편집위원 : 이순석 (ETRI)

SDN 개발 사례 : 고성능 SDN 제어기 MuL

허 찬, 박성용
한국기술교육대학교, 연세대학교

요 약

네트워크 장비의 제어평면과 전송평면을 분리하고 전체 네트워크의 제어평면을 하나의 중앙 제어기에 통합하는 네트워크 아키텍처인 소프트웨어 정의 네트워킹 (Software Defined Networking)는 네트워크 산업과 학계에서 많은 관심을 받고 있다. 개별적으로 동작하는 기존 네트워크 장비와 달리 모든 제어평면의 기능이 하나의 중앙 제어기에 집중되기 때문에 제어기의 성능, 안정성, 유연성은 해당 제어기가 통제하는 네트워크 전체에 지대한 영향을 미치게 된다. 또 SDN의 제어계층과 전달계층 사이에 표준통신 인터페이스로 확고한 자리를 잡은 OpenFlow는 세계 유수의 글로벌ICT들이 참여하여 SDN열풍의 중심에 위치해있다. 본고 에서는 SDN의 개념에 대해 살펴보고, OpenFlow의 소개와 동작을 설명할 것이다. 또한 국내에서 최초로 개발된 새로운 OpenFlow 제어기인 MuL 제어기 플랫폼을 소개하고 다른 제어기들과의 비교분석을 통해 어떤 차별점을 갖는지 알아보고자 한다.

Ⅰ. 서 론

정보화 시대에 이르러 컴퓨터의 하드웨어와 소프트웨어는 비약적인 발전을 거듭하게 되었다. 오늘날 스마트폰으로 대표되는 하드웨어와 운영체제, 데이터베이스, 분산 시스템 등의 소프트웨어를 손에 꼽을 수 있다. 하지만 다양한 컴퓨터 자원을 연결해 주는 네트워크 (Network)는 다른 분야와 비교해 혁신적인 발전을 이루어 내지 못하고 있다. 물론 단말 시스템 (End-system)에서 돌아가는 TCP 등의 일부 프로토콜은 지속적으로 향상되었지만, 단말과 단말 사이에서 데이터 전송과 전달을 담당하는 액세스 코어 네트워크의 프로토콜이나 기타 기능들은 대형 벤더들에 의해 이미 대부분 개발이 끝나서 적용 및 서비스되고 있는 단계이기 때문에 새롭게 수정하거나 개발하기 어렵다. 또한 네트워크의 관리 환경도 일부 운영자나 엔지니어에게

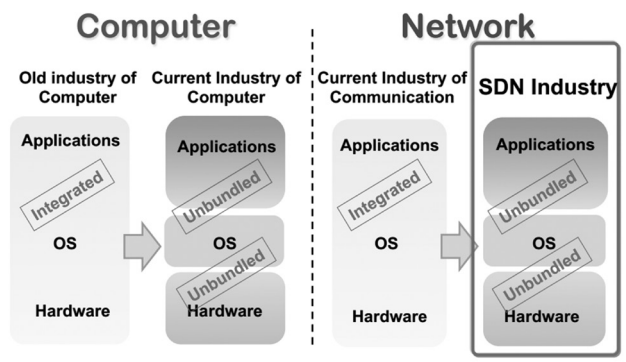


그림 1. 컴퓨터와 네트워크의 발전 방향[18]

종속되어 있는데다가 이마저도 전문적인 지식이나 기술을 요구하므로 사용하기 쉽지 않다. 소프트웨어 정의 네트워킹 (SDN)는 구조는 단순하지만 관리가 복잡하고 인터페이스가 어려운 네트워크의 현실을 해결하기 위한 새로운 기술로 화두 되고 있다[1].

SDN은 인프라 동작이 다양한 서비스와 환경에 따라 동적으로 제어될 수 있는 유연한 구조를 충족시키기 위하여 사용자가 원하는 형태의 네트워크를 범용 하드웨어와 소프트웨어를 사용하여 사용자가 설정하고 관리하는 것을 가능하게 하는 네트워크 아키텍처이다. 구체적으로 살펴보면 SDN은 네트워크 스위치 (Switch)나 라우터 (Router)에서 제어평면 (Control Plane)과 전송평면 (Data Plane)을 분리하는 개념이다. 여기

