ETRI

2012
Electronics and
Telecommunications
Trends

# SDN 프로그래밍 기술 연구 동향

## A Survey on Programming Methods for Software-Defined Networking



**남기혁** (K.H. Nam) 미래인프라표준연구팀 선임연구원
**신명기** (M.K. Shin) 미래인프라표준연구팀 책임연구원
**김형준** (H.J. Kim) 미래인프라표준연구팀 팀장
**강미영** (M.Y. Kang) 고려대학교 정보통신기술연구소 연구교수
**최진영** (J.Y. Choi) 고려대학교 융합소프트웨어전문대학원 교수

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

최근 네트워킹 분야에서 활발히 연구되고 있는 SDN(Software-Defined Networking) 기술에서, 보다 편리하고 정확한 방법으로 네트워크를 구축하기 위한 SDN 프로그래밍 기술에 대해 최근 발표된 논문을 중심으로 연구 동향을 분석한다.

## Ⅰ. 서론

OpenFlow는 미국 NSF의 지원으로 스탠포드 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 확장되어, 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 기술과 달리, 간결한 메커니즘을 토대로, 기존의 네트워크 장비의 보조적인 역할을 수행하던 소프트웨어의 역할을 보다 확대하고, 분산 시스템과 운영체제, 데이터베이스와 같은 소프트웨어에서 정립된 개념을 적극적으로 도입하여 혁신의 속도를 높일 수 있는 구조와 생태계를 구성하는 데 노력하고 있다. 이처럼 소프트웨어의 역할을 극대화하여 혁신의 속도를 높일 수 있다는 장점의 이면에는, 이러한 소프트웨어 모듈의 오류가 전체 네트워크에 악영향을 미칠 수 있다는 위험도 존재하는데, 본고에서는 SDN/OpenFlow에서 소프트웨어 오류로 인한 문제를 최대한 줄이기 위해 최근 활발히 연구되고 있는 SDN 프로그래밍 관련 기술 동향을 언어와 검증 기법, 가상화라는 세 가지 관점으로 분석한다.

## Ⅱ. 언어

### 1. FML

FML(Flow-based Management Language)은 엔터프라이즈 네트워크에 적용할 정책(policy)을 선언적인 방식으로 정의하기 위한 목적으로 개발된 논리 기반 언어다[3]. 단순한 형태의 인터페이스만 제공하던 기존의 라우터나 스위치와 달리, 네트워크 설정 및 정책(policy)을 논리 기반 언어로 정의하여, ACL이나 VLAN, 라우팅 정책 등을 보다 쉽고 견고하게 구현할 수 있는 장점이 있으며, 최근 등장한 Frenetic이나 Nettle 등과 같은 다른 SDN 언어는 대부분 FML에 적지 않은 영향을 받았다[4]-[7].

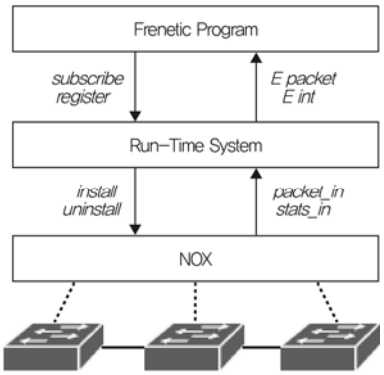
FML에서는 네트워크 정책을 if-then 형식의 규칙들의 집합으로 정의하는데, 이때 규칙들이 서로 재귀적으로 얽히지 않아야 한다. 각 규칙에 대해 별도의 순서를 부여하지 않기 때문에, 언어에 배경 지식이 부족한 네트워크 관리자도 필요한 규칙을 나열하는 방식으로 쉽게 작성할 수 있지만, 이로 인해 여러 개의 규칙이 서로 상충되거나 중복되는 문제도 발생할 수 있는데, policy cascade와 conflict resolution 계층을 도입하여 해결하고 있다.

FML 논문에서는 접근 제어, QoS, NAT 등과 같은 설정을 FML로 표현하는 예를 소개하고 있으며, NOX[8] 컨트롤러에 기반하여 대략 10,000 라인의 C++와 Python 코드로 프로토타입을 구현하였고, 현재 SNAC 컨트롤러의 policy manager에서도 FML을 사용하고 있다[9].

