논문 2011-48TC-9-5

KREONET OpenFlow 네트워크 테스트베드 기반의 QoS 라우팅 경로 제어 구현

(Implementation of a QoS routing path control based on KREONET OpenFlow Network Test-bed)

김 승 주*, 민 석 홍*, 김 병 철**, 이 재 용**, 홍 원 택***

(Seung Ju Kim, Seok Hong Min, Byung Chul Kim, Jae Yong Lee, and Won Taek Hong)

요 약

미래인터넷은 효과적인 이동성 처리, 유연한 트래픽 엔지니어링 기술 및 다양한 새로운 응용 지원이 이루어져야 한다. 이에 따라 많은 트래픽 엔지니어링 기술들이 제안되고 개발되어 왔지만 이를 실제 운용 중인 상용 인터넷 망에 적용하는 것은 매우 어려운 것이 사실이다. 이 문제를 해결하기 위해 다양한 네트워킹 관련 응용을 지원하는 컨트롤러를 사용해 망의 장비를 제어할 수 있는 OpenFlow 기법이 제안되었다. 이는 소프트웨어 적으로 정의된 망으로 연구자들이 자신 만의 트래픽 엔지니어링 기법을 컨트롤러에 적용하여 그 유용성을 검증할 수 있다. 한편 고속 패킷 처리를 지원하는 OpenFlow 망의 구축을 위해 4개의 1G 인터페이스를 가지는 프로그래밍 가능한 NetFPGA 카드와 상용 Procurve 스위치 들이 사용될 수 있다. 본 논문에서는 하드웨어 가속 기능이 지원되는 NetFPGA 카드와 Procurve 스위치를 KREONET 망에 적용한 OpenFlow 테스트베드를 구축하고 가장 보편적인 QoS 라우팅 기법인 CSPF 알고리즘을 구축한 대규모 테스트베드 상에 적용하여 멀티미디어 트래픽 엔지니어링 기법의 성능 및 유효성 검증을 수행하였다.

Abstract

Future Internet should support more efficient mobility management, flexible traffic engineering and various emerging new services. So, lots of traffic engineering techniques have been suggested and developed, but it's impossible to apply them on the current running commercial Internet. To overcome this problem, OpenFlow protocol was proposed as a technique to control network equipments using network controller with various networking applications. It is a software defined network, so researchers can verify their own traffic engineering techniques by applying them on the controller. In addition, for high-speed packet processing in the OpenFlow network, programmable NetFPGA card with four 1G-interfaces and commercial Procurve OpenFlow switches can be used. In this paper, we implement an OpenFlow test-bed using hardware-accelerated NetFPGA cards and Procurve switches on the KREONET, and implement CSPF (Constraint-based Shortest Path First) algorithm, which is one of popular QoS routing algorithms, and apply it on the large-scale testbed to verify performance and efficiency of multimedia traffic engineering scheme in Future Internet.

Keywords : Future Internet, OpenFlow, NetFPGA, NOX controller, QoS routing, CSPF

* 학생회원, ** 평생회원, 충남대학교 정보통신공학과
(Chungnam National University)
*** 정회원, 한국과학기술정보연구원
(Korea Institute of Science and Technology Information)
※ 이 논문은 2010 년도 한국과학기술정보연구원 연구개발사업 위탁과제 및 방송통신위원회의 미래인터넷 인프라를 위한 가상화 지원 프로그래머블 플랫폼 및 핵심원천 기술 개발 사업의 연구결과로 수행되었음
(KCA-2011-09913-05006)
접수일자: 2011년3월15일, 수정완료일: 2011년9월16일

Ⅰ. 서 론

근래의 네트워크에 비해 비교적 고정적이고 작은 규모의 연구망으로부터 시작된 인터넷은 오랜 기간의 진화 과정을 통해 현재에 이르게 되었다. 진화는 사용자의 요구와 서비스의 특성을 만족하는 방향으로 진행되어 이동성 지원, 신뢰성 있는 통신, 실시간 전송과 같은 프로토콜들이 추가되었다. 그러나 아직도 IP 주소의 고갈, 라우팅 테이블의 확장성, 신뢰성 있는 이동성 처리, IP 계층의 과부하 등의 문제점이 발생하고 있으며 연구자들은 이를 해결하고자 여러 프로토콜이나 알고리즘들을 개발하여 실험하고 있다. 하지만 이런 해결책들을 실제 운용중인 인터넷 망에 적용하여 실험하기는 거의 불가능하다. 개발한 프로토콜이나 알고리즘을 각 사업자가 관리하는 인터넷 망에 적용하기도 어려울 뿐만 아니라 이것을 광범위한 망에 배치하는 것 역시 불가능하다. 만약 일부 인터넷 망에 적용되었다 할지라도 만에 하나 잘못될 경우, 망은 혼잡에 빠지고 보안 문제가 발생하거나 사용자에게 불편을 초래할 수 있다. 그렇기 때문에 대부분의 개발 프로토콜이나 알고리즘은 가상의 시뮬레이션 툴로 확인하고 기능을 검증하는 것이 보통이다. 따라서 이들을 실제 망에 적용했을 경우, 기존 망과의 호환성 문제나 고려하지 못했던 오작동 같은 문제점이 발생할 수 있다. 또한, 개개의 연구자가 고가의 네트워크 장비를 사들여 망을 설치하고 실험하는 것 역시 불가능하다.

스탠포드 대학에서 개발한 OpenFlow[1]는 이런 점을 고려해 기존 망의 일반 트래픽과 실험적인 트래픽을 따로 관리하며 일반 PC에서도 구현할 수 있는 소프트웨어 기반의 네트워크 구현을 가능하게 한다. 일반 이더넷 카드만 여러 개 있다면 이것을 평소 우리가 사용하던 PC에 장착하여 라우터나 스위치 같은 네트워크 장비로 활용할 수 있다. OpenFlow는 주로 스위치 기능에 초점을 두고 있어 OpenFlow 스위치로도 많이 불리지만 사용자의 목적에 따라 다른 목적의 네트워크 장비(라우터, 트래픽 모니터 등)로도 전환이 가능하다. 또한 OpenFlow 네트워크 상의 일반 인터넷 트래픽과 실험을 위해 발생한 트래픽을 각각 따로 처리할 수 있어 실제 인터넷 망에 하나의 장비로 활용할 수 있다.

비슷한 시기에 개발된 NetFPGA[2]는 하나의 네트워크 장치로 프로그래밍이 가능한 이더넷 카드이다. NetFPGA에는 4개의 물리 포트가 있으며 각각은 1Gbps의 대역폭을 갖는다. 이를 활용해 PC에 NetFPGA를 장착하고 OpenFlow를 설치하면 OpenFlow 네트워크를 구성할 수 있으며 여기에 OpenFlow의 관리와 제어를 담당하는 NOX 컨트롤러[3]를 추가하면 OpenFlow 테스트베드가 구성된다. NOX 컨트롤러는 관리자 역할을 하는 소프트웨어로 각 OpenFlow 장비들의 기능을 설정하고 관리하는 역할을 한다. 따라서 NOX 컨트롤러에 실험자가 요구하는 기능을 프로그래밍하면 OpenFlow 네트워크는 그에 따라 동작하게 되는 것이다.

이런 컨트롤러 기반의 프로그래머블 네트워크는 현 인터넷의 특징인 대용량, 고대역폭, 실시간 트래픽에 의해 발생하는 서비스 개선 요구에 대한 해결책을 제공한다. 높은 대역폭을 차지하며 실시간으로 전송되는 플로우가 늘어나는 현 시점에서, 통신은 요구 대역폭을 만족하면서 최단의 경로를 통해 이루어져야 한다. 그러기 위해서는 트래픽 엔지니어링을 통한 QoS 라우팅 기술이 필요하며, 이런 기술을 프로그래밍하여 네트워크에 구현할 수 있는 OpenFlow는 미래 인터넷에서의 서비스 개선을 위한 테스트베드로 적합하다.

