.

1.1. OpenDaylight SFC

서비스 기능 체이닝 기술이 중요한 요구사항으로 부상되고 있음에 따라 세계적으로 SDN에서 서비스 기능 체이닝에 대한 다양한 형태의 개념 검증 및 프로토타이핑이 진행되고 있다. 대표적으로 오픈 소스 기반의 SDN 컨트롤러인 OpenDaylight 에서는 서비스 체이닝에 대한 소스 코드를 포함하고 있다. 1.1에서는 OpenDaylight SFC 프로젝트에서 서비스 기능 체이닝 연구 동향 및 구조에 대해 분석한다.

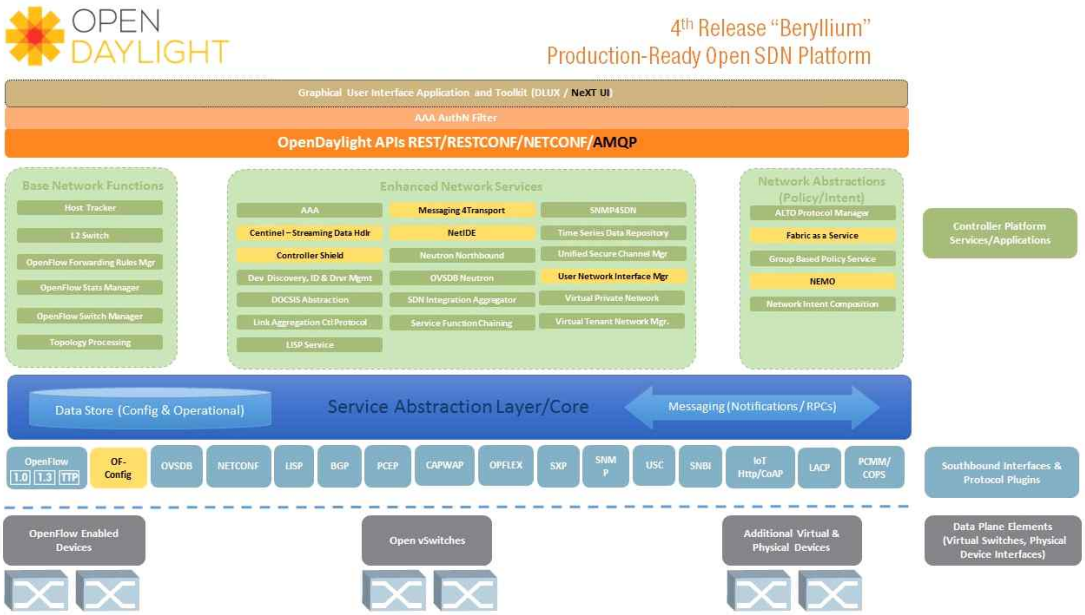

그림 1. OpenDaylight 구조

그림 1은 SDN 기반 오픈 소스 컨트롤러인 OpenDaylight 프로젝트의 버전인 Beryllium의 논리적인 구조를 나타낸다. OpenDaylight은 다양한 네트워크 서비스들을 모듈화된 OSGi 플러그인 형태로 제공한다. 각 모듈은 OpenDaylight 프로젝트에 참여하고 있는 다양한 벤더들에 의해 구현된다. 먼저 가장 상위 계층인 Network Apps & Orchestration계층은 네트워크 제어 및 모니터링을 할 수 있는 네트워크 애플리케이션들로 구성되어 있다. 예를 들어, 네트워크 관리자는 DLUX UI 애플리케이션을 통해 SFC 서비스를 이용할 수 있다. 중간 계층에 위치한 Controller Platform 계층은 다양한 네트워크 서비스들 (네트워크 토폴로지 매니저, 스위치 매니저, SFC, GBP 등) 및 OpenFlow, LISP, SNMP, NETCONF 등의 다양한 Southbound 프로토콜들을 플러그인 형태로 네트워크 애플리케이션에게 제공한다. 하위 계층의 Physical & Virtual Network Devices 계층은 네트워크의 모든 종단점을 연결하는 물리적인 혹은 가상화된 스위치와 라우터 등으로 구성된다. 서비스 기능 체이닝 및 Group Based Policy 또한 OpenDaylight 프로젝트의 하위 프로젝트로써 플러그인 형태로 개발 중에 있다. OpenDaylight 의 SFC 개발 프로젝트에서는 IETF SFC 작업 그룹에서 제안하는 SFC 구조를 기반으로 하여 개발을 진행하고 있다. 아래는 해당 구조에서 등장

하는 핵심 용어에 대한 설명을 나타낸다.

- Service Function Chain (SFC) : 서비스들의 논리적인 순서를 나타낸다.
- Service Function Path (SFP) : 주소를 갖는 서비스 인스턴스들의 순서를 나타낸다. 특정 SFP는 유일한 SFP 식별자를 통해 구별된다.
- Service Index (SI) : 특정 트래픽 (플로우 혹은 패킷)이 받아야 할 서비스 인스턴스들의 갯수를 나타내며 SFP의 길이에서 서비스를 받을 때마다 1씩 감소한다.
- Network Service Header (NSH) [5] : NSH에는 SFP 식별자 정보와 SI 정보가 담겨 있다. NSH를 통해 서비스 평면이라 불리는 오버레이 네트워크를 구성한다.
- Service Function (SF) : 미들박스 기능을 제공하는 요소이다.
- Service Function Forwarder (SFF) : SFP를 따라 스위칭 역할을 수행하는 요소이다. 서비스 오버레이 네트워크의 스위치로 동작한다.
- Service Classifier (SC) : 유입되는 트래픽을 분류하여 NSH 인캡슐레이션을 수행하는 요소이다.

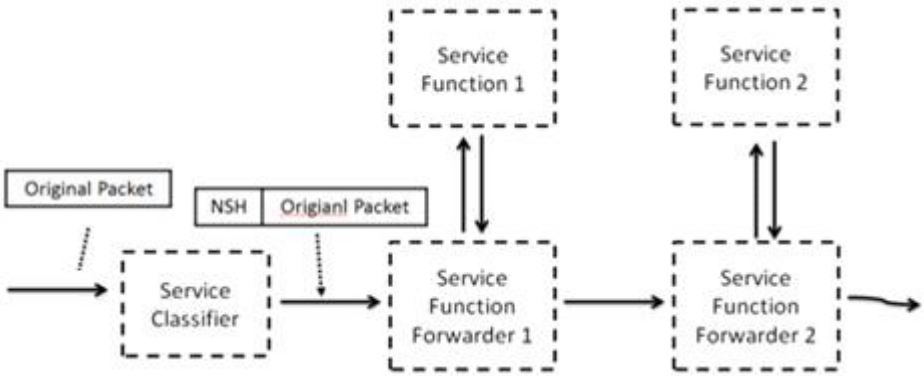

그림 2. OpenDaylight SFC 논리적 구조

그림 2는 서비스 기능 체이닝의 논리적인 구조를 나타낸다. SC는 미리 정의된 정책에 기반하여 유입된 외부 트래픽이 어떠한 SFP를 따르게 될 지를 정하고 결정된 SFP에 대한 식별자 정보를 NSH에 추가한다. NSH 인캡슐레이션이 된 패킷들은 경로 상의 첫 번째 SFF로 전달된다. SFF는 SFP 식별자, SI, 다음 서비스 홉 (NH: Next Hop)으로 구성된 서비스 라우팅 테이블을 유지하여, NSH 패킷이 도달하면 NSH의 SFP 식별자와 SI에 매칭되는 SF로 패킷을 포워딩 한다. 그림 3은 OpenDaylight SFC의 동작 과정을 나타낸다. 박스 부분은 OpenDaylight 컨트롤러의 기능을 나타낸다. 먼저 OpenDaylight SFC 웹 UI를 통해 SFP를 구성, 컨트롤러의 SFC 저장소에 저장한다. SFC 저장소에 등록된 SBI 플러그인들은 SFP 정보를 전달받고, 이를 기반으로 데이터 평면에 위치한 SFF의 서비스 라우팅 테이블을 조작한다. 이 때 OpenFlow 프로토콜 혹은 REST 프로토콜 등으로 SFF를 조작할 수 있다.

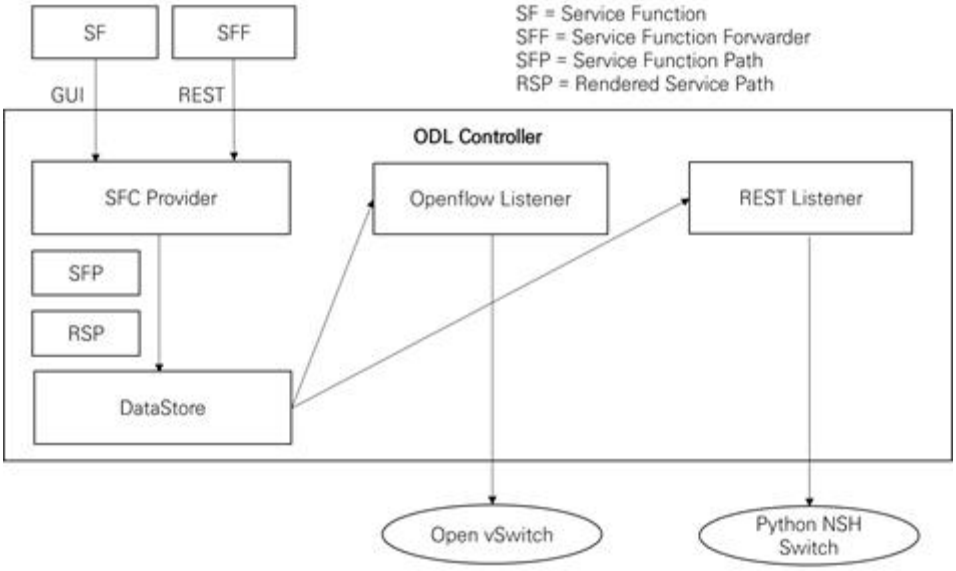

그림 3. OpenDaylight SFC의 동작과정

1.2. OPNFV SFC 프로젝트 개요

ETSI ISG 산하에 NFV라는 표준 그룹은 NFV 기술 분야에서 통신사업자 및 산업체가 요구하는 산업 규격을 정의한다. ETSI ISG가 제안한 규격을 기반으로 OPNFV(Open Platform for Network Functions Virtualization)는 NFV에 대한 오픈 플랫폼을 개발하고 있다. 아래 그림은 NFV기술을 실현하기 위한 프레임워크를 나타내는 그림이다.

NFV 프레임워크가 만족해야하는 구조적인 특징은 다음과 같다. Virtualized Network Functions (VNF)를 관리하고 조율할 수 있는 EMS, OSS/BSS와 같은 시스템이 필요하다. 그리고 VNF는 서로 다른 하이퍼바이저 위에서 동작하고 자유롭게 컴퓨팅 자원에 접근이 가능해야 한다. VNF를 하드웨어적으로 구성할 수 있는 요건이 만족된다면, Virtualized Infrastructure Manager (VIM)을 통해 물리적인 하드웨어 자원을 추상화하여 VNF의 자원의 상태 및 가상 네트워크를 구성한다. Orchestrator은 VIM의 도움으로 가상화된 네트워크 상태를 바탕으로 VNF Forwarding Graph (VNFFG)를 구성하며, 정책 및 결정을 내리는 역할을 하고, 그리고 컴퓨팅, 저장소, 네트워크 기능을 지원하는 물리적인 하드웨어 자원을 가상화하여 VNF를 실행시킬 수 있도록 지원하는 역할을 하는 Network Function Virtualization Infrastructure (NFVI) 구성 요소와 네트워크 서비스의 자원을 조율한다.

OPNFV는 네트워크 장비 및 기능을 가상화하기 위한 NFV의 오픈 플랫폼이다. 네트워크 기술이 빠르게 발전함에 따라 고가의 네트워크 장비에 대한 투자비용과 운용비용 문제를 해결하기 위해 네트워크 및 장비를 가상화하기 위한 NFV 기술에

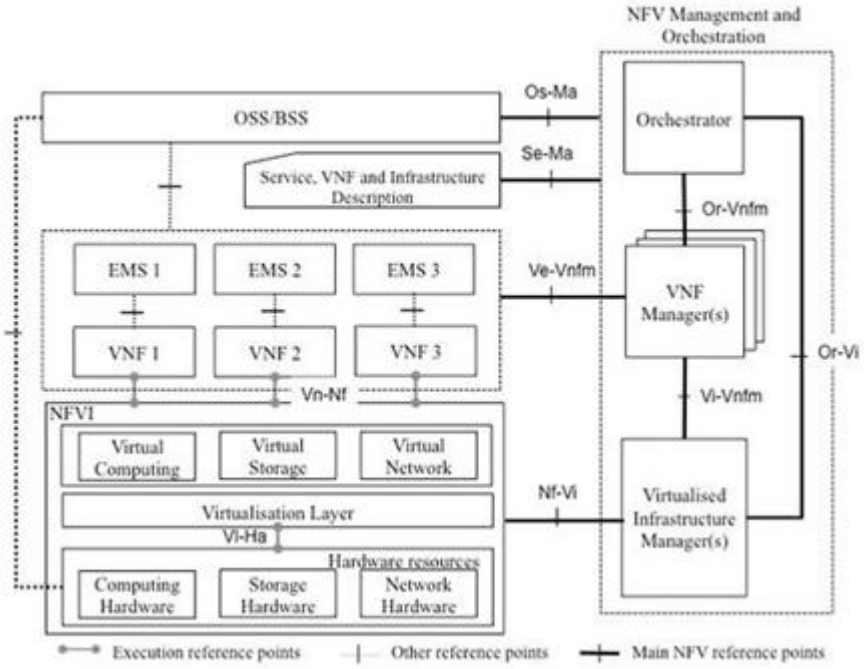

그림 4. NFV 프레임워크

대한 연구가 활발해졌고, NFV 플랫폼 및 솔루션 개발에 대한 필요성이 대두되었다. 이에 따라 최근 Linux Foundation이 공개 소프트웨어 기반의 OPNFV 프로젝트를 발표하였다. OPNFV에서 개발하려고 하는 부분은 각 네트워크의 요소 자체의 기능을 개발하는 것보다 각 네트워크 요소들 간의 인터페이스가 서로 간에 호환이 잘 될 수 있도록 지원하는 것이다. 즉, 모든 통신 산업 개발자들이 NFV를 개발하는데 있어서 각 요소들이 쉽게 장착될 수 있도록 호환성을 제공하는 데 목적을 둔다.