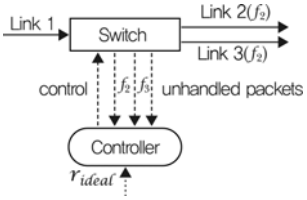
### 2. Frenetic, NetCore

Frenetic은 FML이 제안된 이듬해인 2010년부터 프린스턴 대학 연구팀에서 발표한 OpenFlow 전용 언어로서, FML과 마찬가지로 선언적인 형태의 데이터베이스 쿼리 언어에 기반을 두고 있다[4],[10].

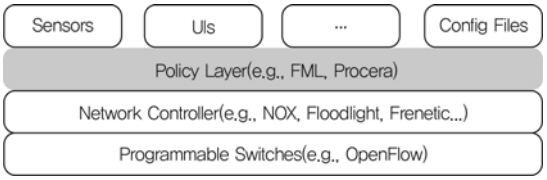
Frenetic은 NOX 컨트롤러의 애플리케이션을 작성하는 과정에서 겪는 여러 가지 한계를 극복하기 위해, 기존 언어에서 제공하지 못한 모듈화와 단일 티어 추상화를 제공하며, 경쟁 조건(race condition)에 대한 처리를




(그림 1) Frenetic 구조




(그림 2) 연속 개념 적용 예




(그림 3) Procera 구조

언어 런타임 차원에서 지원함으로써, NOX 애플리케이션 개발자가 원하는 작업을 구성하는 데 보다 집중할 수 있다((그림1) 참조).

Frenetic은 DATALOG와 같은 선언적인 DB 쿼리 언어와 Yampa[11]와 같은 FRP(Funtional Reactive Programming) 기반 언어의 영향을 받았으며, 컴비네이터 라이브러리 형태로 policy를 관리할 수 있다. 이러한 특성은 뒤에서 소개하는 Nettle과 유사하다.

Frenetic 연구팀에서는 NetCore[5]라는 언어 및 이론적 토대도 제안했는데, 사용자 편의성과 모듈화를 비롯한 고수준 언어의 특성을 제공하기 위한 목적으로 NOX의 임베디드 언어 형태로 구현된 Frenetic과 달리, 네트워크 모델과 정형 의미론(formal semantics)을 이용하여 정의한 언어를 제공함으로써, 고수준의 표현을 OpenFlow 규칙 형태로 변환하는 컴파일러 제작과, 다양한 policy에 대한 정형 검증을 위한 이론적 기반을 제공한다.

### 3. Nettle, Procera

Nettle은 Frenetic과 비슷한 시기에 등장한 Haskell 기반 언어로서, OpenFlow 컨트롤러와 스위치가 서로 주고 받는 이벤트와 메시지를 하나의 스트림으로 추상화했다[6]. 또한 FRP 방식을 채용하여 이산적인 이벤트 발생과 연속적인 시간에 따른 값이나 신호의 변화로 SDN 동작을 모델링하는 특징을 가지고 있다.

Nettle 논문에서는 다양한 OpenFlow 응용 예를 보여주고 있는데, 그 중에서도 로드 밸런싱 애플리케이션처럼 연속적인 변화량을 다루는 경우에도 쉽게 적용할 수 있다는 장점이 있다((그림 2) 참조).

Procera[7]는 2012년도에 발표된 SDN 언어로 Haskell 기반 FRP 방식의 언어라는 점에서 Nettle과 유사하지만, 보다 고수준의 표현으로 policy를 정의할 수 있으며, Lithium[12]을 비롯한 네트워크 컨트롤러 기반으로 동작하고 시제 연산자를 통해 이벤트 히스토리를 다룰 수 있다((그림 3) 참조).

## Ⅲ. 검증 기술

### 1. 개요

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(Open Networking Summit) 등을 통해 소개된 기법을 중심으로 소개한다[13]–[15].

## 2. NICE

NICE는 OpenFlow 네트워크의 프로그래밍 오류를 검출하기 위해, 모델 체킹(model checking) 기반의 검증 및 테스트 도구다[16].

NICE 논문에서는 OpenFlow 네트워크 프로그래밍 과정에서 (그림 4)와 같은 경우에 발생하는 오류를 해결할 수 있도록, 모델 체킹 기법을 활용한 검증 방법을 제안했다[17]. 또한 기호 실행 기법(symbolic execution)을 적용하여 탐색할 상태 공간을 줄이면서[18], 좀 더 OpenFlow에 적합한 구조를 통해, SPIN이나 JPF와 같은 기존 모델 체킹 도구보다 성능을 향상시켰다[19],[20].

