편집위원 : 신명기 (ETRI)

SDN/OpenFlow 전용 언어 및 신뢰성 검증 방법 연구 동향

남기혁, 신명기, 김형준
한국전자통신연구원

요 약

최근 활발히 연구되고 있는 SDN(Software-Defined Networking) 기술과 관련하여, 보다 편리하고 정확한 방법으로 네트워크를 구축하기 위한 SDN 전용 언어와 신뢰성 검증 방법에 대하여 2012년도에 발표된 논문을 중심으로 최신 연구 동향을 분석한다.

Ⅰ. 서 론

OpenFlow는 미국 NSF의 지원으로 Stanford 대학에서 진행된 미래인터넷 연구의 결과물로서, 라우터나 스위치와 같은 네트워크 장치에 종속적인 형태로 제공되던 제어 기능을 논리적으로 중앙 집중적인 형태로 분리하여, 표준 API를 통해 통신하는 구조를 토대로 개발된 기술이다[1]. 현재 OpenFlow는 SDN(Software-Defined Networking)이라는 보다 넓은 개념으로 확장하여, Google, Facebook, Verizon, Cisco 등과 같은 업체 중심으로 새롭게 결성된 표준화 기구인 ONF(Open Networking Foundation)를 통해 현실 적용에 필요한 표준 규격과 기술을 개발하고 있다[2].

SDN과 OpenFlow는 기존 네트워킹 기술과는 다른 패러다임을 추구하고 있으며, 특히 기존의 네트워크 장비의 보조적인 역할을 수행하던 소프트웨어의 역할을 보다 적극적으로 활용하여, 분산 시스템과 운영체제, 데이터베이스와 같은 소프트웨어에서 정립된 개념을 적극적으로 도입한, 새로운 형태의 생태계를 구성하는데 초점을 맞추고 있다.

현재 SDN/OpenFlow 커뮤니티와 표준화 기구에서는 이러한 개념을 구체적으로 실현하기 위한 아키텍처와 프로토콜, 그리고 이를 구성하는 SW 및 HW 규격을 개발하는데 초점을 맞추고 있지만, SDN에서 소프트웨어의 역할을 극대화하여 혁신의 속도를 높일 수 있다는 장점의 이면에는, 이러한 소프트웨어 모듈의 오류가 전체 네트워크에 악영향을 미칠 수 있다는 위험성도 존재한다.

본 논문에서는 SDN/OpenFlow을 위한 프로그래밍 작업의 효율을 높이고, 설정 오류로 인한 문제를 최대한 줄이기 위해 최근 활발히 연구되고 있는 주제를, SDN 전용 언어와 검증 기법이라는 두 가지 관점으로 분석한다.

Ⅱ. 언 어

현재 활발히 진행되고 있는 SDN/OpenFlow 전용 언어 중에서 최근 학회나 ONS[3]를 통해 소개된 것으로 FML, Frenetic, NetCore, Nettle, Procera 등이 있다. FML은 이러한 방향으로 초기 단계에 등장한 대표적인 언어로서, 최근 등장한 다른 언어에 적지 않은 영향을 미쳤다. Frenetic과 NetCore는 Princeton과 Cornell 대학 연구팀을 중심으로, Nettle과 Procera는 Yale과 Georgia Tech, Maryland 대학 연구팀을 중심으로 현재 활발한 연구 개발이 이루어지고 있다.

1. FML

FML은 가장 초기에 제안된 SDN을 위한 로직 기반 언어로서, 엔터프라이즈 네트워크에 적용할 정책(Policy)를 선언적인 방식으로 정의하기 위한 목적으로 개발됐다[4]. 기존의 라우터와 스위치에서는 설정 및 정책 정의를 위해 단순한 형태의 인터페이스만 제공했는데, FML에서는 Logic 및 데이터베이스를 위한 선언적인 언어를 수학 및 논리학 기반으로 정의함으로써, ACL이나 VLAN, 라우팅 정책 등을 보다 쉽고 견고하게 정의할 수 있다.

FML에서 네트워크 정책은 서로 재귀적으로 구성되지 않은, if-then 형태의 규칙으로 표현한다. 여러 규칙간에 순서를 부여하지 않기 때문에, 언어에 배경 지식이 부족한 네트워크 관리자가 필요한 규칙을 나열하는 방식으로 쉽게 작성할 수 있는 반면, 이로 인해 여러 개의 규칙이 서로 상충되거나 중복될 수 있는데, FML에서는 Policy Cascade를 추가하고, conflict resolution 계층을 도입하여 이러한 문제를 해결한다.

FML 논문에서는 접근 제어, QoS, NAT 등과 같은 설정을 FML로 표현하는 예를 소개하고 있으며, NOX[5] 컨트롤러에 기반하여 대략 10,000 라인의 C++와 Python 코드로 프로토타입을 구현하였고, 현재 SNAC 컨트롤러의 Policy 정의 모듈 형태로도 구현되어 있다[30].