OPNFV SFC 프로젝트는 오픈 소스 기반의 SDN 컨트롤러인 OpenDaylight을 활용하여 OPNFV 플랫폼 상에서 관리하는 VNF들 간에 서비스 체인을 구성할 수 있는 기능을 구현하는 것을 목표로 한다. 한편, Opendaylight의 버전인 Beryllium에서는 IETF SFC 구조를 기반으로 Opendaylight 컨트롤러가 관리하는 데이터 평면상에 서비스 체인을 구성하는 기능을 제공하고 있다. 기존의 OpenDaylight SFC에서는 OPNFV 플랫폼 상에서의 서비스 체이닝을 고려하고 있지 않기 때문에, OPNFV SFC에서는 Opendaylight SFC 프로젝트를 업스트림 프로젝트로 하여 추가적으로 필요한 인터페이스 및 기능들에 대한 정의 및 개발을 계획하고 있다.

아래 그림은 OPNFV SFC프로젝트의 초기 구상도를 나타낸다. VIM (Virtualized Infrastructure Manager)에서 네트워크 자원에 대한 관리는 Opendaylight 컨트롤러를 사용하고, 컴퓨팅 자원에 대한 관리는 OpenStack을 사용함을 알 수 있다. VNFM (VNF Manager)로는 OpenStack Tacker가 사용된다. 한편, 노란색 박스로 표현된 Opendaylight SFC, SFF, SC는 OPNFV SFC 프로젝트에서 개발을 진행하는 컴포넌트들을 나타낸다. Opendaylight SFC는 OpenDaylight 컨트롤러의 SFC 플러그인

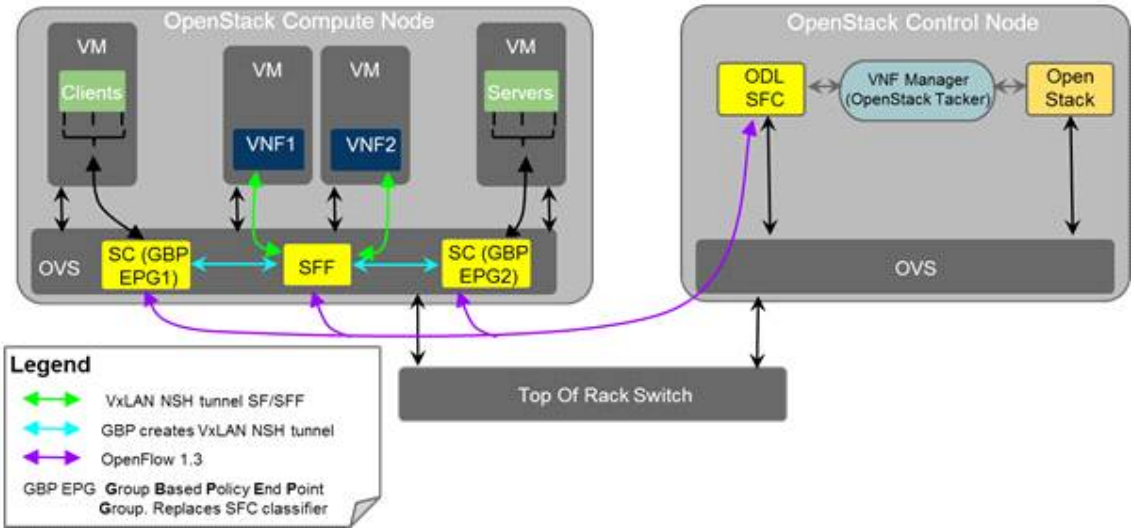

그림 5. OPNFV SFC 초기구상도

을 의미하며 VNFM으로부터 전달받은 서비스 체인 요구사항을 데이터 평면에 렌더링하기 위해 데이터 평면의 SC 및 SFF를 조작한다. 먼저 SC는 미리 정의된 정책에 따라 유입되는 트래픽을 분류하여 NSH 인캡슐레이션을 수행하는 요소이다. 이 때, Opendaylight 의 GBP (Group-based Policy) 프로젝트에서 정의하는 EPG (End-point Group) 형식으로 트래픽을 분류한다. 그림 5의 예시에서는 클라이언트들 및 서버들이 각각 GBP EPG1 그리고 GBP EPG2로 정의된다. 이 때, 정의된 정책은 EPG1과 EPG2사이의 트래픽들에 대해서 VNF1 및 VNF2로 정의되는 서비스체인을 지나도록 하는 것이다. 왼쪽의 SC는 이에 따라 VM으로부터 들어오는 트래픽들 중 소스가 EPG1이고 목적지가 EPG2인 트래픽에 대해 VNF1->VNF2에 해당하는 SFP 식별자 정보를 담은 NSH 인캡슐레이션을 수행한다. 오른쪽 SC는 VM으로부터 들어오는 트래픽들 중 소스가 EPG2이고 목적지가 EPG1인 트래픽에 대해 VNF2->VNF1에 해당하는 SFP 식별자 정보를 담은 NSH 인캡슐레이션을 수행한다. 인캡슐레이션이 끝나면 SFP상의 첫 번째 SF가 연결된 SFF에게로 패킷을 전달한다. 한편 SFF는 SFP를 따라 스위칭 역할을 수행하는 요소로 Opendaylight SFC는 SFF 의 서비스 라우팅 테이블에 VNF1->VNF2 그리고 VNF1->VNF2로 정의되는 서비스 체인에 대한 <SFP 식별자, SI> 엔트리와 이에 대한 NH들을 설치한다. SFF는 SC로 부터 NSH 인캡슐레이션된 패킷을 받으면 매칭되는 엔트리의 NH으로 패킷을 전달한다. 이러한 일련의 과정을 거침으로써 EPG1->EPG2로의 트래픽은 VNF1->VNF2 순서로 서비스를 받게 되고 EPG2->EPG1로의 트래픽은 VNF2->VNF1 순서로 서비스를 받게 된다.

1.3. Tacker SFC 프로젝트 개요

본 장에서는 클라우드 컴퓨팅 오픈 소스 프로젝트 OpenStack 프로젝트의 하위 프로젝트인 Tacker 프로젝트에 대해 설명한다. Tacker 프로젝트는 아래 그림 6에서 VNFO와 NFVO 역할을 타겟으로 하는 프로젝트이다. VNF를 고려하여 서비스 기능 체이닝을 구성하기 위해서는 NFV 환경에서 VNF를 관리하는 OpenStack과 같은 플랫폼이 필요하다. OpenStack은 클라우드 컴퓨팅 오픈 소스 프로젝트로, 2010년 미 항공우주국과 락 스페이스사가 오픈스택을 설립하였고, 2012년 9월 비영리 단체인 오픈스택 파운데이션 (OpenStack Foundation) 으로 다시 출범하게 되었다. 첫 번째 버전인 오스틴 (Austin) 을 시작으로, 매 해 6개월 주기로 2번씩 새로운 버전이 릴리즈 되고 있다. OpenStack의 개념적인 구조는 아래 그림 7과 같다. 클라우드 컴퓨팅 시스템을 구축하기 위한 것으로, 크게 컴퓨트 서비스를 Nova로, 오브젝트 스토리지 서비스는 Swift, 운영체제 이미지 관리를 위한 서비스는 Glance로 구성되어 있다. 또한 위의 Nova, Swift, Glance와 같은 서비스들의 인증을 담당하는 Keystone, 서비스들을 쉽게 사용하기 위해 사용자들에게 대시보드를 제공하는 Horizon, 오케스트레이션 서비스르 제공하는 Heat, 미터링 서비스 Ceilometer도 OpenStack의 버전업에 따라 새롭게 서비스가 추가되었다.

Tacker 프로젝트는 NFV 환경에서 VNF를 관리하는 VNF Manager 기능과 NFV 서비스들을 관장하고 관리하는 NFV Orchestrator 기능을 지원하기 위한 프로젝트이다. 즉, Tacker 프로젝트에서 제공하는 기능은 아래 그림에서 빨간색 박스 부분이다. 가상화된 리소스를 바탕으로 VNF를 관리하고 구성하는 역할을 담당하는 VNFM과 NFV 환경에서 네트워크의 전체적인 조율 및 관리에 대한 기능을 수행하는 NFVO로 구성된다. VNFM과 NFVO에서 VNF의 배치와 네트워크 서비스를 관리하는 데는 Network Service Descriptor (NSD)와 VNF 카탈로그에 저장되어 있는 Virtualized Network Function Descriptor (VNFD)를 기반으로 이루어진다.

Tacker의 구조는 아래 그림과 같다. VIM 부분의 VNF 생성을 위한 부분과 관리를 위한 드라이버 모니터링을 위한 드라이버가 존재하며, 현재 ping과 HTTPping 그리고 alarm-based monitoring이 있다. NFVO에서는 포워딩그래프와 네트워크 서비스 오케스트레이션, 다중 VIM 환경에서의 VNF 배치. 리소스 체크 및 할당 위한 VNFM으로 구성되어 있다. Tacker는 ETSI에서 논의되고 있는 기본 형태를 기반으로 제안하고 있기 때문에 특정 벤더에 종속되지 않고 멀티 벤더로부터 VNF를 배치하는 것이 가능하다.

Tacker 프로젝트는 NFV Orchestration 기능을 타겟으로 한 프로젝트로 대표적으로 VNF manager 기능이 있다. VNF manager의 기능은 다음과 같다. OpenStack과 같은 VIM 위에서 VNF의 lifetime을 관리한다. 즉, VNF의 생성 및 삭제, 모니터링, 스케일링을 통해 전반적으로 VNF를 관리한다. VNF를 배치하는 과정은 다음과 같다. Tacker 프로젝트는 template 기반으로 VNF를 생성 및 관리하기 때문에,

Horizon 혹은 Command line을 통해 Tacker VNFD catalog에 TOSCA VNFD template 을 온보딩 시켜야 한다. VNFD에는 기본적으로 VNFD의 id, 기본정보, vdu 정보, virtual link 정보, 모니터링 및 오토 스케일링 관련 파라미터 등이 포함되어 있다. 이를 통해, VNF의 생성뿐만 아니라, 생성이후에 필요한 관리 기능에 대한 정의 및

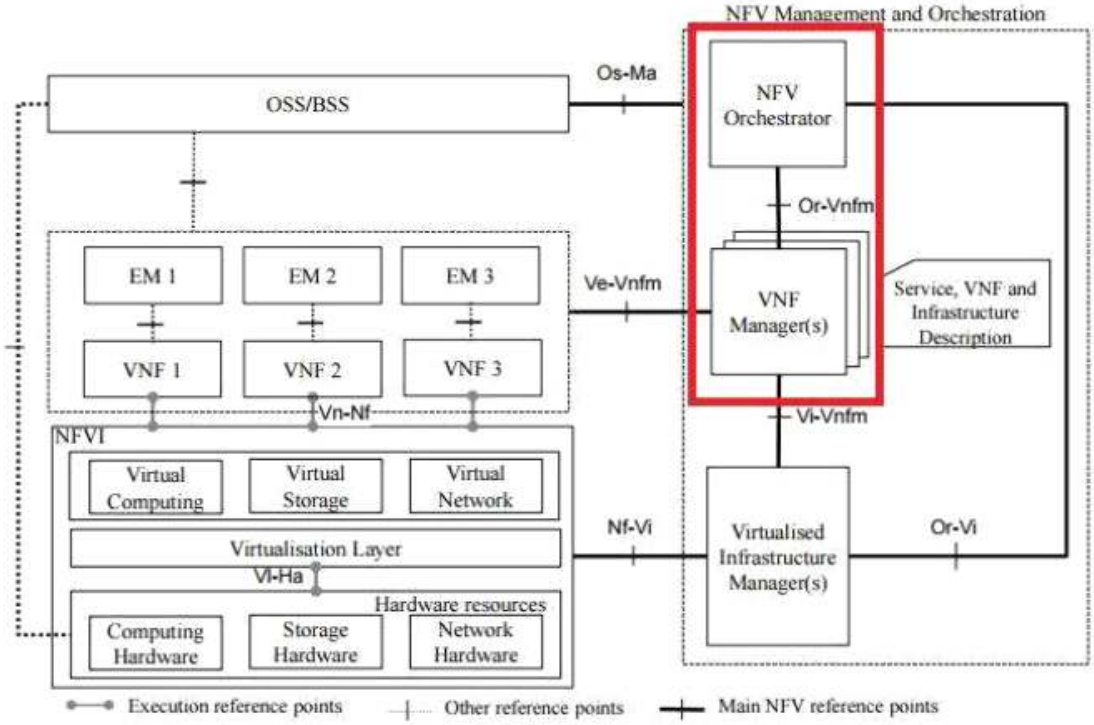

그림 6. OpenStack Tacker target 기능

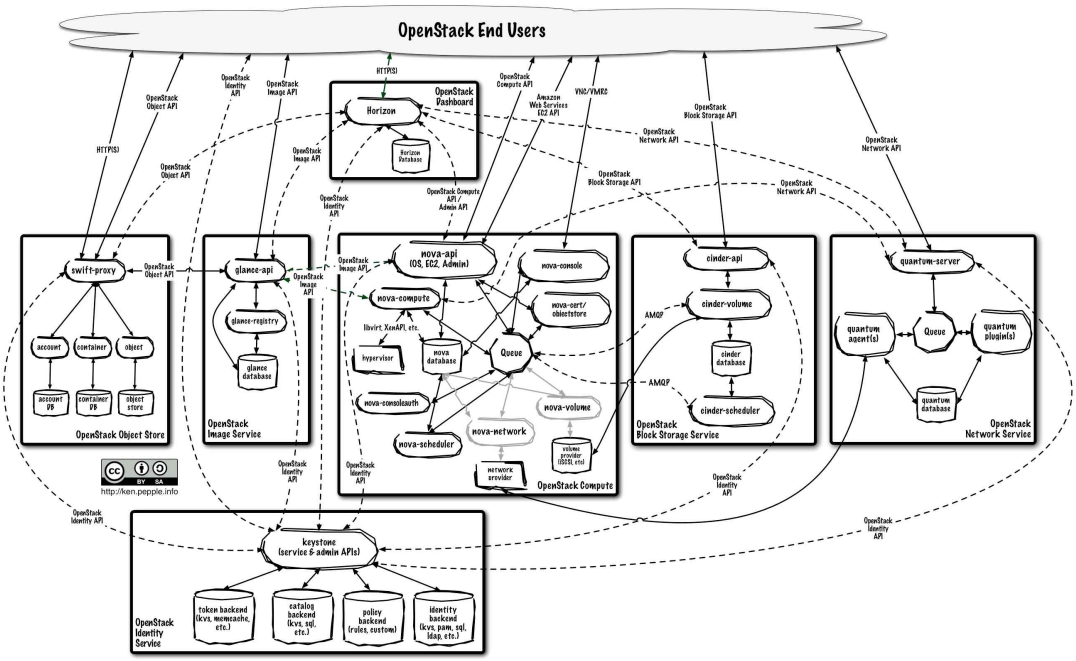

그림 7. OpenStack 개념적 구조

모니터링 이후의 액션에 대한 정의도 포함되어 있어 유용하다.