본 논문에서는 트래픽 엔지니어링[4]의 하나인 QoS 라우팅을 OpenFlow 네트워크에 활용하기 위해 OpenFlow 컨트롤러 상에 구현하였고 그 성능을 분석하였다. 사용자의 요구에 따라 플로우를 구분하고 원하는 대역폭을 갖도록 경로를 설정하여 신뢰성 있는 서비스를 보장하는 CSPF 알고리즘[5]을 적용하였다. 다음 Ⅱ장에서는 관련 기술인 OpenFlow와 NOX 컨트롤러의 특성 및 구성에 대해 설명하였고 Ⅲ장은 CSPF 알고리즘의 설명과 구현 사항에 대해 기술하였다. Ⅳ장은 OpenFlow 테스트베드의 구성과 실험 시나리오 및 실험 결과를 나타내었으며 마지막 장은 간략한 결론 및 추후 개선사항을 설명하며 논문을 마친다.

Ⅱ. OpenFlow 네트워크

1. OpenFlow[1]

현 인터넷의 문제점을 해결하고 미래 인터넷 기술 개발에 누구나 손쉽게 참여할 수 있는 여건을 만들고자 개발된 OpenFlow 프로토콜은 연구자로 하여금 다양한 실험을 가능하게 하며 자료 및 기술의 공유를 위한 열린 커뮤니티를 제공하고 있다.

OpenFlow는 PC 기반의 네트워크 장비 구현 기술로

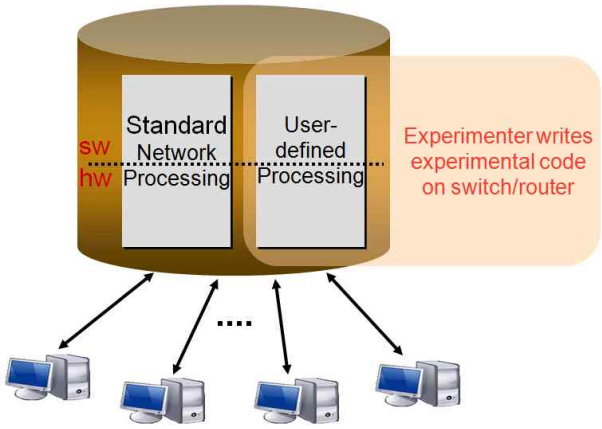


그림 1. OpenFlow 프로토콜의 트래픽 분리
Fig. 1. Traffic separation of OpenFlow protocol.

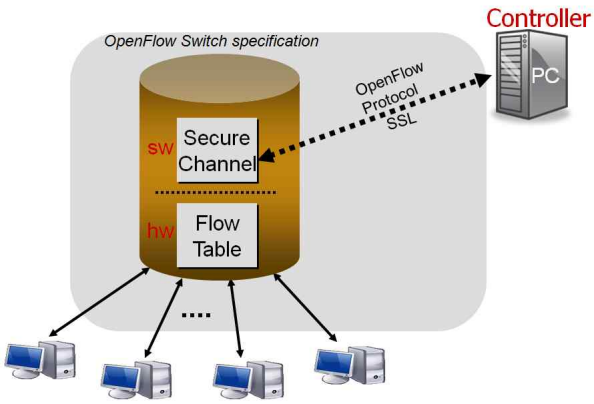


그림 2. OpenFlow에서 S/W와 H/W의 역할
Fig. 2. Functions of S/W and H/W in OpenFlow.

일반 PC에 여러 이더넷 카드만 있다면 소프트웨어를 설치함으로 구현이 가능하다. 따라서 고가의 장비 없이도 네트워크를 구성할 수 있어 경제적이다. 그러나 더 좋은 성능의 OpenFlow 망을 꾸미고 싶다면 OpenFlow를 지원하는 상용스위치나 NetFPGA 카드를 별도로 설치하여 구현할 수 있다. 또한, OpenFlow는 <그림 1>처럼 일반적인 네트워크 기능과 연구자의 실험을 별개로 진행하기 때문에 기존 망과 연동하여 사용할 수 있는 장점이 있다. OpenFlow 스위치는 컨트롤러에 의해 동작하는 네트워크 장비이다. 컨트롤러에서 구현된 기능에 따라 필요한 명령이 전해지고 각 OpenFlow 스위치는 그에 따라 움직이게 된다.

<그림 2>는 컨트롤러와 OpenFlow의 운영 관계를 나타낸다. 초기 컨트롤러와 OpenFlow들 간에 SSL이나 TCP를 통해 연결이 맺어지면 그 이후부터는 둘 사이에 OpenFlow 프로토콜 메시지를 교환하여 Secure Channel을 생성한다. Secure Channel은 소프트웨어 계

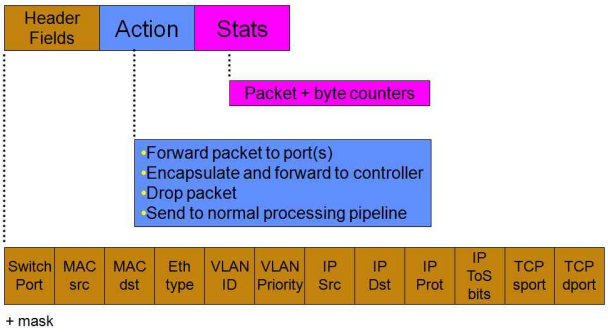


그림 3. OpenFlow의 Flow Table
Fig. 3. OpenFlow's Flow Table.

층에서 수행되며 컨트롤러와 OpenFlow들 간에 명령이나 정보를 주고받는 역할을 한다. 받은 명령은 하드웨어 계층에서 수행되며 이 명령의 형태는 Flow Table이란 명칭으로 실행된다.

Flow Table은 <그림 3>과 같은 구조로 되어 있다. Flow Table이란 명칭에서 알 수 있듯 플로우를 연구자의 목적에 맞게 분리하고 관리하는 기능을 한다. Header Fields는 각 계층의 ID에 따라 플로우를 정의하는 역할을 하며 10-Tuple이라 불리고 현재 OpenFlow 버전에서는 12-Tuple까지 지원이 가능하다. 플로우 정의의 단위는 최소 MAC ID로 시작해 최대 TCP 포트번호까지이다. Action은 정의한 플로우를 어떻게 처리할 것인지를 나타내며 Stats은 정의된 플로우의 통계치를 일컫는다. 따라서 실험자가 원하는 계층만을 사용해 패킷을 전송할 수 있어 스위치나 라우터 개념의 포워딩이 모두 가능한 것이다. 만약 서비스에 따라 플로우를 나누고 각각 다른 경로를 통해 플로우를 전송하고 싶다면 TCP 포트까지 사용해 Flow Table을 작성하면 된다. Flow Table의 Stats에는 각 물리포트의 통계치를 반환하는 기능이 있다. 그 중 'port_stats'란 항목을 사용해 링크의 가용 대역폭을 확인할 수 있으며 이를 통해 대역폭을 제약사항으로 하는 CSPF 알고리즘을 구현할 수 있다.

2. NOX 컨트롤러[3]

OpenFlow 망은 컨트롤러가 관리하는 망으로 컨트롤러가 없을 경우 일반 스위치와 같은 동작만을 하게 된다. 따라서 OpenFlow 망에서 컨트롤러의 역할은 중요하다. 주로 OpenFlow 컨트롤러로 NOX 컨트롤러를 사용하며 주요 코드는 C++로 되어 있고 간단한 인터페이스를 제공하기 위해 Python도 쓰인다.

