소프트웨어 정의 네트워킹 기술을 위한 컨트롤 플레인 설계 방법

Atto-Research Korea 연구소 ▌송 용 주
Texas A&M University ▌신 승 원

1. 서 론

2008년 OpenFlow[8] 프로젝트가 발표된 이후로 소프트웨어 정의 네트워킹 기술(SDN)은 네트워크의 새로운 패러다임으로 주목받아왔다. Google[2], MS[3] 등 대기업에서는 이미 자체적으로 SDN을 구축/운영해왔을 정도로 SDN은 네트워크의 거부할 수 없는 흐름으로 이미 자리 잡았다. 이에 따라 Nox[4]를 필두로 수많은 오픈소스 및 상용 컨트롤 플레인이 쏟아져 나오고 있다. 본 논문에서는 SDN 기술을 소개하고 현존하는 다양한 OpenFlow 컨트롤 플레인들의 특징을 살펴보고자 한다. 더 나아가, 분석된 특징들을 기반으로 새로운 컨트롤 플레인을 설계할 때 고려해야 할 사항을 짚어보고 각 요소를 결정할 때 추천되는 설계도 제시해본다.

2. 소프트웨어 정의 네트워킹 기술

미국의 스탠포드 대학과 버클리 대학의 연구진들은 기존의 복잡하고 벤더 종속적인 네트워크 장비들의 단점에 주목했다. 4D[5], Ethane[6], SANE[7] 등은 네트워크의 중앙 집중 관제를 다양한 영역에서 소프트웨어만으로 시도한 선행 연구들로서, Software Defined Networking(SDN)의 개념과 유사한 것을 제시하고 있다. SDN은 이들 연구와 유사하긴 하나 새로운 네트워크 구조를 제공한다.

2008년 발표된 OpenFlow[1] 연구 프로젝트는 SDN 기술을 세상에 각인 시키고 많은 연구자들과 네트워크 관련 회사들이 SDN에 관심을 가지게 만든 계기가 되었다. OpenFlow의 핵심 아이디어는 기존 네트워크 장비들의 데이터 플레인과 컨트롤 플레인을 분리하는 것이다. 컨트롤 플레인은 외부에 연결된 장비로 옮겨졌고, 보통 OpenFlow Controller라고 지칭한다. 이 연구는 SDN에서 매우 중요한 컨트롤 플레인과 데이터 플레인 간 인터페이스 및 데이터 플레인의 기능 정의를 포함하고 있다는 점에서 매우 중요한 의의를 가진다. OpenFlow가 SDN에 대한 모든 내용을 포함하고 있지는 않으나, 비로소 SDN을 현실적으로 개발하고 연구할 수 있게 되었다. Open Networking Foundation[8]이 이를 이어받아 2010년부터 현재까지 표준화를 진행하고 있다. 기본적인 OpenFlow의 구조는 그림 1과 같다.

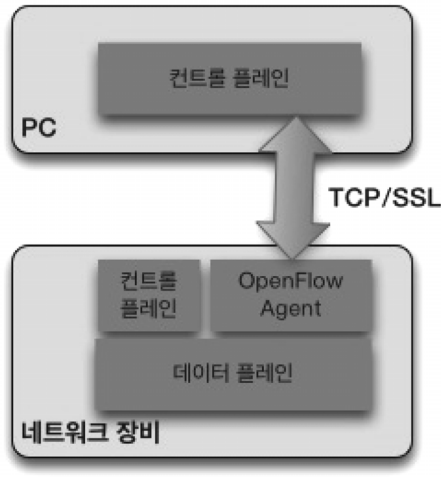

그림 1 OpenFlow 개념도

기존 네트워크 장비는 패킷을 분석하고 경로를 결정하는 복잡한 로직인 컨트롤 플레인과 실제 패킷을 받고 보내는 데이터 플레인을 동시에 가지고 있었다. OpenFlow는 복잡한 컨트롤 플레인의 대부분을 외부 PC로 옮기고 최소한의 것만 네트워크 장비에 남겨둔다. 외부의 컨트롤 플레인은 SSL/TCP 통신 채널을 통해 네트워크 장비와 필요한 정보를 주고받는다. OpenFlow Agent는 네트워크 장비 내에서 데이터 플레인과의 연결을 담당한다. 복잡한 컨트롤 플레인의 기능이 대부분 외부로 나가 있기 때문에 OpenFlow를 지원하는 네트워크 장비들은 간단한 데이터 플레인 기능만 수

행하게 된다.

