SDN 관련 연구의 간략한 역사 및 다양한 연구 주제들

국민대학교 ▌이 상 환*

1. 서 론

최근 들어 인터넷에서는 다양한 새로운 서비스들이 선보이고 있다. 페이스북(Facebook)이나 트위터(Twitter) 등의 Social Network Service가 등장하였고, 클라우드 컴퓨팅을 이용한 서비스도 등장하고 있다. 이러한 서비스들의 가장 큰 특징 중 하나는 바로 중앙 집중식 서비스라는 점이다. 즉 모든 서비스가 하나의 서비스 제공자에 의해 제공되고 있다. 이러한 방식의 서비스를 효과적으로 제공하기 위해 각 서비스 별로 다양한 형태의 데이터 센터 네트워크를 구성하여 서비스를 제공하고 있다. 이러한 데이터 센터 관련 연구는 SDN (Software Defined Network, 소프트웨어 정의네트워크)의 등장으로 새로운 전기를 맞이하게 된다. 특히 OpenFlow[1]로 대표되는 표준화된 스위치 인터페이스가 제시되고, NOX [2]와 같은 OpenFlow 컨트롤러들이 제시되어 비약적인 발전을 맞게 된다. 이러한 SDN 개념은 실제로 많이 적용되고 있는데, 예를 들면 구글의 경우 Map/Reduce를 사용하는 자체 데이터 센터 네트워크를 OpenFlow를 기반으로 하는 SDN으로 구성하였다[3]. 이렇게 새로이 등장하는 서비스들은 많은 경우 데이터 센터 네트워크를 통해 서비스를 제공하고, 이 데이터 센터 네트워크를 SDN을 통해 효율적으로 관리하고자 하는 움직임이 최근 2-3년 사이에 지속적으로 진행되고 있다. 아래에서는 최근 새로운 네트워크 패러다임으로 주목받고 있는 SDN에 대해 기술하는데, 우선 SDN이 등장하게 된 배경 및 현재까지의 대표적인 연구 진행 과정을 살펴보고, SDN을 활용한 다양한 연구 주제에 대해 살펴보도록 한다.

2. 소프트웨어 정의 네트워크 (Software Defined Network)

SDN의 연구가 진행되는 과정을 살펴보면, 우선 중앙집중식 컨트롤이라는 개념이 먼저 도입되고, 이를 표준화된 인터페이스로 제공하고자 하는 노력이 이어졌음을 알 수 있다. 이러한 전개 과정을 아래에서 간략하게 설명한다.

2.1 중앙집중식 컨트롤 플레인(Control Plane)의 시작

소프트웨어정의 네트워크(SDN)는 [4]에서 처음으로 일반 IP 라우터에서 Inter-domain 라우팅을 분리하여 Inter-domain 라우팅 기능을 수행하는 독립적인 플랫폼인 Routing Control Platform(RCP)을 제안한 것에서 시작되었다. 현재는 한 도메인 내의 모든 IP라우터가 Inter-domain 라우팅을 담당하는 BGP 기능을 수행하여 독립적으로 경로를 설정하게 하였는데, 이는 각 IP 라우터가 가지는 정보의 불일치 등으로 인해 많은 문제를 일으키고 있기 때문에 RCP라는 독립적인 서버가 도메인 전체에 대한 하나의 일관된 상태를 바탕으로 경로를 설정하여 각 IP 라우터에 해당 라우팅 정보를 전송하여 IP 라우터는 단순하게 포워딩만을 수행하도록 하였다. 이 구조에서 가장 큰 장점 중 하나는 도메인 전체에 대한 다양한 정책(Policy)을 일관성 있게 수행할 수 있다는 점이다. 즉 특정 policy(예를 들면, 특정 IP로부터 오는 패킷은 모두 누락시킬 것)등을 기존에는 각 ingress 라우터의 설정을 일일이 수정해 주었으나, RCP를 이용할 경우 RCP내의 policy만 변경해주면, RCP가 알아서 각 IP 라우터에 그 policy가 구현되도록 라우터의 설정을 변경해준다. 이러한 방식은 매우 다양한 장점을 가지고 있지만, 이 방식의 가장 큰 문제점은 확장성(scalability)이라고 할 수 있다. [5]에서는 실제로 RCP 시스템을 구현하여 기존의 iBGP 구조의 라우팅 정보 수렴 속도와 비슷한 속도를 가짐을 보였다. 하지만 이러한 확장성의 문제는 RCP의 철학을 이어받은 SDN에서도 지속적으로 문제로 제기되고 있다. 참고로 필자가 이 논문을 처음 접하던 때에는 아직 SDN에 대한 논의가 시작되지 않은 시점이었기 때문에 중앙집중적인 방식의 Routing이라는 개념은 인터

* 종신회원

넷이 처음 등장하게 된 시점의 문제의식과는 완전히 반대의 개념이라서 별 효용성이 없을 것이라고 판단했다.