NICE에서는 OpenFlow에 최적화 된 네 가지 휴리스틱(huristic) 기법을 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.

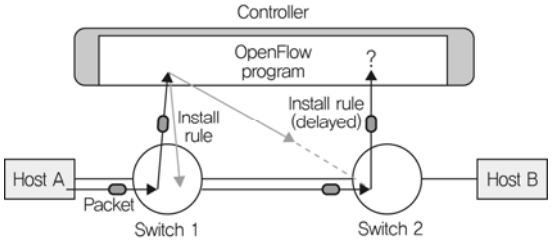
NICE는 대표적인 OpenFlow 컨트롤러인 NOX를 기반으로 동작하며 Python으로 구현됐다[21]. Python 인터프리터를 수정하지 않으면서 동적으로 기호 실행 기법을 처리하기 위해 concolic execution 기법을 적용했으며[22], MAC 러닝 스위치와 웹 서버 로드밸런서, 에너지 효율 트래픽 제어 등의 예를 통해 도구의 실제 적용 가능성을 입증했다.

NICE는 실시간 시스템이나 하드웨어 설계 검증에 주로 적용되던 모델 체킹 기법을 SDN/OpenFlow 문맥에 단순히 적용하는데 그치지 않고, SDN에 맞게 적용 및 보완했다는 점에서 SDN에 대한 모델 체킹 기법의 대표적인 사례로 손꼽을 수 있다.

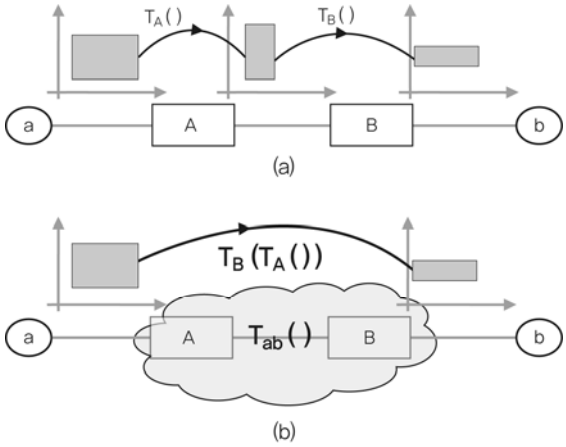
## 3. Header-Space Analysis

HSA(Header Space Analysis)는 네트워크의 프로토콜과 장치가 패킷 헤더의 내용을 토대로 동작이 결정된다는 점에 착안하여, 헤더를 구성하는 비트 패턴을 다차원의 기하 공간으로 표현하고, 스위치나 라우터와 같은 네트워크 장치를 거쳐는 부분을 변환 함수로 처리하여, 네트워크의 동작과 속성을 정적으로 분석하는 기법이다[23]. HSA에서는 패킷이 최종 목적지까지 도달하는 과정을 변환 함수의 합성으로 표현한다((그림 5) 참조).

논문에서는 Python 2.6으로 Hassel이라는 라이브러리 형태로 프로토타입을 구현하고[24], 스탠포드 대학




(그림 4) OpenFlow 네트워크 설정 과정에서 발생하는 문제




(그림 5) 변환 함수로 표현한 패킷 전달 과정

백본망에서 적용하여 도달 가능성과 포워딩 루프, 트래픽 분할 등과 같은 속성을 검증하고 성능을 분석했다. Hassel에서는 Cisco IOS용 파서도 제공하는데, 이를 사용하면 Cisco 라우터의 동작과 속성을 분석할 수 있다.

HSA 논문에서는 엔터프라이즈 네트워크 검증을 위해, 15,000명의 학생과 2,000명의 교수진이 사용하는, 다섯 개의 IPv4 기반 서브넷으로 구성된 스탠포드 대학 백본망을 대상으로 Hassel을 적용하여 상용 단계의 적용 가능성을 시험하고, 12개의 무한 루프 경로를 발견하는 등, 다양한 네트워크 설정 오류와 도달 가능성 속성을 검증했다. 이외에도 Flowvisor[25]를 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 얻었다. 또한 새로운 프로토콜을 설계하는 과정에서도 HSA 기법을 적용하여, 미래 인터넷 연구의 보조 도구로 활용할 수 있는 가능성도 확인했다.