OpenFlow는 다양한 액션을 표준으로 정의해서 기존 네트워크 장비들의 기능을 거의 모두 수행할 수 있도록 했다. 네트워크 패킷을 전송하거나 버릴 수 있으며, 헤더를 변경할 수도 있다. 네트워크 패킷 헤더를 변경하는 기능은 기존 네트워크 라우터 혹은 스위치의 기능에서 더 진보된 것으로, 이를 바탕으로 더욱 다양한 네트워크 기능을 수행할 수 있게 되었다. 예를 들어, 간단한 Network Address Translation(NAT) 장비를 만드는 경우를 생각해보자. 대부분의 기존 네트워크 장비들은 네트워크 IP 주소와 TCP 포트 번호를 변경할 수 없기 때문에, 별도의 NAT장비를 설치하거나 추가 기능을 설치하는 것이 최선이었다. 그러나 OpenFlow를 이용하는 경우 패킷 헤더를 변경하는 기능을 이용하면 NAT 기능을 SDN 네트워크 장비만을 이용하여 구현할 수 있다.

그림 2는 SDN 네트워크 장비들을 실제로 제어하는 방법을 설명한다. 네트워크 스위치는 OpenFlow에서 제공하는 인터페이스를 이용해서 제어하며, OpenFlow controller를 이용하여 응용 프로그램을 작성한다고 가정하자. 그림 2는 간단한 Layer 2 포워딩 기능을 나타낸다. 네트워크 스위치에 연결된 2개의 호스트(A와 B)들 간의 네트워크 패킷을 주고받는 것이 주 역할이다. 호스트 A가 호스트 B로 패킷을 전송하는 경우 (1), 네트워크 스위치에 관련된 플로우 룰이 없기 때문에 (2), 스위치는 OpenFlow Controller로 플로우 룰 요구 메시지를 보내고 (3), Layer 2 포워딩 응용 프로그램은 패킷을 호스트 B로 전달하는 간단한 플로우 룰을 스위치로 전달한다. (4) 그러면, 이 룰은 스위치 내의 플로우 테이블에 저장되고 (5), 마지막으로 스위치는 이 룰을 바탕으로 패킷을 호스트 B로 전달한다(6).

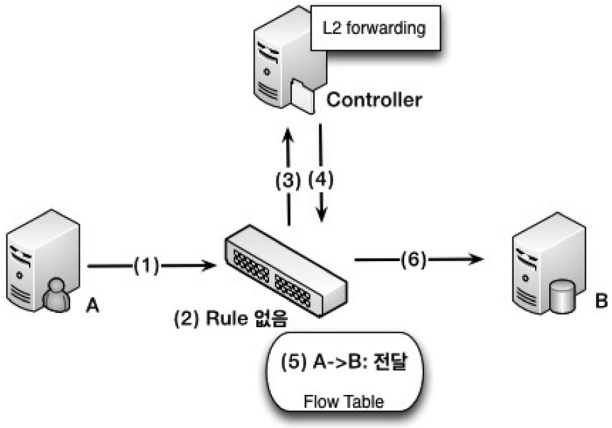

그림 2 OpenFlow를 이용한 L2 스위치 구현 예제

2.1 용어 정의

일반적으로 네트워크 장비 내에는 두 가지 요소가 있다. 하드웨어 부분을 일컫는 데이터 플레인 요소와, 소프트웨어 기능을 말하는 컨트롤 플레인 요소이다. 소프트웨어 정의 네트워킹은 컨트롤 플레인을 데이터 플레인과 분리하여 컨트롤 플레인에 다양한 기능을 운용하기 위한 개념이다.