2. Frenetic, NetCore

FML이 제안된 이듬해인 2010년부터 Princeton 대학 연구팀에서 Frenetic이라는 OpenFlow 전용 언어를 발표했는데, FML과 마찬가지로 선언적인 형태의 데이터베이스 쿼리 언어에 기반을 두고 있다[6][7].

Frenetic은 NOX 컨트롤러를 위한 애플리케이션 작성 단계에서 부딪히는 여러 가지 한계를 극복하기 위해, 기존 언어에서 제공하지 못한 모듈화와 단일 티어 추상화를 제공하며, 경쟁 조건(race condition)에 대한 처리를 언어 런타임 차원에서 지원하여, NOX 애플리케이션 개발자는 좀 더 원하는 작업을 구성하는데 집중할 수 있다.

Frenetic의 초기 버전은 NOX 컨트롤러의 임베디드 언어 형태로 구현됐으나, 향후 전용 컨트롤러가 보강될 것으로 예상된다.

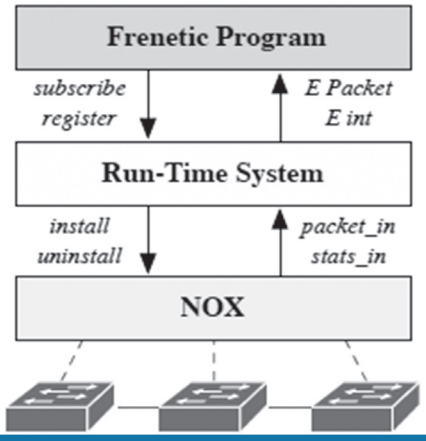


그림 1. Frenetic 구조

Frenetic은 Yampa의 영향을 받아 FRP(Funtional Reactive Programming) 기반 언어로 개발했으며, 컴비네이터 라이브러리 형태로 policy를 관리할 수 있다. 이러한 특성은 뒤에서 소개하는 Nettle과 유사하다[8].

Frenetic 연구팀에서는 NetCore[9]라는 언어 및 이론적 토대도 제안했는데, 사용자 편의성과 모듈화를 비롯한 고수준 언어의 특성을 제공하면서, NOX의 임베디드 언어 형태로 구현된 Frenetic과 달리, 정형 의미론(formal semantics)에 기반을 둔 언어를 제공함으로써, 고수준의 표현을 OpenFlow 규칙 형태로 컴파일하기 기반을 마련하고, 다양한 네트워크 설정 및 정책을 엄격하게 검증하는 기반을 제공한다. 다만 풍부한 이론적인 배경으로 인해, 실제 구현은 Frenetic 보다 시간이 걸릴 것으로 예상되며, Frenetic및 Nettle을 보완하는 형태로 발전할 가능성도 있다.

3. Nettle, Procera

Nettle은 Frenetic과 비슷한 시기에 등장한 FRP 기반 언어로서, discrete과 continuous를 동시에 다룰 수 있다는 점이 특징이다[10]. Nettle에서는 OpenFlow 컨트롤러와 스위치가 서로 주고 받는 이벤트와 메시지를 하나의 스트림으로 추상화한 오브젝트 단위로 취급할 수 있다.

Nettle은 Haskell 언어에 기반을 두고 있으며, 다음과 같은 계층 구조를 이루고 있다.

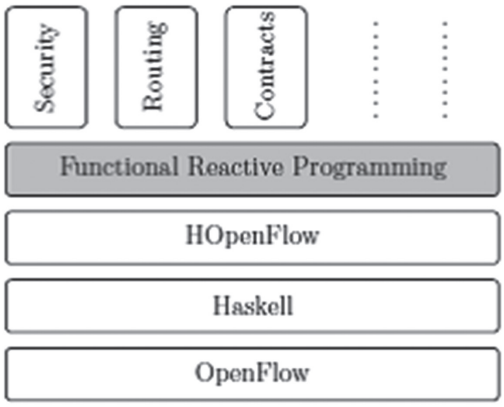


그림 2. Nettle 구조

Nettle 논문에서는 다양한 OpenFlow 응용 예를 보여주고 있는데, 그 중에서도 로드 밸런싱 애플리케이션처럼 연속적인 변화량을 다루는 경우에도 쉽게 적용할 수 있다는 장점이 있다.

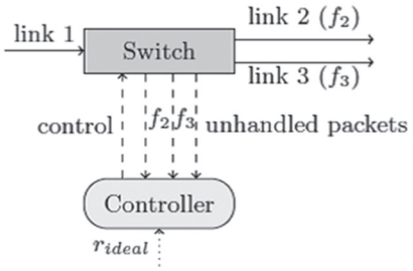


그림 3. 연속 개념 적용 예

Procera[11]는 2012년도에 발표된 SDN 언어로서, Haskell 기반 FRP 방식이라는 점에서 Nettle과 유사하지만, Nettle 보

다 고수준의 언어로 네트워크 설정 및 정책을 정의할 수 있으며, Lithium[12] 컨트롤러 기반으로 동작하고 시제 연산자를 통해 이벤트 히스토리를 다룰 수 있다.