RCP의 문제의식을 더 발전시켜, [6]에서는 네트워크에서 제어와 관리 부분을 포워딩에서 완전하게 분리하는 4D 접근 방식을 제안하였다. 4D 구조는 4개의 평면을 가지는데, 이는 Decision, Dissemination, Discovery, Data의 네 가지이다. 즉 라우팅의 결정, 그 정보의 배포, 네트워크 상태 발견, 데이터의 네 부분으로 분리하고, IP 라우터는 데이터 평면만을 담당하여 데이터를 포워딩하는데 전념하고, 나머지 기능을 라우터로 분리하는데 그 의의가 있다. 기존의 RCP는 iBGP 등을 사용하여 이전 네트워크 구조와의 호환성을 유지하고 있었는데, 이 4D 구조는 기존 프로토콜에서 완전히 벗어난 새로운 구조를 주장하는 clean slate 접근 방식으로 보다 근본적인 연구 과제를 제시하였다.

이러한 중앙집중적인 Control Plane의 개념은 이후 다른 연구들을 통해 점점 확장되어 SDN으로 진화하게 된다. 우선 [7]에서는 RCP의 개념을 한 기관의 내부 네트워크(Intra-Domain Network)에 적용하는 Ethane이라는 시스템을 제시하였다. 즉 한 기관의 내부 네트워크는 스위치들로 연결된 하나의 도메인을 구성하는데, 이 스위치들의 기능을 대폭 축소하여 단지 패킷을 포워딩하는 역할만 하게하고, 중앙 컨트롤러(Centralized Controller)를 두어, 이 Ethane 컨트롤러가 네트워크 전체에 대한 정책(policy)을 담당하고, 이 policy를 각 스위치에 적용하도록 하는데, 그 적용 방식을 스위치의 flow table의 entry를 중앙 집중적인 방식으로 컨트롤러가 수정하는 것이다. 구체적으로 살펴보면 새로운 flow의 패킷이 발생하면 이 패킷을 처음 받은 스위치는 이 패킷을 Ethane 컨트롤러에 전송한다. Ethane 컨트롤러는 이 패킷에 해당하는 flow 테이블의 entry가 존재하는지 검사하여, 존재하지 않으면, policy검사를 거친 후 경로를 생성한 후, 이 경로에 해당하는 각 스위치별 flow entry를 생성하고, 그 내용을 해당 스위치들에 전송하여 스위치들이 이 새로운 flow에 대한 flow entry를 가지도록 한다. 이러한 방식으로 각 flow에 대한 policy가 쉽게 구현되는 것이다. 기존의 Ethernet 스위치는 ARP 등의 메시지로 인해 flooding 방식으로 패킷이 발생하기 때문에 많은 flow가 생성되어 flow table의 크기가 커지지만, Ethane 스위치의 경우에는 ARP 등은 Ethane 컨트롤러에서 담당하기 때문에 실제적인 flow만 flow table에 기록되게 되어 flow table 크기가 작아도 된다. 이 연구진은 이러한 방식을 300여개의 호스트가 있는 스탠포드 대학 구내에서 실험하여 그 유효성을 입증하였다. Ethane도 RCP와 마찬가지 개념을 적용한 것인데, 이 논문에서는 2 계층 기기인 스위치를 대상으로 이 기능을 적용한 것이다. 또한 이러한 policy의 표현을 쉽게 하도록 POL-ETH라는 policy 언어도 제시하였다.