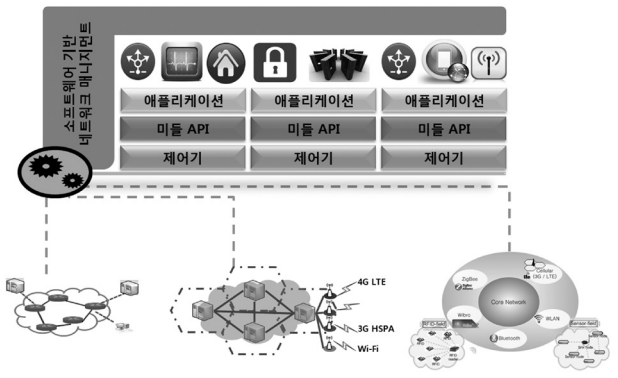


그림 2. SDN개념도

서 제어평면은 흔히 제어기 (Controller)라고 불리는 소프트웨어로 구현되어 전체 네트워크 장비의 정보를 수집하고 통제하는 역할을 맡아, 라우팅 정책 (Policy), 네트워크 가상화 (Virtualization) 등을 수행하게 된다. 이러한 구조는 연구자나 사용자가 쉽고 효율적인 라우팅/스위칭 정책 변경을 가능하게 하므로 네트워크 산업과 연구자 사이에서 주목을 받게 되었다 [2-3].

SDN기반 프로토콜 중에서도 ONF (Open Network Foundation)에서 표준화 중인 OpenFlow 프로토콜 (Protocol)이 대표적이다. ONF는 마이크로소프트, 구글 등과 같은 다양한 80 여 글로벌 ICT 업체들이 회원사로 참가하고 있으며 국내에는 SKT, KT, 삼성전자가 참여하고 있다. OpenFlow에는 Stanford의 주도로 개발된 NOX 제어기[3], BigSwitch Networks에서 공개한 Floodlight 제어기[4] 등을 비롯한 다양한 OpenFlow 제어기 들이 발표되었고 지속적으로 개발되고 있다. 각각의 제어기들은 각자 다른 지향점을 가지고 구현이 되었는데 아직까지 제어기의 중요한 부분 중에 하나인 성능 (Performance)에 초점을 맞춘 제어기는 미비한 것이 현실이다.

이에 본고에서는 SDN의 개념과 현황을 고찰하고 국내에서 최초로 개발하여 오픈소스로 공개한 OpenFlow 기반 제어기인 MuL 제어기 플랫폼을 소개한다. 또, MuL제어기가 기존에 나와있는 다른 제어기와 어떤 차별 점을 갖는지 알아보고자 한다.

Ⅱ. 본 론

1. SDN

소프트웨어 정의 네트워킹(Software Defined Networking;SDN)은 네트워크의 제어평면 (네트워크 제어기능)이 전송평면 (패킷 포워딩)과 분리되어 전송평면에 직접 프로그래밍을 할 수 있게 해주는 새로운 네트워크 아키텍처이다. 일반적으로는 개별 네트워크 장비에서의 제어기능이 하드웨어에서 분리되지 않는다. 하지만 SDN에서는 전송평면과 제어평면이 분리되어 있다. 이 때 분리된 전송평면은 고속 전송 기능만을 담당하는 부분으로, 판단기능이 없이 단순한 하드웨어 전송만을 한다. 라우팅이나 스위칭 등 복잡한 판단 기능은 제어평면으로 분리된다. 이 때 분리된 제어평면을 일반 사용자가 개별적으로 프로그램 할 수 있도록 컴퓨터로 옮기고 C나 Python 등 일반적 프로그램 언어로 제어평면의 기능을 구현할 수 있다. SDN 또는 OpenFlow라 하는 것은 이렇게 분리된 제어평면과 전송평면이 어떻게 정보를 주고 받을 것인가에 대한 API를 정의한 것에 불과하다. 하지만, 이런 간단한 변경만으로 일반 사용자가 논리적 또는 가상의 실체로서 하나 또는 다수의 하드웨어가 구성하고 있는 네트워크를 관리, 제어 할 수 있는 어플리케이션 및 네트워크 서비스로의 이용을 가능케 해줄 수 있다는 것이 SDN이 각광받는 이유이다.

SDN 아키텍처의 논리적인 구조<그림 2>를 보면, 통신장비의 핵심 기능인 제어평면을 소프트웨어 기반의 SDN 제어기에 집중해 구현하고 있다. 따라서 사용자가 새로운 기능을 구현하려면 일반 응용프로그램을 만들듯이 SDN제어기 상에서 프로그램하고 컴파일 하면 된다. 기존 방식으로 새로운 통신기능을 구현하려면 국제적 표준화 작업으로 시작해 4-5년간의 수많은 논의를 거쳐 기능이 구현되었지만, SDN을 활용한 새로운 네트워크 기법을 이용하면 자가 망과 기존 망을 접속하는 부분의 표준만 준수하면 자가 망의 기능을 임의로 만들어 쓸 수 있다. 특히 네트워크 운영자 및 관리자는 분산되어있는 다양한 네트워크 장비에서 해당 장비에 맞는 커맨드 라인의 입력을 통해 설정하지 않고, SDN의 프로그래밍 방식으로 보다 단순화 하여 네트워크를 설정할 수 있다. 또 IT 부서는 SDN 제어기의 중앙집중형 특징을 활용하여 실시간으로 발생하는 네트워크 문제를 해결할 수 있으며, 새로운 어플리케이션과 네트워크 서비스를 배치하는데 소요되던 몇 주 또는 몇 달의 시간을 몇 분 또는 몇 시간으로 단축할 수 있다. SDN은 네트워크의 설정, 관리, 보안 등에 있어 관리자에게 유연성을 제공하고 역동적이고 자동화된 SDN 프로그램을 통해 네트워크 리소스를 최적화 할 수 있다. 마지막으로 네트워크 공급업체 고유의 특징 또는 네트워크 장비 분야에서의 폐쇄적인 소프트웨어 환경을 따르지 않고 운영자 및 관리자가 네트워크 어플리케이션을 스스로 작성할 수 있다.