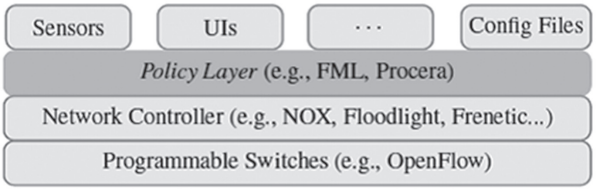


그림 4. Procera 구조

Ⅲ. 검증 및 테스팅

네트워크에 대한 검증과 테스팅 관련 연구는 1990년대부터 꾸준히 발표됐지만, SDN/OpenFlow 문맥에 직접적으로 관련된 연구는 최근에 활발히 진행되고 있다. SDN/OpenFlow가 향후 널리 적용될 경우에 발생할 수 있는 잠재적인 문제점을 보완해준다는 점에서 중요한 연구 주제로 다룰 필요가 있으며, 2012년 NSDI와 SIGCOMM, ONS 등을 통해 소개된 기법을 중심으로 다섯 가지 기법 및 도구를 소개한다.

1. NICE

NICE는 OpenFlow 네트워크의 프로그래밍 오류를 검출하기 위해, 모델 체킹과 symbolic execution 기법을 결합한 도구다[13][14][15].

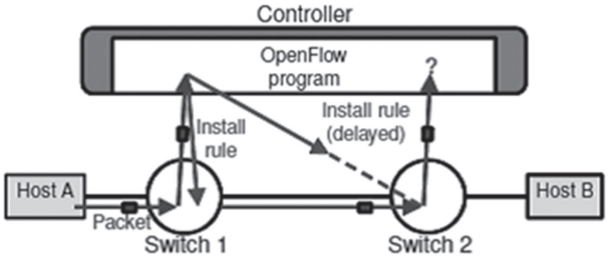


그림 5. OpenFlow 네트워크 설정 과정에서 발생하는 문제

NICE 논문에서는 OpenFlow 네트워크 프로그래밍 과정에서 발생하는 오류로서, 그림 5와 같은 경우를 제시하면서, 단순히 모델 체킹만을 사용할 때 발생하는 상태 공간이 너무 커지는 문제점을 보완하기 위해 symbolic execution 기법을 접목하여, 보다 OpenFlow 네트워크에 적합한 형태의 테스팅 도구를 개발하여, SPIN이나 JPF와 같은 기존 모델 체킹 도구보다 성능을 향상시켰다.

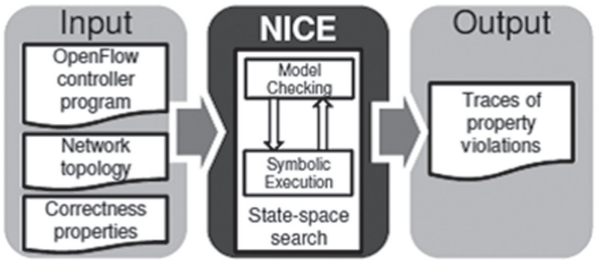


그림 6. NICE 동작 구조

NICE에서는 OpenFlow에 최적화 된 네 가지 휴리스틱을 개발하여, 이벤트 발생 순서에 따른 경우의 수를 크게 줄이고, 잠재적인 오류를 찾는데 좀 더 신경 쓸 수 있도록 구성했다. 또한 주요 검증 속성을 커스터마이즈하여 다양한 상황에 적용할 수 있으며, 이렇게 구성된 속성을 라이브러리 형태로 구축할 수도 있다. 대표적으로 제시된 속성으로는 포워드 루프나 블랙홀의 존재 여부, 경로 도달 가능성, 패킷 손실 여부 등을 제시했다.

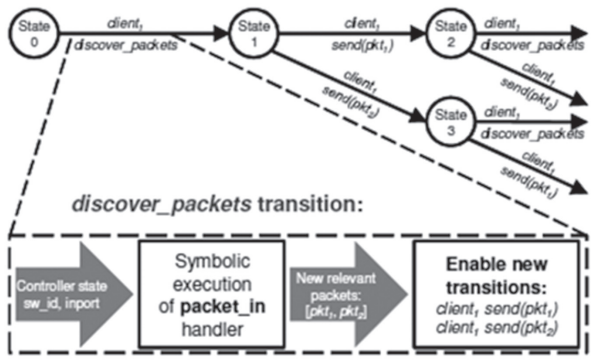


그림 7. NICE에서 Symbolic Execution으로 패킷 처리 과정을 관리하는 과정

NICE는 대표적인 OpenFlow 컨트롤러인 NOX를 기반으로 동작하며 Python으로 구현됐다[16]. Python 인터프리터를 수정하지 않으면서 동적인 symbolic execution을 구현하기 위해 concolic execution 기법을 적용했으며[17], MAC 러닝 스위치와 웹 서버 로드밸런서, 에너지 효율 트래픽 제어 등의 예를 통해 도구의 실제 적용 가능성을 입증했다.

NICE는 실시간 시스템이나 하드웨어 설계 검증에 주로 적용되던 모델 체킹 기법을 SDN/OpenFlow 문맥에 단순히 적용하는데 그치지 않고, 그 동안 꾸준히 제기된 상태 공간이 커지는 문제를 SDN에 맞게 보완했다는 점에서 SDN에 대한 모델 체킹 적용의 대표적인 사례로 손꼽을 수 있다.