소프트웨어 정의 네트워킹을 실제로 구현하는 사례에서 OpenFlow가 가장 유명하지만 OpenFlow 자체가 소프트웨어 정의 네트워킹을 말하는 것은 아니다. OpenFlow는 컨트롤 플레인과 데이터 플레인 간의 규격 표준을 뜻한다. OpenFlow 시스템에서 자주 등장하는 컨트롤러라는 용어는 앞서 설명한 컨트롤 플레인을 지칭한다.

3. 컨트롤 플레인의 설계 방법

소프트웨어 정의 네트워킹을 위한 컨트롤 플레인은 일반적으로 두 가지 형태를 보이고 있다. 가장 기본적이고 일반적인 구조는 중앙 집중적인 구조로, 하나의 컨트롤 플레인이 모든 데이터 플레인을 관리하는 구조이다. 중앙 집중적 구조는 하나의 중앙 컨트롤 플레인으로 부하가 몰릴 수밖에 없기 때문에 성능 문제가 항상 제기된다. 이를 개선한 것이 분산 형태의 컨트롤 플레인이다. 본 장에서는 각 형태의 상세한 구조와, 이런 구조를 반영한 컨트롤 플레인의 예제를 선보일 것이다.

3.1 중앙 집중 형태의 설계 방법

처음 OpenFlow와 같은 소프트웨어 정의 네트워킹 구조를 실현할 방법이 제시되었을 때, 기본적으로 등장한 컨트롤 플레인의 구조는 중앙 집중적인 형태였다. 이는 그림 3에 보이는 바와 같이, 다수의 데이터 플레인들을 하나의 컨트롤 플레인이 관리하는 구조이다. 이러한 구조는 하나의 컨트롤 플레인이 모든 데이터 플레인들의 정보를 알 수 있기에 전체 네트워크의 구조와 상태, 변화 등을 알아내기 쉬운 구조이다. 동시에, 하나의 컨트롤 플레인에서 모든 정보를 받아 관리하기 때문에 관리 측면에서 많은 장점을 보인다.

현재 이러한 형태 구조를 가지고 있는 컨트롤 플레인들은 다음과 같은 것들이 있다.

- NOX-classic[9]

스탠포드 대학에서 개발되었으며, OpenFlow 프로토콜을 지원하는 가장 먼저 개발된 컨트롤 플레인으로

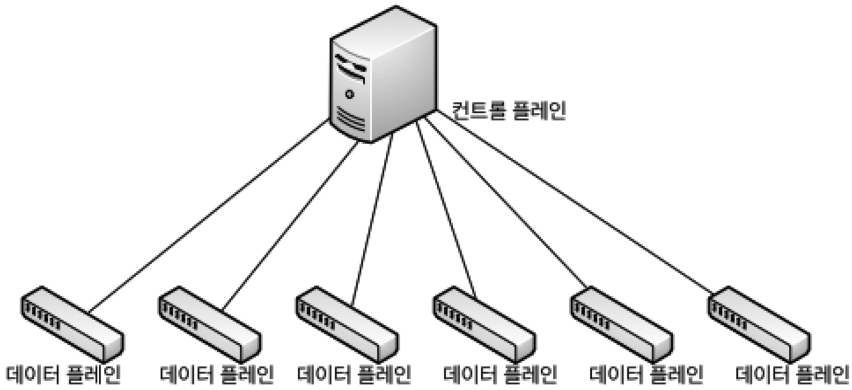

그림 3 중앙 집중 형태의 컨트롤 플레인

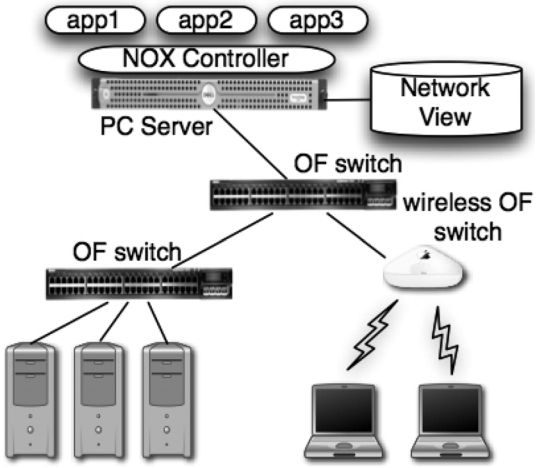

그림 4 NOX-classic 중앙 집중 구조[9]

유명하다. C++ 언어로 개발되었으며, Python과 C++ 등의 언어로 응용 프로그램을 개발할 수 있도록 하였다. 가장 먼저 개발되고, 많은 연구 등에서 쓰인 컨트롤 플레인이었으나, 현재 더 이상 지원이 되지 않고 있으며 개발 역시 멈춰진 상태이다.