이와 같은 SDN 아키텍처는 라우팅(Routing), 멀티 캐스트 (Multi-Cast), 보안 (Security), 액세스 제어 (Access-Control), 대역폭 관리 (Bandwidth Manage), 트래픽 엔지니어링(Traffic Engineering), 프로세서 및 스토리지 최

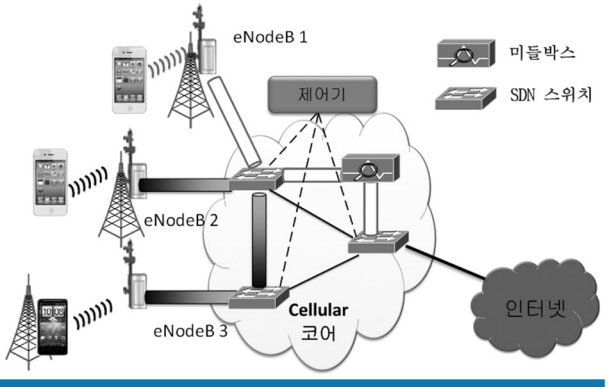


그림 3. SDN 기반의 LTE 서비스[7]

적화(Optimization), 서비스 품질(QOS), 에너지 사용량(Consumption of Energy), 모든 형태의 정책관리(Policy Manage)를 포함하여 사용자가 비즈니스 목표에 부합하는 맞춤형으로 네트워크 서비스를 구현할 수 있도록 API(Application Programming Interface)의 집합을 제공한다.

현재 SDN연구는 초창기 대학 캠퍼스 네트워크 레벨에서의 서비스 모델에서[1], 현재의 데이터 센터 네트워크 서비스[4], 코어 네트워크에서의 트래픽 엔지니어링 서비스[6], 홈 네트워크 서비스[5], 최근에는 무선 LTE 쪽에서의 SDN 기반의 서비스[7]등 다양한 서비스 모델들이 연구 중에 있다. 이와 함께, 제어기의 성능 평가에 관한 연구도[8] 상당 수 진행되어 왔고, 실제 적용 시 제어기의 scalability 문제가 항상 이슈가 되어왔다[9]. 이를 해결하기 위한 많은 연구가 단일 제어기 기반의 제어기가 아닌, 복수개의 제어기와 역할을 나누어 Master 와 Slave 식의 역할 관계를 활용한 접근법[10], 스위치 레벨에서 제어기 레벨로 전달되는 메시지를 최소화 하기 위한 접근법[11] 등 다양한 방면에서의 연구가 진행 중에 있다.

2. OpenFlow[2]

OpenFlow는 SDN 아키텍처의 제어평면과 전송평면 사이에 정의된 최초의 표준 통신 인터페이스이다. OpenFlow 프로토콜을 이용하여 SDN 아키텍처에서의 스위치, 라우터 등 네트워크 장비의 패킷 전달기능(Forwarding Plane)에 직접 접속하여 조작할 수 있다.

〈그림 1〉에 언급했던 것처럼, 기존 네트워크 장비들은 오래된 메인프레임과 같이 획일적이고 폐쇄적인 특징을 가졌다. 그러나 시장의 요구사항과 환경의 변화로 네트워크 제어 기능을 네트워킹 장비-하드웨어 에서 분리하여 논리적으로 중앙 집중화된 제어기-소프트웨어로 이동시키는 요구가 발생 하게 되었고, 이에 표준화된 OpenFlow와 같은 프로토콜이 필요하게 되었다. 이러한 OpenFlow 프로토콜은 네트워크 장비와 SDN 제어기 사이에 존재하는 인터페이스(Interface)로서 양쪽 영역에서 실행된다.

SDN의 핵심 컴포넌트인 OpenFlow는 일종의 통신 프로토콜로서 OpenFlow 제어기와 OpenFlow 지원 네트워크 장비 사이에서의 커뮤니케이션 역할을 담당한다. 현재 IP 기반의 라우팅에서는 다양한 요구수준에 상관없이 두 종단점(Endpoint) 사이의 데이터 흐름을 네트워크 안에서 동일한 경로로 흐르게 하여 결국 사용자에게 어플리케이션(Layer 7) 및 세션계층(Layer 5)에서의 컨트롤을 제공할 수가 없다. 하지만 OpenFlow를 이용하면 사용 패턴, 어플리케이션, 클라우드 자원 등과 같은 가시적인 파라미터를 바탕으로 트래픽을 제어하는 것이 가능하여 진다.