VNFFG Manager는 Tacker project에 새롭게 추가된 기능이다. NFV 진영에서는 단순히 VNF를 생성하고 관리하는 단계를 넘어, 생성된 VNF를 SFC 기술을 이용하여 플로우의 요구사항에 따라 VNF를 논리적으로 연결하여 그래프를 구성하는 단계까지를 필요로 한다. 최종적으로, 단순히 논리적이고 추상적으로 구성된 그래프를 오버레이 네트워크 아래의 물리적인 네트워크에 실질적으로 렌더링하는 것이다. 현재 Tacker project에서는 단일 경로의 SFC만 지원 가능한 상태이다. 이 문제를 해결하기 위해서, 다중 경로 그래프를 여러 개의 단일 경로 SFC로 파싱하여 더 확장된 VNFFG를 생성할 수 있도록 한다.

2. Multisite 네트워크와 in network 모니터링

2.1. Multisite 네트워크

현재의 소프트웨어 정의 인프라 패러다임에서 핵심 기술들인 SDN, NFV 및 클라우드 네트워크에 대한 연구는 단일 가상 네트워크 인프라(NFVI), 즉 단일 사이트를 타겟으로 꾸준히 진행되고 있다. 그러나 최근에 SDN, NFV 및 클라우드 기술들이 대규모의 인프라 환경에 도입하게 됨에 따라 단일 사이트 뿐만 아니라 여러 사이트들이 공존하는 네트워크에 대한 고려가 필요해졌다. 즉, 단일 사이트를 타겟으로 하는 연구 뿐만 아니라, 여러 네트워크 관리자가 관리 및 소유하고 있는 사이트들을 아우르는 환경인 멀티 사이트에서 앞서 언급한 기술들의 연구 필요성이 증가하고 있다.

아래 그림은 멀티 사이트 네트워크 구조를 띠는 K-ONE Playground 환경을 나타낸다. K-ONE Playground에서는 각 기관에 K-Cluster들이 배포되어 있고 각 K-Cluster는 소프트웨어 정의 인프라 패러다임에 대응하는 다양한 SDN, NFV 및 클라우드 실증을 제공하기 위한, Edge-Cloud 모델에 대응하는 소규모 클러스터 형태의 테스트베드로 정의된다. 이와 관련된 사항들은 'K-ONE 기술문서 16#_멀티사이트 클라우드 실증환경에 대응하는 K-Cluster 중심의 K-ONE 공용개발환경 설계 및 활용 방안' 에 자세히 설명되어 있다 [6]. 따라서 각 K-Cluster들은 하나의 사이트에 대응되며 각 사이트들은 독립적인 SDN, NFV 및 클라우드 네트워크 인프라를 구성하고 있다. K-Cluster들을 관리하는 각 사이트를 하나의 인프라로 연결함으로써 멀티 사이트 네트워크 구조를 띠고 있다. 각 사이트의 K-Cluster들은 중앙 기관에 존재하는 K-DevOps Tower에서 관리가 가능한 계층적 구조로 배포되어 있다.

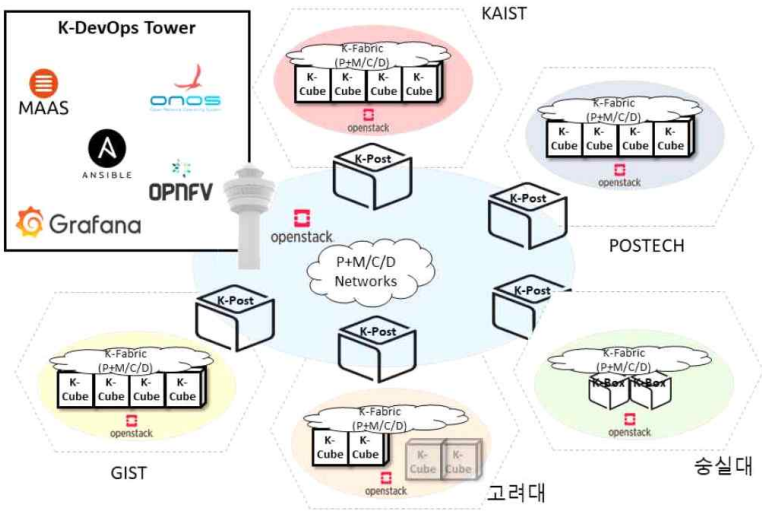

그림 8. 멀티 사이트 구조의 K-ONE Playground의 환경

2.1.1. Multisite 네트워크의 특징

단일 사이트의 SDN, NFV 및 클라우드 네트워크 인프라에서는 네트워크 관리자가 Virtual Infrastructure Manager(VIM)를 통해 자신의 사이트 내에 있는 모든 네트워크 자원(서버 및 스위치 등)들의 상황 및 트래픽 상황을 파악할 수 있기 때문에 상황에 따라 관리자가 원하는 대로 자원 배포와 트래픽 관리가 가능하여 단일 사이트에서의 네트워크 관리를 효율적으로 할 수 있다. 그러나 멀티 사이트 네트워크 환경에서 각 사이트 네트워크 관리자는 다른 사이트의 네트워크 자원 및 트래픽 상황을 알 수 없기 때문에 트래픽들이 다른 사이트를 지나야 하는 경우에는 로드 밸런싱 및 최적의 라우팅을 위해 트래픽 경로를 제어하는 방법에 한계가 있다는 특징이 있다.

2.1.2. Multisite 네트워크 관련 오픈소스 프로젝트

OPNFV의 Multi-site Virtualized Infrastructure(Multisite) 프로젝트는 대표적인 VIM인 OpenStack이 다수 연결된 멀티 사이트 환경에서 NFV 시나리오의 지원을 모티브로 하는 프로젝트이다.

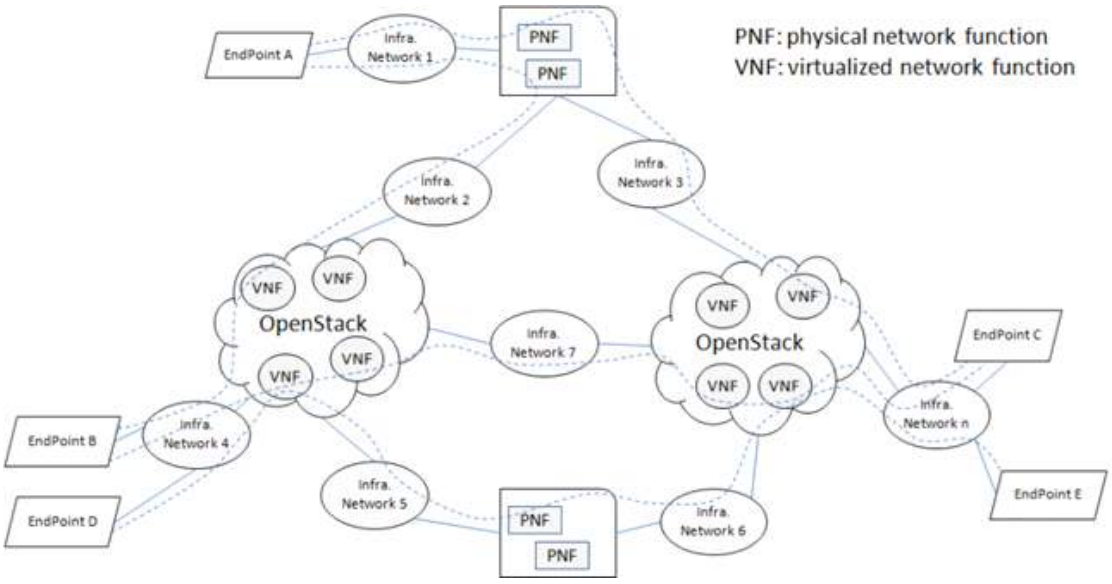

그림 9. OPNFV의 Multi-site Virtualized Infrastructure 환경

해당 프로젝트는 위 그림과 같이 NFV 클라우드 및 VIM들이 멀티 사이트 내에서 지역적으로 분포되어 있고 NFV 클라우드는 가상화되지 않은 전형적인 텔코 네트워크와 공존할 수도 있는 상황을 고려하였다. 해당 프로젝트는 VNF의 관리 및 네트워크 서비스 카탈로그와 같은 기능들처럼 VNFM(Virtual Network Function Manager)나 NFVO(Network Function Virtualization Orchestrator)와 관련된 기능들은 프로젝트

타겟으로 하고 있지는 않고 멀티 사이트를 지원하기 위한 OpenStack의 모든 구성 요소들(Nova, Cinder, Neutron, Glance, Ceilometer, Keystone)의 개선에 초점을 맞추고 있다.

Multisite 프로젝트의 유즈케이스로 VIM 들간의 High Availability 지원, 여러 지역의 사이트들에 어플리케이션 레벨의 데이터 및 설정들을 복제하는 Geo-site Redundancy(GR)케이스, 사이트들 간의 로드 밸런싱을 수행할 수 있는 Geo-Site Load Balancing(GLB)가 있다 [7].

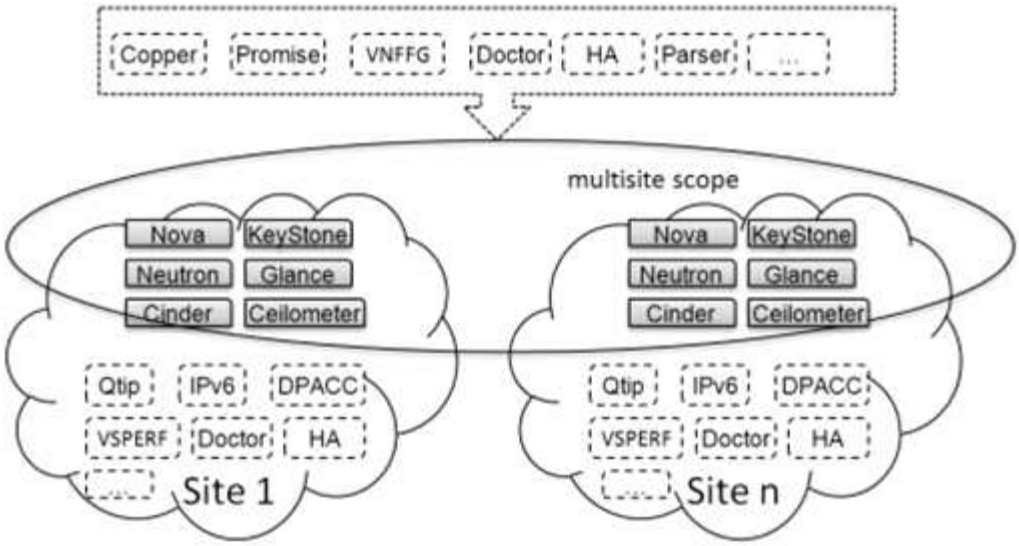

그림 10. OPNFV Multisite 프로젝트의 스코프

또 다른 멀티 사이트 환경을 타겟으로 하는 오픈소스 프로젝트로 ONAP(Open Networking Automation Platform)의 Multi VIM/Cloud for Infrastructure Providers(Multi VIM/Cloud) 프로젝트가 있다. ONAP은 네트워크 가상 인프라에서 여러 네트워크 서비스들과 VNF들을 배포, 동작 및 관리하는 오픈소스 프로젝트이다. 서비스 프로바이더들은 가상 및 클라우드 인프라를 구현할 때, 필요에 따라 원하는 배포가 가능한 유연성을 갖길 원한다. 예를 들면, 사업장 내의 개인 클라우드, 퍼블릭 클라우드 또는 하이브리드 클라우드들을 사업자가 원하는 방식으로 배포할 수 있길 원하는데, 이를 지원하기 위해 제안된 프로젝트가 Multi VIM/Cloud 프로젝트이다 [8].

Multi VIM/Cloud 프로젝트는 ONAP 플랫폼이 멀티 인프라 환경의 배포 및 동작을 가능하도록 하는 것을 목표로 한다. 예를 들면, 여러 OpenStack 사이트들로 이루어진 멀티 인프라 환경이 있을 수 있고, OpenStack 이외에 퍼블릭 및 개인 클라우드들인 VMware, Azure, 그리고 마이크로 서비스 컨테이너들이 공존하는 환경도 목표로 하고 있다. 해당 프로젝트는 Multi-VIM/Cloud Mediation Layer를 제공하여 다수의 인프라들을 지원함과 동시에, 벤더에 종속되는 것을 막기 위해 네트워크 백엔드들까지 지원하도록 한다. 또한, ONAP 플랫폼을 underlying 클라우드 인프라의 발전과

분리시킴으로써, underlying 클라우드 인프라를 업그레이드하는 동안, 배포된 ONAP에 미칠 수 있는 영향을 최소화시키도록 하였다.

아래 그림은 Multi-VIM/Cloud Mediation Layer의 구조를 나타낸다. 해당 레이어는 여러 모듈 구조를 포함하고 있는데, Provider Registry 모듈은 인프라 사이트, 위치, 영역과 각각의 특징 및 용량들을 등록하는 역할을 수행한다. Infra Resource는 Service Orchestration으로부터 오는 자원 요청들을 관리하여 VM과 VNF들이 올바른 인프라 내에서 생성되고 실행되도록 한다. SDN Overlay는 대응되는 클라우드 인프라를 로컬 SDN 컨트롤러들을 통해 오버레이 네트워크를 설정하는 역할을 한다. VNF Resource LCM은 VM 라이프 사이클 관리를 담당하며, FCAPS는 인프라에서 사용되는 자원 메트릭들을 DCAE(Data Collection, Analytics and Events)에게 보고하는 역할을 담당한다.