2. Header Space Analysis

HSA(Header Space Analysis)는 네트워크의 프로토콜과 장

치가 패킷 헤더의 내용을 토대로 동작이 결정된다는 점에 착안하여, 헤더를 구성하는 비트 패턴을 다차원의 기하 공간으로 표현하여, 네트워크의 동작과 속성을 정적으로 분석하는 기법이다[18].

HSA 기법으로 분석할 수 있는 속성으로 도달 가능성과 포워딩 루프, 트래픽 분할 등을 대표적인 예로 제시했으며, Python 2.6으로 Hassel이라는 라이브러리 형태로 프로토타입을 구현하였다 [19]. 또한 이렇게 구현한 도구를 Stanford 대학 백본망에서 적용하여 대표적인 속성을 검증하고 성능을 분석했다. Hassel에서는 Cisco IOS용 파서도 제공하는데, 이를 사용하면 Cisco 라우터 설정을 HSA 변환 함수로 생성하여, 라우터의 동작과 속성을 분석할 수 있어서, 기존 장비에 대한 분석도 수행할 수 있다는 장점이 있다.

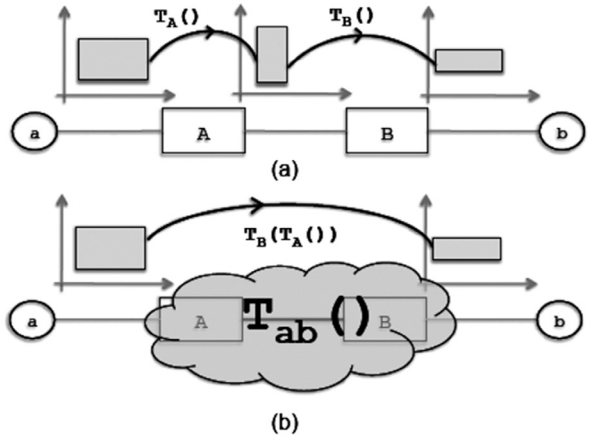


그림 8. 패킷이 여러 장비를 거쳐가는 과정을 변환 함수 형태로 표현

구현 과정에서 〈표 1〉와 같은 다섯 가지의 최적화 과정을 적용하여, 실제 망에 대한 적용 가능성도 실험했다.

표 1. 도달가능성과 루프 검사 과정에서 측정한 최적화 기법의 성능

Disabled Optimization	T.F. Generation	Reach. Test	Loop Test
None	160s	12s	11s
(1) IP Table Compression	10.5x	15x	19x
(2) Lazy Subtraction	1x	〉400x	〉400x
(3) Dead Object Deletion	1x	8x	11x
(4) Lookup Based Search	0.9x	2x	2x
(5) Lazy T.F. evaluation	1x	1.2x	1.2x

HSA 논문에서는 세 가지 예를 보여주고 있는데, 먼저 엔터프라이즈 네트워크 검증을 위해, 15,000명의 학생과 2000명의 교수진이 사용하는, 다섯 개의 IPv4 기반 서브넷으로 구성된 Stanford 백본망을 대상으로, Hassel을 적용하여 상용 단계의 적용 가능성을 시험하고, 12개의 무한 루프 경로를 발견하는 등, 다양한 네트워크 설정 오류와 도달 가능성 속성을 검증했다. 이외에도 Flowvisor를 통해 슬라이스(slice) 단위로 가상화하여, VLAN을 대체할 수 있을 정도로 빠르게 새로운 가상 네트워크를 생성할 수 있는지를 HSA 기법으로 분석하여, 아주 복잡하지 않은 슬라이스를 기준으로 500개까지 빠른 속도로 처리할 수 있다는 결과를 얻기도 했다. 또한 현재 네트워크 설정에 관련된 속성을 분석하는데 그치지 않고, 새로운 프로토콜을 설계하는 과정에서도 적용하여, HSA 기법을 미래 인터넷 연구에 활용할 수 있는 가능성도 확인했다.

HSA는 앞서 소개한 NICE에서 SDN/OpenFlow 기반 네트워크의 실제 동작을 추상화하여 모델체킹이나 정리 증명과 같은 기존 검증 기법을 적용하여 특정 속성에 대한 반례를 찾는데 집중하는 것과 달리, 현재 정의된 네트워크 설정 및 프로토콜 설계를 토대로 패킷의 흐름을 정적으로 분석하고, 문제가 되는 패킷 전체를 알려준다는 점이 특징이다.

3. Abstraction for Network Updates

SIGCOMM 2012의 메인 세션에서 SDN 전용 언어 및 검증 기법과 관련하여 네트워크 설정의 업데이트 과정의 신뢰성 향상을 위한 구조와 방법을 정형적으로 정의한 논문이 발표됐는데, 다양한 이슈 중에서 SDN과 관련하여 언어 및 검증과 관련된 주제가 메인 세션에서 대표적으로 채택되어 특히 주목할 만 하다 [20].