이러한 이유는 기존의 네트워크 장비가 하드웨어 기반의 플로우테이블(Flow Table)과 소프트웨어 기반의 라우팅 프로토콜이 밀접하게 결합되어 프로그래머블한 환경을 제공하지 못하였던 반면, SDN 은 OpenFlow 프로토콜과 같은 통신 프로토콜을 이해할 수 있는 소프트웨어 제어기를 통해 플로우테이블을 조작하고 제어함으로써 네트워크 장비의 프로그램 가능성을 제공할 수 있다. 따라서 앞서 얘기했듯이 SDN 아키텍처는 L1-7 레이어에 이르는 광범위한 범위에서의 컨트롤을 가능하게 해준다. 결국 이는 네트워크가 사용자, 어플리케이션, 세션계층 수준에서의 실시간 변화에 대응할 수 있도록 해주고 프로그래머블한 환경을 제공함으로써 다양한 서비스에 특화된 네트워크를 운용하는 것이 가능하게 해준다.

3. MuL 제어기[12-13]

쿨클라우드에서 개발한 MuL 제어기는 다른 OpenFlow 제어기와는 다르게 성능과 안정성을 중점으로 두고 설계된 OpenFlow 제어기이다. 네이티브 C언어로 개발되었고 멀티쓰레딩 (Multi-Threading)을 고려하여 설계하여 미션 크리티컬 (Mission Critical)한 네트워크의 현실을 반영하여 실제네트워크에서 SDN의 활용이 가능하도록 개발하였다. 또한, Two-Level API (Application Programming Interface)를 지원하여 사용자의 다양한 요구조건을 유연하게 수용하고자 한 것도 특징이다.

MuL제어기의 내부영역은 크게 아래와 같이 4가지로 나눌 수 있다.

1) 스위치나 App의 요청을 처리하는 멀티쓰레드 코어 영역
2) Mid-level API (MLAPI)를 사용하는 MLAPI App 영역
3) RESTful 노스 바운드 (North-Bound) API (NBAPI)를 사용하는 NBAPI App 영역
4) Openflow와 같은 SDN 프로토콜을 통해 라우터와 통신하는 사우스 바운드 (South-Bound) 영역

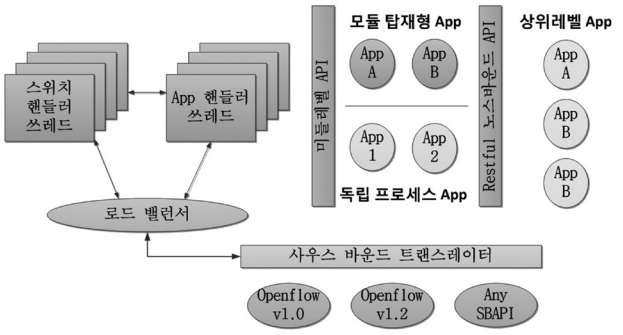


그림 4. MUL 제어기의 아키텍처

3.1 MuL 코어 영역

코어 영역에서는 어플리케이션에서 내려지는 명령을 처리하여 스위치로 전달하거나 스위치의 정보를 원하는 어플리케이션에 전송하는 역할을 한다. 다중 스레드를 사용하여 작업을 처리하여 멀티코어 CPU의 활용을 극대화 하였고 Cache-aware 프로그래밍을 적극 활용하여 성능을 최적화 (특히 인텔 계통 CPU에서) 한 것이 특징이다. 그리고 로드 밸런싱 기능을 통하여 내부 프로세스간 통신 시의 자원 사용의 효율성을 최대화 하도록 설계하였다.

3.2 MLAPI Application 영역

C로 작성된 MuL Mid-level API(MLAPI)는 사용성 이나 범용성 보다는 성능에 초점을 맞춘 API이다. MLAPI를 사용하는 어플리케이션은 독립적인 프로세스로 실행되어 제어기와 통신할 수도 있지만, 최대한의 성능을 위해 MuL 제어기안에 모듈(Module) 형태로 탑재될 수도 있다. 전자의 경우 사용자의 의도에 따라 어플리케이션을 어떤 요구에 따라(On-Demand) 등록할 수 있도록 하는 장점이 있는 반면에, 후자의 경우 제어기와 같은 프로세스를 사용하여 명령 처리 시간을 줄이고 성능을 극대화 할 수 있는 장점이 있다.