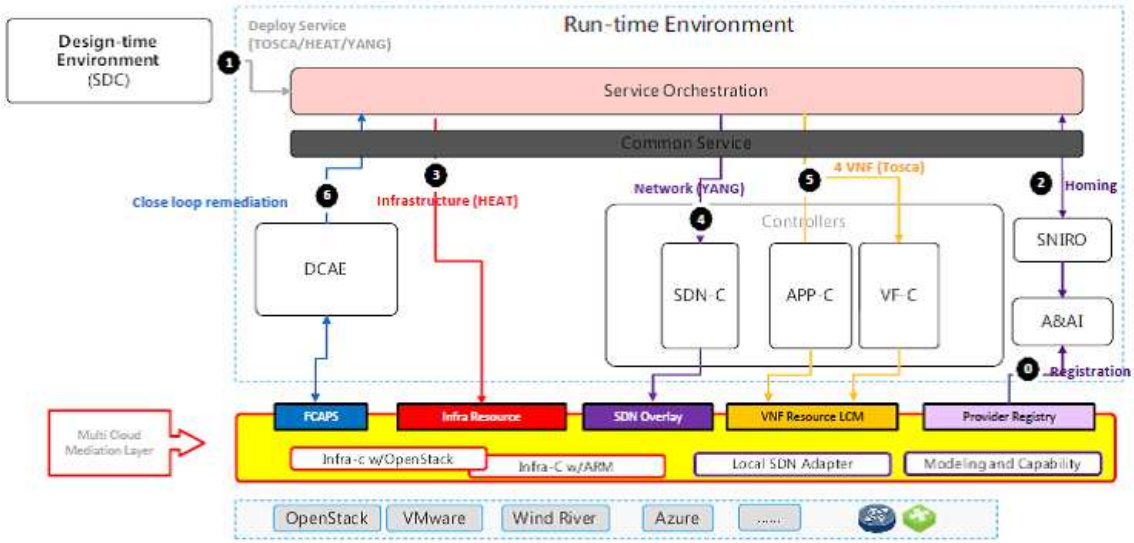

그림 11. Multisite VIM/Cloud Mediation Layer 구조

또한, 범용 northbound 인터페이스와 앞서 말한 기능적인 모듈들의 Multi-Cloud API들을 제공하여 SO, SDN-C(SDN Controller), APP-C(Application Controller), VF-C(Virtualized Function Controller)와 DCAE 등이 이용할 수 있게 한다. 프로젝트 전반적으로, 여러 기능들은 ONAP 유저들이 해당 기능들을 사용할지 안할지를 결정할 수 있도록 구현된다.

2.2. In network 모니터링 기법

네트워크 정보를 수집하는 방식으로는 크게 In-band 와 Out-of-band 의 두 가지 방식으로 나뉜다. Out-of-band 방식은 별도의 탐지 패킷을 생성하여 네트워크 정보를 수집하는 반면, In-band 방식은 기존 데이터 트래픽에 네트워크 정보를 삽입하여 별도의 탐지 패킷 생성 없이 네트워크 정보를 수집하는 방식이다. 본 절에서는 In-band 방식의 네트워크 정보를 수집하는 프레임워크인 P4진영에서 제안하는 In band Network Telemetry와 CISCO 에서 제안하는 In-situ Operations, Administration, and Maintenance를 분석한다.

2.2.1. P4 기반의 In band Network Telemetry 기능

In-band Network Telemetry기능은 데이터 평면 프로그래밍 언어인 P4를 통해 데이터 평면의 네트워크 정보에 대한 수집을 가능케 해주는 기능이다. INT의 기본적인 정보 수집 방식은 P4를 통해 헤더의 자유로운 처리가 가능하다는 점을 이용하여 패킷 헤더의 VXLAN Option Field(per GPE extension), Geneve Option Field, Network Service Header Option Field, TCP/UDP Option Field를 이용하여 네트워크 스위치에서 수집할 수 있는 메타데이터 정보를 해당 Option Field에 캡슐화 형식으로 포함시켜 전달한다. INT헤더는 아래 그림과 같이 정의된다.

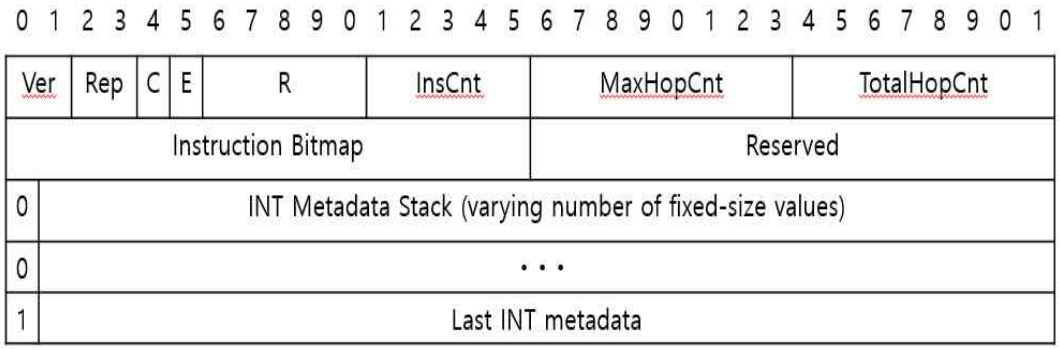

그림 12. INT헤더의 포맷

헤더 포맷의 첫 번째와 두 번째 스택은 INT instruction을 나타낸다. 즉, 스위치에서 수집할 정보는 두 번째 스택에 16-bits의 Instruction Bitmap으로 정의되며, INT specification에서 정의하고 있는 Instruction Bitmap은 아래와 같다.

- bit0 (MSB): Switch ID (스위치 레벨의 정보)
- bit1: Ingress port ID (Ingress 레벨의 정보)

- bit2: Hop latency (Egress 레벨의 정보)
- bit3: Queue occupancy (버퍼 레벨의 정보)
- bit4: Ingress timestamp (Ingress 레벨의 정보)
- bit5: Egress port ID (Egress 레벨의 정보)
- bit6: Queue congestion status (버퍼 레벨의 정보)
- bit7: Egress port tx utilization (Egress 레벨의 정보)
- The remaining bits are reserved

정의된 Instruction Bitmap에서 확인할 수 있듯이 8-bits는 새로운 모니터링 요소를 위한 공간이다. 현재 정의된 8-bits의 모니터링 요소 외의 추가적인 정보가 필요하면 사용자는 비어있는 8-bits의 공간에 새로운 요소를 추가하여 얻고자 하는 네트워크 정보를 추가할 수 있다. instruction 헤더를 제외한 나머지 스택은 INT 메타데이터를 나타낸다. 모니터링 경로에 있는 INT 스위치는 Instruction 헤더를 보고 해당하는 스위치 정보를 메타데이터 형식으로 INT 헤더에 삽입한다.

아래의 그림은 INT를 통한 네트워크 정보 수집의 예제를 나타낸다.

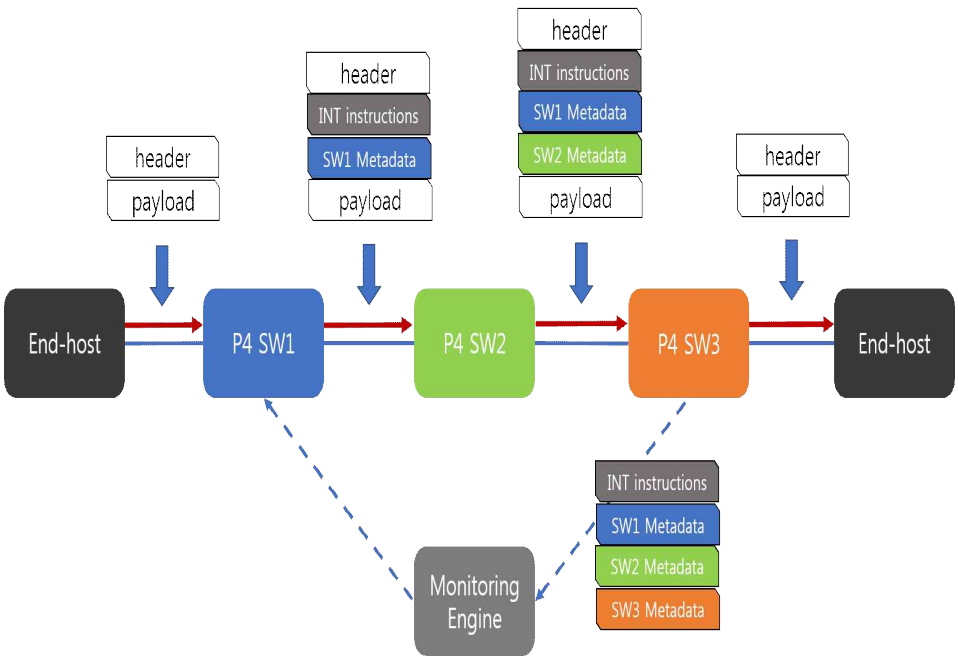

그림 13. INT를 통한 네트워크 정보 수집의 예제

해당 예제는 패킷이 거쳐 온 스위치의 ID와 해당 스위치들의 큐 상태를 사업자가 INT기능을 통해 모니터링 하는 예제이다. 우선 사업자는 모니터링 할 네트워크의 시작점과 끝점을 정한다. 상기의 예제에서는 패킷인 SW1 ==> SW2 ==> SW3로 흐른다고 가정 할 때, 사업자는 SW1에서부터 SW3까지 네트워크 정보를 수집 할 것이라고 결정하였다. 이때 SW1은 INT source노드로 설정되며, source노드는 아래와 INT instruction 헤더를 패킷에 삽입하는 기능을 수행하며, 또한 Instruction 헤더에 정의된 네트워크 정보를 메타 데이터로 삽입시킨다.

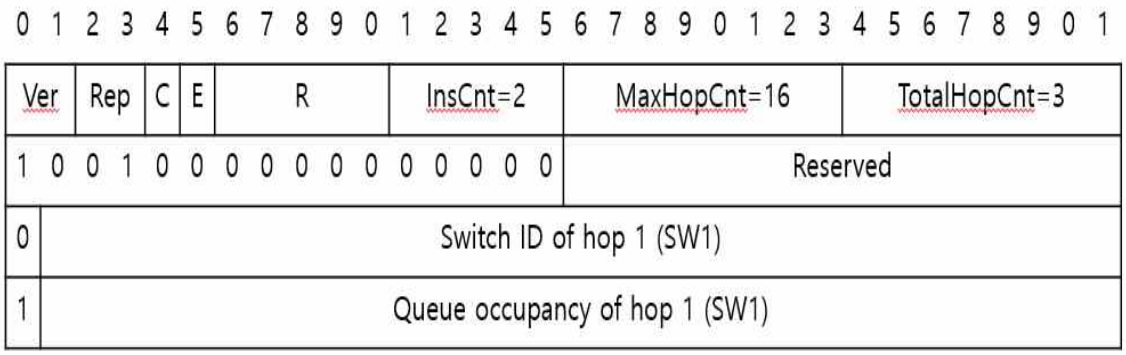

그림 14. Source (SW1)에서의 INT 메타데이터 정보

SW2의 경우 모니터링 경로의 중간 노드이다. 해당 노드를 Transit 노드로 정의된다. SW2는 SW1로부터 전달 된 INT 패킷의 INT instruction 헤더에 따라 스위치 정보를 아래 그림과 같은 메타데이터 형식으로 패킷에 삽입한 후에 다음 홉으로 전달한다.

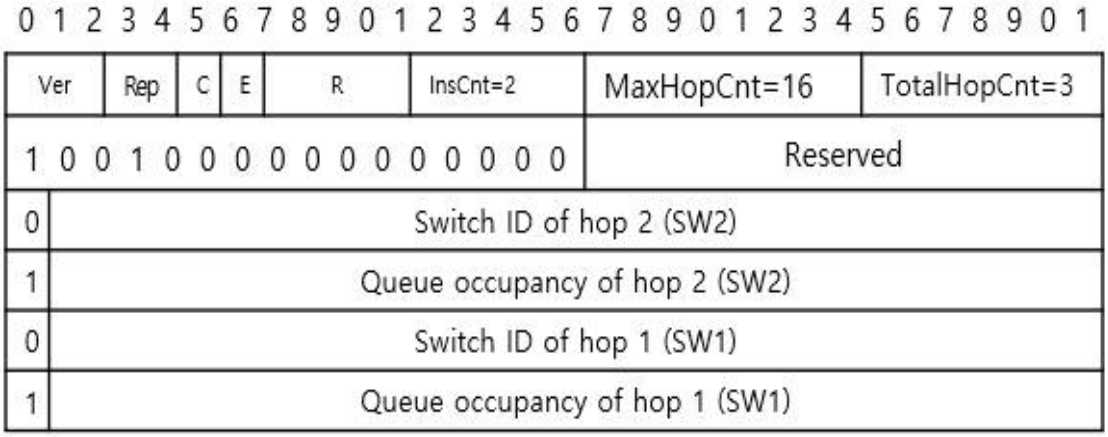

그림 15. Transit(SW2)에서의 메타데이터 정보

끝으로, SW3와 같이 모니터링 경로의 마지막 노드를 INT Sink 노드라고 명명한다. SW3는 SW2와 같이 스위치 정보를 추가하고 INT 헤더를 추출하여 INT헤더를 제외한 나머지 패킷 (원래의 패킷)을 end host에게 전송하고, 아래 그림과 같

이 최종적으로 축출한 메타데이터들을 모니터링 엔진으로 전달하는 기능 제공한다.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Ver	Rep	C	E	R	InsCnt=2	MaxHopCnt=16	TotalHopCnt=3
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0						Reserved	
0	Switch ID of hop 3 (SW3)						
0	Queue occupancy of hop 3 (SW3)						
0	Switch ID of hop 2 (SW2)						
0	Queue occupancy of hop 2 (SW2)						
0	Switch ID of hop 1 (SW1)						
0	Queue occupancy of hop 1 (SW1)						

그림 16. Sink (SW3)에서의 INT 메타데이터 정보

앞선 예제에서 보았듯이, INT는 스위치를 프로그래밍 하여 스위치 내부의 (큐, Ingress/Egress, Switch 메타데이터) 정보를 바탕으로 한 패킷 레벨의 데이터 평면의 정보를 제어 평면의 개입 없이도 수집 할 수 있는 기술이다. 이와 관련하여 데이터 평면 프로그래밍을 통한 정보 수집 기술을 이용하여 정보를 수집하고 분석하기 위해 여러 연구가 진행되고 있다.