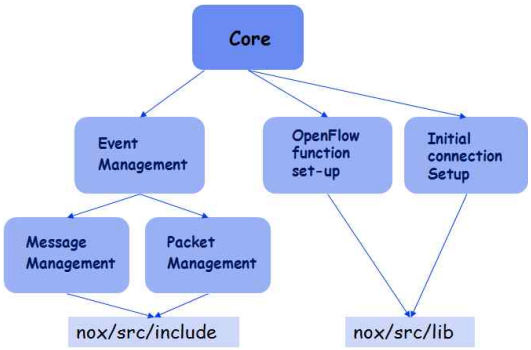


그림 4. 주요 코드 구성
Fig. 4. Core code composition.

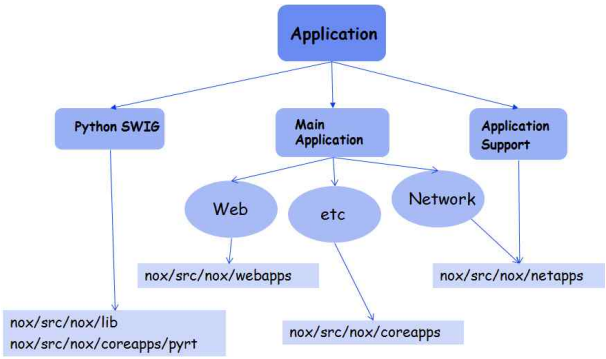


그림 5. 어플리케이션 관련 코드
Fig. 5. Application-related code.

NOX 컨트롤러의 코드 구성은 <그림 4>의 주요 코드 부분과 이것들을 활용한 <그림 5>의 어플리케이션 관련 코드들이 있다. 주요 코드는 OpenFlow와의 Secure Channel 생성과 OpenFlow 기능 설정 및 통계치 요청, 망 상황의 관리, OpenFlow 메시지 전송 등의 기능을 한다.

OpenFlow 기능 설정은 STP(Spanning Tree Protocol) 사용여부나 IP fragmentation에 대한 처리, 통계기능 활성화 등을 설정하며 망 상황의 관리는 OpenFlow 네트워크에 패킷이 들어오거나 만료되었을 경우, 또는 OpenFlow 메시지가 발생했을 경우 이를 감지해 어떻게 처리할지 물어보는 역할을 한다. 또한 사용자가 정한 명령을 Flow Table로 만들어 메시지로 OpenFlow에게 전송하는 것 역시 NOX의 주요 기능이다. 어플리케이션 관련 코드는 위의 주요 코드들을 활용해 실제 하나의 네트워크 기능을 하도록 만든 사용자 정의의 코드이다. 또한 여러 어플리케이션 코드들을 한꺼번에 NOX에서 구동할 수도 있는데 그 예로 STP 어플리케이션[6]과 NOX에서 제공하는 기본 스위치인 'pyswitch' 어플리케이션의 조합을 들 수 있다. 또 다른 예로는 하나의 메인 어플리케이션을 만들고 부속 어플리케이션으로 다른 어플리케이션을 추가하면 메인 어플리케이션이 구동하는 동안 부속 어플리케이션도 같이 동작하는 효과를 얻을 수 있다. 즉, 루프를 방지하면서 OpenFlow를 스위치로 사용하려면 STP 어플리케이션과 'pyswitch' 어플리케이션을 같이 사용하면 되는데, 이때 STP 어플리케이션에는 'discovery'란 어플리케이션이 부속 어플리케이션으로 지정되어 있어 따로 구동시키지 않아도 함께 동작하는 결과를 낼 수 있다.

어플리케이션 코드의 구성은 C++로 된 함수들을 간단하게 Python에서 사용하기 위한 SWIG(Simplified Wrapper and Interface Generator) 코드와 메인 어플리케이션 코드, 그리고 어플리케이션을 지원하는 부속 어플리케이션 코드로 나눌 수 있다. 메인 어플리케이션 코드는 네트워크 라우팅과 웹 관련 코드로 나눌 수 있으며 그 밖의 예제 코드나 모니터링 관련 코드, Python 메인 함수 코드 등이 그 외의 코드로 존재한다. <그림 4>와 <그림 5>에서 맨 하위 텍스트는 각 코드가 위치한 디렉토리의 이름을 나타낸 것이며 코드 구성은 NOX의 버전에 따라 조금 다른 부분이 있을 수 있다.

본 논문에서는 새로운 메인 어플리케이션을 만들어 CSPF 알고리즘을 구현하였으며 NOX에서 제공해주는 STP 어플리케이션을 함께 사용하였다.

Ⅲ. OpenFlow QoS 라우팅 알고리즘 구현

1. CSPF 알고리즘[5]

OpenFlow 프로토콜은 Flow Table에서 각 스위치의 통계치를 NOX로 보고하는 기능이 있다. 통계자료로는 [표 1]과 같이 여러 항목이 있으나 실제 NetFPGA와 상용 스위치 장비를 연동하여 실험해 본 결과, 물리포트의 누적 바이트 수와 패킷 수가 신뢰성 있고 정확하였다. 이런 OpenFlow 프로토콜의 통계 기능과 이것을 바탕으로 획득한 대역폭, 유선망에서의 지연과 손실이 가용 대역폭에 비해 고려 가치가 떨어진다는 사실을 감안하면, OpenFlow 네트워크에서의 QoS 라우팅은 가용 대역폭에 제약을 둔 CSPF 알고리즘이 적합하다는 결론에 이르렀다.

따라서 본 논문은 가용 대역폭을 제약사항으로 한 CSPF 알고리즘의 구현에 목적을 두었으며 이를 위해 통계 자료에 근거한 가용 대역폭을 산출하였다. CSPF 알고리즘은 <그림 6>에서 보는 것과 같이 제약사항에

표 1. Flow Table의 통계자료
Table 1. Statistics of Flow Table

Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

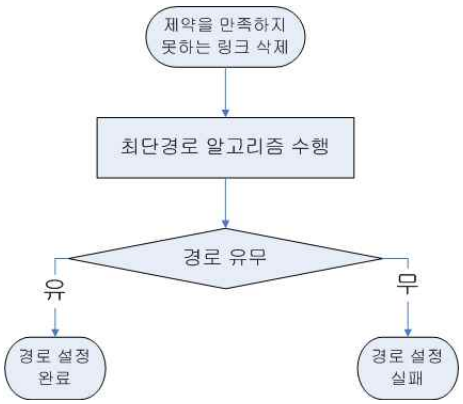


그림 6. CSPF 알고리즘
Fig. 6. CSPF algorithm.

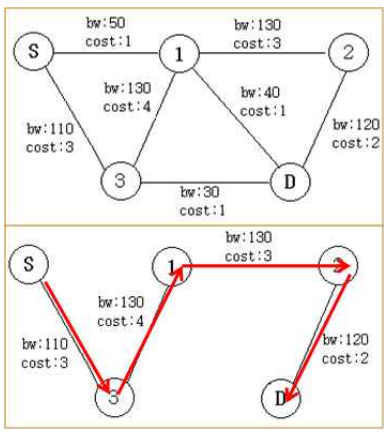


그림 7. 가용 대역폭 기반의 CSPF 알고리즘
Fig. 7. Available bandwidth based CSPF algorithm.

만족하지 않는 네트워크 요소를 제거한 후 최단 경로를 찾아내는 알고리즘으로, 제약사항으로는 지연, 손실, 가용 대역폭 등을 꼽을 수 있다. <그림 7>에서처럼 가용 대역폭을 예로, 사용자가 원하는 가용 대역폭이 100Mbps라면 그 이하의 링크들은 제거한 후 최단경로 알고리즘을 수행하는 기법이 CSPF 알고리즘이다.

최단경로 알고리즘으로는 Dijkstra 알고리즘[7]을 선택하였으며 본 실험에서 각 링크의 코스트는 모두 1로 설정하였고, QoS를 보장받지 못하는 플로우의 경우는 단순히 Dijkstra 알고리즘에 의해 생성된 경로로 패킷이 전송된다.