3.3 NBAPI Application 영역

RESTful(Representational State Transfer) 한 North-Bound API(NBAPI)는 성능 측면에서는 MLAPI보다 떨어지지만, 범용적인 RESTful 한 요청(Request)/응답(Response) 체계를 가짐으로써 네트워크에 연결된 좀더 다양한 클라이언트에서 편하고 쉽게 사용될 수 있는 API 이다. 이러한 SOA(Service Oriented Architecture) 아키텍쳐 설계를 통하여 제어기에 추상화된 서비스 기반의 접근이 가능해지고, 이를 통하여 네트워크 어플리케이션 개발이 용이하여지게 된다.

3.4 South-Bound 영역

사우스바운드(South-Bound) 영역은 제어기가 전송평면 사이에서 통신하는 부분으로써 먼저 어플리케이션 영역으로부터 제어기가 요청을 받으면 각 전송평면에 맞는 프로토콜로 변환시키는 사우스바운드 변환기(Translator)를 통해 OpenFlow 1.0, OpenFlow 1.2 등 SDN 프로토콜로 변환시켜 전송평면에 전달하는 구조를 갖고 있다. 이러한 구조는 추후 다른 SDN 프로토콜에 대응하는 전송평면의 지원을 용이하게 한다. 또한 MuL 제어기의 사우스바운드 영역은 OpenFlow 만으로 한정하지는 않는다. 이 외에 SNMP(Simple Network Mangement Protocol), YANG/NETCONF(Network Configuration Protocol)와 같은 기존의 네트워크 매니지먼트 시스템과의 연동을 제공함으로써 기존의 legacy 장비 기반의 네트워크 매니지먼트 시스템과의 연동을 통한 통합 환경을 제공하여 기존의 네트워크와의 연동성을 제공하는 것을 목적으로 한다.

4. SDN 제어기 성능 비교 분석

이번 절에서는 Openflow 프로토콜이 발표된 2009년 이후 Stanford의 주도로 개발된 NOX 제어기, BigSwitch Networks에서 공개한 FloodLight 제어기, 그리고 국내에서 새로 개발된 MuL 제어기의 성능을 측정, 비교분석 하여 MuL 제어기의 강점을 확인한다.

4.1 성능 측도 및 측정방법

제어기의 성능에는 다양한 성능측도가 적용될 수 있다. 본고에서는 제어기의 실질적인 성능을 나타내는 플로우 처리속도(Flow setup requests / second)를 다양한 조건하에서 측정하여 제어기 간의 성능을 비교하였다.

성능 측정을 위해 사용된 2대의 서버는 Intel XEON 프로세서의 8G RAM 기반의 Ubuntu 12.10 버전의 환경에서 실험을 하였다. 실험을 위해 1대의 서버는 제어기의 프로세스들이 동작을 하게 되고, 다른 1대의 서버는 Stanford 대학에서 개발한 cbench[14]라는 벤치마크 툴을 throughput 모드로 사용하였다. 로컬 레벨에서의 PCI BUS 기반이 아닌 실제 네트워크 환경에서의 성능 측정을 위하여 각 서버는 10Gbps 82599 Intel 카드를 사용하여 연결 시켰고, 2대의 서버 간의 대역폭을 tc[15] 툴을 사용하여 100Mbps 부터 10Gbps 까지 변화시켜 가면서 성능 측정을 하였다.

cbench는 가상의 스위치들을 시뮬레이션 하기 위한 툴로써 이를 통해 생성된 가상의 스위치들은 중앙 제어기에 연결하여 플로우 요청 (Openflow packet-in 메시지)를 보내고 제어기가 보내오는 플로우 설정 명령(Openflow flow-mod, packet_out 메시지)에 대한 응답 패킷을 Counting 함으로써 전체적으로 처리된 플로우 요청 패킷의 평균 갯수를 구하게 된다.

본 연구에서는 스위치 개수를 32개로 고정 시킨 상태에서 총 16번의 시험 반복을 통한 평균 값을 기본으로 작성을 하였다. 이 과정에서 제어기 레벨에서의 쓰레드 수에 의한 영향을 측정하기 위하여 쓰레드의 갯수를 1, 4, 8개 단위로 조정해 가며 성능을 측정하였다. 일반적으로 네트워크가 이상적인 상황이라면 제어기와 스위치로부터 들어오는 플로우 요청 메시지의 Aggregated 된 링크의 대역폭이 증가함에 따라 성능이 일

정하게 증가하다가 steady-state 상태에 도달하게 될 것이다. 그러나 제어기들의 구현상의 문제로 플로우 요청 메시지의 Aggregated 된 링크의 대역폭이 증가함에 따라 Thread 간의 경쟁 문제, 제어기 OS의 작업 스케쥴링 문제 등으로 인하여 성능의 저하가 나타날 수 있기 때문에 링크의 대역폭과 제어기의 쓰레드 개수에 변화를 주어가며 성능을 측정하였다.

4.2 성능 측정 결과

4.1에서 언급된 바와 같이 NOX 제어기, FloodLight 제어기, MuL 제어기의 성능을 동일한 환경에서 각각 측정한 후 결과를 비교하였다.

