http://dx.doi.org/10.3745/KIPSTC.2011.18C.6.423

KREONET 기반의 광역 규모 오픈플로우 네트워크 구축 및 성능 분석

홍 원 택[†] · 공 정 욱[††] · 정 진 욱[†††]

요 약

최근 네트워크 가상화 기능 및 망에 대한 프로그래밍의 용이성을 제공해 줄 수 있는 기반 기술로 오픈플로우 기술이 주목 받고 있다. 국내에서는 캠퍼스 실험실 수준의 로컬 망에서 오픈플로우 기술의 적용 및 Layer 3을 경유한 캠퍼스 망 간의 연동이 이뤄지고 있지만 IP 계층에서의 네트워크 지연 등으로 인한 성능 저하가 문제되고 있다. 본 논문에서는 로컬 망 수준이 아닌 국가과학기술연구망 기반의 광역 규모 오픈플로우 망을 순수 Layer 2상에서 설계 및 구축하고 종단간 Round-trip Time 측정, TCP/UDP 성능 테스트를 통해 오픈플로우 망과 오픈플로우 적용 전 일반 망의 성능을 비교 분석한다. 분석 결과, 광역 규모의 오픈플로우 망은 일반 망과 비교할 때, UDP 스트림에 대한 초반 패킷 손실 문제를 제외하고 대등한 수준의 성능을 보였다. 또한, 초반 UDP 패킷 손실로 인한 성능 저하 현상은 컨트롤러 상에 반복 유입되는 동일한 "Packet_in" 이벤트에 대한 예외 처리를 구현함으로써 개선시킬 수 있었다.

키워드 : 오픈플로우, 국가과학기술연구망, 종단간 성능, 프로그래밍 용이성, 컨트롤러

Deployment and Performance Analysis of Nation-wide OpenFlow Networks over KREONET

WonTaek Hong[†] · JongUk Kong[††] · JinWook Chung[†††]

ABSTRACT

Recently, OpenFlow has been paid attention to as a fundamental technology which provides a function of virtualization and programmability in network. In Korea, deployment of OpenFlow networks in campuses and the interconnection between them through tunneling in layer 3 has been performed. However, the performance of the interconnected networks is decreased due to delay in IP layer. In this paper, we design and deploy nation-wide, not local, OpenFlow networks in a pure layer 2 environment over KREONET. After that, we do end-to-end Round-trip Time measurements and TCP/UDP performance tests in OpenFlow and normal networks, and do comparison and analysis on the test results. The results show that the nation-wide OpenFlow networks provide equal performance to normal networks except for the initial packet loss for UDP streaming. In regards to the performance decrease due to early UDP packet loss, we can mitigate it by implementing exceptional procedures in a controller which deal with the same continuous "Packet_in" events.

Keywords : OpenFlow, KREONET, End-to-End Performance, Programmability, Controller

1. 서 론

일반적으로 네트워크 가상화는 하나의 물리적인 네트워크 상에 논리적으로 독립된 다수의 네트워크가 공존할 수 있는 개념으로서 서비스의 다양성, 망 운영의 효율성 및 유연성을 제공해 주는 측면에서 망 설계 및 구축 시 지속적으로 고려되고 있다. 또한, 각각의 가상화된 네트워크 상에서 네트워크 응용별 서비스의 다양화, 차별화를 구현하기 위해 망에 대한 프로그래밍의 용이성(Programmability)이 함께 요구되고 있다.

더욱이 최근에는 현재 인터넷의 문제점을 지적하고 망에 대한 다양한 요구사항을 반영시키기 위한 미래인터넷에 대한 연구가 국내외에서 진행되고 있고[1-3], 가상화 및 망에 대한 프로그래밍의 용이성 등은 필요한 핵심 기술들로 인식되고 있다. 이러한 배경 속에서 최근 활발히 연구되고 있는 오픈플로우[4,5] 기술은 망의 가상화 및 프로그래밍의 용이성 등을 제공해 줄 수 있는 기반 기술로 주목 받고 있다.