2. 구현 사항

가. OpenFlow 네트워크의 토폴로지 획득

OpenFlow 프로토콜의 지원 어플리케이션 중 'discovery'라는 어플리케이션은 LLDP(Link Layer Discovery Protocol) 패킷을 생성해 OpenFlow 장비로 전송해주는 기능을 갖고 있다.

<그림 8>에서와 같이 NOX 컨트롤러는 각각의 OpenFlow 스위치로 물리포트 개수만큼의 LLDP 패킷을 보내준다. 그 후 LLDP 패킷들을 받은 OpenFlow 스위치는 <그림 9>처럼 LLDP 패킷에 적힌 포트번호에 맞게 LLDP 패킷을 해당 포트로 전송한다. 각각의 LLDP 패킷은 각 포트에 연결된 노드로 전송될 것이며 각 노드는 이 패킷을 처리할 것이다. 이 때 일반 터미널 단말이나 NOX 같은 경우는 이 패킷을 무시하고 버리지만 LLDP 패킷을 받은 노드가 OpenFlow 스위치라면 이를 NOX에 'packet_in'이라는 OpenFlow 메시지로 보고하게 된다. 'packet_in'의 기능은 패킷이 OpenFlow

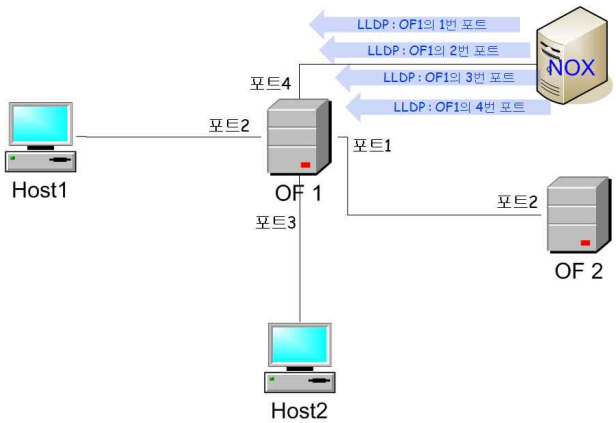


그림 8. NOX에서의 LLDP 패킷 전송
Fig. 8. Forwarding LLDP packets from NOX.

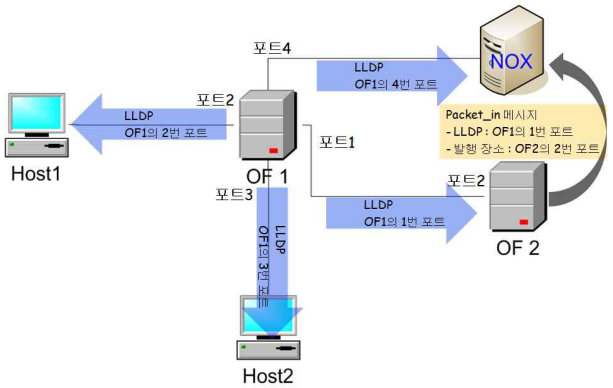


그림 9. OpenFlow 네트워크에서의 LLDP 패킷 전송
Fig. 9. Forwarding LLDP packets from OpenFlow Network.

네트워크에 들어올 경우 패킷을 'packet_in'이라는 메시지로 감싸 NOX에 보고하는 것이다. 이 메시지 안에는 패킷을 받은 OpenFlow 스위치의 주소와 물리포트의 정보가 담겨 있는데 이를 통해 네트워크의 토폴로지를 획득할 수 있다. LLDP 패킷을 'packet_in'이라는 메시지로 NOX에 보고할 경우 <그림 9>에서 볼 수 있듯이 OpenFlow 스위치 1의 1번 포트와 OpenFlow 스위치 2의 2번 포트가 서로 연결되어 있다고 NOX에서 판단할 수 있다. 이런 방식을 사용하면 토폴로지가 확장하더라도 NOX는 언제나 정확한 토폴로지를 획득할 수 있다.

나. OpenFlow 네트워크의 가용 대역폭 획득

이전에 설명했듯이 OpenFlow는 통계 기능을 갖고 있으며 그 중 물리포트의 누적 바이트 수를 사용해 가용 대역폭을 산출해 낼 수 있다. 실험자가 원하는 주기를 정해 OpenFlow 스위치로 'port_stats_request' 메시지를 보내면 OpenFlow 스위치는 그동안 포트 별로 누적해 갖고 있던 통계치를 'port_stats_reply' 메시지로 NOX에 보고한다. 주기의 경우 실험자가 개발하고자하는 목적에 맞게 조절할 수 있으며 본 실험에서는 1초 간격으로 정하였다. 받은 메시지의 통계 자료를 분류하여 사용하고자 하는 누적 바이트 수만을 추려낸 후 현재 받은 누적 바이트 수를 A로 저장한다. 다음 주기에 같은 포트에서 받은 누적 바이트 수가 들어오면 그 값으로 A를 갱신하고 이전 주기에 받은 누적 바이트 수는 B에 저장한다. 이 값을 사용하면 (식 1)처럼 한 포트의 사용 대역폭을 계산할 수 있다.

$$\text{사용 대역폭} = \frac{A - B}{\text{한 주기}} \quad (1)$$

실험에 쓰인 NetFPGA나 상용 장비인 ProCurve 스위치는 모두 1Gbps의 대역폭을 갖기 때문에 가용 대역폭은 고정적으로 1G에서 사용대역폭을 빼준 값을 사용하였다. 이렇게 산출한 가용 대역폭은 개발 어플리케이션에서 CSPF에서의 제약사항으로 쓰인 가용 대역폭과 비교되며 제약사항보다 작다면 그 링크는 경로 계산 시 제거된다.

다. CSPF 기능 구현

위에서 획득한 두 가지 정보를 사용해 CSPF 알고리즘을 Python 프로그래밍 언어로 구현하였다. CSPF 기능의 구현은 크게 가지치기(pruning) 부분과 Dijkstra 경로 설정 부분으로 나뉜다.

가지치기 부분은 제약사항으로 지정된 대역폭에 만족하지 못하는 링크를 최단 경로 계산 전 토폴로지 변수에서 지워주는 기능을 한다. 따라서 경로를 계산하는 동안, 제약사항을 만족하지 못한 링크는 최단 경로 알고리즘 수행에서 제외되는 것이다.

다음 단계는 가지치기로 얻은 토폴로지에 최단 경로 설정 알고리즘인 Dijkstra를 적용하는 것이다. 일반 플로우와 QoS 플로우의 경로 계산 시 차이점은 가지치기 과정의 수행 여부로 일반 플로우의 경우는 가지치기 없이 바로 최단경로 알고리즘을 수행한다. QoS 플로우는 가지치기 후 최단 경로 알고리즘을 수행하여 경로가 존재하면 경로 설정은 성공하게 되고, 존재하지 않는다면 실패한다. 실패한 경로는 데이터베이스에 저장된 후 추후 대역폭이 변하면 그때 다시 경로 검색을 수행한다.

라. 네트워크 상황에 따른 대처 기능

CSPF 알고리즘의 구현에 추가적으로 다양한 네트워크 상황에 대처하는 관리 기능을 추가하였다. 첫째로 CSPF의 경우 제약사항에 만족하지 못하는 상황이 발생하면 경로 설정이 실패하는 경우가 있다. 이 때 개발 어플리케이션에서는 실패한 패킷에 대한 정보를 데이터베이스에 저장한 후 주기적으로 망을 감시하여 적당한 경로가 나타나면 경로를 재설정해 패킷을 전송하는 기능을 추가하였다. 이 기능은 주기적으로 각 링크의 가용 대역폭을 점검함으로써 가능하다.