2.2 컨트롤러와 스위치 인터페이스의 표준화 – 오픈플로우의 등장

[7]에서 스위치 내의 flow table을 조작하여 다양한 flow 레벨의 policy의 구현이 가능함을 보였는데, 스위치의 flow table을 수정하기 위해서는 스위치의 firmware나 소프트웨어를 수정해야 할 필요성이 발생한다. 이는 스위치의 종류가 매우 다양하다는 상황을 고려하면 간단한 문제가 아니다. 이에 따라 스위치에 대한 보다 다양한 작업을 표준화된 인터페이스로 정의할 필요성이 대두되어, OpenFlow라는 스위치 프로토콜이 제안되었다[1]. 그림 1은 OpenFlow의 기본 구조를 보여준다. OpenFlow는 표준화된 인터페이스를 통해 스위치의 flow entry를 추가하거나 삭제할 수 있게 하였다. 즉 모든 스위치들이 OpenFlow 기능을 내장하고 있다면, 스위치의 펌웨어 수정 없이 Ethane과 같은 네트워크 구조를 쉽게 생성할 수 있게 된다. 구체적으로 각 OpenFlow 스위치는 컨트롤러와 SSL을 통해 연결을 설정하여 컨트롤러와의 통신을 수행하고, flow 테이블을 관리한다. OpenFlow의 각 flow entry는 간단한 action이 포함되어 있는데, 패킷을 특정 포트로 포워딩하는 것, 패킷을 캡슐화하여 컨트롤러에 전송하는 것, 그리고 패킷을 폐기(Drop) 시키는 것 등 3가지 action이 있다. 이러한 기능을 통해 쉽게 관리자가 원하는 네트워크 구조를 구현할 수 있게 된다.

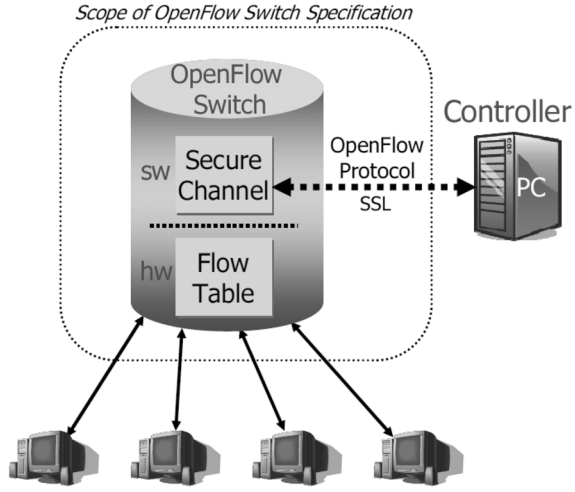

그림 1 OpenFlow 관련 컴포넌트[1]

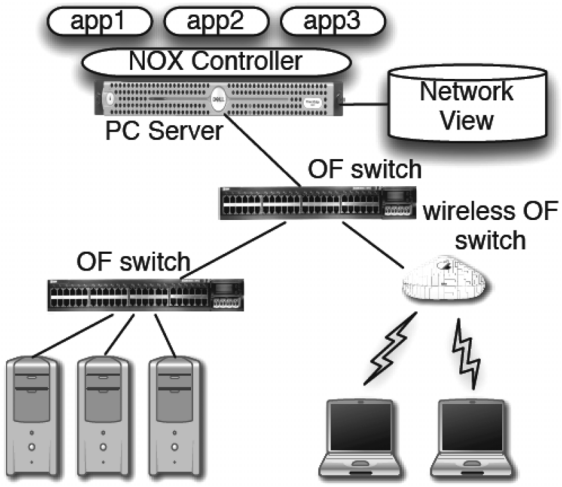

그림 2 NOX 기반 네트워크의 구조 [2]

OpenFlow의 등장은 네트워크 관리가 표준화된 방식으로 쉽게 이루어질 수 있다는 점을 보여주었는데, [2]에서는 이러한 표준화된 스위치를 바탕으로 네트워크 운영 체제(Network Operating System)이라는 새로운 개념을 제시하였다. 즉 어떤 특정한 형태의 네트워크 라우팅은 네트워크 운영체제에서 수행되는 하나의 응용으로 생각하고, 네트워크 운영체제 위에서 복수개의 응용이 수행되는 개념을 제시한 것이다. 이에 따라 여러 가지 다른 형태의 네트워크나 응용이 하나의 물리적인 네트워크 위에 존재하게 된다. 예를 들면, 각 사용자별로 VLAN을 사용할 수 있는 기능을 하나의 응용이 수행하고, 외부의 포트 스캔 공격을 탐지하는 기능을 또 다른 응용으로 구현하는 것이 가능하다. 이 논문 [2]에서 제시한 네트워크 운영체제의 이름이 NOX이다. 그림 2는 NOX 컨트롤러와 OpenFlow를 사용한 네트워크의 구조를 보여주고 있다. NOX는 각 응용 개발자가 NOX를 사용할 수 있는 프로그래밍 인터페이스를 제시하여, 쉽게 특정 응용을 구현할 수 있도록 하였다. 참고로 앞에서 언급한 Ethane도 NOX와 OpenFlow를 이용하면 쉽게 구현 가능하다.