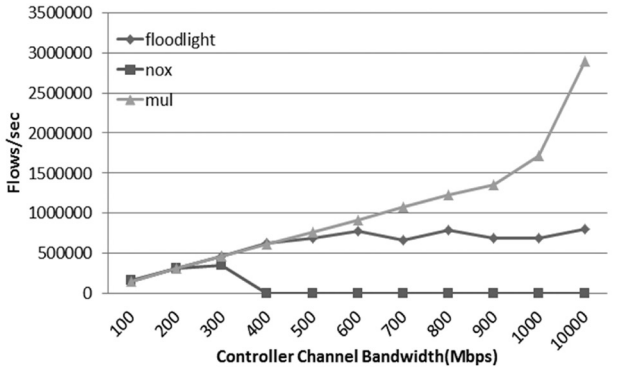


그림 5. 8 Threads 와 32 스위치 환경 성능 평가

〈그림 5〉에서는 사용된 가상 스위치의 개수 32개, 쓰레드의 수를 8개로 지정한 cbench에서의 플로우 처리 속도이다. MuL 제어기는 제어기와 Aggregated 된 링크의 대역폭 10Gbps 에서 최대 3,000,000 responses/sec를 보여주고 있다. 그에 반해 NOX 제어기는 대역폭 300Mbps 환경에서 최대 500,000 responses/sec 나타내고, 오히려 링크의 대역폭이 증가해 감에 따라 성능이 나빠지는 경향을 보이고 있다. 이와 같은 이유는 링크의 대역폭이 증가되어 감에 따라 처리해야 하는 Flow의 수는 많아지는데, NOX 의 어플리케이션 레벨에서 이를 처리하지 못하게 되고 이에 의해 전체적인 성능에 영향을 미치게 되어 성능 저하가 발생하게 된다. 반면에 FloodLight은 NOX에 비해서는 링크의 대역폭에 따라 안정적인 성능을 보여주고는 있지만 최대 750,000 responses/sec 정도를 보여주고 있고 링크의 대역폭이 500Mbps 만 되어도 steady-state로 접어 든다. 이를 MuL 제어기와 비교할때 MuL 제어기가 링크의 대역폭 10Gbps 환경에서 최대 4배 정도의 성능이 증가됨을 확인할 수 있다. 만약 Floodlight을 프로그래밍 관점에서의 편리성으로 인하여 사용해야 할 필요성이 있을 경우는 스위치 레벨로 부터 올라오는 Aggregated 된 링크의 대역폭이 500Mbps 이하로 작은 네트워크 환경에서 사용하기에 적합할 것으로 보인다. 이에 반해 Large scale 의 성능이 중요한 이슈가 되는 환경에서는 MuL 제어기의 사용이 적합하다.

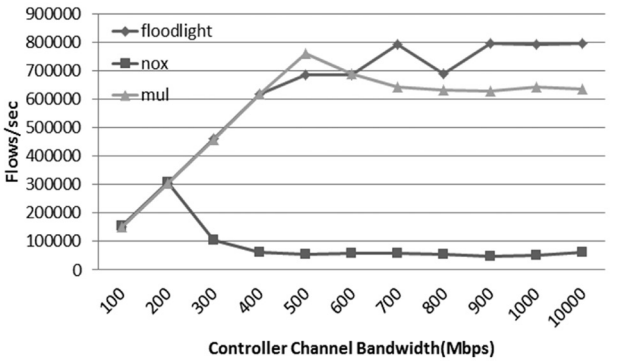


그림 6. 1 Thread 와 32 스위치 환경 성능 평가

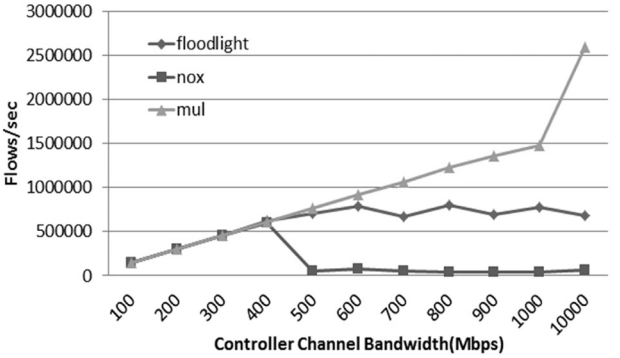


그림 7. 4 Threads 와 32 스위치 환경 성능 평가

〈그림 7〉은 사용된 가상 스위치의 개수 32개, 쓰레드의 수를 4개로 지정한 cbench에서의 플로우 처리 속도인데 〈그림 5〉의 그래프와 유사한 경향을 보이고 있다. 〈그림 6〉은 쓰레드가 1개인 매우 저사양의 환경에서의 제어기의 성능 비교로써 이와 같은 저사양 환경에서도 NOX 제어기의 사용은 불가능 할 것으로 보이고 이에 반해 Floodlight 과 MuL 제어기는 어느 정도 비슷한 성능을 보이기 때문에 개발자 또는 운영자의 선호도에 따라 선택하는 것이 적합할 것으로 보인다.