앞서 소개한 NICE가 SDN/OpenFlow 기반 네트워크의 실제 동작을 추상화하여 모델 체킹과 같은 기존 검증 기법을 적용하여 특정 속성에 대한 반례를 찾는데 집중하는 것과 달리, HSA는 현재 정의된 네트워크 설정 및 프로토콜 설계에 대한 패킷의 흐름에 대한 모델을 정의하고, 모든 패킷에 대한 동작을 정적으로 분석하는 점이 특징이다.

### 4. Kinetic

Kinetic은 네트워크 설정의 업데이트 작업에 대한 신뢰성 향상을 위한 구조와 방법에 대해 SIGCOMM 2012의 메인 세션에서 발표된 논문에서 제안한 기법으로, SDN의 이론적인 모델을 수학적으로 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[26].

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 switch rules with timeouts, wildcard cloning, end-host feedback 등으로 구분하여 메커니즘을 분석하여, 기존 OpenFlow 문맥에서 제기된 여러 가지 상황에 대응할 수 있는 기법을 제시했다.

이러한 기법은 NOX 컨트롤러 위에 동작하는 Kinetic 모듈 형태로 프로토타입을 구현하고, Mininet[27]을 이용한 가상 네트워크 환경을 통해 이를 검증했다. Kinetic은 OpenFlow 1.0을 기반으로 구현했으며, 앞서 소개한 두 종류의 추상화와 관련된 기능은 per_packet_update와 per_flow_update라는 함수 형태로 구현했다. Kinetic에서는 버전 정보를 기록하기 위해 OpenFlow의 VLAN 필드를 사용하는 다소 제한적인 방식으로 구현했는데, 1.1 이상의 OpenFlow 규격에서는 이러한 제약 사항을 해결할 것으로 전망된다.

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, 당장 구체적인 형태로 구현하기에는 다소 복잡한 면이 없지 않으나, 향후 이를 뒷받침할 만한 다양한 도구와 기법의 결합을 통해 강력한 플랫폼으로 성장할 수 있는 잠재력이 있고, Frenetic 연구팀에서 제안한 점을 감안할 때, Frenetic이나 Nettle 플랫폼과 결합되는 형태로도 발전할 가능성이 있다.

### 5. FortNOX, VeriFlow

FortNOX는 다양한 보안 위협 상황과 부주의한 네트

워크 설정으로 인한 OpenFlow 규칙 간의 충돌을 실시간으로 검사하기 위해, SRI와 Texas A&M 대학의 공동 연구팀에서 개발한 NOX 컨트롤러의 확장한 모듈이다[28].

Flowvisor[25]와 같은 가상화 기법을 이용하여 플로우 단위로 구분한 가상 네트워크(슬라이스)가 서로 간섭하지 않도록 최소한의 보안 속성을 보장하는데 그치지 않고, 하나의 슬라이스 안에서도 여러 가지 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 함수에 500 라인 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.

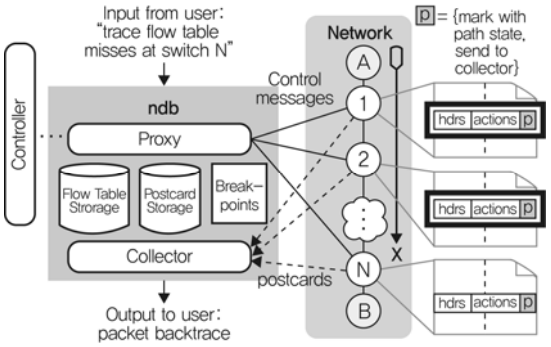
VeriFlow 역시 FortNOX와 마찬가지로 네트워크에 대한 주요 속성을 실시간으로 검증하는 기법으로서, 컨트롤러와 네트워크 장비 사이의 계층에서 동작하며, 검증으로 인한 지연 시간을 최소화하기 위한 기본 메커니즘을 제시하고, Mininet으로 프로토타입을 검증했다[27], [29].