프로그래머블 스위치로 구성된 네트워크 환경에서 SDN 을 활용하여 INT 메타데이터를 수집하기 위한 구조가 여러 연구 중 대표적인 연구이다. 대표적인 SDN 컨트롤러인 ONOS를 통해 프로그래머블 스위치에 INT 프로그램 배포, INT 모니터링 조건 제어, 모니터링 결과에 따른 네트워크 요구 사항 반영 등의 기능을 수행케 한다. ONOS 컨트롤러는 각 프로그래머블 스위치에 INT 패킷 처리 동작을 정의한 P4 프로그램을 배포한다. 더불어, 어떤 네트워크 정보를 수집 할지 정의하여 프로그래머블 스위치에 알려준다. 아래의 그림은 ONOS상의 구현되는 INT 프레임워크 구조를 나타낸다.

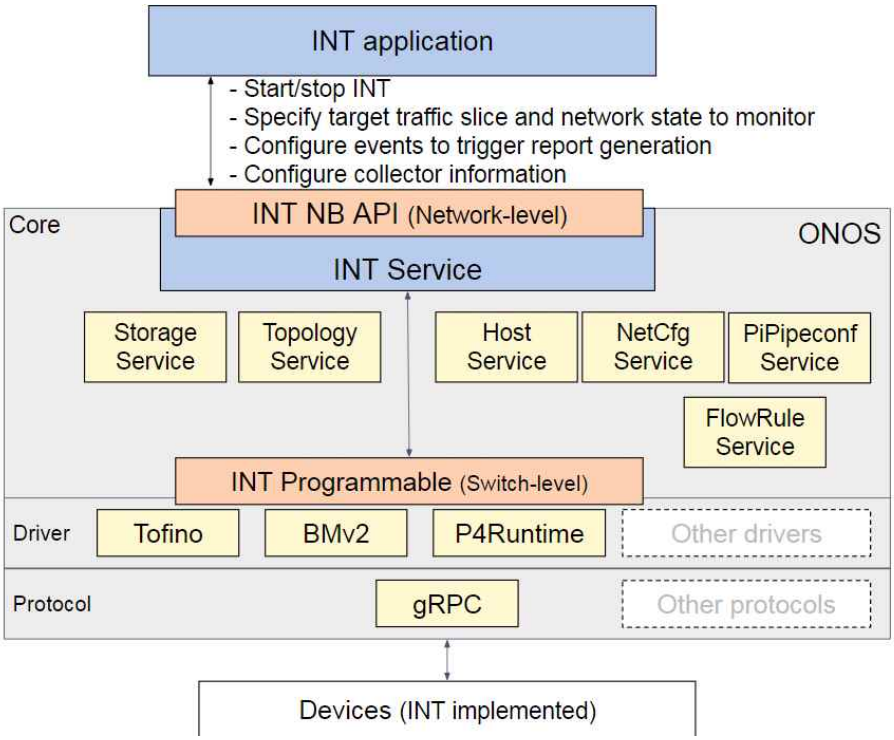

그림 17. ONOS기반의 INT 구조

2.2.2. In-situ Operations, Administration, and Maintenance (iOAM)

P4 진영의 INT와 더불어 CISCO에서는 iOAM 프로토콜을 in-network 모니터링 기능으로 제안하였다. 해당 프로토콜은 IETF에서 표준화 중에 있다. 아래의 그림은 iOAM의 개념도를 나타낸다. 앞 절에서 기술한 INT와 마찬가지로 네트워크 정보를 메타데이터 형식으로 패킷에 삽입한다. INT와 마찬가지고 source/ transit/ sink 노드로 구분되며, source와 sink사이의 네트워크를 iOAM 도메인이라고 명명한다.

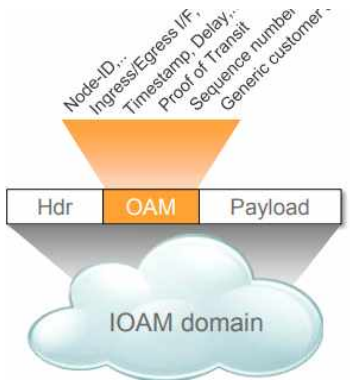

그림 18. iOAM 개념

iOAM은 3가지 기능 (proof of transit, trace 그리고 edge to edge)을 제공하고 있

다. 아래의 그림은 Proof of transit 기능과 Tracing 기능의 개념도를 나타낸다.

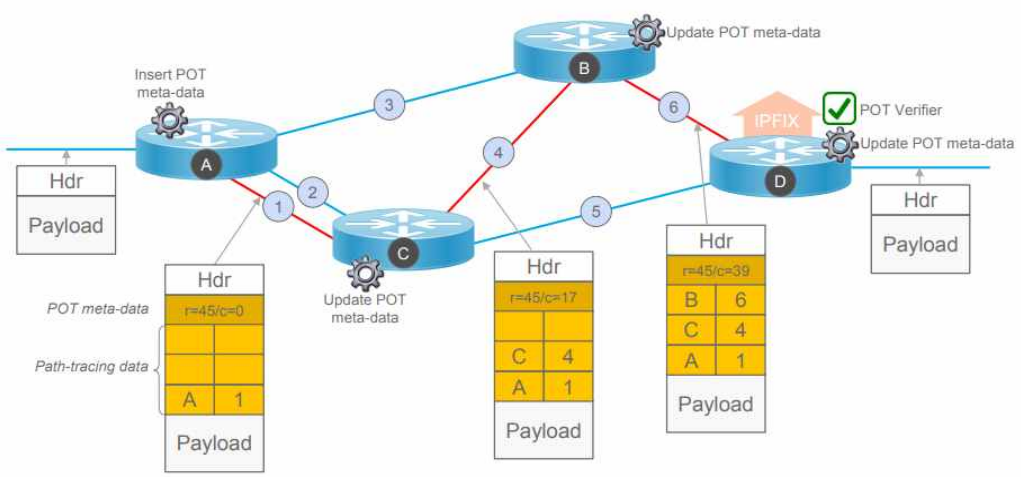

그림 19. Proof of transit 기능과 Tracing 기능의 개념도

상기의 그림에서는 A가 source노드 역할을 수행한다. A노드는 tracing 할 네트워크 정보가 담긴 헤더를 패킷에 삽입하고, 자신의 네트워크 정보 또한 삽입한다 (A:1). B와 C는 transit노드로써 A노드가 삽입한 헤더를 통해 자신의 네트워크 정보를 삽입한다. 끝으로 D는 sink노드로써 삽입된 메타데이터를 축출하여 모니터링 엔진으로 전송하고, 원래의 데이터형식으로 end-host에게 전송한다. Proof of transit 기능은 외부 컨트롤러로부터 비밀키 값을 부여받는다. 패킷이 지날 때 마다 비밀키 값을 삽입하여 sink노드에서 비밀키 들을 조합하여 올바른 경로로 지났는지 검증하게 된다.

iOAM 기능을 통해 CISCO에서 제안한 네트워크 검증과 모니터링을 위한 프레임워크는 아래 그림과 같다. 아래 그림에서는 SDN 컨트롤러로 Opendaylight 컨트롤러를 사용하였으며, 해당 컨트롤러를 통해 어떤 네트워크 정보를 수집 할지, 수집한 정보를 어디로 보낼지 등을 결정하며, 더불어 경로 검증 기능을 위해 Opendaylight 컨트롤러에서 비밀 키 값등을 데이터 평면에 내려준다. 해당 그림에서의 encap/decap은 sorce노드 (A)와 sink노드 (C)에서 iOAM 헤더를 삽입하고 축출하는 기능을 나타낸다. iOAM으로 수집된 데이터들은 데이터 분석 툴인 PNAD등을 통해 분석 하고, 분석 된 정보를 다시 컨트롤러에 제공하여 더 나은 서비스를 네트워크 사업자는 제공 할 수 있다.

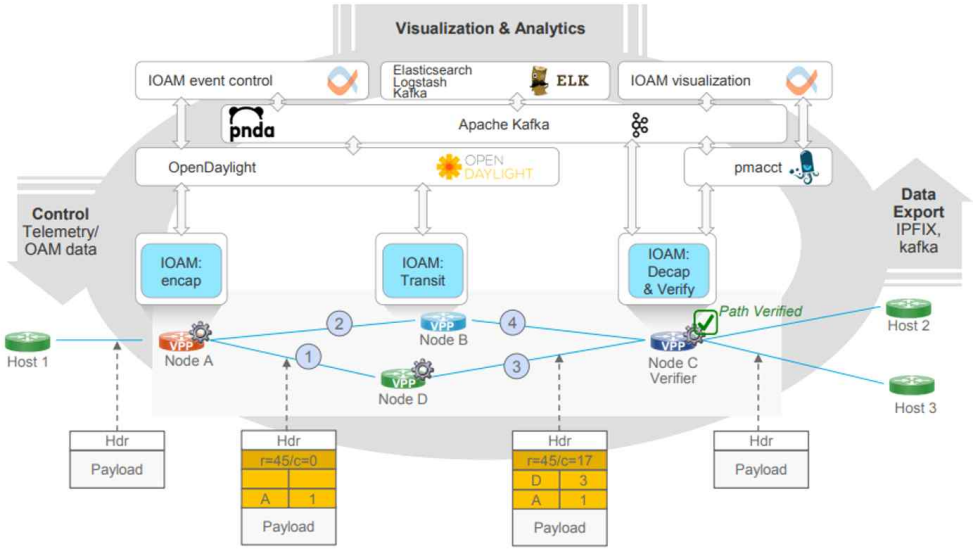

그림 20. iOAM 기반의 네트워크 모니터링 프레임워크

3. Multisite Service Function Chaining

3.1. Multisite SFC의 개요

멀티 사이트 SFC는 SFC의 구현을 단일 사이트에 존재하는 SF들로만 RSP를 구성하는 것이 아닌 멀티 사이트들에 존재하는 여러 SF들 간에 RSP를 구성할 수 있는 기능을 의미한다.

3.2. Multisite SFC의 필요성

멀티 사이트 SFC 지원이 필요한 이유는 멀티 사이트에서의 SFC 지원 및 그에 따른 RSP를 구성했을 때 다음과 같은 경우들에 대해 단일 사이트에서 SFC 를 구성하여 처리하는 것보다 트래픽 전송의 지연시간 및 자원관리의 효율성 측면에서 유리할 수 있기 때문이다.

첫 번째 경우로, 특정 사이트 내에서 트래픽이 수행 받아야 할 해당 사이트의 SF에 문제가 발생했을 경우가 있다. 이 경우에, 해당 사이트의 SF가 복구될 때까지 기다리거나 혹은 사전에 설정한 failure management 시스템을 통해 다른 노드에 해당 SF와 같은 기능을 수행하는 SF를 새로 배포하는 등의 방법이 존재할 수 있다. 그러나 두 경우 SF와 해당 SF가 동작하는 VM에 따라 복구되거나 새로 배포되는데 높은 지연시간이 요구될 수 있기 때문에 효율적이지 않을 수 있다. 반면, 멀티 사이트 환경에서 SFC가 지원된다면, 이웃에 있는 다른 사이트에 문제가 발생한 SF와 같은 기능을 수행하는 SF가 멀지 않은 곳에 존재한다면, 해당 SF를 수행 받을 수 있도록 RSP(Rendered Service Path)를 구성하여 지연 시간을 줄일 수 있다.

두 번째 경우로, 자신의 사이트에 수행 받아야 할 SF가 존재하지 않을 경우가 있다. 사실 이 경우는 기존의 하나의 사이트를 관리하는 VIM과 해당 사이트에 존재하는 SDN 컨트롤러가 하나이기 때문에 존재하지 않는 SF를 이용하여 SFC를 구성하는 것은 불가능하다고 볼 수 있다. 그러나 멀티 사이트 환경에서 SFC 구성을 지원할 수 있다면, 다른 사이트의 SF를 이용하여 RSP를 구성하고 이를 이용해 사이트 간의 SF 노드들을 협력적으로 사용할 수 있다.

따라서 멀티 사이트 SFC를 구성할 수 있게 되면 위와 같이 다양한 경우들에 대해 기존의 단일 사이트에서만 SFC를 구성하고 그에 따른 RSP 구성을 했을 때보다 발생 상황에 따라 유연하고 효율적으로 트래픽이 SF 처리를 받을 수 있게 되어 RSP 경로를 완료하는데 걸리는 지연시간의 감소를 가져올 수 있을 것으로 예상된다. 또한, 사이트 관리자들 간에는 실시간으로 각자가 관리하는 사이트 내의 자원들을 협력적으로 이용할 수 있어 배포한 SF를 자원의 효율성도 증대시킬 수 있을 것으로 예상된다.

멀티 사이트 SFC 지원을 위해서 여러 사이트들의 SF, SFF(Service Function

Forwarder), SFC Classifier의 정보를 사전에 수집할 수 있는 오케스트레이터의 역할을 할 수 있는 개체가 필요하고, 또한 해당 정보들을 토대로 기존의 단일 사이트 내에서의 RSP 구성 뿐만 아니라 다른 사이트의 SF들을 포함하여 RSP를 구성할 수 있게 하는 네트워크 계층의 개체가 필요하다.

3.3. ODL SFC 기반 Multisite RSP 구성

본 장에서는 ODL SFC 기반으로 멀티 사이트 RSP 구성을 위한 기능을 구현하였다. ODL 컨트롤러가 하나의 사이트를 관리한다고 가정하는 환경에서, 두 개의 사이트를 거쳐서 RSP를 구성하도록 테스트를 진행하였다.

3.3.1. Multisite RSP 구성 모듈 테스트

멀티 사이트 간에 RSP의 구성을 테스트하기 위해, 우선 아래 그림과 같이 계층적인 SDN 컨트롤 플레인의 구조를 만들어서, 각 사이트를 관리하는 SDN-S (ODL Controller)에게 멀티 사이트 SFC구성을 수행할 수 있는 SDN-O를 도입하는 방식을 택했다다. SDN-O는 파이썬 기반으로 구현되었으며, SDN-S에게 RSP 구성 요청과 SF 타입들을 받아오는 기능, 그리고 구성된 RSP를 저장하는 기능을 구현 중에 있다. 우선 SDN-C와 같은 VM에 SDN-O를 파이썬으로 구현하였고, SDN-C에 DPI와 FW로 구성된 SFC와 두 사이트를 거치도록 구성된 멀티 사이트 RSP까지 구성하도록 하였다.