두 번째 기능은 전송하고 있던 경로 상의 링크가 끊어졌을 경우 다른 경로를 탐색하고 경로를 바꿔주는 것이다. 여기에서는 LLDP 패킷의 전송 상태를 통해 링크의 단절을 인식할 수 있는데 일정 주기로 전송되던

LLDP 패킷이 어느 시간 이상으로 나타나지 않는다면 그 링크를 끊어진 것으로 판단하는 것이다.

또 다른 기능으로는 호스트의 이동을 감지하고 옳은 경로로 재설정해주는 것이다. 이 때 한 가지 문제가 있다. 한 호스트에서 발행한 패킷이 OpenFlow 네트워크를 경유하면 거쳐 간 모든 OpenFlow 장비는 이를 NOX에 보고해 NOX는 이를 보고한 모든 장비에 호스트가 위치한다고 판단할 수 있다. 이 문제의 해결을 위해 다음의 정보를 사용하여 정확한 위치를 추적하였다. NOX에서는 이미 LLDP 패킷을 사용해 OpenFlow 장비끼리 서로 어느 포트에 연결되어 있는지 알고 있다. 그리고 패킷을 생성한 호스트는 OpenFlow 장비끼리 연결된 포트가 아닌 다른 포트에 연결되어 있다. OpenFlow 장비에 패킷이 들어왔다는 보고 메시지에는 몇 번 장비의 몇 번 포트에 패킷이 전송됐는지 적혀있기 때문에 패킷이 OpenFlow 장비끼리 연결된 포트로 들어왔는지 알 수 있으며 이 경우 OpenFlow 장비는 중계 노드이며 호스트에 직접 연결된 것이 아니라고 판단한다. 이 방법을 사용해 호스트의 위치를 확인할 수 있으며 이동 역시 감지할 수 있다. 위치가 파악된 호스트가 패킷을 전송하면서 다른 스위치로 옮겨간 경우, 이 호스트는 다른 OpenFlow 장비의 다른 포트에 재연결될 것이다. 호스트에서는 계속 패킷을 내보내므로 재연결된 OpenFlow 장비는 패킷이 들어왔다는 메시지를 NOX에 보고할 것이다. 이 때 보고한 내용 역시 장비의 ID와 포트 번호가 포함되어 있어 NOX에서는 이 포트가 다른 OpenFlow 장비와 연결된 포트인지 확인한다. OpenFlow 장비끼리 연결된 포트가 아니라면 패킷을 생성한 호스트는 이동했다고 판단한다. 이 경우 NOX는 기존의 경로를 전부 삭제한 후 새로운 경로를 탐색해 연결을 지속시킨다.

마지막으로 루프 방지 기능이다. OpenFlow 프로토콜은 Flow Table에 정의되어 있지 않은 패킷이 들어오게 되면 그것들을 일일이 'packet_in'으로 감싸 NOX로 보고한다. 이런 특성으로 인해 종종 루프가 발생한다. 실험한 바로는, Flow Table에 없는 고대역의 UDP 패킷이 OpenFlow 네트워크에 들어올 경우 상당히 많은 패킷들이 NOX로 보고된다. 이 때 보고 메시지는 OpenFlow 프로토콜의 Secure Channel에서 처리하고 이 채널은 소프트웨어 계층에서 동작하다보니 LLDP 패킷 처리에 지장을 줘 LLDP 패킷의 전송이 늦어지게 된다. 그러면 STP 어플리케이션에서는 그 링크가 끊어졌다고 판단하고 그에 따라 망은 루프가 없는 토폴로지로 잘못 인식되어 STP는플러딩하는 패킷들을 더 이상 막지 않는다. 이로 인해 루프가 생기는데 이 경우를 대비해 패킷의 초당 플러딩 횟수를 측정해 적정 수준 이상이 되면 차단하는 기능과 한꺼번에 밀려오는 패킷에 대비해 플로우의 첫 번째 패킷만을 통해 경로를 계산하고 Flow Table을 내려주는 기능을 추가하였다.

위에서 언급한 기능들의 동작 시나리오와 성능 측정 결과는 Ⅳ장에 자세히 나타나 있다.

Ⅳ. 테스트 베드 망 구성 및 성능 측정

1. OpenFlow 테스트베드 구성

테스트베드는 로컬 망과 KREONET 망, 두 가지를 대상으로 구현하였으며 로컬 망은 <그림 10>과 같이 NetFPGA 기반의 OpenFlow 장비 2대와 HP ProCurve 스위치 1대를 사용하였고, KREONET 망은 <그림 11>과 같이 HP ProCurve 스위치만을 사용하였다. 로컬 망의 경우 사설 IP를 할당하여 외부 연결 없이 실험하였고 KREONET 망의 경우는 Secure Channel은 공용 IP로 설정하고 각 단말은 사설 IP를 사용해 실험하였다. KREONET 망의 테스트베드는 실제로 각 장비들이 서울과 대전, 광주에 배치되어 있으며 각 장비들은 백본망을 통해 연결되어 있다.

두 테스트베드 모두 링크의 최대 대역폭은 1Gbps이며 마지막 실험인 비디오 전송 테스트의 경우만 100Mbps로 제한하였다. 테스트에 사용한 모든 트래픽은 UDP로 링크 최대 대역폭을 사용하였기 때문에 장비에 따라 다를 수 있으며 보통 900Mbps 정도이다. 비디

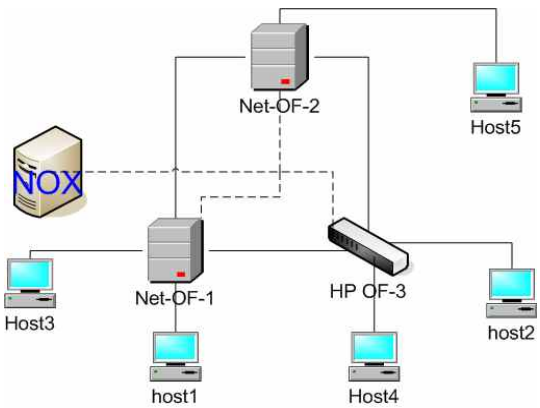


그림 10. 로컬 OpenFlow 테스트베드
Fig. 10. Local OpenFlow Test-bed.

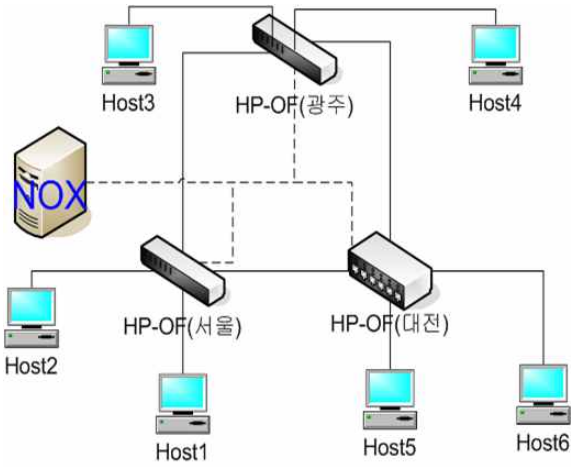


그림 11. KREONET OpenFlow 테스트베드
Fig. 11. KREONET OpenFlow Test-bed.

오 전송 실험에서는 백그라운드 트래픽을 85Mbps로 보내고 30Mbps 비디오 스트림을 전송하였다.

2. 실험 시나리오 및 결과

실험은 5가지 시나리오와 비디오 스트리밍 전송 시나리오로 구성되어 있다. 실험에 쓰인 툴로는 'iperf'[8]와 'DVTS'[9]를 사용하였으며 비디오 스트리밍 전송에는 별도로 bandwidth limiter가 사용되었다. Limiter의 사용 목적은 실험에 쓰인 30Mbps 비디오 스트림을 1Gbps 망에 전송할 경우 백그라운드 트래픽에 상관없이 비디오가 잘 전송되기 때문에 망의 최대 대역폭을 낮추기 위해 사용되었다.