## 6. ndb

ndb는 오픈 소스 소프트웨어에서 많이 사용하는 gdb라는 디버거에서 힌트를 얻어, 스탠포드 대학 연구팀에서 개발한 SDN/OpenFlow 네트워크를 위한 디버거다[30]. 네트워크에서 디버깅 작업은 수행하기 까다로운 것으로 악명이 높은데, ping이나 traceroute, tcpdump, netflow 등에서 제공하는 기본적인 기능만으로는, 수시로 상태가 변하는 L2 러닝 스위치나 L3 라우팅에서는 다양한 분산 프로토콜에 기반한 복잡한 상태를 재현하기에는 한계가 있다. 더구나 다른 검증 기법 관련 논문에서 공통적으로 지적하듯이, 논리적으로 중앙 집중적인 구조를 가진 제어기를 통해 다양한 네트워크 장치의 설정과 상태를 관리하는 SDN에서는, 이를 구성하는 소프트웨어의 정확한 동작이 매우 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 기법을 적용하는 다른 기법보다 좀 더 실용적인 접근 방식을 취하여, breakpoint, backtrace, single-step, watch, continue 등과 같은 gdb의 기능을 SDN 기반의 네트워크 문맥으로 재구성하여, 포워딩 상태와 패킷 로그 조작을 통해 에러의 원인이 되는 부분을 추적할 수 있다.

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 6)에서 보는 바와 같이, 장치를 제어하는 메시지를 전달하는 동시에, 각 장치에서 전송되는 postcard와 개발자가 지정한 breakpoint를 수집 관리하는 모듈로 구성된다. 그러나 이러한




(그림 6) ndb 구조

과정에서 플로우 테이블과 패킷에 대한 모호성이 발생할 수밖에 없는데, ndb 구현 과정에서 이러한 문제를 해결하기 위한 몇 가지 테크닉을 제안했으며, 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, ndb 논문에서 지적한 개선사항을 반영하고, JTAG과 같은 하드웨어 지원도 뒷받침된다면, SDN/OpenFlow 네트워크의 디버깅 작업에 핵심적인 도구로 발전할 가능성이 있다.

## Ⅳ. 가상화

앞서 살펴본 SDN 프로그래밍 언어 및 검증 도구는 결국 SDN/OpenFlow 환경과 밀접하게 연계하여 동작할 수밖에 없다. 특히 이러한 환경을 제공하기 위한 플랫폼에서는 네트워크 가상화를 주요 요소 기술로 다루고 있는데, SDN 가상화 관련 논문과 오픈 소스 기반 소프트웨어 스위치인 Open vSwitch, 그리고 클라우드를 위한 오픈 인터페이스 기반 플랫폼인 OpenStack을 중심으로 가상화 관련 기술 동향도 간략히 정리한다.

### 1. 슬라이스 추상화

네트워크 가상화와 관련하여 GENI 프로젝트와 OpenFlow의 가상화 솔루션인 Flowvisor 등에서 슬라이스라는 용어를 사용하는데[25], 각 문맥마다 의미하는 바가 약간씩 다르긴 하나, 가상 네트워크들이 서로 방해하지 않는 형태의 모델에 기반을 둔다는 점은 공통적이다.

2012년 SIGCOMM HotSDN 워크숍에서 발표된 논문에서는, VLAN과 같은 저수준의 메커니즘에 의존하는 현재의 네트워크 가상화 기술을 대체할 수 있도록, 슬라이스라는 네트워크 추상화 단위를 프로그래밍 모델 차원에서 지원하는 새로운 메커니즘을 제시했다[31]. 이 논문에서는 앞서 소개한 Frenetic/NetCore 언어를 이용한 슬라이스 단위의 가상 네트워크 모델을 정의하고, 이렇게 표현된 네트워크 명세를 OpenFlow 스위치 코드로 변환하는 컴파일러와, 이 과정의 정확성을 검증하기 위한 기법을 제안했다. 슬라이스 컴파일러에 대한 프로토타입은 웹에 공개했으며[32], 컴파일러의 정확성에 대해서는 직접 검증하지 않고, translation validation과 같은 접근 방식을 취하여, 고수준 언어로 명세한 네트워크 모델이 만족해야 하는 속성을 CTL이라는 시제 논리로 표현하여 NuSMV라는 모델 체커로 검증할 수 있는 방안도 제시했다[33]-[35].