• NOX[10]

이전 NOX-classic의 이름을 이어받아 새롭게 개발된 컨트롤 플레인으로, 스탠포드 대학이 아닌, 버클리 대학의 ICSI 연구소에서 개발을 주도하고 있다. 이전 NOX-classic의 Python 인터페이스를 제거하고, 순수하게 C++ 위주로 개발을 할 수 있도록 하였으며, NOX-classic에 비해서 많은 성능 향상을 이루어 냈다. 그러나 발표된지 오래 되지 않은데다 안정성 면에서도 문제점을 보이고 있어서, 아직 많은 곳에서 쓰이지는 않고 있다.

• POX[11]

NOX 최신 버전을 개발하면서 동시에 이를 Python 언어로도 개발하였는데, 이것이 바로 POX이다. 당연히 NOX에 비해서 성능은 좋지 않다. Python 언어로 개발되었기 때문에 확장 및 변경이 매우 쉽다는 장점을 가지고 있어서 연구 목적으로 많이 사용된다.

• Floodlight[12]

소프트웨어 정의 네트워킹 분야의 초기 벤처 회사인 BigSwitch에서 발표한 JAVA 기반의 오픈 소스 컨트롤 플레인으로, 현재 가장 활발히 사용되고 있다. OpenFlow 프로토콜 발표 이후 NOX-classic과 마찬가지로 초기에 발표되었으나, NOX-classic과는 달리 현재도 꾸준히 지원되고 있다. 다양한 주변 프로젝트들도 같이 발전하고 있다.

• Beacon[13]

Floodlight의 전신이 되는 컨트롤 플레인이다. 스탠포드 대학에서 연구 프로젝트로 시작되었으며, 지금은 On.LAB[14]에서 이를 이어받아 발전시켜 나가고 있다. Floodlight와 같이 JAVA 언어로 개발되었다.

이와 같이 다양한 컨트롤 플레인들이 중앙 집중적인 구조로 개발되었다. 현존하는 소프트웨어 정의 네트워킹을 위한 대다수의 컨트롤 플레인들이 이런 중앙 집중적인 구조를 취하고 있는 것이 현실이다.

그러나 중앙 집중 형태의 컨트롤 플레인은 구현 및 관리가 용이하다는 장점과 별개로 본질적인 성능 문제를 가지고 있다. 하나의 컨트롤 플레인이 다수의 데이터 플레인을 제어하고 정보를 받아 들여야 하기 때문에 컨트롤 플레인이 병목이 될 가능성이 항상 존재한다. 데이터 플레인의 수가 늘어나는 경우, 혹은 특정 데이터 플레인에서 대량의 정보를 보내는 경우에 하나의 컨트롤 플레인에서 이를 모두 처리하기는 쉽지 않다.