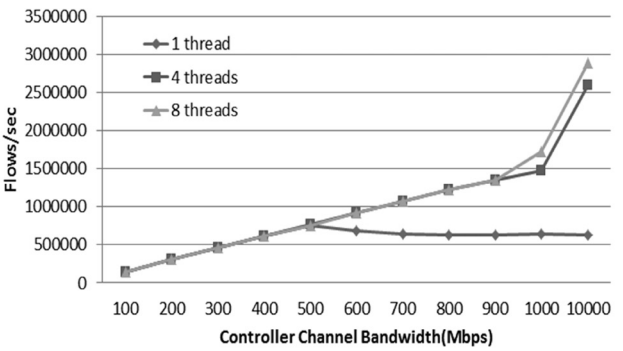


그림 8. MuL 제어기의 Thread 수 별 성능 평가

〈그림 8〉은 MuL 제어기에 한정지어 링크의 대역폭과 쓰레드의 수에 따른 성능을 비교한 것으로써 링크의 대역폭이 500Mbps 까지는 어느 정도 모두 비슷하게 나오다가 그 이후에는 쓰레드의 수에 의해 큰 영향을 받는 것을 볼 수 있다. 하지만 쓰레드 4개 이상에서는 별 차이가 없는 것으로 보아 성능에 초점을 맞춘 SDN 네트워크에서 MuL 제어기를 사용할 시도 링크의 대역폭이 500Mbps 까지는 1개의 쓰레드를 사용하는 것으로 충분하고 500Mbps 이상의 환경에서는 쓰레드를 4개로 증가시켜 주는 것이 자원 사용의 효율성 측면에서 바람직하다.

5. MuL 제어기 적용 사례[16]

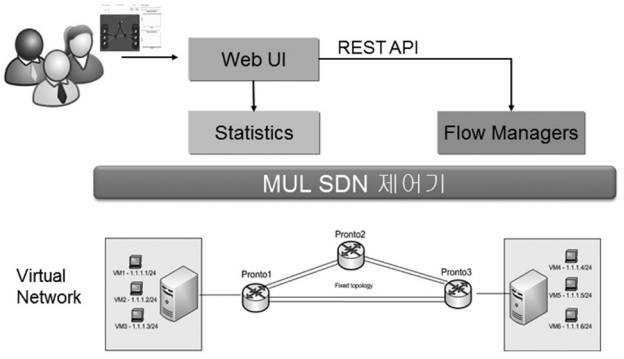


그림 9. MuL 제어기 NBAPI 기반 프로그래밍

MuL 제어기는 프로세스의 성능 최적화를 하였을 뿐 아니라, NBAPI 기반의 인터페이스를 통하여 SOA(Service Oriented Architecture) 기반의 프로그래밍 모델을 또한 제시한다. 〈그림 9〉와 〈그림 10〉은 그 일례로써 NBAPI 를 기반으로 한 모니터링 시스템을 일례로써 보여준다. 본 시스템의 목적은 SDN 운영자가 네트워크 상태를 모니터링 하면서 네트워크의 상태에 따라 On-demand 로 네트워크를 매니지 하는 것을 보여준다. 이를 위해 MuL 제어기는 Flow 를 매니지 하기 위한 어플리케이션과 Flow 의 통계 정보를 제공하기 위한 어플리케이션이 돌아가고 NBAPI 는 각 어플리케이션에 의해 제공되는 정보와 원격 제어 기능을 웹 서비스 기반의 API로 제공함으로써 손쉽게 웹 기반의 모니터링 시스템을 구현 하는 것이 가능함을 보이고 있다.

〈그림 10〉은 위와 같은 기능을 활용하여 제작된 웹페이지 서비스의 일례로써, 웹 페이지의 각 레이아웃 부분들의 기능이 MuL 제어기 기반의 NBAPI 웹 서비스와 연동되어 손 쉽게 서비스를 구축하는 것이 가능함을 보이고 있다.

이 뿐만이 아니라, MuL 제어기는 현재 Openstack[16] 이라는 클라우드 매니지먼트 시스템의 Quantum 이라는 네트워크 시스템과의 연동을 통한 클라우드 환경에서의 네트워크 서비스

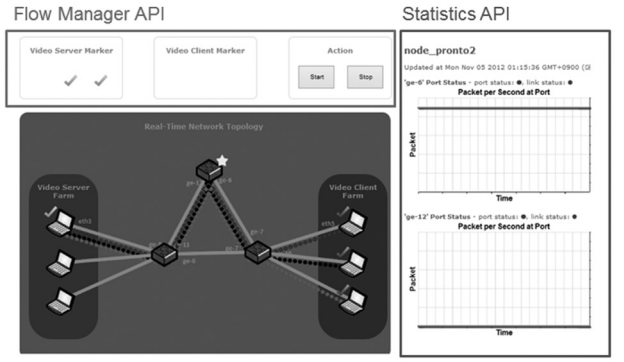


그림 10. MuL 제어기 기반의 모니터링 시스템

를 제공하고 있고 클라우드 환경에서의 L4 이상의 레이어 에서의 로드 밸런싱 방화벽과 같은 서비스를 구현 중에 있다. 이는 추후 ONS 2013(Open Networking Summit)[17] 에서 시연을 하기 위하여 연구 및 구현 중에 있다.