여기서는 먼저 SDN의 이론적인 모델을 정의하고, 네트워크 설정 업데이트 과정을 패킷 단위와 플로우 단위의 두 단계로 추상화하여, 네트워크 업데이트 과정에 발생하는 여러 가지 동작과 관련하여 네트워크에서 보장해야 하는 주요 속성을 엄밀하게 분석할 수 있는 이론적인 토대를 제공했다.

네트워크 업데이트 과정을 패킷 단위로 분석할 때 보장해야 할 주요 속성을 No Loops, Egress, Waypointing, Blacklisting 등으로 선별하여, 모델 체커에서 주로 사용하는 시제 논리인 CTL로 표현했다. 플로우 단위로 분석할 경우에는 스위치 룰 타임아웃, 와일드카드 클로닝, 엔드 호스트 피드백 등으로 구분하여 메커니즘을 분석하여, 기존 OpenFlow 문맥에서 제기된 여러 가지 상황에 대응할 수 있는 기법을 제시했다.

이러한 기법은 NOX 컨트롤러 위에 동작하는 Kinetic모듈 형태로 프로토타입을 구현하고, Mininet[21]을 이용한 가상 네트워크 환경을 통해 이를 검증했다. 앞서 소개한 두 종류의 추상화와 관련된 기능은 per_packet_update와 per_flow_update라는 함수 형태로 구현했다. Kinetic은 OpenFlow 1.0을 기반으로 구현하여 OpenFlow의 VLAN 필드에 버전을 기록했는데, 1.1 이상의 버전을 활용하거나 향후 OpenFlow 규격에 대한 새

로운 제안을 통해, 이러한 제약 사항을 극복할 수 있을 것으로 사료된다.

언어 및 검증과 관련하여 탄탄한 이론적인 토대를 마련하고, 업데이트와 관련된 SDN의 전반적인 동작 과정에 대한 세밀한 분석을 통해 다양한 메커니즘을 제시했다는 점에서 큰 의의가 있으며, 향후 이를 뒷받침할 만한 다양한 도구와 기법의 결합을 통해 강력한 플랫폼으로 성장할 수 있는 잠재력이 있지만, 당장 구체적인 형태로 구현하기에는 다소 복잡한 형태를 띠고 있다. 앞서 소개한 언어 중 Frenetic 연구팀과 구성원이 중첩된 점을 감안할 때, Frenetic이나 Nettle 플랫폼을 최대한 활용하는 측면에서 접근할 수도 있다고 예상된다.

4. FortNOX

FortNOX는 SRI와 Texas A&M 대학의 공동 연구팀에서 기존 NOX 컨트롤러에 보안 기능을 강화하기 위해 개발한 확장 모듈로서, 다양한 보안 위협 상황을 동적으로 대처하고, OpenFlow 규칙간의 충돌 상황을 방지하기 위한 기능을 SDN/OpenFlow 컨트롤러에 추가하여, 네트워크를 제어하는 과정에서 자연스럽게 보안 관련 속성을 보장해주는 방식으로 구현했다 [22].

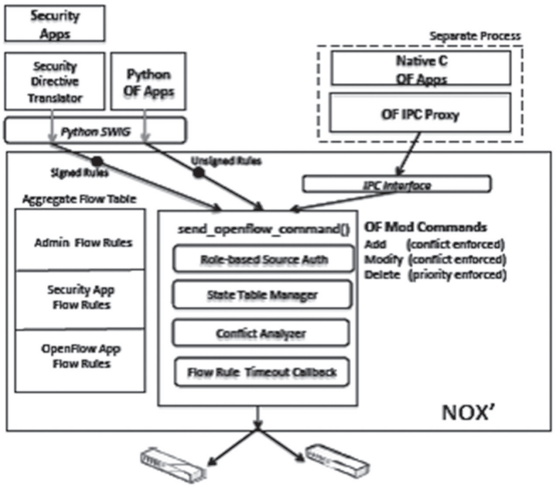


그림 9. FotNOX 구조

기존 SDN/OpenFlow 기술로는 Flowvisor와 같은 가상화 기법을 이용하여 플로우를 여러 개의 슬라이스 단위로 구분하여, 서로 간섭하지 않는다는 최소한의 보안 속성만을 보장해줬는데, FortNOX에서는 한 단계 더 나아가, 하나의 슬라이스 안에서도 보장되어야 할 여러 보안 속성을 보장해줄 수 있도록, NOX에서 OpenFlow 커맨드를 네트워크 장치로 전달하기 전에, 역할 기반 소스 인증을 수행하는 모듈과, 여러 개의 규칙 간의 충돌 상황을 감지하고 회피하기 위한 규칙 최적화 단계를 추가했다.

FortNOX는 앞서 언급한 기능을 기존 NOX 컨트롤러의 send_openflow_command 함수에500라인 가량의 코드를 추가한 형태로 구현하여, 이 과정을 통과한 플로우 규칙이나 명령만 네트워크 장치로 전달하고, 그렇지 않은 것은 반환하도록 구현했다. 이러한 기능을 수행하기 위해 플로우 규칙을 저장하고, 보안 속성을 관리하는 모듈도 부가적으로 구현했다.