### 2. Open vSwitch

Open vSwitch(OVS)는 오픈 소스 프로젝트로 활발히 개발하고 있는, 네트워크 가상화를 위한 소프트웨어 스위치다[36],[37]. OVS는 Xen과 같은 가상 머신을 위한 소프트웨어 스위치로 출발하여, 현재는 OpenStack 기반의 클라우드나 데이터 센터를 위한 가상 네트워크를 구성하는데 필요한 기능도 추가됐다[38]. 특히 가상 머신 간의 동적인 네트워크를 구성하기 위해서 STT (Stateless Transport Tunneling)라는 터널링 프로토콜과 OpenFlow 제어 및 관리를 위한 데이터베이스 모듈인 OVS-DB에 대해 IETF 기고서 형태로 발표했으며[39],[40], 본격적인 NaaS(Network as a Service) 서비스를 위한 핵심 모듈로 자리잡고 있다. 현재는 OpenFlow 프로젝트에 참여했던 스탠포드 대학 출신 연구원을 주축으로 구성된 스타트업인 Nicira Networks에서 OpenStack Quantum과 관련하여 연구 개발이 활발히 진행되고 있으며, 2012년 상반기에 VMWare에 합병되어 VXLAN과 같은 기존 기술과 시너지를 이룰 것으로 예상된다.

### 3. OpenStack

최근 몇 년간 소프트웨어 분야에서 끊임없이 등장하는 키워드로 클라우드를 빼놓을 수 없는데, 여기서도 SDN 기술은 중요한 역할을 담당하고 있다. 최근 Amazon EC2와 Citrix의 CloudStack과 더불어, 대표적인 플랫폼으로 OpenStack이 등장하고 있는데[38], 컴퓨팅과 스토리지, 네트워크 등을 담당하는 이종 모듈끼리 메시지 큐와 REST API 기반의 플랫폼 독립적인 인터페이스로 상호 작용함으로써, Xen을 비롯한 다양한 요소 기술을 구현한 상용 및 오픈 소스 제품을 최대한 수용하면서 기술 개선의 속도를 높일 수 있는 구조를 가진 것이 큰 특징이다. SDN/OpenFlow 기술은 OpenStack의 네트워크 가상화를 담당하는 Quantum에서 중요한 역할을 담당하고 있으며, Nicira Networks에서 Open vSwitch와 관련하여 Quantum 프로젝트에도 적극적으로 참여하고 있으며, 자사의 상용 플랫폼인 NVP에서도 OpenStack과의 연동을 지원하고 있다.

## Ⅴ. 표준화 활동

SDN 프로그래밍 기술은 학회나 ONS[15]뿐만 아니라, ITU-T나 IETF 등과 같은 표준화 기구에서도 논의되고 있다. ONF[2]에서는 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[41],[42], 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[43].

## Ⅵ. 향후 연구 전망

지금까지 SDN/OpenFlow 기술 실현의 주요 수단으로 등장한 프로그래밍 기술을 언어와 검증 기법, 가상화 관점에서 간략히 살펴봤다. 우선 SDN 언어는 FRP 기반의 Nettle, Procera, Frenetic과 logic 기반의 FML의 두 갈래로 구분할 수 있는데, 모두 DATALOG와 같은 선언적인 형태의 DB 쿼리 언어에 기반을 두고 있으며, 현재 활발히 연구되는 Frenetic과 Nettle, Procera만을 보면 모두 FRP 기법을 적용하고 있다. 특히 Nettle은 언어뿐만 아니라 컨트롤러 프레임워크의 의미도 강한데, 향후 Frenetic이나 NetCore와 결합하여 서로 시너지를 이룰 수도 있을 것으로 전망된다.

또한 현재 활발히 개발되고 있는 FRP 기반 언어와 달리, 프로세스 대수 등과 같은 다른 형태의 접근 방법도 시도되고 있는데[44], 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 의미론에 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 정형적인 특성과 검증 도구 구현의 용이성에 치우치면, 네트워크 프로그래머의 학습 곡선이 높아져 사용자 편의성이 떨어질 가능성이 있는데, 상용화를 염두에 둔 도구를 개발할 경우, 이러한 상반된 특성을 잘 조합하는 것이 중요한 요소로 작용할 것이다.

또한 현재로선 언어와 검증 기법에 대해 다소 학문적인 접근이 두드러지는 경향을 보이고 있지만, 과거에도 라우팅 관련 policy 설정 및 동작 오류 문제가 꾸준히 제기되었고[45],[46], 특히 소프트웨어 중심의 혁신을 추구하는 SDN에서는 이를 실현하는 소프트웨어 및 주요 모듈에 대한 동작의 신뢰성이 더욱 강조될 수 밖에 없으므로, 앞으로도 연구 활동의 규모와 활성도가 지금보다 확대될 것으로 전망된다.