예를 들어, 어떤 소프트웨어 정의 네트워킹 환경에

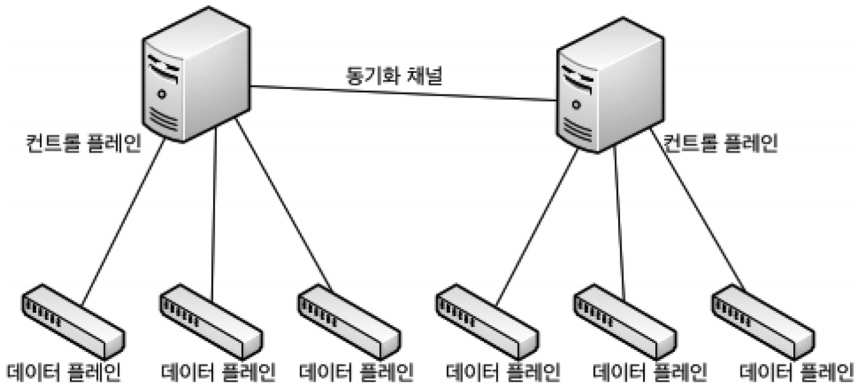

그림 5 분산 형태의 컨트롤 플레인

서 하나의 컨트롤 플레인이 10대의 데이터 플레인을 관리하고 있다고 가정해 보자. 이 때, 네트워크의 규모가 커져서, 추가로 100대의 데이터 플레인을 더 설치한다고 하면, 이런 경우 기존의 컨트롤 플레인으로 추가된 100대의 데이터 플레인 역시 관리할 수 있을까? 만약, 네트워크 관리자가 컨트롤 플레인을 10대의 데이터 플레인 정도만을 처리하기 위하여 성능을 맞추었다면 추가된 100대의 데이터 플레인을 처리하는 것은 불가능 할 것이다. 물론 더 성능이 좋은 컨트롤 플레인을 대신 설치하는 것도 방법이 될 수 있다. 그러나 계속 데이터 플레인의 수가 늘어난다면 컨트롤 플레인의 확장성 문제는 언젠가는 발생할 수밖에 없다.

3.2 분산 형태의 설계 방법

중앙 집중 형태의 성능 문제를 해결하기 위하여 분산 형태를 가지고 있는 컨트롤 플레인들이 제안되었다. 하나의 컨트롤 플레인으로 모든 데이터 플레인들을 제어 및 관리하는 중앙 집중 형태와 달리, 그림 5와 같이 여러 개의 컨트롤 플레인들을 두고, 다수의 데이터 플레인들을 각각 나누어서 관리하는 형태로 되어 있다.

각 컨트롤 플레인은 다수의 데이터 플레인들을 나누어 관리한다. 데이터 플레인의 수가 늘어나게 되는 경우 컨트롤 플레인을 추가하여 추가분을 손쉽게 처리할 수 있다.

현재, 분산 구조를 지원하는 컨트롤 플레인들은 다음과 같은 것들이 있다.

- ONIX[15]

구글과 Nicira(현 VmWare)에서 같이 연구 개발한 분산 구조의 컨트롤 플레인으로, 구글 내에 있는 대규모의 네트워크(데이터 플레인들)를 관리하기 위한 목적으로 개발되었다. 각 관리 네트워크 별로 컨트롤 플레인을 따로 두었으며, 이들 간에는 NIB(Network Information Base)를 주고받는 것으로 동기화 문제를 해결하였다. 구글 내에서 개발한 것이라 자세한 정보는 공개되지 않았고, 컨트롤 플레인 역시 공개되지 않았다.

- ONOS[16]

OpenFlow 및 소프트웨어 정의 네트워킹 기술을 발전시키기 위하여 출범한 비영리 기관인 On.LAB[14]에서 개발한 분산형 컨트롤 플레인이다. 2013년 Open Networking Summit[17]에 처음 발표되었으며, 오픈소

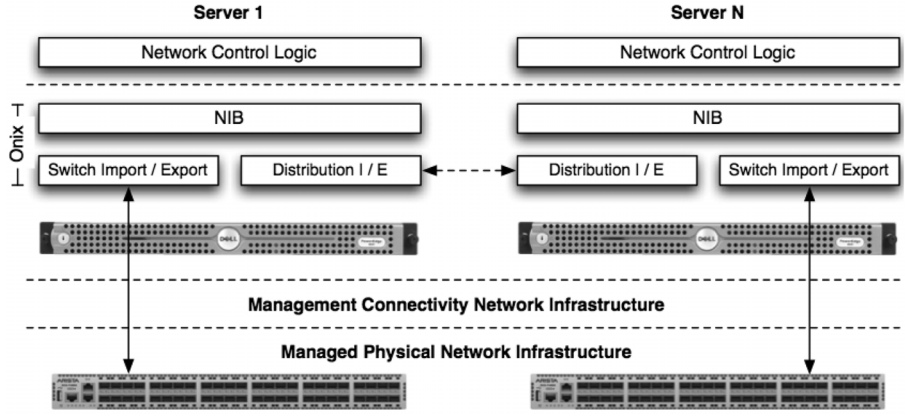

그림 6 ONIX 분산 구조[15]

스로 발표된 최초의 분산형 컨트롤 플레인이기에 많은 사람들의 기대를 받고 있다. 기본 구조는 ONIX에서 발표한 구조를 많이 참조한 것으로 알려져 있다.