시나리오 1은 개발 어플리케이션의 기본 기능을 검증하기 위한 실험으로 <그림 12>에서 실선은 백그라운

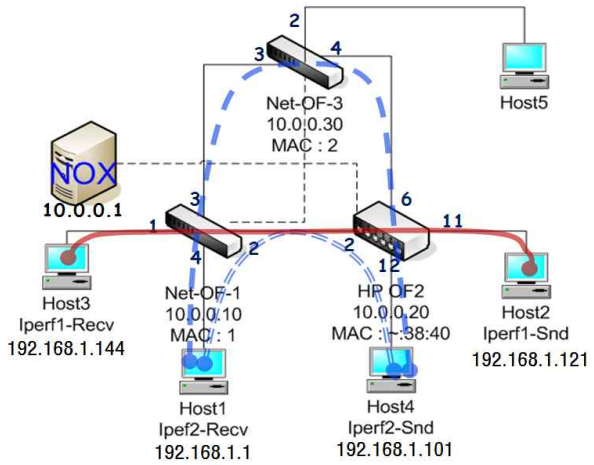
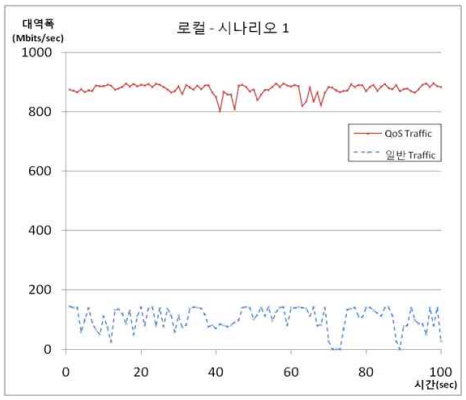
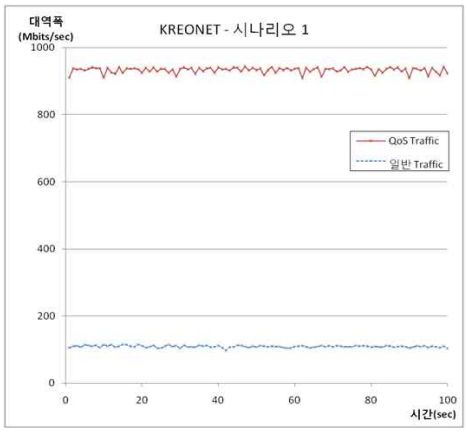


그림 12. 시나리오 1의 토폴로지
Fig. 12. Topology of Scenario 1.



(a) Local



(b) KREONET

그림 13. 시나리오 1의 성능 그래프
Fig. 13. Performance Graph of Scenario 1.

드 트래픽으로 가용 대역폭을 줄이기 위해 사용하였다.

이중 점선은 QoS를 보장받지 못하는 플로우로 Dijkstra 알고리즘을 통해 최단경로로만 설정되어 망의 상황을 반영하지 못해 충돌을 일으켜 <그림 13> 그래프의 아래 실선과 같이 대역폭이 낮게 나타난다. 일반 점선은 QoS를 보장받는 플로우로 CSPF 알고리즘을 통해 제약사항을 만족하는 경로로 우회하여 전송되는 것을 <그림 13> 그래프의 위쪽 실선을 통해 확인할 수 있다. <그림 13>의 (a)는 로컬 망에서의 실험 결과이며 (b)는 KREONET 망에의 결과로 ProCurve의 성능이 약간 더 좋은 것을 볼 수 있다.

시나리오 2는 QoS를 보장받은 플로우의 경로에 다른 백그라운드 트래픽이 발생할 경우 더 나은 경로로 이동하는 기능을 보여준다. <그림 14>에서처럼 기존에 일반 점선으로 표시된 QoS 플로우가 실선의 백그라운드 트래픽에 의해 충돌이 발생하면 QoS 플로우는 이중 점선으로 표시된 경로로 이동하는 내용으로 <그림 15>의

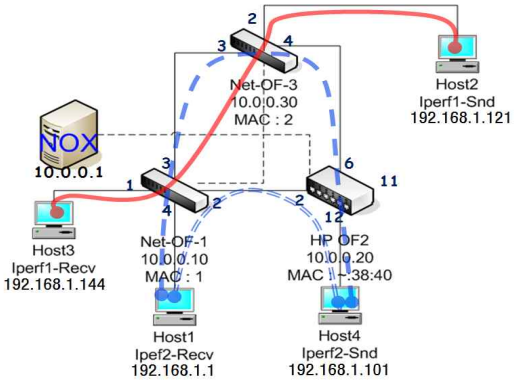
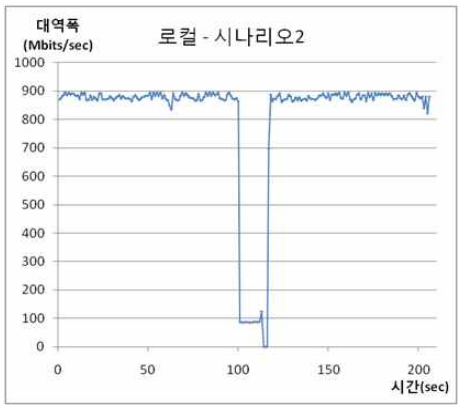
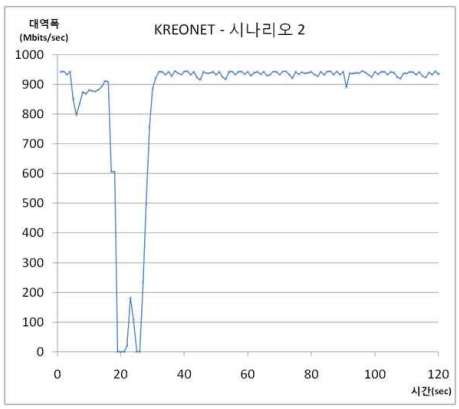


그림 14. 시나리오 2의 토폴로지
Fig. 14. Topology of Scenario 2.



(a) Local



(b) KREONET

그림 15. 시나리오 2의 성능 그래프
Fig. 15. Performance Graph of Scenario 2.

그래프에서 대역폭이 급격히 떨어지는 부분이 경로를 재설정하는 시간이다.

시나리오 3은 CSPF의 경로 설정 실패에 관한 것으로 <그림 16>에서 두 종류의 점선(Host3과 Host2 간, Host5와 Host6 간)은 백그라운드로 Host1에서 Host4로 갈 수 있는 모든 링크의 대역폭을 사용한다. 그러면

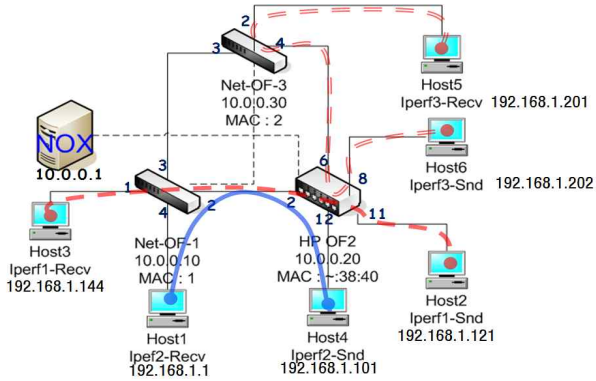
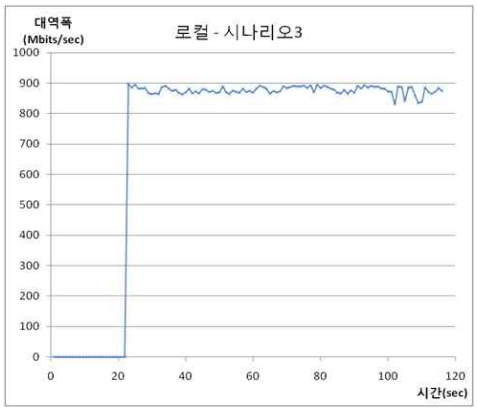
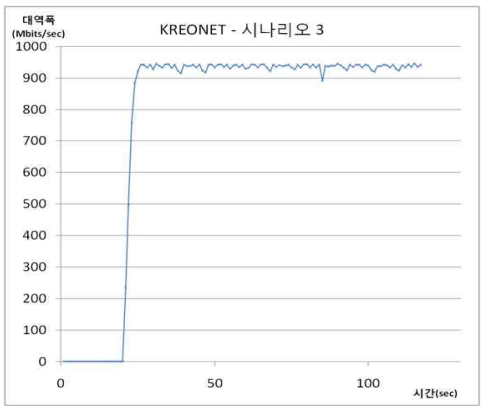


그림 16. 시나리오 3의 토폴로지
Fig. 16. Topology of Scenario 3



(a) Local



(b) KREONET

그림 17. 시나리오 3의 성능 그래프
Fig. 17. Performance Graph of Scenario 3.