표준화 기구 입장에서도 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[47], 최근 캐리어 네트워크 사업자를 중심으로 결성된 ETSI 산하의 NFV 그룹이나 Cisco나 Huawei, Ericsson에서 제안한 SDN 모델에서 다양한 네트워크 장비의 유기적인 결합으로 고차원 서비스와 기능을 효율적인 구현을 추가한다는 점을 감안할 때[48],[49], 앞으로 다양한 언어와 프레임워크가 등장할 뿐만 아니라, 향후 표준화 기구에서도 지속적으로 토론이 이루어질 것으로 전망된다.

**용어해설**

**정형 기법(Formal Methods)** SW/HW 시스템의 명세와 검증을 위한 수학 및 논리학 기반 기술이다. 소프트웨어 공학의 한 분야로, 계산이론, 프로그래밍 의미론을 비롯한 여러 가지 전산학의 이론에 기반을 두고 있으며, 시스템의 요구사항을 엄격히 만족시키고 신뢰성을 보장해야 하는 실시간 임베디드 시스템을 비롯한 다양한 분야에서 주로 활용된다.

**정형 의미론(Formal Semantics)** 프로그래밍 언어의 의미를 수학 및 논리학 기반으로 엄밀히 정의한 것으로, 언어 및 컴파일러 구현이나 프로그램 검증에서 주로 활용된다.

**FRP(Functional Reactive Programming)** 함수형 언어 기반의 반응형(reactive) 프로그래밍을 위한 방법론으로서, 연속적인 시간의 흐름에 따라 동작과 신호가 변하며, 이산적인 이벤트에 반응하는 모델에 기반을 두고 있다.

## 약어 정리

| | |
|---|---|
| FML | Flow-based Management Language |
| FRP | Funtional Reactive Programming |
| NaaS | Network as a Service |
| ONF | Open Networking Foundation |
| ONS | Open Networking Summit |
| OVS | Open vSwitch |
| SDN | Software-Defined Networking |
| STT | Stateless Transport Tunneling |

## 참고문헌


[1] N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM CCR*, vol. 38, no.2, Apr. 2008.

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

[3] T.L. Hinrichs et al., "Practical Declarative Network Management," *WREN*, 2009, pp. 1-10.

[4] N. Foster et al., "Frenetic: A Network Programming Language," ICFP, Sept. 2011, pp. 279-291.

[5] C. Monsanto et al., "A Compiler and Run-time System for Network Programming Languages," *POPL*, Jan. 2012, pp. 217-230.

[6] A. Voellmy and P. Hudak. "Nettle: Functional Reactive Programming of OpenFlow Networks," *PADL*, Jan. 2011.

[7] A. Voellmy, H. Kim, and N. Feamster. "Procera: A Language for High-level Reactive Network Control," *SIGCOMM HotSDN*, Aug. 2012.

[8] N. Gude et al., "NOX: Towards an Operating System for Networks," *SIGCOMM CCR*, vol. 38, no. 3, July 2008, pp. 105-110.

[9] OpenFlowHub, SNAC. http://www.openflowhub.org/display/Snac/SNAC+ Home

[10] Frenetic. http://frenetic-lang.org

[11] A. Courtney, H. Nilsson, and J. Peterson. "The Yampa Arcade," *Haskell Workshop*, Aug. 2003, pp. 7-18.

[12] H. Kim et al., "Lithium: Event-driven Network Control," SCS Technical Report, GT-CS-12-03, Georgia Institute of Technology, 2012.

[13] NSDI 2012. https://www.usenix.org/conference/nsdi12

[14] SIGCOMM 2012. http://conferences.sigcomm.org/sigcomm/2012/

[15] ONS 2013. http://www.opennetsummit.org

[16] M. Canini et al., "A NICE Way to Test OpenFlow Applications," *NSDI*, Apr. 2012.

[17] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Programming Languages Syst. (TOPLAS)*, vol. 8, no. 2, Apr. 1986, pp. 244-263.

[18] S. Bucur et al., "Parallel Symbolic Execution for Automated Real-World Software Testing," EuroSys, 2011, pp. 183-198.