2.3 SDN 컨트롤러의 확장성

이후 NOX를 활용한 응용들이 많이 제시되었는데, 앞에서도 언급했다시피 중앙 집중적 방식은 항상 각 flow의 첫 번째 패킷이 캡슐화되어 컨트롤러로 전송된 후 컨트롤러가 새로운 flow entry를 계산해야 하기 때문에 확장성에 문제가 있다. DIFANE은 NOX 컨트롤러의 이러한 확장성 문제를 해결하기 위해 Authoritative 스위치라는 좀 더 고급의 스위치를 도입하여, 이 복수의 Authoritative 스위치들이 다양한 policy 규칙의 일부를 저장하도록 하였다[8]. NOX 컨트롤러는 전체 규칙을 분할하여 적절하게 각 Authoritative 스위치에 분배하는 역할을 하고, Ingress 스위치는 각 flow의 첫 번째 패킷을 NOX 컨트롤러에 보내는 대신에 그 패킷의 헤더에 해당하는 Authoritative 스위치에 전송하게 하여 NOX 컨트롤러의 부하를 분산하였다. 또한 이 논문은 규칙을 효율적으로 분할하는 알고리즘을 제시하여, 분할된 규칙의 수를 최소한으로 줄이도록 하였다. [9]도 컨트롤러의 기능을 스위치에 양도하는 방법을 통해 컨트롤러의 flow 관리의 부하를 줄이는 기법을 제시하였다. 우선 OpenFlow 기반의 SDN에서 부하가 발생하는 부분을 파악해 내고, 이를 해결하는데, 구체적으로 살펴보면, 우선 wildcard를 사용하는 flow entry를 exact match를 사용하도록 확장하여, 그 내용을 포워딩 테이블에 저장하도록 하고, wildcard를 저장할 수 있는 TCAM에 저장되는 flow entry의 수를 줄였다. 그리고 링크 실패가 발생했을 때 컨트롤러에 보고하면서 패킷 전송을 지속하기 위해 지역적인 대응(Local Rerouting)을 도입하였다. 또한 각 스위치로부터 통계 정보를 수집할 때 샘플링이나 추정 카운터(Approximate Counter)를 사용하여 컨트롤러로의 메시지 부하를 줄였다. [10]도 NOX 컨트롤러의 부하를 줄이기 위한 방식을 제시하였다. 이 논문에서 제시한 시스템은 ONIX이고, 매우 큰 규모로 실제 동작하는 네트워크에서 분산된 제어를 담당하는 시스템이다. 구체적으로 복수 개의 ONIX 서버를 실행하고, 네트워크에 대한 정보는 NIB (Network Information Base)에 저장한 후, 이를 복수의 ONIX 서버들이 실시간으로 공유하도록 하였다. 각 스위치는 이 ONIX 서버들 중 하나를 컨트롤러로 사용하게 되어, 단일 컨트롤러의 부하를 복수의 ONIX 서버에 분산하도록 하였다.

2.4 네트워크 패브릭(Network Fabric)

SDN에서 추구하는 네트워크 제어와 데이터 전송의 분리는 필연적으로 각 스위치의 기능을 단순하게 만들게 되었다. 즉 간단하게 flow 테이블에 의해 데이터를 전송하는 작업만 제대로 수행하면 나머지 라우팅 결정 등 제어에 관한 부분은 중앙 컨트롤러가 담당하도록 하게 된다. 제어에 관한 부분의 시작은 Ingress Edge 스위치에서 새로운 flow의 패킷을 발견하고 이를 중앙 컨트롤러에 보내주는 것인데, 이 기능은 내부의 다른 스위치는 보유할 필요가 없는 기능이다. [11]은 이러한 문제의식에서 출발하여, 네트워크의 중앙 컨트롤러를 분리하여, Edge컨트롤러와 fabric 컨트롤러라는 두 가지 컨트롤러를 두도록 하였다. 그리하여 Edge 컨트롤러는 제어 및 flow 생성 등에 관한 부분만을 담당하

게 되고, fabric 컨트롤러는 각 스위치의 데이터 전송에 관한 부분만을 전담하도록 하였다. 즉 이러한 분리를 통해 fabric 컨트롤 기능만 따로 개선하거나 다른 구조를 사용할 수도 있고, fabric은 상관없이 edge 컨트롤러의 기능을 업그레이드 할 수 있는 유연한 구조를 확립하게 된다. 아직 이 구조에 대한 검증은 없지만, 향후 많은 연구가 필요한 연구 주제라고 할 수 있겠다.