CSPF의 경로 설정은 실패하게 되는데 일반 점선의 백그라운드 트래픽을 멈추면 그때 경로가 재설정되어 통신이 되는 것을 나타냈다. <그림 17>에서 초기 0의 대역폭은 백그라운드 트래픽이 없어질 때까지의 시간이다.

시나리오 4와 5의 경우 물리적으로 멀리 떨어진 KREONET 망에서는 수행할 수 없어 로컬 테스트베드

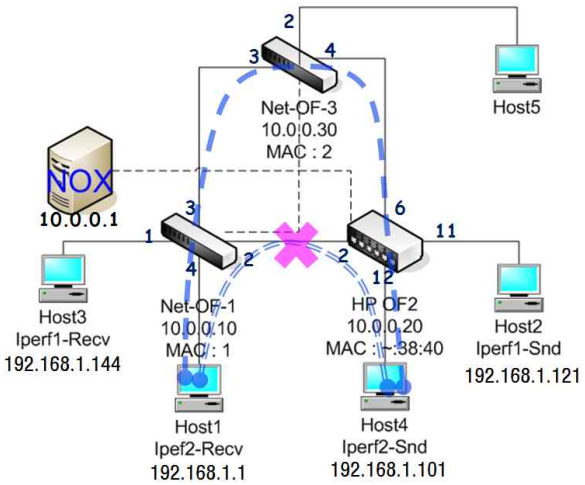


그림 18. 시나리오 4의 토폴로지
Fig. 18. Topology of Scenario 4.

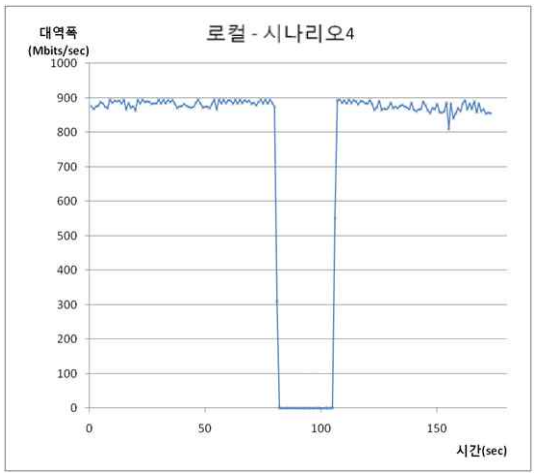


그림 19. 시나리오 4의 성능 그래프
Fig. 19. Performance Graph of Scenario 4.

에 대한 결과만 나타내었다.

시나리오 4는 통신 중이던 링크가 끊어졌을 경우 이에 대처하는 기능으로 <그림 18>에서 이중 점선의 플로우가 일반 점선으로 이동하는 것을 나타내며 <그림 19>의 그래프에서 대역폭이 0인 부분은 링크의 단절을 감지하고 경로를 재설정하는데 걸리는 시간이다.

시나리오 5는 단말의 이동에 따른 경로 재설정으로 <그림 20>에서처럼 실선에서 점선으로 경로가 재설정되며 <그림 21>에서의 대역폭이 0인 구간은 경로 재설정에 걸리는 시간이다.

마지막은 영상 전송에 대한 실험으로 시나리오 1을 적용하였고 결과는 전송된 화질의 선명도로 나타내었다. DVTS[9] 툴을 사용해 영상을 전송할 경우 최대 사용 대역폭이 30Mbps 밖에 되지 않아 1Gbps망에서는

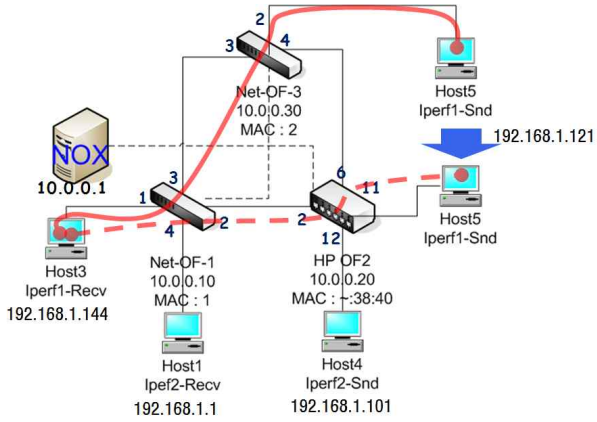


그림 20. 시나리오 5의 토폴로지
Fig. 20. Topology of Scenario 5.

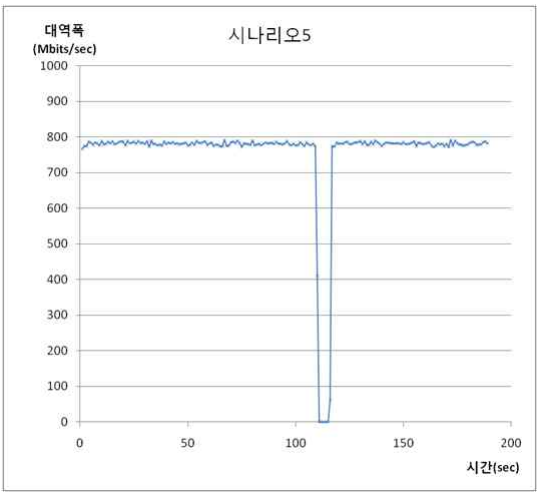


그림 21. 시나리오 5의 성능 그래프
Fig. 21. Performance Graph of Scenario 5.

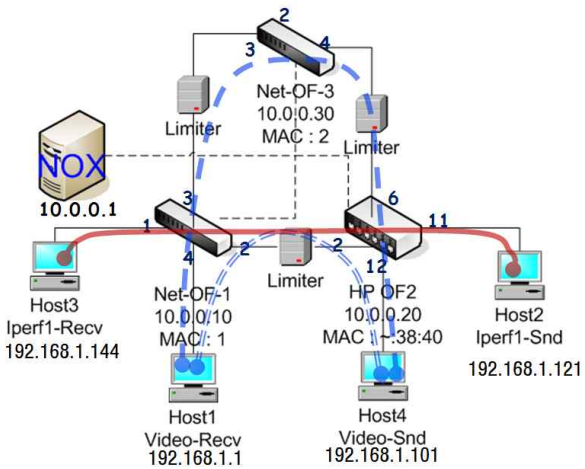
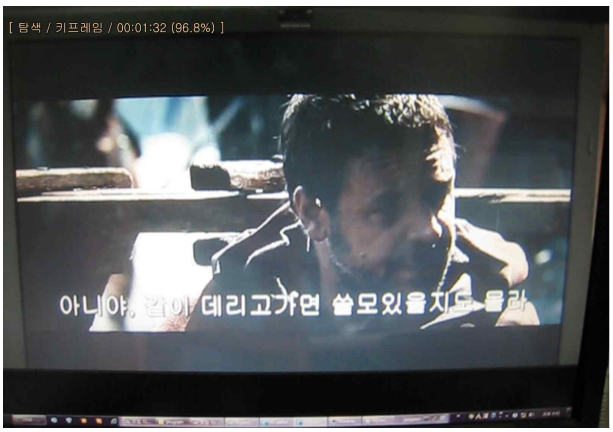


그림 22. 영상 전송 실험
Fig. 22. Video Streaming Test.