5. ndb

ndb는 오픈 소스 소프트웨어 개발에 많이 사용되는 gdb라는 디버거에서 힌트를 얻어, 스탠포드 대학 연구팀에서 개발한 SDN/OpenFlow 네트워크를 위한 디버거다[23]. ndb는 특히 브레이크 포인트breakpoint와 패킷 백트레이스packet backtrace라는 두 개의 기본 요소를 구현하여, 포워딩 상태와 패킷 로그 조작을 통해 에러의 원인이 되는 부분을 추적할 수 있다.

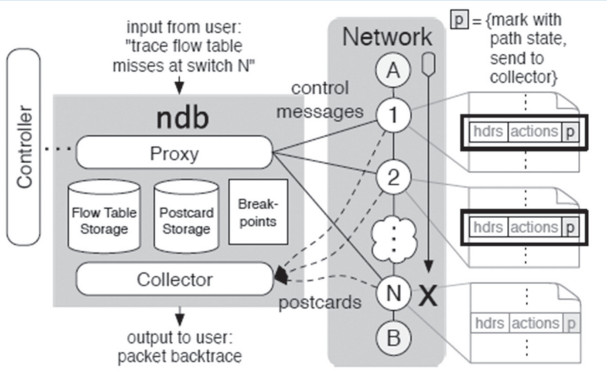


그림 10. ndb 구조

네트워크에서 디버깅 작업은 수행하기 까다로운 것으로 악명이 높은데, ping이나 traceroute, tcpdump, netflow 등과 같은 도구의 도움을 받을 수 있지만, 수시로 상태가 변하는 L2 러닝 스위치나 L3 라우팅에서는 다양한 분산 프로토콜에 기반한 복잡한 상태를 재현하기에는 턱없이 기초적인 기능만 제공한다는 점에서 한계가 있다. 더구나 다른 검증 기법 관련 논문에서 공통적으로 지적하듯이, 논리적으로 중앙 집중적인 제어기를 통해 표준 API 형태로 제공되는 다양한 네트워크 장치의 설정과 상태를 관리하는 SDN에서는, 이를 기반으로 동작하는 소프트웨어의 정확성이 중요하므로, 기존 소프트웨어 분야에서 축적된 노하우를 적극 수용할 필요가 있다. 현재 GCC를 이용한 오픈 소스 소프트웨어를 개발할 때, gdb라는 디버거를 많이 활용한다는 점에 착안하여, 정형 기법 보다는 좀 더 실용적인 접근 방식을 취하여, breakpoint, watch, backtrace, single-step, continue 등과 같은 gdb의 기능을 SDN 기반의 네트워크 문맥으로 재구성했다.

ndb는 backtrace를 구성할 수 있도록, 패킷이 스위치와 같은 특정한 네트워크 장치를 지나칠 때마다 postcard라는 메시지를 전달하여, 이론상 모든 플로우의 시작점에 해당하는 패킷마다 특정한 표시를 남기는 효과를 구현했다. 따라서 ndb는 〈그림 10〉에서 보는 바와 같이, 장치를 제어하는 메시지를 전달하는 동시에, 각 장치에서 전송되는 postcard와 개발자가 지정한 breakpoint를 수집 관리하는 모듈로 구성된다. 그러나 이러한 과정에서 플로우 테이블과 패킷에 대한 모호성이 발생할 수밖에 없는데, ndb 구현 과정에서 이러한 문제를 해결하기 위한 몇 가지 테크닉을 가미해야 했다.

또한 ndb 구현 과정에서 병렬적인 상황에서 플로우 테이블을 atomic하게 업데이트하는 기능과, L2 캡슐화 기능, 포워딩 액션에 대한 추가 기능 등과 같은, 현재 OpenFlow 규격에 대한 개선점도 도출했다.

현재는 기본적인 기능 구현을 통해 기술 검증 수준의 프로토타입을 제시했지만, 논문에서 지적한 개선 사항을 반영하고, JTAG과 같은 하드웨어 지원도 뒷받침된다면, SDN/OpenFlow 네트워크의 디버깅 작업에 필수적인 도구로 발전할 가능성이 있다.

Ⅳ. 표준화 활동

SDN 전용 언어 및 검증 기법에 대해서는 학회나 ONS뿐만 아니라, ITU-T나 IETF 등과 같은 표준화 기구에서도 논의되고 있다. ONF에서는 언어나 검증 기술에 대해 토론 그룹에서 잠시 논의된 바 있지만, 현재는 SDN의 핵심 요소 기술에 보다 주력하고 있다.

ITU-T에서는 미래 네트워크를 연구하는 SG13/Q.21 그룹에서 SDN을 위한 프레임워크와 정형 명세 및 검증 요구사항에 대한 기고서가 제안되어 표준 문서 작업을 시작하고 있으며[24][25], 이와 관련하여 ONF와도 문서의 방향과 활동 방식에 대해 두 차례의 Liaison 교환을 통해 논의된 바 있다.