• HyperFlow[18]

NOX 최신 버전을 개발한 연구원들이 개발한 분산형 구조의 컨트롤 플레인으로, 성능 향상에 주 초점을 맞추고 있다. 각 컨트롤 플레인들 간의 동기화를 위해서 WheelFS[19]을 쓰고 있다.

분산 형 구조의 컨트롤 플레인에서 중요한 것은 컨트롤 플레인들 간의 정보 동기화이다. 소프트웨어 정의 네트워킹의 기본 개념이자 가정 중 하나는 컨트롤 플레인에서 전체 데이터 플레인들의 정보를 볼 수 있다는 것이다. 중앙 집중 형태의 구조에서는 하나의 컨트롤 플레인이 모든 데이터 플레인에 대한 정보를 가지고 있기 때문에 당연한 개념이다. 반면에 분산 형태의 경우 각 컨트롤 플레인이 관리하는 데이터 플레인들이 다르기 때문에 각각의 컨트롤 플레인은 전체 데이터 플레인들의 정보를 알 수 없다. 전체 데이터 플레인들의 정보를 아는 것은 토폴로지 구성 등 SDN의 핵심기능에 반드시 필요한 부분이다.

가장 명확한 해결책은 각 컨트롤 플레인들끼리 서로 필수 정보를 공유하는 것이지만 몇 가지 이유로 쉬운 일은 아니다. 먼저, 어디에 어떤 정보가 있는지 알아내는 것부터 쉽지 않다(Lookup Problem). 어떤 컨트롤 플레인에서 자신이 관리하지 않는 데이터 플레인의 정보를 보고 싶은 경우, 다른 컨트롤 플레인에서 받아 와야 한다. 이 데이터 플레인에 대한 정보를 가진 컨트롤 플레인이 어느 것인 알아야 정보를 받아 올 수 있다. 두 번째로, 정보를 받아왔더라도 언제까지 이 정보가 유효한지 알기가 어렵다(Consistency Problem). 데이터 플레인의 속성 등이 변경된 경우 이전에 받아온 정보는 틀린 정보가 된다. 물론, 받아온 정보를 별도로 관리하지 않고 매번 정보를 받아 오면 이 문제를 부분적으로나마 해결할 수 있으나 또 다른 성능 문제를 야기할 수 있다. 또한, 정보를 받아온 직후에 바로 데이터 플레인의 속성이 변경되는 경우를 고려하면 유효성 문제는 여전히 존재한다. 마지막으로, 다수의 데이터 플레인들을 각 컨트롤 플레인들에 어떤 분포로 연결할지 결정해야 하는 문제가 있다(Distribution Problem). 각 컨트롤 플레인들에게 같은 수의 데이터 플레인들을 공평하게 연결하는 방법이 간단하지만, 네트워크의 구조와 기타 운영을 고려하였을 때 효율적이지 못할 경우가 많다.

4. 중앙 집중 형태와 분산 형태의 컨트롤 플레인 비교

앞서 살펴보았듯이, 중앙 집중 형태의 컨트롤 플레인과 분산 형태의 컨트롤 플레인들은 각기 장단점을 가지고 있다. 이 장에서는 이들의 장점과 단점 그리고 특징들을 요약하여 정리한다.