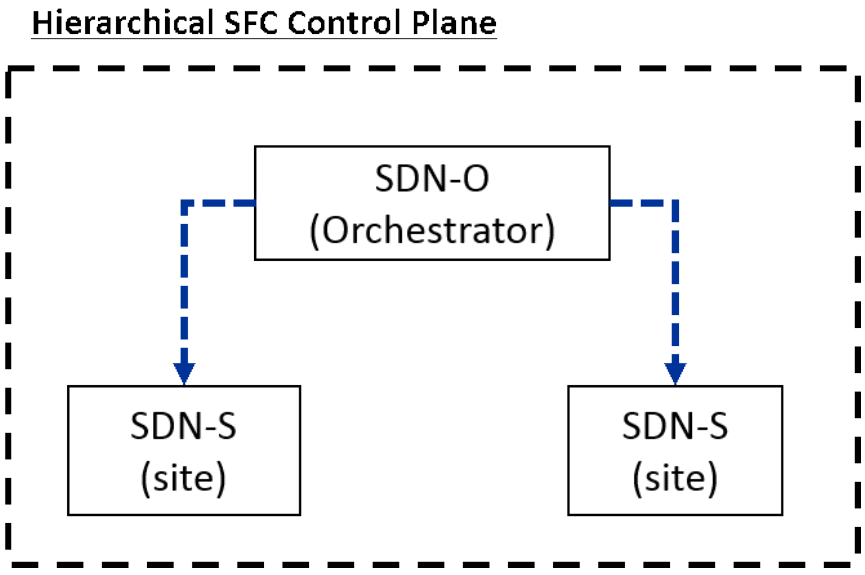

그림 21. 테스트를 위한 계층적 SFC 제어 평면

그 결과 아래 그림은 모듈을 통해 요청하고, 설정된 멀티 사이트 RSP를 나타낸다. FW=>DPI 의 서비스 기능체인을 위해 아래와 같이 두 개의 RSP를 구성하도록 하였

다.

RSP1: DPI (사이트1: PathID1 / SI255) => FW (사이트2: PathID1 / SI254)

RSP2: DPI (사이트2: PathID2 / SI255) => FW (사이트1: PathID1 / SI254)

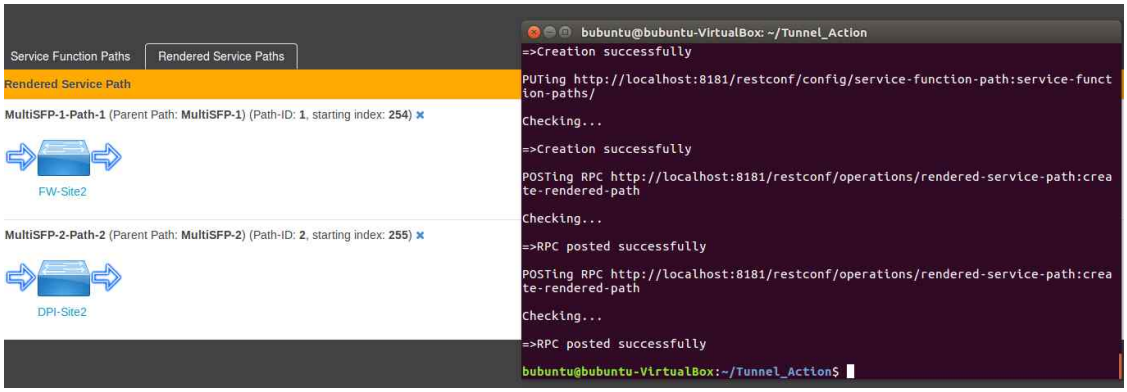

그림 22. RSP 기능 구현 확인

위의 그림을 통해 RSP1에 FW 인스턴스가 할당되고, RSP2에 DPI 인스턴스가 할당되었음을 확인할 수 있다. 또한 SI가 서로 다름을 확인할 수 있는데 이는 멀티 사이트 SFC에서 NSH의 번호를 맞추기 위해 설정하였다.

그림 23. 두 사이트에서의 RSP 구성 (왼쪽 : Site1, 오른쪽 : Site2)

위의 그림을 통해 사이트 별로 같은 Path Number를 가진 RSP가 생성되고 서로 스타팅 포인트가 할당되는 것을 확인할 수 있다. 또한, 아래의 그림을 통해 RSP2의 정보로 사이트 2의 DPI가 Path ID 2와 SI 255, 사이트 1의 FW가 Path ID 2와 SI 254를 할당받은 것을 확인할 수 있다.

그림 24. 두 사이트에 구성된 RSP2의 정보 (왼쪽 : 사이트1, 오른쪽 : 사이트2)

3.4. Openstack Tacker 기반 개발 제안

Tacker 프로젝트는 Liberty 버전까지는 Tacker 서버가 하나의 VIM만 관리 가능하였으며, 해당 VIM을 통해서 VNF를 배포할 수 있었다. 하지만 Mikata 버전부터 아래 그림과 같이 Multisite VIM 기능이 추가되면서, 여러 사이트에 존재하는 VIM을 하나의 Tacker서버에서 동시에 제어 및 관리가 가능하게 되었다. Multisite VIM은 NFV Orchestrator의 기능으로 추가 되었으며, 각 사이트들의 OpenStack 버전과는 상관없이 통합적으로 관리가 가능하다. NFVO 내에서 각 사이트들의 SF들의 대한 정보를 수집하여 전체의 VNFFG를 구성한 이후, 멀티 사이트간 RSP를 구축할 수 있는 모듈을 제공하는 모듈을 제안한다. 해당 개발은 차후 기술문서에서 구현할 예정이다.

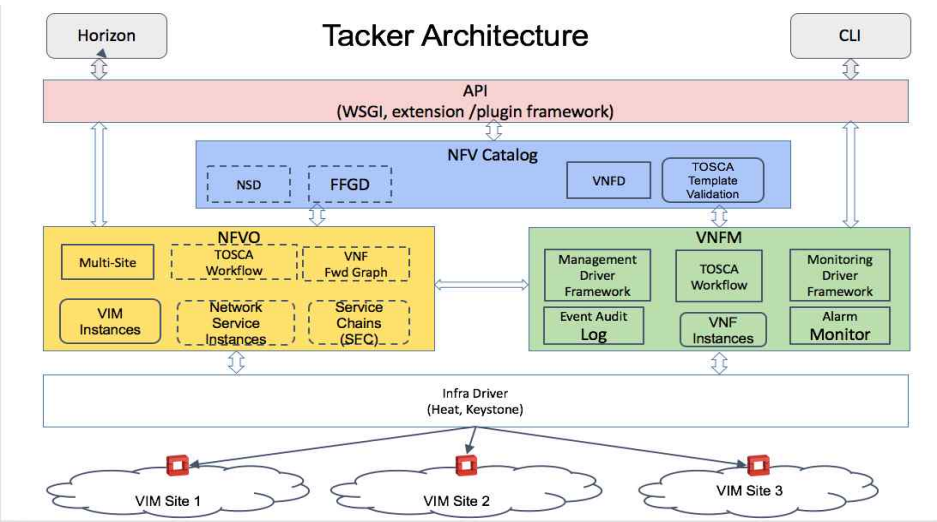

그림 25. Tacker Multi-VIM 구조

4. OPNFV/ODL SFC에서의 SFC monitoring Framework 개발

본 장에서는 OPNFV/ODL SFC 컨트롤러에서 구성한 rendered service function (RSP)의 경로 검증, 성능 측정 그리고 고 가용성 측정을 위한 SFC 모니터링 프레임워크를 제안한다.

4.1. SFC를 위한 In-situ Operation, Administration and Maintenance (iOAM) Framework

PNFV/ODL SFC 컨트롤러에서 구성한 rendered service function (RSP)의 경로 검증, 성능 측정 그리고 고 가용성 측정을 위해 IETF 표준 프로토콜인 iOAM [9]을 사용한다.

4.1.1. SFC iOAM 구조

iOAM을 통한 RSP 모니터링 프레임워크 구조는 아래 그림과 같다.

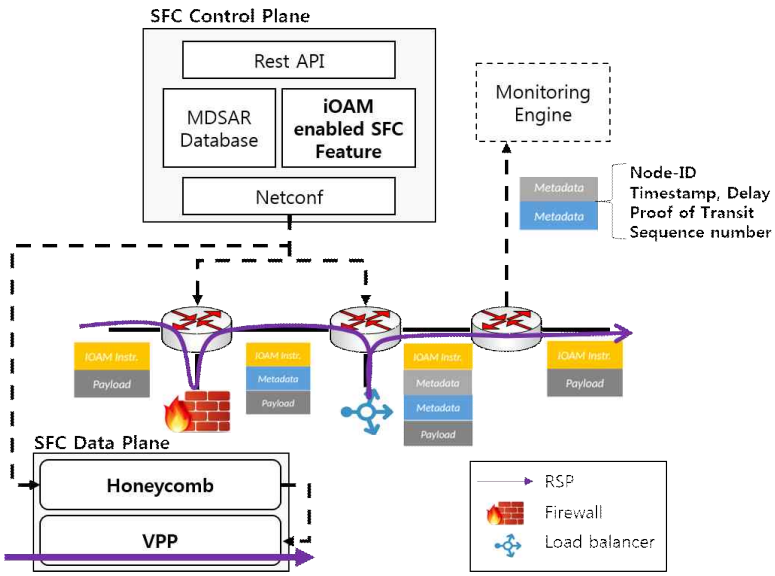

그림 26. iOAM enabled SFC 제어/데이터 평면 구조

위의 그림과 같이 전체적인 구조는 제어평면과 데이터평면으로 분리된다. 기존 SFC제어평면에서는 사용자가 요구하는 SFC에 충족하게 데이터평면에 RSP를 구성한다. 본 구조에서는 기본적인 RSP 구성기능 뿐만 아니라, 구성된 RSP의 경로검증, 성능측정 그리고 패킷손실 측정 등의 기능이 추가로 지원된다. 사용자가 데이터 플랜에 구성된 RSP의 성능 등을 측정하고 싶을 때, REST API를 통해 SFC 모니터링

기능을 요청 할 수 있다. 이러한 요청은 iOAM enabled SFC Feature에서 처리되어, 최종적으로는 Netconf 인터페이스를 통해 SFC 데이터 평면에 모니터링 기능을 활성화 시킨다. 본 구조의 데이터 평면에서는 NSH기반의 RSP처리와 같은 기존의 SFC 데이터 플랜에서의 기능 제공뿐만 아니라, RSP의 성능에 대한 정보를 NSH의 Meta-data에 삽입하는 기능을 제공한다. 해당 기능을 제공하기 위해 본 프레임워크에서는 Vector Packet Processing (VPP) platform [10]과 Netconf/Restconf 기반의 관리 에이전트인 Honeycomb [11]를 사용한다. 아래의 그림과 SFC 컨트롤 평면 (e.g., Opendaylight SFC)에서 모니터링 기능 활성화 명령을 Netconf 인터페이스를 통해 데이터 평면에 전달하게 되면, 데이터 평면에서의 Honeycomb 에이전트가 해당 명령을 받아 low level API를 통해 VPP로 전달하게 된다. VPP는 요구하는 모니터링 기능에 따라 들어오는 RSP에 Metadata형식으로 모니터링 된 정보를 삽입한다.
위에 그림과 같이 사용자가 특정 플로우에 대해 Firewall => Load balancer 기능을 순차적으로 처리하기 원하면서, 동시에 해당 플로우를 모니터링 하고 싶은 경우 컨트롤 평면에서는 사용자의 요구사항을 종합하여 RSP를 구성하고, 구성된 RSP 내의 SFF (e.g. VPP)에 모니터링 기능을 활성화 시킨다. 데이터 플랜에서는 들어오는 플로우는 체인 요구사항에 맞게 처리되며 동시에 SFF에서는 Timestamp, 혹은 Sequence Number, 혹은 난수 등을 Metadata형식으로 NSH Metadata field에 삽입하여 모니터링 기능을 제공한다.

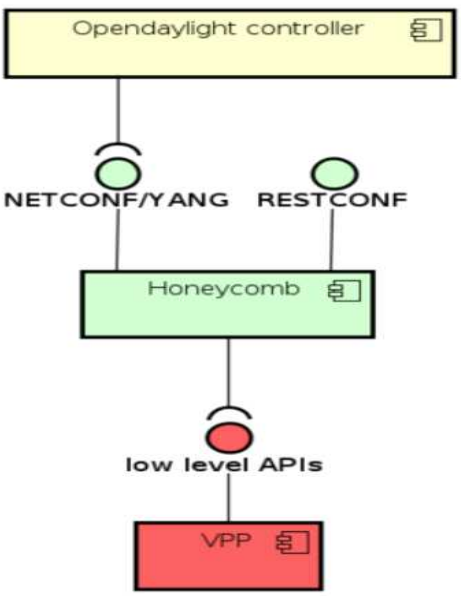

그림 27. Honeycomb enabled VPP

4.1.2. NSH Encapsulation for iOAM Data

본 절에서는 데이터 평면에서 사용되는 헤더를 기술한다. 해당 데이터 평면에서는 NSH의 next protocol header 기능을 이용하여, NSH로 모니터링 된 데이터를 캡슐화 한다. NSH로 캡슐화하는 이유는 NSH를 통해 RSP를 포워딩만 제공하는 데이터 평면과의 호환성을 위해서이다. 본 내용은 IETE의 기고된 문서를 [12] 참고하였다. NSH로 iOAM 데이터 (모니터링 된 데이터)를 캡슐화한 프로토콜은 아래 그림과 같다.

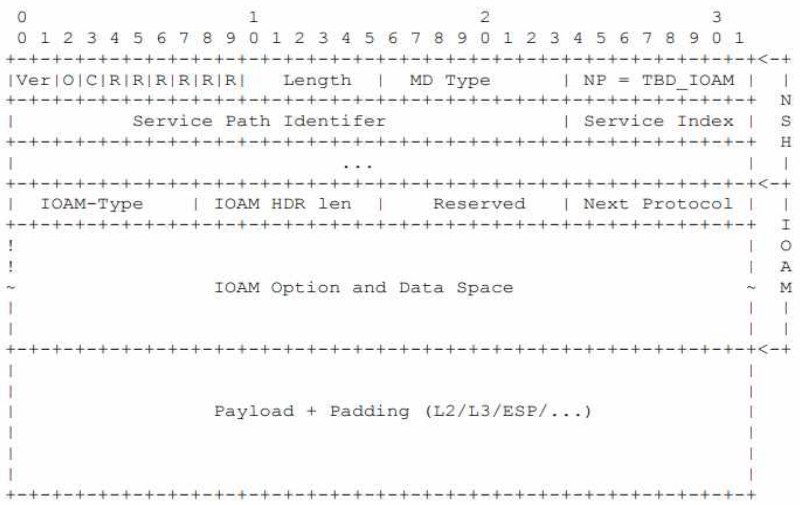

그림 28. In-situ OAM NSH encapsulation packet

본 패킷 구조에서는 iOAM 데이터를 NSH로 캡슐화하기 위해 NSH의 Next Protocol (NP)를 TBD_IOAM으로 세팅한다. TBD_iOAM은 IETF 표준상 정의되지는 않았으면, 임의로 정의하였다. NSH내의 iOAM관련 필드와 정의된 기능은 다음과 같다.