3. SDN 관련 다양한 연구 주제들

OpenFlow의 등장으로 SDN에 대한 기본적인 연구 환경이 갖춰졌다고 생각된다. 이를 통해 다양한 분야에서 연구를 진행하여 SDN 기술을 향상시키게 되는데, 아래에서는 다양한 관련 연구 내용을 살펴본다.

3.1 OpenFlow 및 OpenFlow 컨트롤러의 활용

NOX의 개념을 동적인 접근 제어에 활용하는 방법을 제시한 시스템 중 하나가 Resonance이다[12]. 네트워크 내의 다양한 시스템(DHCP 서버, VLAN Management Policy Server, 방화벽)이 보내오는 정보를 Resonance가 종합하여 policy와 비교한 후, 이에 대응하는 flow entry를 생성하여 각 스위치에 적용시키는 방법이다. 이 방법을 Georgia Tech의 네트워크 접근 제어 시스템에 적용하여 그 효용성을 입증하였다. 역시 이 방식도 확장성과 반응 시간이 가장 큰 문제가 되는데, 캠퍼스 네트워크에 적용하는 것은 큰 무리가 없음을 주장하였다.

네트워크 제어가 중앙에서 이루어지는 SDN 구조의 특성을 활용하여 실제 동작중인 네트워크(Production Network)를 테스트베드로 사용할 수 있는 시스템이 제안되었다[13]. 이 논문에서 제안하는 FlowVisor는 복수개의 컨트롤러를 지원하며, 이 다양한 컨트롤러 별로 자원을 분배 한 후 각 컨트롤러가 flow의 생성이나 삭제를 요구할 때 미리 분배된 자원의 용량에 맞는지를 검사하여 그 요청에 대응하도록 하였다. 각 컨트롤러의 입장에서는 FlowVisor가 OpenFlow 플랫폼으로 인식되고, 각 스위치의 입장에서 FlowVisor는 컨트롤러로 인식이 되어 기존의 OpenFlow 기반의 SDN에서 아무 무리 없이 동작할 수 있도록 하였다. 이 FlowVisor가 지원하는 복수의 컨트롤러 중 하나는 Production 네트워크를 담당하는 컨트롤러로 하고, 테스트베드로 실험하고자 하는 컨트롤러는 제한된 자원을 할당 받아 같은 물리 네트워크를 사용하도록 한다. 이로써 새로 개발되는 컨트롤러가 실제 동작 네트워크 환경에서 성능을 시험받을 수 있게 된다. FlowVisor는 각 컨트롤러가 요청하는 CPU, 메모리, Bandwidth의 사용량이 미리 배분된 사용량을 초과하지 않도록 하는 다양한 기법을 제공한다. 하지만 아직 정확하게 독립적인 네트워크처럼 동작하지는 않는다.

3.2 SDN 응용 프로그램 작성 및 검증 기술

SDN은 네트워크의 관리를 매우 쉽게 해줄 수 있지만, 실제로 SDN 응용을 올바로 작성하는 것은 다른 문제이다. [14]는 프로그래머들이 NOX 프로그램을 쉽게 작성할 수 있는 Frenetic이라는 새로운 언어를 제공하였다. 즉 NOX에서 제공하는 Low Level의 프리미티브 대신 High Level 언어를 사용하여 프로그램을 할 수 있도록 했는데, 이렇게 프로그램된 내용을 Frenetic Run Time 시스템이 NOX의 문법으로 번역하여 NOX 명령을 내림으로써 실행이 되도록 하는 것이다. 스위치에서 보내는 이벤트 메시지는 NOX가 받아서 Runtime에 넘겨주고, 이 Runtime이 다시 Frenetic에 정의된 이벤트 핸들러로 전송하여 처리하도록 하였다. 이러한 고차원 언어는 사용하는 SDN 컨트롤러의 종류에 상관없이 Runtime만 그 컨트롤러를 지원하게 되면 정상적으로 동작하도록 되어 있어서 매우 이식성이 높아진다고 할 수 있다.