항목	중앙 집중 형태	분산 형태
특징	– 하나의 컨트롤 플레인만 존재. – 모든 데이터 플레인들이 하나의 컨트롤 플레인에 연결되어 통제를 받음.	– 다수의 컨트롤 플레인 존재. – 데이터 플레인들이 각각의 컨트롤 플레인에 연결됨. – 각 컨트롤 플레인들은 서로 정보를 주고받음.
장점	– 관리가 쉬움. – 컨트롤 플레인 하나에서 모든 데이터 플레인들의 정보를 볼 수 있음	– 성능 이슈가 적음(컨트롤 플레인에 접속하는 데이터 플레인 수를 조절할 수 있음). – 확장성이 좋음.
단점	– 다수의 데이터 플레인이 연결하는 경우에 성능문제 발생(확장성 문제). – 적은 수의 데이터 플레인이 있더라도, 많은 양의 정보가 전달되면 컨트롤 플레인의 성능문제 발생	– 컨트롤 플레인들 간의 정보 동기화가 어려움. – 데이터 플레인들을 나누어 관리하는 문제가 어려움
예제	NOX, Floodlight, POX	ONIX, ONOS, HyperFlow

5. 컨트롤 플레인 설계 제안

지금까지 소프트웨어 정의 네트워킹 환경을 위한 컨트롤 플레인들의 구조와 특징들을 알아보았다. 이런 상황에서, 새로운 컨트롤 플레인을 설계할 때 고민해 봐야할 문제를 짚어보자.

• **구조 선택**

위에서 언급한 중앙 집중 구조와 분산 구조 중에서 하나를 선택하는 것이 가장 중요하다. 분산 형태의 구조로 일단 개발한 후, 하나의 컨트롤 플레인으로 모든 데이터 플레인들의 관리가 가능하다면 하나의 컨트롤 플레인만 이용하고 부족한 경우만 여러 컨트롤 플레인들을 운영할 수 있게 만드는 것이 가장 이상적인 설계일 수 있다. 그러나 여러 개의 컨트롤 플레인들이 필요 없을 때는 의미 없는 개발로 인한(컨트롤 플레인들 간의 동기화 해결 기능 등등) 비용 및 인력의 낭비가 발생한다.

따라서 현재 처한 환경에서(혹은 앞으로도) 여러 컨트롤 플레인들이 필요 없는 경우에는, 중앙 집중 구조로 설계하는 것이 가장 바람직하다고 할 수 있다.

• 개발 언어 선택

개발 언어 역시 여러 가지 선택이 가능하다. SDN 컨트롤 플레인에 국한하지 않고 일반적으로 성능을 가장 우선적으로 두는 개발은 C 혹은 C++ 언어를 이용하여 개발하는 것이 좋다. 이 두 가지 개발 언어를 이용하면 다양한 커널/네트워크의 튜닝/성능 향상이 가능하기 때문에 성능 우선의 컨트롤 플레인에 가장 적합하다. JAVA의 경우, 성능은 C/C++에 약간 뒤지지만 개발 편이성이 매우 높기 때문에 좋은 대안이 될 수 있다. Python 등과 같은 고수준 언어는 개발이 많이 쉬워지는 장점이 있으나, 성능을 상당부분 포기해야 하는 단점이 있다. 성능 보다는 빠른 개발을 필요로 하는 연구 목적의 프로토타입 등과 같은 경우에 Python 등과 같은 언어가 적합하다.

• 응용 프로그램 인터페이스 선택

컨트롤 플레인을 개발할 때는 자체의 성능 및 구조뿐만 아니라 그 위에서 실행되는 응용 프로그램들의 성능 및 개발 편이성도 고려해야 한다. 가장 좋은 것은 하나의 컨트롤 플레인에서 다양한 인터페이스(C, C++, JAVA, Python 등등)를 제공하는 것이지만, 이 역시 개발 비용 및 시간이 들어가기 때문에 가급적 특정 인터페이스에 집중하는 것이 좋다.

일반적으로 인터페이스는 컨트롤 플레인의 개발 언어에 의존하는 경우가 많다. 보통 C로 개발한 컨트롤 플레인의 경우 C로 된 인터페이스를 먼저 지원하게 된다. 하지만, 빠른 응용 프로그램의 개발을 위해서, Python 등과 같은 고수준의 언어 역시 지원하는 것이 바람직하다.