† 정 회 원 : 한국과학기술정보연구원 선임연구원, 성균관대학교 전자전기컴퓨터공학과 박사과정
†† 정 회 원 : 한국과학기술정보연구원 선임연구원
††† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2011년 7월 20일
수 정 일 : 1차 2011년 9월 5일
심사완료 : 2011년 9월 6일

오픈플로우 기술은 스탠포드 대학에서 연구 진행하는 공개된 표준 기술로서, 실험적인 네트워크 프로토콜들을 실제 네트워크 상에서 구동하도록 하는 인터페이스를 제공함으로써 가상화된 형태의 프로그래밍이 용이한 망의 구성을 가능하게 해 준다. 오픈플로우 망은 오픈플로우 스위치, 컨트롤러, 스위치와 컨트롤러 사이의 통신을 정의하는 오픈플로우 프로토콜로 구성된다.

오픈플로우 망 사용자는 원하는 시나리오에 따라 망의 동작을 플로우 기반으로 정의 및 컴포넌트 형태로 구현하여 컨트롤러 상에 반영시킨 후, 컨트롤러에 등록된 오픈플로우 스위치들을 원하는 시나리오에 따라 플로우별로 제어할 수 있다. 이러한 메커니즘을 통해 오픈플로우 네트워크는 오픈플로우 스위치들로 구성되는 데이터 평면과 컨트롤러들로 구성되는 제어 평면이 분리될 수 있고 망 관리의 유연성을 높일 수 있다. 또한, 망의 특성 및 동작과 관련한 다양한 아이디어들이 NOX[6]와 같은 공개된 컨트롤러 상에 플러그인 형태로 반영될 수 있다.

이러한 오픈플로우의 기술적 특성은 커뮤니티별로 다양한 네트워크 요구사항을 갖는 KREONET(Korea Research Environment Open NETwork)[7]의 네트워크 응용들에 대해 망에 대한 프로그래밍의 용이성을 제공함으로써 서비스의 다양성을 높일 수 있는 기술로 활용될 수 있고 향후 연구망의 진화 관점에서도 주목해야 할 기술로 판단되고 있다.

국내에서는 캠퍼스 망 차원에서 오픈플로우 기술이 도입되어 실험실 수준의 로컬 망에서 적용 및 관련 응용 연구가 진행되고 있고 상호 캠퍼스 오픈플로우 망 사이에서 Layer 3 네트워크를 터널링 형태로 경유한 연동이 시도되었다. 그러나 이러한 연동은 IP계층에서의 네트워크 지연 등으로 인해 종단간의 성능이 감소한다. 이와는 달리 백본 망 수준에서의 오픈플로우 기술 적용은 Layer 3 네트워크를 경유하지 않고 일반적으로 VLAN 기술 등을 활용한 순수 Layer 2에서 이뤄진다.

본 논문에서는 KREONET 백본 상에 구축된 국내 광역 규모 오픈플로우 망에 대해 망의 설계, 동작 및 성능을 분석한다. 먼저, KREONET 백본 상에 오픈플로우 망을 적용하기 앞서 검토되었던 연동될 일반 망 기술들에 대한 비교 및 오픈플로우 망의 설계 과정을 설명한다. 그리고 선택된 IEEE 802.1ah PBB (Provider Backbone Bridges) [8,9]/VLAN 상에 오픈플로우 망을 적용할 때 오픈플로우 기술이 적용되기 이전의 PBB/VLAN 망과 오픈플로우 기술을 적용한 후의 망 성능을 RTT(Round-trip Time), TCP/UDP 성능 테스트를 통해 비교 분석한다. 마지막으로 오픈플로우 망에서 UDP 성능 측정 시 일반 망과 달리 성능이 저하되는 현상이 발생하는 원인을 분석하고 오픈플로우 망의 컨트롤러 상에서 이러한 문제점을 개선시키는 방법을 제시한다. 이러한 실험 및 분석을 통해 광역 규모의 백본 망에 적용된 오픈플로우 망의 성능이 일반 망과 필적할 수 있음을 보일 수 있었고 오픈플로우의 기술적 특징인 망에 대한 프로그래밍의 용이성을 활용하여 컨트롤러 상에 특정 이벤트 처리 모듈을 구현함으로써 UDP 트래픽에 대한 성능 저하 현상을 개선시킬 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 연구망에서의 네트워크 가상화와 관련된 기존의 연구 및 접근 방법을 소개하고, 3장에서 KREONET 오픈플로우 망의 설계 및 구축 과정을 서술한다. 4장에서는 일반 망과 오픈플로우 망의 성능 측정 방법 및 실험 환경에 대해 서술하고 성능 측정 결과를 비교 분석한다. 그리고 5장에서는 오픈플로우 망의 컨트롤러 상에 이벤트 처리 모듈 구현을 통해 망 성능을 개선시키는 방법의 제시 및 성능 분석을 하고 마지막으로 6장에서 본 논문의 결론 및 향후 연구 과제를 서술한다.