- IOAM-Type: 8-bit 필드로써, iOAM Option Type을 명시한다. iOAM 옵션으로는 성능을 측정하는 trace옵션과 경로검증을 위한 proof-of-transit 옵션 그리고 패킷의 손실 등을 측정하기 위한 edge-to-edge 옵션이 존재한다.
- IOAM HDR Len: 8-bit 필드로써, iOAM header의 길이를 4-octet 단위로 명시한다.
- Reserved bits: 예약된 필드로써, 향후 추가되는 기능 등을 위해 남겨둔 필드이다. 따라서, 해당 필드는 항시 0x0으로 설정돼야 한다.
- Next Protocol: 8 비트 부호 없는 정수로써, IOAM 프로토콜에 따라 헤더 유형을 결정한다.
- IOAM Option and Data Space: IOAM-Type 필드에 옵션에 따라 지정된다.

본 프로토콜을 적용하기 위한 고려사항은 다음과 같다.

1. “NSH MD Type 2“로 iOAM 데이터 필드를 캡슐화해야 한다. 각 iOAM 데이터 필드 옵션 (trace, proof of transit, edge to edge)은 옵션 유형별로 지정되며, 다른 iOAM 데이터필드는 해당 옵션 유형 내의 TLV이다. NSH MD Type 2를 사용하는 이유는 가변 길이의 메타 데이터를 지원하기 때문이다. 해당 길이 필드는 6 비트이며, 최대 256 옥텟이 지원 가능하다.

2. 앞서 기술한 바와 같이 Next Protocol 필드를 사용하여 iOAM 데이터 필드를 캡슐화하고, 각 iOAM 데이터 필드 옵션 (trace, proof of transit, edge to edge)은 자체 “Next Protocol“로 지정 할 수 있다.

3. “Next Protocol“로 iOAM 데이터 필드를 캡슐화 할 때, 단일 NSH 프로토콜 유형 코드 포인트가 iOAM에 할당된다. “하위 유형“ 필드는 어떤 iOAM 옵션 유형 (trace, proof of transit, edge to edge)이 운반되는지를 나타낸다.

4.2. SFC Proof of Transit 기능

본 절에서는 ODL SFC에서 제공하는 iOAM 기능 중 Proof of Transit 기능을 분석한다.

4.2.1. SFC Proof of Transit 기능 개요

SFC Proof of Transit기능은 플로우 내의 패킷들이 SFC 컨트롤 평면에서 데이터 평면에 구성한 특정 경로 (RSP)에 따라 처리되고 있는지를 검증하는 기능이다. 플로우 내의 패킷들이 올바른 경로를 통과하고 있는지 확인하기 위한 기능은 데이터 플랜에서 플로우 운반을 담당하는 SFF에 구현되며, 사용자가 특정 RSP에Proof of Transit기능을 활성화 시키고 싶을 때 컨트롤러에 활성화 요청을 하면, 컨트롤러에서는 해당 RSP를 구성하는 SFF에 Proof of Transit 기능을 활성화 시킨다.
패킷이 지정된 경로 또는 기능 체인을 따르는 지 확인하기 위해 메타 데이터가 패킷에 앞 절에서 기술 한 바와 같이 헤더 형식으로 추가된다. Proof of transit 옵션에서의 메타 데이터는 ’share of a secret’에 기반으로, 보안 채널을 통해 컨트롤 평면에서 각 데이터 평면내의 노드로 할당된다. 이 메타 데이터는 각각의 서비스 홉에서 업데이트되는 반면, 검증을 위해 지정된 노드는 수집 된 메타 데이터를 통해 제대로 패킷이 흘러왔는지 검증한다.

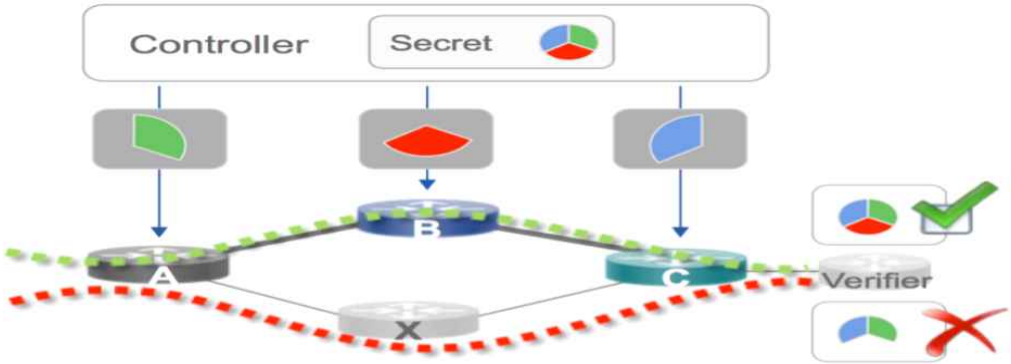

그림 29. SFC에서의 Proof of transit 예시

상기의 그림은 Proof of transit의 예시를 나타낸다. ODL SFC에서의 Proof of transit기능은 Shamir의 비밀 공유 알고리즘을 사용한다. 위 그림과 같이 RSP내의 노드에 원호와 같은 키를 컨트롤 평면으로부터 부여받는다. 키를 부여 받은 노드들은 패킷이 지날 때 마다, 패킷의 해더에 해당 킷값을 삽입하게 되고, 검증을 담당하는 노드는 헤더에 존재하는 킷값을 모아서 해당 패킷이 경로대로 제대로 처리되었는지 검증된다. 즉, 위의 그림과 같이 원호 형태의 킷값들을 제어평면으로부터 부여받았다면, 검증 노드에서 모든 킷값을 조합하면 하나의 원이 생성된다. 만일, 올바른 경로로 오지 않았다면 원이 생성되지 않을 것이다.

4.2.2. SFC Proof of Transit 구조 및 구현

SFC Proof of Transit 기능은 SFC내의 경로 검증을 위해 iOAM의 Proof of Transit 기능을 통해 구현되었다. 구현은 크게 응용 프로그램의 Northbound (NB)와 Southbound (SB) 측면으로 구분된다. 사용자의 요청에 따라 RSP에 Proof of Transit를 활성화하기 위해 Northbound 인터페이스에서는 사용자 입력을 받는 기능과, Southbound 인터페이스에서는 주기적으로 매개 변수(key값)를 생성하고, NETCONF 프로토콜을 통해 데이터평면의 노드 (e.g., VPP node)에 전달하는 기능이 구현되었다.

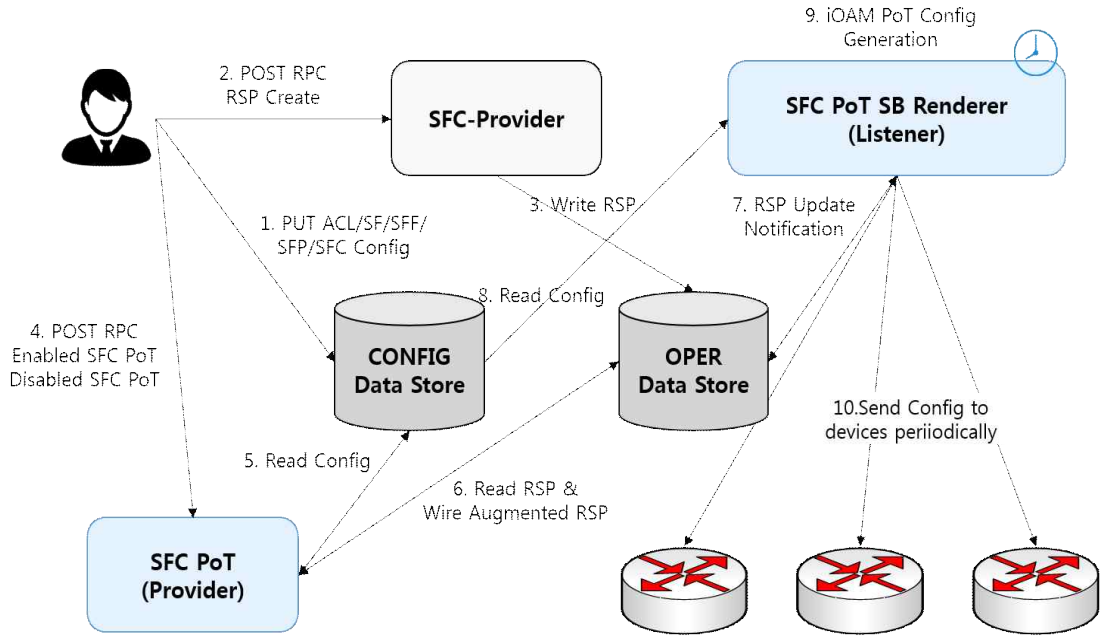

그림 30. SFC Proof of Transit 구조

위의 그림은 SFC Proof of Transit의 구조와 각 모듈간의 동작과정을 나타낸다. 동작과정은 아래와 같다.

1. 사용자는 Northbound 인터페이스를 통해 ACL/SF/SFF/SFP/SFC 등을 컨트롤러를 통해 설정한다.
2. 설정된 데이터 평면내에서 RSP를 구성하기위해 RSP 구성요청을 SFC Provider 모듈에 보낸다.
3. 해당 RSP 구성요청에 맞게 SFC Provider는 OPER 타입의 데이터 스토어 해당 정보를 작성하고, 데이터 평면 RSP를 구성한다.
4. 사용자가 구성된 RSP의 경로검증을 위해 특정 RSP에 Proof of transit 기능을 활성 요청을 SFC PoT Provider에게 전송한다.
5. 해당 요청을 받은 SFC PoT Provider는 RSP의 SF와 SFF에 대한 정보 그리고 RSP에 대한 정보를 CONFIG와 OPER 타입의 데이터 스토어에서 획득한다.
6. 회득된 정보를 통해 Augmented RSP를 OPER 타입의 데이터 스토어에 기입한다.
7. OPER 타입의 데이터 스토어 내에 Augmented RSP 정보가 변하는 경우 SFC PoT SB Renderer에게 알림 전송된다.
8. 해당 알림을 받은 후에 FC PoT SB Renderer는 CONFIG 데이터 스토어에서 RSP내의 SF와 SFF 정보를 획득한다.
9. 해당 알림을 받은 SFC PoT SB Renderer는 주기적으로 킷값을 생성한다.
10. 생성된 킷 값은 RSP를 구성하는 데이터평면내의 각 노드에게 전송한다.

상기의 과정을 다이어그램을 나타내면 아래 그림과 같다.

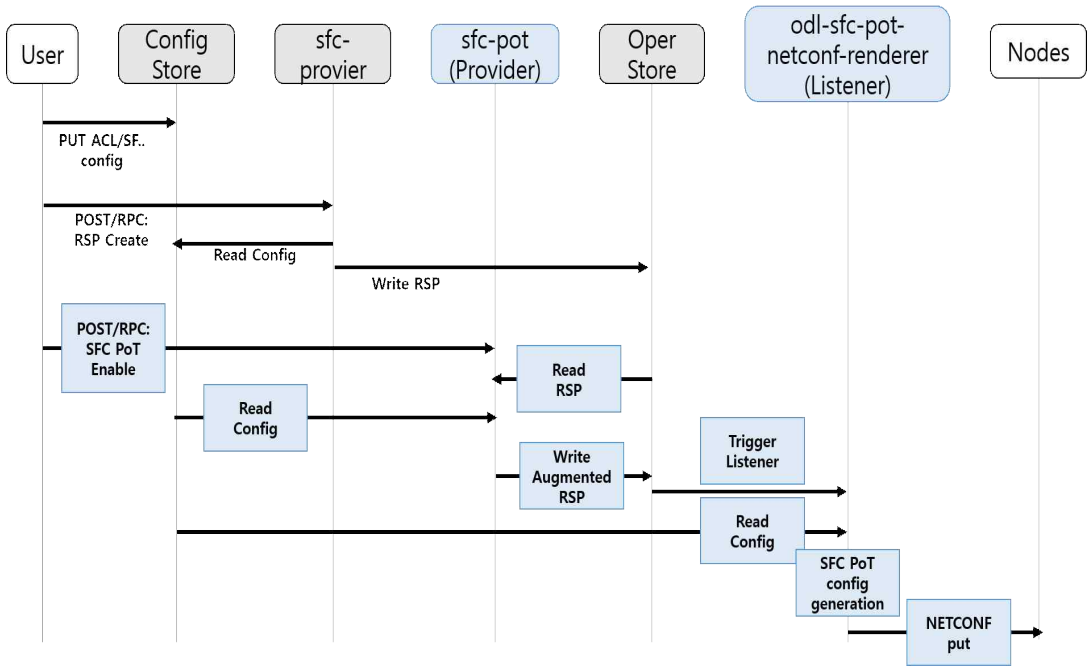

그림 31. SFC Proof of Transit의 Sequence Diagram

4.2.3. SFC Proof of Transit 사용법

사용자가 특정 RSP의 PoT 기능을 검증하고 싶은 경우 REST API를 통해 POST 메시지를 작성한다. POST 메시지는 다음과 같다.

POST-http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-pot:enable-sfc-ioam-pot-rendered-path/Message-

```
{
    "input":
    {
        "sfc-ioam-pot-rsp-name": "sfc-path-3sf3sff",
        "ioam-pot-enable":true,
        "ioam-pot-num-profiles":2,
        "ioam-pot-bit-mask":"bits32",
        "refresh-period-time-units":"milliseconds",
        "refresh-period-value":5000
    }
}
```

그림 32. SFC PoT를 위한 POST 메시지 예시

해당 메시지는 RSP이름이 “sfc-path-3sf3sff” 인 RSP에 Proof of transit 기능을

활성화 시키는 예제로, 각 노드의 킷값은 5000msec주기로 변동되는 것을 세팅하였다. 이와 반대로 사용자가 특정 RSP의 PoT 기능을 검증하고 그만하고 싶은 경우에도 REST API를 통해 POST 메시지를 작성한다. POST 메시지는 다음과 같다.