[19] G. Holzmann, *The Spin Model Checker - Primer and Reference Manual*, Addison-Wesley, Reading, Massachusetts, 2004.

[20] W. Visser et al., "Model Checking Programs," *Automated Softw. Eng.*, vol. 10, no. 2, 2003, pp. 203-232.

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

[22] P. Godfroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," *PLDI*, 2005, pp. 213-223.

[23] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," *NSDI*, Apr. 2012.

[24] http://stanford.edu/~kazemian/hassel.tar.gz

[25] R. Sherwood et al., "Can the Production Network Be the Testbed?," *OSDI*, 2010.

[26] M. Reitblatt et al., "Abstraction for Network Update," *SIGCOMM*, Aug. 2012, pp. 323-334.

[27] Mininet. http://openflow.org/mininet

[28] P. Porras et al., "A Security Enforcement Kernel for OpenFlow Networks," *HotSDN*, Aug. 2012, pp. 121-126.

[29] A. Khurshid et al., "VeriFlow: Verifying Network-wide Invariants in Real Time," *HotSDN*, Aug. 2012, pp. 49-54.

[30] N. Handigol et al., "Where is the Debugger for My Software-defined Network?," *HotSDN*, Aug. 2012, pp. 55-60.

[31] S. Gutz et al., "Splendid Isolation: a Slice Abstraction for Software-defined Networks," HotSDN, Aug. 2012, pp. 79-84.

[32] https://github.com/frenetic-lang/slices

[33] A. Pnueli, M. Siegel, and E. Singerman, "Translation Validation," *Int. Conf. Tools Algorithms Construction Anal. Syst. (TACAS)*, Lisbon, Portugal, Mar. 1998, pp. 151-166.

[34] A. Cimatti et al., "NuSMV 2: An Opensource Tool for Symbolic Model Checking," *Int. Conf. Comput. Aided Verification (CAV)*, Copenhagen, Denmark, July 2002, pp. 359-364,

[35] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Programming Languages Syst. (TOPLAS)*, vol. 8, no. 2, Apr. 1986, pp. 244-263.

[36] B. Pfaff et al., "Extending Networking into the Virtualization Layer," *HotNets*, 2009.

[37] Open vSwitch http://www.openvswitch.org

[38] OpenStack. http://www.openstack.org

[39] B. Davie and J. Gross, "A Stateless Transport Tunneling Protocol for Network Virtualization (STT)," draft-davie-stt-02, Aug. 31th, 2012. http://tools.ietf.org/html/draft-davie-stt-02

[40] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol," draft-pfaff-ovsdb-proto-00, Aug. 20th, 2012. http://tools.ietf.org/html/draft-pfaff-

ovsdb-proto-00

[41] Draft Recommendation of Y.FNsdn, "Framework of Software-Defined Networking for Carrier Networks in Future Networks," ITU-T.

[42] Draft Recommendation of Y.FNsdn-fm, "Requirement of Formal Specification and Verification Methods for SDN," ITU-T, 2012.

[43] M.-K. Shin et al., "Formal Specification for Software-Defined Networks (SDN)," draft-shin-sdn-formal-specification-01, June 29th, 2012. http://tools.ietf.org/html/draft-shin-sdn-formal-specification-01

[44] M. Kang et al., "Formal Specifications for Software-Defined Networking," *Proc. 7th Int. Conf. Future Internet Technol. (CFI)*, Sept. 2012, pp. 51-52.

[45] T. Benson, A. Akella, and A. Shaikh. "De-mystifying Configuration Challenges and Trade-offs in Network-based ISP Services," *SIGCOMM Comput. Commun. Review*, vol. 41, no. 4, Aug. 2012, pp. 302-313.

[46] R. Mahajan, D. Wetherall, and T. Anderson. "Understanding BGP Misconfiguration," *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002, pp. 3-17.

[47] Network Heresy, "What Might an SDN Controller API Look Like? (and should we standardize it?)," Aug. 9th, 2011. http://networkheresy.com/2011/08/09/what-might-an-sdn-controller-api-look-like-and-should-we-standardize-it/

[48] OFN, "Network Function Virtualisation group," Oct. 23th, 2012. https://www.opennetworking.org/?p=380&option=com_wordpress&Itemid=72

[49] Broadband World Forum, 2013. http://www.broadbandworldforum.com/conference/presentations/