이 Frenetic은 프로그래머가 프로그램을 하는데 좀 더 쉽게 할 수 있도록 도와주는 고급 언어라고 할 수 있는데, 프로그램이라는 것이 사람이 하는 것이므로 프로그램 자체에 버그가 발생할 수 있다. 즉 SDN 컨트롤러 프로그램 자체에 문제가 있다면 원하는 네트워크를 얻을 수 없을 것이다. 이에 따라 SDN 컨트롤러가 제대로 동작하는 지를 검증하고자 하는 연구가 진행되었는데, [15]는 SDN용 Debugger를 제시하였다. 예를 들면 어떤 패킷이 어느 스위치에서 폐기되었을 때, 왜 그런 상황이 발생하는지를 알고 싶을 수가 있다. 이를 위해서는 그 패킷이 어느 경로를 통해 전달되어 왔는지를 파악해야 하는데, 이미 그 정보는 사라지고 없게 된다. [15]에서는 이를 해결하기 위해 각 패킷이 하나의 스위치를 방문할 때마다 그 내용을 postcard라는 메시지로 디버거 서버에 전송하여, 디버거가 패킷의 이동 경로를 저장할 수 있도록 한다. 추후 운영자가 특정 패킷이 어디에서 폐기되었는지 등에 관한 질의를 하면, 그 내용을 보고할 수 있고, 어떤 flow entry에 의해 폐기 되었는지를 파악하게 해준다. 이를 바탕으로 컨트롤러를 수정할 수 있게 된다. 하지만, 패킷 별로 매 스위치에서 postcard 메시지를 발생시키는 것은 많은 오버헤드를 발생시킨다. 따라서 보다 효율적인 디버깅 기능이 필요하다.

SDN에 대한 디버깅 뿐만 아니라, 실제로 SDN이 개발될 때 실제 네트워크에 설치하지 않고, 테스트를 해 볼 수 있는 시뮬레이션 환경이 필요한데, [16]에서는 Mininet이라는 VM 기반의 SDN 개발용 시뮬레이터를 제시하였다. 패킷 레벨 시뮬레이터인 NS2를 이용하는 것은 노드의 수에 따른 제한이 있기 때문에 SDN을 테스트하는데 사용하기는 곤란하여, 보다 단순하면서 SDN의 기본 기능을 테스트 할 수 있는 환경으로 Mininet을 제시한 것이다. Mininet은 토폴로지 구성을 자유롭게 할 수 있고, 각 스위치의 종류를 다양하게 선택할 수 있게 하여 매우 실제적인 환경에서 컨트롤러를 테스트 해 볼 수 있게 한다. 이와 비슷하게 [17]에서는 다양한 OpenFlow 스위치를 평가할 수 있는 OFLOPS를 제안하였다. 즉 OFLOPS에서 패킷을 생성하여 스위치로 전송하여 얼마나 잘 처리하는지를 평가할 수 있고, 컨트롤러에서 발생하는 메시지들도 OFLOPS에서 생성하여 스위치로 전송한 후 얼마나 빨리 처리할 수 있는지를 평가할 수 있다. 즉 OFLOPS에서 데이터 트래픽과 컨트롤 트래픽을 생성하여 스위치로 전송한 후 스위치의 처리 상태를 분석할 수 있다는 것이다. 이 논문에서는 OFLOPS의 구조를 제시하고, 이 OFLOPS를 이용하여 5가지 종류의 OpenFlow 스위치의 성능을 평가하고 그 결과를 공개하였다. OpenFlow 스위치 개발자는 이러한 평가 도구를 사용하여 자신의 스위치 구현을 검증할 수 있게 된다.

SDN에서는 flow entry가 추가되거나 삭제될 때 스위치는 컨트롤러에 메시지를 보내야 한다. 이런 상황에서 해당 flow가 몇 개의 패킷을 그 스위치로 전송했는지를 같이 메시지에 포함시켜서 보내면 링크의 사용률을 추가적인 프로브 메시지 없이 알아 낼 수 있게 된다. [18]에서는 이러한 방식을 제안하였는데, 이 때 발생하는 가장 큰 문제는 flow가 생성될 때는 그 시점에 확실하게 메시지가 컨트롤러로 가지만, flow가 삭제되는 것은 패킷이 일정 시간 동안 그 flow에서 발생하지 않을 때 삭제되기 때문에 flow가 끝나는 시간을 정확하게 알 수는 없다는 점이다. [18]에서는 이러한 문제로 발생하는 사용률 계산 오류가 크지 않다는 점을 실험으로 보였다.