인터넷 기술에 대한 표준 기구로서 대표적인 IETF에서도 올해 상반기에 결성된 SDN RG를 통해 학계와 업계의 SDN에 대한 주요 방향과 프레임워크에 대해 활발히 논의되었으며, ETRI에서는 SDN을 위한 정형 언어에 대한 기고도 발표한 바 있다[26].

현재 표준화 기구 입장에서는 SDN을 실현하기 위한 구조와 프레임워크, 프로토콜, 그리고 이를 위한 HW/SW 규격에 대한 이슈를 중심으로 논의되고 있으며, 오픈 소스 및 산업계 중심의 생태계 형태로서 발전을 추구한다는 점에서 표준화에 대한 중요성이 아직 부각되고 있지 않지만[27], 앞으로 다양한 언어와 프레임워크가 등장할 것으로 예상되는 만큼, 향후 표준화 기구에서도 지속적으로 토론이 이루어질 것으로 전망된다.

Ⅴ. 향후 연구 전망

지금까지 SDN/OpenFlow 기술 실현의 주요 수단으로 등장한 전용 언어와 검증 기법에 대해 간략히 살펴봤다. 언어는 FRP 기반의 Nettle, Procera, Frenetic과 Logic 기반의 FML의 두 갈래로 구분할 수 있는데, 모두 DATALOG와 같은 선언적인 형태의DB 쿼리 언어에 기반을 두고 있으며, 현재 활발히 연구되는 Frenetic과 Nettle, Procera만을 보면 모두 FRP기법을 적용하고 있다. 특히 Nettle은 언어 뿐만 아니라 컨트롤러 프레임워크의 의미도 강한데, 향후 Frenetic 이나 NetCore와 결합하여 서로 시너지를 이룰 수도 있을 것으로 전망된다.

또한 현재 활발히 개발되고 있는 FRP기반 언어와 달리, 프로세스 대수 등과 같은 다른 형태의 접근 방법도 시도되고 있는데[31], 이를 통해 함수형 언어 기반의 FRP 방식의 한계를 점검할 수 있을 뿐만 아니라, 새로운 언어의 개발을 촉진하여 SDN/OpenFlow 생태계에 기여한다는 점에서 의미가 있을 것이다.

한편 검증 기법과 관련하여 현재 활발히 진행되고 있는 연구는 크게 모델 체킹을 비롯한 전통적인 정형 기법을 최대한 SDN/OpenFlow 문맥에 맞게 최적화하고, 기존 모델 체킹에서 흔히 지적되던 상태 공간 문제를 해결하는 기법을 가미하거나, SDN 문맥에 최적화된 디버깅 기능을 제공하는 것처럼 외부 도구를 활용하는 방식과, FortNOX나 Kinetic처럼 SDN/OpenFlow컨트롤러와 같은 핵심 구성 요소에 보안 및 신뢰성 보장 모듈을 추가하여, 컨트롤러 동작 과정에서 자동으로 주요 속성을 검증하는 방식으로 구분할 수 있다. 전자의 방법 중 스탠포드 대학 연구팀에서 제시한 HSA 기법은 패킷이라는 기본 단위의 특성을 이용하여 범용적인 정적 분석 도구를 개발했다는 점에서, 모델 체킹이나 정리 증명 도구에 의존하는 다른 기법과는 색다른 시도라 볼 수 있다. 또한 ndb와 FortNOX는 정형 기법과 같은 특별한 배경 지식이 없어도 사용할 수 있을 형태로 구성하거나, 기존 OpenFlow 개발자의 관점을 충실히 반영한 도구를 제공한다는 점에서, 정형 기법이나 함수형 언어에 기반한 기법보다 쉽게 현장에 적용할 수 있을 것으로 예상된다.

언어와 검증 기법은 서로 별개의 연구 주제로 다룰 수도 있지만, 언어의 의미론을 잘 정의하면, 보다 원활한 검증을 수행할 수 있는 토대를 제공한다는 측면에서 서로 밀접하게 관련이 있다. 가령, SDN 전용 언어를 정형 의미론(formal semantics)에 기반하여 정의하면, 보다 엄밀한 분석을 수행할 수 있을 뿐만

아니라, 일반 언어로 구현할 때보다 모델 체커나 정리 증명기와 보다 손쉽게 결합할 수 있다는 장점이 있다.다만 정형 이론의 특성과 검증 도구 구현의 용이성에 치우치면, 네트워크 프로그래머 또는 관리자의 학습 곡선이 높아지거나, 사용자 편의성이 떨어질 가능성이 있는데, 상용화를 염두에 둔 도구를 개발할 경우, 이러한 상반된 특성을 잘 조합하는 것이 중요한 요소로 작용할 것이다.

Ⅵ. 결 론

본고에서는 최근 학계뿐만 아니라 네트워킹 업계에서 부각되고 있는 SDN/OpenFlow 기술에 대한 논의의 장에서 새롭게 부각되고 있는 SDN 전용 언어와 검증 기법에 대한 주요 연구 동향을 알아봤다.