Ⅲ. 결 론

본고에서는 네트워크의 새로운 패러다임을 가져온 SDN 아키텍처에 대한 개념과 정의를 살펴보고, 국내에서 최초로 개발된 SDN의 중요한 구성요소 중 하나인 MuL 제어기 플랫폼의 구조에 관하여 살펴보았다. 그리고 실제 환경에서 제어기와 Aggregated 된 스위치의 링크 대역폭을 조절해 가면서 각 제어기의 성능 비교 분석을 통해 각 제어기의 특징을 파악하고 링크의 대역폭과 쓰레드의 수가 각 제어기에 미치는 영향을 분석하였다. 그 결과 MuL 제어기가 기존 제어기 대비 4배 정도 성능의 향상을 확인할 수 있고, 기존 제어기 대비 Large Scale 의 네트워크 환경에 더 적합함을 확인할 수 있었다.

이와 더불어 본고에서는 MuL 제어기의 실제 적용 사례를 보였다. MuL 제어기의 NBAPI 기능을 활용한 네트워크 매니지먼트 시스템을 일례로 들었고, SOA 기반의 인터페이스를 통한 MuL 제어기 상의 어플리케이션에 의해 제공되는 정보 및 접근 권한을 사용자에게 제공함으로써 더욱더 다양한 서비스를 제공하는 것이 가능함을 보였다. 이와 같이 앞으로는 MuL 제어기를 활용하여 다양한 서비스를 제공하기 위한 연구가 지속되어야 할 것이다.

참 고 문 헌

[1] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In HotNets, pages 1-6, 2010.

[2] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks."ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, and N. McKeown. NOX: Towards an Operating System for Networks. In ACM SIGCOMM CCR, July 2008.

[4] Floodlight : floodlight.openflowhub.org

[5] A. Voellmy, J. Kim, and N. Feamster. Procera: A Language for High-Level Reactive Network Control. InProceedings of ACM Sigcomm HotSDN Workshop, 2012.

[6] Urs Hoelzle, Openflw @ Google. Open Networking Summit 2012.

[7] http://www.bell-labs.com/user/erranlli/publications/CellSDN-TR12.pdf

[8] A. Tootoonchian, S. Gorbunov, Y. Ganjali, and M. Casado, "On controller performance in software-de?ned networks," in USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 2012.

[9] S. Hassas Yeganeh and Y. Ganjali, On Scalability of Software-Defined Networking., Communications Magazine, IEEE. Feb.

[10] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications", Proc. HotSDN '12 Wksp., 2012, pp. 19-24.

[11] A. R. Curtis et al., "DevoFlow: Scaling Flow Management for High-Performance Networks," Proc. ACM SIGCOMM '11, 2011, pp. 254-265.

[12] MuL Controller : http://sourceforge.net/projects/mul

[13] 박준우, 박성용 "Performance Comparison Study on High-speed Openflow Controller MuL: an Openflow Controller designed for Performance, Reliability, and Flexibility", 2012년 추계 학술대회.

[14] Rob Sherwood and Kok-Kiong Yap, Cbench: an Open-Flow Controller Benchmarker. (http://www.openflow.org/wk/index.php/Oflops).

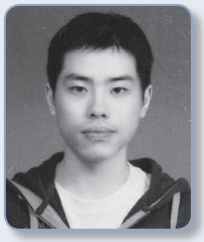
[15] TC tool : http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2

[16] Openstack : http://www.openstack.org

[17] ONS 2013 : http://opennetsummit.org

[18] Kaoru Yano, "SDN is Ready to go!", NEC Corporation. Open Networking Summit 2012.(http://opennetsummit.org/archives/apr12/yano-tue-keynote.pdf)

약 력



허 찬

2006년~현재 한국기술교육대학교 컴퓨터공학부 재학

관심분야: UX, 모바일 어플리케이션, 네트워크, 접근성

박 성 용

1991년 연세대 전자공학 학사

1996년 U. of Illinois at Urbana-Champaign, ECE 석사

2006년 U. of Illinois at Urbana-Champaign, ECE, 박사

2001년~2004년 Cisco Systems Senior 소프트웨어 엔지니어

2004년~2007년 삼성전자 수석엔지니어

2007년~현재: 연세대 전기전자공학과 연구 교수