3.3 SDN 의 데이터 센터에의 적용

SDN을 활용하는 연구 중 대표적인 것이 데이터 센터의 성능을 개선하는 부분이다. 이에 관한 연구는 [19]가 처음으로 시도하였다. 이 논문에서는 NOX 컨트롤러를 사용하여 flow entry에 대한 action을 수정함으로써 기존에 제안된 새로운 데이터 센터 구조인 VL2나 Portland 등을 구현할 수 있음을 보였다. 예를 들면 VL2에서 사용하는 VLB(Valiant Load Balancing)을 ECMP (Equal-Cost Multiple Path)나 캡슐화를 사용하지 않고 구현할 수 있음을 보였고, Portland의 PMAC 매핑을 flow entry를 수정함으로써 유지할 수 있음을 보였다. 이렇게 기존의 데이터 센터 구조를 NOX로 그대로 복제할 수 있음을 보였는데, 더 나아가서 데이터 센터의 성능을 향상시키는데도 SDN이 사용될 수 있음을 보인 논문이 [20]이다. 기존의 CLOS 구조[21]를 사용하는 데이터 센터는 ECMP(Equal-Cost Multiple Path)를 활용하여 부하 분산(Load Balancing)을 진행한다. 하지만, 이 부하 분산은 랜덤한 방식으로 이루어지기 때문에 최악의 경우 같은 목적지 호스트를 가진 플로우가 같은 Core 스위치에 할당되는 상황이 발생하여, 다운링크의 부하가 증가하게 된다. 이 논문에서는 이러한 문제를 회피하기 위해 SDN 기술을 사용하는 Hedera라는 시스템을 제시하였다. Hedera에서는 각 플로우별 요구량을 계산하여, 그 중 요구량이 많은 플로우의 Core 스위치를 변경하여 전체적으로 링크의 사용량이 균일하게 유지되도록 하였다. 이를 위해 플로우별 Bandwidth 요구량을 파악하는 알고리즘을 제시하고, 이를 바탕으로 각 플로우별 최적의 경로를 Simulated Annealing을 통해 계산하는 알고리즘을 제시하였다. 플로우의 경로를 변경하는 방법은 OpenFlow 스위치를 이용하여 flow entry를 Hedera 컨트롤러에서 변경할 수 있도록 하였다. 하지만 이 방법은 요구량을 계산하는데 있어서 어느 정도 패킷 전송에 대한 통계를 수집해야 하는데, 대부분의 플로우가 매우 짧은 상황에서는 요구량을 계산하는 것이 어려울 수가 있다[22]. 또한 이 연구들은 플로우별 Bandwidth 계산이 TCP를 기준으로 TCP가 공평하게 자원을 분배할 경우 할당 받게 되는 Bandwidth를 계산하기 때문에 실제 플로우가 사용하는 Bandwidth와는 다를 수가 있다. 따라서 보다 일반적인 플로우별 Bandwidth 사용량을 측정할 수 있는 기법을 연구할 필요가 있다.

SDN 기술을 데이터 센터에 적용하는 또 다른 연구는 [23]인데, 데이터 센터에서 가장 많이 사용하는 빅데이터 응용의 성능을 향상시키기 위해 실시간으로 네트워크를 수정하는 방법을 제시하였다. 구체적으로 살펴보면 빅데이터 응용으로 많이 사용하는 HaDoop 응용에서 데이터의 aggregation시 발생하는 각 Mapper와 Reducer들 간의 네트워크 트래픽을 통해 그 들 간의 가상적인 토폴로지를 파악한 다음, 이 가상적인 토폴로지에 맞게 OpenFlow 스위치의 flow table을 수정하여,

링크별로 오버플로우가 발생하지 않도록 하였다. 즉 데이터의 흐름과 일치되는 가상 토폴로지를 실제 네트워크에서 실시간으로 구현해 주는 것이다. 하지만 실제 물리적인 토폴로지가 미치는 영향에 대해서는 간과하였기 때문에 최상의 성능을 발휘하지 못한다. 또한 트래픽 요구량을 계산하는 것은 상당한 시간이 필요하기 때문에 실시간으로 처리하는 데는 어려움이 있을 수 있다.

3.4 WAN에서의 SDN 적용

사실 많은 SDN응용은 LAN에서 L2 계층의 라우팅을 쉽게 하는데 사용되었다. 즉 하나의 Domain 내에서 중앙집중식 컨트롤을 통해 보다 직접적으로 네트워크를 관리하고자 하는 것이다. 이러한 문제의식은 WAN으로도 확장되었는데, 일반적인 WAN에서의 적용은 아직 시기 상조이고, 하나의 기업에서 구성하고 있는 전 지구적인 규모의 네트워크에서 WAN을 적용하는 사례들이 있다. 예를 들면 구글의 데이터센터가 전 세계 여러 지역에 걸쳐서 존재하고 있는데, 이러한 상황에서 SDN을 활용하여 데이터 센터간의 네트워크 트래픽을 좀 더 효율적으로 관리하는 것이다. [24]에서는 SDN을 활용하여 전지국적인 구글 데이터 센터간의 Traffic Engineering을 효율적으로 달성할 수 있음을 보여주었다.