6. 결 론

이 논문에서는 현재 많은 화제가 되고 있는 소프트웨어 정의 네트워킹의 핵심 기술인 컨트롤 플레인에 대해서 설명하고 있다. 현재 많이 쓰이는 컨트롤 플레인들의 기술 방식에 - 중앙 집중적인 구조와 분산 구조 - 대해서 설명하고, 각각의 장단점에 대해서 기술하였다. 또한, 이런 지식들을 바탕으로 차후 컨트롤 플레인을 설계할 때 어떤 방식으로 해야 하는지에 대한 것을 기술하였다.

참고문헌

[1] OpenFlow, OpenFlow Swtch Specification} version 1.1.0, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[2] B4: Experience with a Globally-Deployed Software Defined WAN Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat

[3] Achieving High Utilization with Software-Driven WAN Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer

[4] Natasha Gude and Teemu Koponen and Justin Pettit and Ben Pfaff and Martín Casado and Nick McKeown and Scott Shenker, NOX: Towards an Operating System for Networks, Proceedings of ACM SIGCOMM Computer Communication Review, July, 2008

[5] Albert Greenberg and Gisli Hjalmtysson and David A. Maltz and Andy Myers and Jennifer Rexford and Goeffrey Xie and Hong Yan and Jibin Zhan and Hui Zhang, A Clean Slate 4D Approach to Network Control and Management, Proceedings of ACM Computer Communications Review, 2005

[6] Martin Casado and Michael J. Freedman and Justin Pettit and Jianying Luo and Nick McKeown and Scott Shenker, Ethane: Taking Control of the Enterprise, Proceedings of ACM SIGCOMM, 2007

[7] Martin Casado and Tal Garfinkel and Michael Freedman and Aditya Akella and Dan Boneh and Nick McKeowon and Scott Shenker, SANE: A Protection Architecture for Enterprise Networks, Proceedings of Usenix Security Symposium, 2006

[8] Open Networking Foundation, http://www.opennetworking.org/

[9] NOX-classic, http://www.noxrepo.org/nox/nox-classic - repo/

[10] NOX, http://www.noxrepo.org/nox/

[11] POX, http://www.noxrepo.org/pox/

[12] FloodLight, http://www.projectfloodlight.org/floodlight/

[13] Beacon, https://openflow.stanford.edu/display/Beacon/Home

[14] ON.LAB, http://onlab.us/

[15] Koponen, Teemu and Casado, Martin and Gude, Natasha

and Stribling, Jeremy and Poutievski, Leon and Zhu, Min and Ramanathan, Rajiv and Iwata, Yuichiro and Inoue, Hiroaki and Hama, Takayuki and Shenker, Scott, Onix: a distributed control platform for large-scale production networks, Proceedings of the 9th USENIX conference on Operating systems design and implementation, 2010

[16] ONOS, http://onlab.us/tools.html

[17] Open networking summit, http://www.opennetsummit.org/

[18] Amin Tootoonchian, A Yashar Ganjali, HyperFlow: a distributed control plane for OpenFlow, Proceedings of the 2010 internet network management conference on Research on enterprise networking

[19] WheelFS file system, http://pdos.csail.mit.edu/wheelfs/doku.php?id=wheelfs

약 력

송 용 주

2000 KAIST 전자전산학과 학사
2002 KAIST 전자전산학과 석사
2007 KAIST 전자전산학과 박사
2007~2013 (주)티베로 BigData팀
2013~현재 (주)아토리서치코리아 재직중
관심분야 : SDN, 네트워크 보안, 빅데이터, 클라우드, 분산시스템
E-mail: yongjoo.song@atto-research.com

신 승 원

1998 KAIST 전자전산학과 학사
2000 KAIST 전자전산학과 석사
2013 Texas A&M 컴퓨터공학과 박사
2000~2002 (주)티맥스소프트
2002~2004 ETRI 정보보호연구단
2004~2005 MIT 방문연구원
2005~2009 (주)티맥스소프트
관심분야 : SDN, 네트워크 보안, 클라우드, 분산시스템
E-mail: seungwon.shin@neo.tamu.edu