언어와 검증 기법에 대한 주제는 다소 학문적인 접근이 두드러지는 경향을 보이고 있지만, 과거에도 라우팅 관련 Policy 설정 및 동작 오류 문제가 꾸준히 제기되었고[28][29], 특히 소프트웨어 중심의 혁신을 추구하는 SDN에서는 이를 실현하는 소프트웨어 및 주요 모듈에 대한 동작의 신뢰성이 더욱 강조될 수밖에 없으므로, 앞으로도 연구 활동의 규모와 활성도가 지금보다 확대될 것으로 전망된다.

Acknowledgement

본 연구는 방송통신위원회 및 한국방송통신전파진흥원의 방송통신기술개발사업의 일환으로 수행하였음 (KCA-2012-10913-05003, 미래인터넷 국제협력 연구를 위한 테스트베드 구축)

참 고 문 헌

[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 SIGCOMM CCR, April 2008

[2] http://www.opennetworking.org

[3] http://www.opennetsummit.org

[4] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. Practical declarative network management. In WREN, pages 1-10, 2009

[5] N. Gude, T. Koponen, J. Petit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. SIGCOMM CCR., 38:105-110, July 2008

[6] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, David Walker. Frenetic: A network programming language. In ICFP, September 2011

[7] http://frenetic-lang.org

[8] Antony Courtney, Henrik Nilsson, and John Peterson. The Yampa arcade. In Haskell Workshop, pages 7-18, August 2003

[9] Christopher Monsanto, Nate Foster, Rob Harrison, David Walker. A compiler and run-time system for network programming languages. In POPL, January 2012

[10] Andreas Voellmy and Paul Hudak. Nettle: Functional reactive programming of OpenFlow networks. In PADL, January 2011

[11] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: A language for high-level reactive network control. In SIGCOMM HotSDN, August 2012

[12] Hyojoon Kim, Andreas Voellmy, S. Burnett, Nick Feamster, and R. Clark. Lithium: Event-driven network control. Technical Report GT-CS-12-03, Georgia Institute of Technology, 2012

[13] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transaction on Programming Languages and Systems, vol. 8, pages. 244-263, 1986

[14] S. Bucur, V. Ureche, C. Zamfir, and G. Candea. Parallel Symbolic Execution for Automated Real-World Software Testing. In EuroSys, 2011

[15] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A NICE way to test OpenFlow applications. In NSDI, April 2012

[16] http://code.google.com/p/nice-of/

[17] P. Godfroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In PLDI, 2005

[18] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static checking for networks. In NSDI, April 2012
[19] http://stanford.edu/~kazemian/hassel.tar.gz
[20] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstraction for network update. In SIGCOMM, August 2012
[21] http://openflow.org/mininet
[22] Phillip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In HotSDN, August 2012
[23] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazieres, and Nick McKeown. Where is the debugger for my software-defined network? In HotSDN, August 2012
[24] Draft Recommendation of Y.FNsdn. Framework of Software-Defined Networking for Carrier Networks in Future Networks. ITU-T
[25] Draft Recommendation of Y.FNsdn-fm. Requirement of Formal Specification and Verification Methods for SDN. ITU-T
[26] http://tools.ietf.org/html/draft-shin-sdn-formal-specification-01
[27] http://networkheresy.com/2011/08/09/what-might-an-sdn-controller-api-look-like-and-should-we-standardize-it/
[28] T. Benson, A. Akella, and A. Shaikh. Demystifying configuration challenges and trade-offs in network-based isp services. SIGCOMM Computer Communication Review, 41(4):302-313, Aug. 2011
[29] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In Proc. ACM SIGCOMM, p.3-17, Pittsburgh, PA, August 2002
[30] http://www.openflowhub.org/display/Snac/SNAC+Home
[31] Miyoung Kang, Junkil Park, Jeehoon Shin, Ki-Hyuk Nam, Myung-Ki Shin, and Jin-Young Choi. Formal Specifications for Software-Defined Networking. Proceedings of the 7th International Conference on Future Internet Technologies (CFI'12), pages 51-52. September 2012

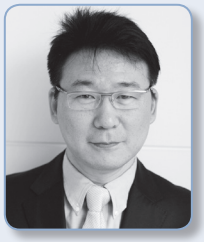
약 력

남 기 혁

2002년 고려대학교 컴퓨터학과 이학사
2004년 고려대학교 컴퓨터학과 공학석사
2004년~현재 한국전자통신연구원 선임연구원
관심분야: SDN, OpenFlow, 가상화, 정형 기법

신 명 기

2003년 충남대학교 컴퓨터공학과 공학박사
1994년~현재 한국전자통신연구원 책임연구원
2008년~현재 한국기술연합대학원대학교(UST) 겸임교수
2004년~2005년 미국NIST (National Institute of Standards and Technology) 초빙연구원
관심분야: 미래인터넷, 네트워크 가상화, SDN, OpenFlow



김 형 준

2007년 충남대학교 공학박사
2007년~2008년 University of Virginia 초빙연구원
1989년~현재 한국전자통신연구원 표준연구센터 팀장/책임연구원
2008년~현재 한국기술연합대학원대학교(UST) 겸임교수
관심분야: 미래인터넷, RFID/USN, M2M