4. 결 론

SDN은 현재 많은 연구자들의 관심을 받고 있는 연구 분야이다. 본 기고에서는 다른 모든 연구에서와 마찬가지이겠지만, 중앙집중식 컨트롤이라는 개념이 등장하고, 이를 손쉽게 지원할 수 있는 다양한 표준 인터페이스가 제시되고, 이를 개선하거나 활용하기 위한 다양한 연구가 진행됨을 보였다. 네트워크 관련 연구자들이나 학생들이 본 고에서 제시하는 기본 지식을 바탕으로 다양한 관련 연구를 섭렵하여 SDN에 대한 연구를 진행하는데 많은 도움이 될 수 있길 바란다.

참고문헌

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” ACM Computer Communication Review, vol. 38, no. 2, Apr. 2008.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: Towards an operating system for networks,” ACM Computer Communication Review, vol. 38, no. 3, Jul. 2008.

[3] Google, SDN, and Open Source, http://www.openflowhub.org/blog/blog/2012/04/30/google-sdn-and-open-source/

[4] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, “The case for separating routing from routers,” in Proc. of FDNA'04, Portland, OR, Aug. 2004.

[5] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, “ “Design and implementation of a routing control platform,” ” in Proceedings of the 2nd Symposium on Networked systems Design and Implementation(NSDI 2005), Boston, MA, May 2005, pp. 15-28.

[6] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, “A clean slate 4d approach to network control and management,” ACM Computer Communication Review, Vol. 35, no. 5, Oct. 2005.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: Taking control of the enterprise,” in Proceedings of ACM SIGCOMM 2007, Kyoto, Japan, Aug. 2007.

[8] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, “Scalable flow-based networking with difane,” in Proceedings of ACM SIGCOMM 2010, New Delhi, India, Aug. 2010.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” in Proceedings of ACM SIGCOMM 2011, Toronto, Ontario, Canada, Aug. 2011.

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: A distributed control platform for large-scale production networks,” in 9th Symposium on Operating systems Design and Implementation(OSDI'10), Vancouver, BC, Canada, Oct. 2010.

[11] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, “Fabric: A retrospective on evolving sdn,” in Proc. of HotSDN'12, Helsinki, Finland, Aug. 2012, pp. 85-89.

[12] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, “Resonance: Dynamic access control for enterprise net-

works," in Proc. of WREN'09, Barcelona, Spain, Aug. 2009.

[13] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed?" in 9th Symposium on Operating systems Design and Implementation(OSDI'10), Vancouver, BC, Canada, Oct. 2010.

[14] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in Proc. of ICFP'11, Tokyo, Japan, Sep. 2011.

[15] N. Handigol, B. Heller, V. Jeyakumar, D. Mazires, and N. McKeown, "Where is the debugger for my software-dened network?" in Proc. of HotSDN'12, Helsinki, Finland, Aug. 2012, pp. 55-60.

[16] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in Proc. of HotNets'10, Monterey, CA, USA, Oct. 2010.

[17] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An open framework for openflow switch evaluation," in Proc. of Passive and Active Measurement Conference, Vienna, Austria, Mar. 2012.

[18] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, Guofei, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in Proc. of Passive and Active Measurement Conference, Hong Kong, China, Mar. 2013.

[19] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter," in Proc. of HotNets-VIII, New York City, NY, USA, Oct. 2009.

[20] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in Proc. of NSDI'10, San Jose, CA, USA, Apr. 2010.

[21] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in Proceedings of ACM SIGCOMM 2009, Barcelona, Spain, Aug. 2009.

[22] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in Proceedings of the Internet Measurement Conference(IMC), Melbourne, Australia, Nov. 2010.

[23] G. Wang, T. S. E. Ng, and A. Shaikh, "Programming your network at run time for big data applications," in Proc. of HotSDN'12, Helsinki, Finland, Aug. 2012.

[24] 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, "B4: Experience with a Globally-Deployed Software Defined WAN," in Proceedings of ACM SIGCOMM 2013, Hong Kong, China Aug. 2013.

약 력

이 상 환

2005 Ph.D University of Minnesota Twin Cities
2005~2006 IBM T.J. Watson Research Center
2006~현재 국민대학교 컴퓨터 공학부 부교수
관심분야 : P2P, 클라우드, 데이터 센터
E-mail : sanghwan@kookmin.ac.kr