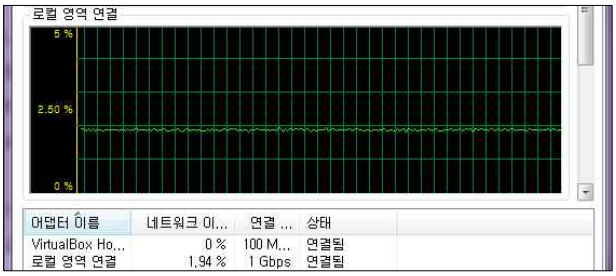
백그라운드가 아무리 많아도 전송된 화질에 변화가 없었다. 그래서 확실한 실험 결과를 위해 망을 100Mbps

(a) Non-QoS

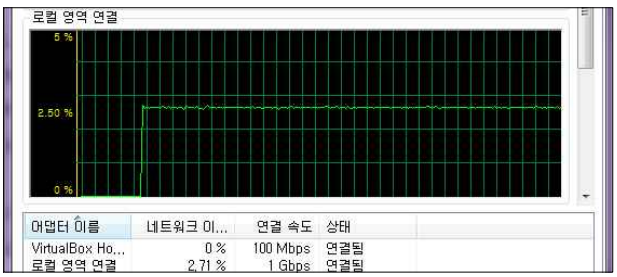


(b) QoS

그림 23. 영상 전송 화면
Fig. 23. Video Streaming Screen.



(a) Non-QoS



(b) QoS

그림 24. 영상 전송 대역폭
Fig. 24. Video Streaming Bandwidth.

로 바꿀 필요가 있었고 <그림 22>에서의 'Limiter'는 대역폭을 1Gbps에서 100Mbps로 제한해 주는 역할을 한다. <그림 23>에서처럼 QoS가 보장되는 것과 그렇지 않은 것의 차이를 영상을 통해 알 수 있다.

실험은 <그림 22>에서 실선으로 표현한 백그라운드 트래픽를 보내고 Host4에서 Host1로 비디오 영상을 전송하였다. QoS(일반 점선)와 Non-QoS(이중 점선) 각각 따로 실험하였다. <그림 22>에서 보듯이 QoS가 제공되지 않은 경우 백그라운드 트래픽으로 인해 영상의 끊김이 빈번히 발생하고 품질 저하가 일어났다. <그림 23>은 각각 전송된 영상의 대역폭을 윈도우의 작업관리자 창에서 확인한 것으로 Non-QoS 플로우의 경우 19Mbps정도의 대역폭을 나타내었고 QoS 플로우의 경우 영상 서비스가 요구하는 27Mbps의 대역폭을 나타내 QoS가 보장되는 것을 확인하였다.

V. 결 론

현 인터넷의 문제점에 대한 해결과 미래 인터넷의 개발에 이바지하고자 많은 연구자들이 노력하고 있으나 실제로 망에 적용하여 실험하기는 거의 불가능하다. 이런 상황을 개선하기 위해 미국의 GENI(Global Environment for Network Innovations) 클러스터 중 하나인 OpenFlow 그룹은 미래 인터넷을 위한 프로토콜 개발의 장으로, OpenFlow 프로토콜을 사용해 공용의 가상 망을 구축하여 많은 연구자들이 이 망을 통해 기술을 개발하고 적용하여 실험할 수 있도록 돕는다.

본 논문에서는 이런 OpenFlow 프로토콜을 사용하여 트래픽 엔지니어링 기술 중의 하나인 CSPF 알고리즘을 구현하여 대역폭을 만족하는 경로로 우회하는 QoS 라우팅이 가능함을 보였다. 또한, 링크의 단절, 호스트의 이동, QoS 경로 재설정 등의 네트워크 관리 및 제어 기능을 설계하여 네트워크 상황에 맞는 동적 라우팅 경로 제어를 구현하였다. 개발 초기의 설계 시점에서 고려하지 못했던 상황들을 구현 과정 속에서 찾아내고 이런 점을 실제 실험을 통해 수정해 나갈 수 있는 것이 이러한 테스트베드의 이점이라는 것을 본 연구를 통해 다시 한 번 확인할 수 있었다.

앞으로의 연구로는 최신 버전의 OpenFlow 프로토콜을 활용해 물리포트의 통계자료를 활용한 라우팅이 아닌 플로우 별 통계자료를 바탕으로 한 라우팅 어플리케이션 개발이다. 또한 OpenWRT[10]를 활용해 논문에서

소개한 테스트베드와 연동하여 유무선 테스트베드를 구축하고 동작 과정에서 나타날 예상치 못한 상황을 파악하여 개선하는 것이 앞으로의 계획이다.

참 고 문 헌

[1] OpenFlow, http://OpenFlow.org
[2] NetFPGA, http://netfpga.org
[3] NOX 컨트롤러, http://noxrepo.org
[4] D. Awduche, J. Malcolm, J. Agogbua and J. McManus, "Requirements for Traffic Engineering Over MPLS", IETF RFC2702, September 1999.
[5] Ziegelmann and Mark, "Constrained Shortest Paths and Related Problems : Constrained Network Optimization", VDM Verlag Dr. Müller. ISBN 978-3-8364-4633-4. December 2007
[6] Spanning Tree protocol 어플리케이션, http://www.openflowswitch.org/wk/index.php/Basic_Spanning_Tree
[7] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik 1: 269-271
[8] iperf, http://sourceforge.net/projects/iperf/
[9] DVTS, http://www.internet2.edu/communities/dvts/
[10] OpenWRT, http://www.openwrt.org/

저 자 소 개

김 승 주(학생회원)
2011년 충남대학교 전자전파 정보통신공학 석사
2009년 충남대학교 전자전파 정보통신공학 학사
<주관심 분야 : QoS 라우팅, 미래 인터넷, 이동 및 위성통신>

민 석 홍(학생회원)
2005년 공주대학교 전기전자정보 공학과 석사
2010년~현재 충남대학교 전자전파정보통신공학과 박사과정
2004년 한국전자통신연구원 BcN 시험기술팀 위촉연구원
2005년 디지피아㈜ 방송장비팀 연구원
2006년~2009년 ㈜엠티아이 연구2실 전임연구원
<주관심분야 : 무선 메쉬 네트워크, 데이터 통신, 미래인터넷>

김 병 철(평생회원)
1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1996년 한국과학기술원 전기 및 전자공학과 박사
1993년~1999년 삼성전자 CDMA 개발팀
1999년~현재 충남대학교 정보통신공학부 교수
<주관심 분야 : 이동인터넷, 이동통신 네트워크, 데이터 통신>

이 재 용(평생회원) - 교신저자
1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1995년 한국과학기술원 전기 및 전자공학과 박사
1990년~1995년 디지콤 정보통신 연구소 선임연구원
1995년~현재 충남대학교 정보통신공학부 교수
<주관심분야 : 초고속통신, 인터넷, 네트워크 성능분석>

홍 원 택(정회원)
1998년 성균관대학교 정보공학과 졸업(학사)
2000년 성균관대학교 대학원 컴퓨터공학부 졸업(석사)
2000년~2002년 (주)콤텍시스템 기술연구소 연구원
2002년~현재 한국과학기술정보연구원 슈퍼컴퓨팅본부 융합자원실 선임연구원
<주관심분야 : 망 관리, 트래픽 분석, 프로그래머블 네트워크>