POST-http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-pot:enable-sfc-ioam-pot-rendered-path/Message-

```
{
    "input":
    {
        "sfc-ioam-pot-rsp-name": "sfc-path-3sf3sff",
    }
}
```

그림 33. SFC PoT를 위한 POST 메시지 예시

4.3. SFC Trace 기능 제안

본 절에서는 ODL SFC에서 구현되지 않은 iOAM Trace 기능을 제안한다.

4.3.1. SFC Trace 기능 개요

SFC Trace 기능은 플로우 내의 패킷들이 SFC 컨트롤 평면에서 데이터 평면에 구성한 특정 경로 (RSP)에 따라 처리되고 있을 때, 플로우의 SFC 처리시간 등의 RSP의 성능을 측정 가능토록 하는 기능이다. 플로우의 성능 정보를 패킷헤더에 삽입하는 기능은 데이터 평면에서 플로우 운반을 담당하는 SFF에 구현되며, 사용자가 특정 RSP에 Trace 기능을 활성화 시키고 싶을 때 컨트롤러에 활성화 요청을 하면, 컨트롤러에서는 해당 RSP를 구성하는 SFF에 Trace 기능을 활성화 시킨다.

플로우 성능을 측정하기 위해 패킷이 SFF를 지날 때 마다 메타 데이터가 패킷에 헤더 형식으로 추가된다. SFF에 의해 추가된 메타 데이터는 마지막 SFF가 패킷을 성능정보를 축출하여 모니터링 엔진 혹은 컨트롤러에게 전달한다. 해당 부분은 본 구현의 out of scope로 정의한다.

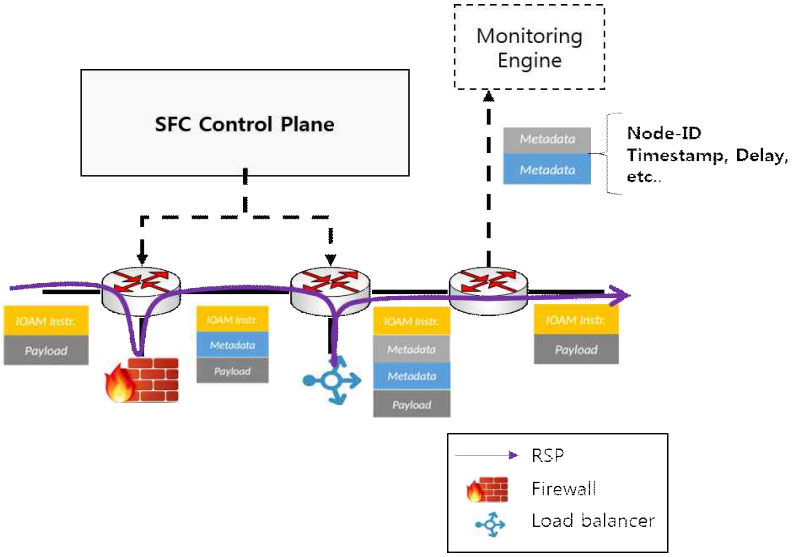

그림 34. SFC Trace 예제

상기의 그림은 SFC Trace 기능예제를 나타낸다. 사용자가 방화벽과 부하분산기능을 순차적으로 처리되도록 SFC 컨트롤 평면에 요청을 하여 RSP가 구축된 상태이다. 그 후에 사용자가 혹은 RSP를 제공하는 사업자가 특정 RSP의 성능을 모니터링하기 위해서 SFC 컨트롤 평면에 특정 RSP의 성능 모니터링 기능을 활성화 시킨다. 활성화 시킬 때, 어떤 정보를 모니터링할지도 결정해야한다. Timestamp와 delay를 결정하면, 상기 그림의 SFF들은 특정 RSP가 지날 때 마다 timestamp와 delay를 메타데이터 형식으로 패킷 헤더에 삽입한다. 가장 끝의 SFF가 패킷 헤더에 존재하는 메타데이터를 축출하여 monitoring 엔진에 전송한다.

4.3.2. SFC Trace 구조 및 구현예제

SFC Trace 기능은 RSP의 성능 모니터링을 위해 iOAM의 trace 기능을 통해 구현된다. 구현은 크게 응용 프로그램의 Northbound (NB)와 Southbound (SB) 측면으로 구분된다. 사용자의 요청에 따라 RSP에 trace 기능을 활성화하기 위해 Northbound 인터페이스에서 사용자 입력을 받는 기능과, Southbound 인터페이스에서는 tracing할 미터를 NETCONF프로토콜을 통해 데이터평면의 노드 (e.g., VPP node)에 전달하는 기능이 구현된다.

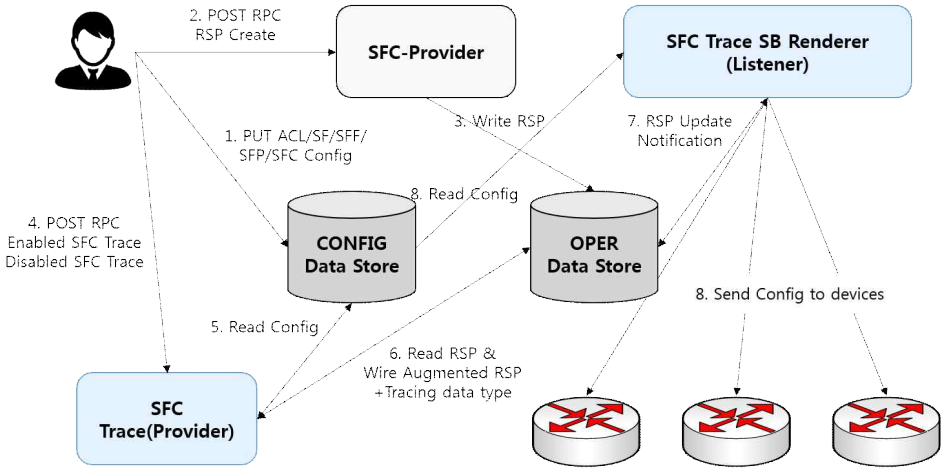

그림 35. SFC Trace 구조

위의 그림은 SFC Proof of Transit의 구조와 각 모듈간의 동작과정을 나타낸다. 동작과정은 아래와 같다.

1. 사용자는 Northbound 인터페이스를 통해 ACL/SF/SFF/SFP/SFC 등을 컨트롤러를 통해 설정한다.
2. 설정된 데이터 평면내에서 RSP를 구성하기위해 RSP 구성요청을 SFC Provider 모듈에 보낸다.
3. 해당 RSP 구성요청에 맞게 SFC Provider는 OPER 타입의 데이터 스토어 해당 정보를 작성하고, 데이터 평면 RSP를 구성한다.
4. 사용자가 구성된 RSP의 성능측정을 위해 trace 기능 활성화 요청을 SFC Trace Provider에게 전송한다.
5. 해당 요청을 받은 SFC Trace Provider는 RSP의 SF와 SFF에 대한 정보 그리고 RSP에 대한 정보를 CONFIG와 OPER 타입의 데이터 스토어에서 획득한다.
6. 회득된 정보를 통해 Augmented RSP를 OPER 타입의 데이터 스토어에 기입한다.
7. OPER 타입의 데이터 스토어 내에 Augmented RSP 정보가 변하는 경우 SFC Trace SB Renderer에게 알림 전송된다.
8. 해당 알림을 받은 후에 SFC Trace SB Renderer는 CONFIG 데이터 스토어에서 RSP내의 SF와 SFF 정보를 획득한다.
9. 해당 알림을 받은 SFC Trace SB Renderer는 모니터링 할 미터에 대한 정보를 RSP를 구성하는 데이터평면내의 각 노드에게 전송한다.

상기의 과정을 다이어그램을 나타내면 아래 그림과 같다.

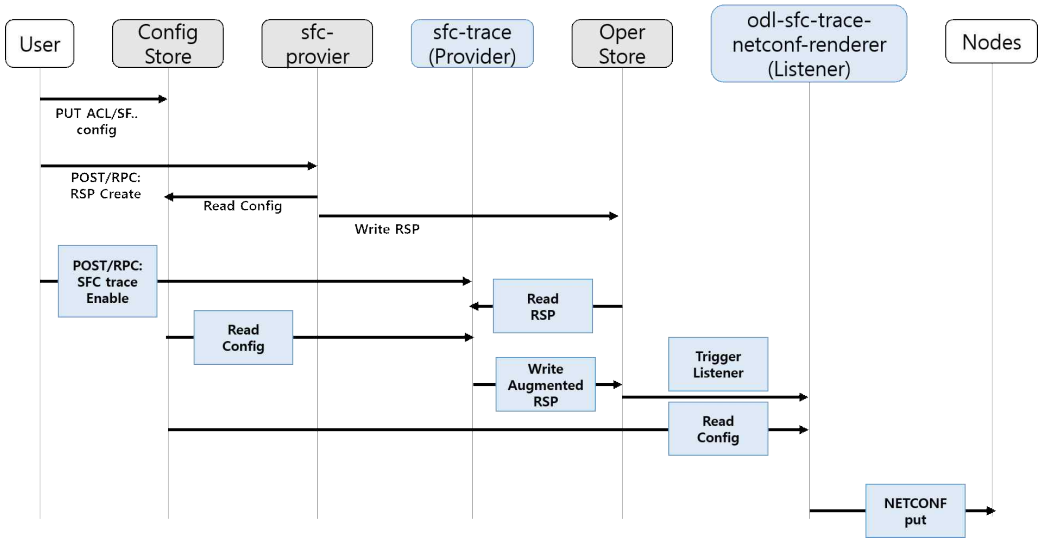

그림 36. SFC Trace의 Sequence Diagram

4.3.3. SFC Trace 사용예제

사용자가 특정 RSP의 성능을 모니터링 하고 싶을 때, REST API를 통해 POST 메시지를 전송한다. SFC Trace 기능을 활성화를 위한 POST 메시지는 다음과 같이 설계 하였다.

POST-
http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-trace:enable-sfc-ioam-trace-rendered-path/

```
{
        “input”:
        {
                “sfc-ioam-tarce-rsp-name”: “sfc-path-ku”,
                “ioam-trace-enable”: true,
                “ioam-trace-type-type”: 0x1f
        }
}
```

그림 37. SFC Trace를 위한 POST 메시지 예시

해당 메시지는 RSP이름이 “sfc-path-ku” 인 RSP에 trace 기능을 활성화 시키는 예제로, 각 노드는 hop limit, node ID, ingress and egress interfaces IDs, timesteamp 그리고 application data를 모니터링 하도록 세팅하였다.

이와 반대로 사용자가 특정 RSP의 Trace 기능을 검증하고 그만하고 싶은

경우에도 REST API를 통해 POST 메시지를 작성한다. POST 메시지는 다음과 같이 설계하였다.

POST-http://ODL-IP:8181/restconf/operations/sfc-ioam-nb-pot:enable-sfc-ioam-pot-rendered-path/Message-

```
{
      "input":
      {
            "sfc-ioam-tarce-rsp-name": "sfc-path-ku",
      }
}
```

그림 38. SFC Trace를 위한 POST 메시지 예시

VPP에서 제공하는 Trace-type은 5가지 (0x1f¦0x7¦0x9¦0x11¦0x19)로 정의된다. 각 Trace type별 메타데이터에 포함되는 정보는 아래와 같다.

- 0x1f : hop limit (8 bits), node ID (24 bits), ingress and egress interface IDs (16 bits each), timestamp (32 bits), application data (32 bits)
- 0x7 : hop limit (8 bits), node ID (24 bits), ingress and egress interface IDs (16 bits each)
- 0x9 : hop limit (8 bits), node ID (24 bits), timestamp (32 bits)
- 0x11: hop limit (8 bits), node ID (24 bits), application data (32 bits)
- 0x19: hop limit (8 bits), node ID (24 bits), timestamp (32 bits), application data (32 bits)

4.4. End to end 기능

End to end기능은 NSH에 시퀀스 번호를 메타데이터 형식으로 삽입하여 미리 정해진 끝점에서 시퀀스번호를 확인한다. 해당 번호를 통해 RSP내의 패킷 손실률을 알 수 있으며, 더불어 고 가용성 지원을 위해 두 경로로 패킷을 전달 할 때 중복 패킷을 제거하고, 패킷의 순서를 재조합하는 기능을 제공한다. 해당 기능은 차후 기술문서에 제안할 예정이다.

5. 결론

본 문서에서는 OpenDaylight / OPNFV SFC 프로젝트의 개요 및 특징들을 분석하였다. 이후 여러 사이트들이 공존하는 Multisite 네트워크에 대한 특징과, Multisite를 타겟으로 하는 관련 오픈소스 프로젝트들을 분석하였다. 대표적인 In network 모니터링 기법들인 P4 기반의 In band Network Telemetry와 iOAM 들을 분석하였다.

Multisite SFC에 대한 개요와 Mutlsite SFC의 필요성에 대해 설명한 이후 ODL SFC 프로젝트를 기반으로 Multisite SFC를 테스트하기 위한 토폴로지를 구축하고 간단한 Multisite RSP의 구성이 가능함을 테스트하였다. 마지막으로 RSP가 구성된 이후에 RSP를 모니터링 할 수 있는 프레임워크를 제안하고 기존의 존재하는 SFC PoT기능에 대해 설명하고 SFC의 트레이싱의 기능을 제안하였다.

References

[1] OpenDaylight, [Online] Available : https://www.opendaylight.org/

[2] Halpern, J., Ed. and C. Pignataro, Ed., “Service Function Chaining (SFC) Architecture,” RFC 7665, DOI 10.17487/RFC7665, October 2015, <http://www.rfc-editor.org/info/rfc7665>.

[3] ETSI ISG NFV, http://portal.etsi.org/portal/server.pt/community/NFV/367

[4] OPNFV, “Open Platform for Network Function Virtualization,” [Online] Available : https://www.opnfv.org/

[5] Quinn, P. and U. Elzur, “Network Service Header“, draft- ietf-sfc-nsh-04, March 2016.

[6] K-ONE 기술문서 #16 ‘멀티사이트 K-ONE Playground 활용을 위한 K-DevOps Tower의 구축 및 활용 방법

[7] https://wiki.opnfv.org/display/multisite/Multisite

[8] https://wiki.onap.org/pages/viewpage.action?pageId=3247262#app-switcher

[9] https://datatracker.ietf.org/wg/ioam/about/

[10] https://wiki.fd.io/view/VPP

[11] https://wiki.fd.io/view/Honeycomb

[12] F. Brockners, et al., “NSH Encapsulation for In-situ OAM Data,” draft-ietf-sfc-ioam-nsh-00

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: 2018/12