2. 관련 연구

세계 각국의 연구망에서는 여러 네트워크 응용들을 대상으로 독립된 망을 제공하기 위해 계층별로 네트워크 가상화를 실현할 수 있는 방법들을 연구해 오고 있다.

UCLP(User Controlled LightPaths)[10,11]는 일반 사용자들이 광 패스를 동적으로 프로비저닝할 수 있도록 해 주는 네트워크 가상화 서비스 프레임워크로서, Layer 1에서의 SONET(Synchronous Optical Network)/ SDH(Synchronous Digital Hierarchy) 기반 논리 망을 제공해 줄 수 있는 기술이다. 이 기술을 통하여 고품질, 고대역을 요구하는 네트워크 응용 사용자들은 원하는 시점에 광 패스를 동적으로 할당 받음으로써 Layer 1에서의 SONET/SDH 기반의 채널 분할을 통한 독립된 논리 망을 생성 및 제거할 수 있다.

IEEE 802.1ah PBB(Provider Backbone Bridges)[8,9]는 Ethernet의 고유 특성을 유지하면서 기존 Ethernet의 VLAN 및 MAC 주소의 확장성 문제를 MAC-in-MAC 방식을 통해 해결하고 사용자 망과 프로바이더 망을 구분함으로써 확장성 있는 Ethernet 기반의 백본 망을 설계 및 구현할 수 있는 기술이다. PBB 기술을 바탕으로 세분화된 논리 망의 구성이 가능하고 종단간 순수 Layer 2 기반의 효율적이고 안정적인 가상화 네트워크의 운영이 가능하다.

Logical router[12]는 하나의 라우터를 논리적으로 독립된 다수의 라우터들로 분리하는 기술로서, 각각의 logical router들은 본래 라우터의 기능들과 속성들을 동일하게 유지하면서 각각 복수 개의 라우팅 테이블들을 독립적으로 운영할 수 있다. 네트워크 장비 벤더별로 구현상의 차이는 존재하지만 Layer 3에서 물리적 라우터의 추가적인 설치 없이 망의 분리 및 유연성을 높이는데 유용하게 이용될 수 있다.

한편, 위에서 언급된 계층별 네트워크 가상화 기술들은 네트워크 응용의 특성에 따라 선택적으로 적용될 수 있다. 실제로 오픈플로우 망은 본래 Ethernet 기반의 Layer 2에서 동작하고 광역 규모의 적용을 위해서는 일반 망과의 연동이 불가피하다. 이러한 과정에서 UCLP를 활용한 Layer 1의 SONET/SDH 논리적 채널들을 할당 받아 일반 망과의 연동을 고려할 수도 있고 Layer 2에서의 PBB, VLAN 기술들을 활용한 터널링을 고려할 수도 있다. 즉, UCLP 서비스 프레임워크와 PBB 망에서는 오픈플로우 망이 일종의 가상화 서비스를 제공받는 또 다른 서비스 형태로 인식될 수 있다.

이와 같이 연구망에서의 네트워크 응용에 따른 네트워크 가상화 프레임워크 제공과 관련하여 다양한 관점에서 기술 적용을 위한 연구가 진행되고 있고 오픈플로우 기술은 향후 연구망에서의 네트워크 가상화 서비스 제공을 위한 잠재적인 요소 기술로 활용될 것으로 예상된다. 이러한 맥락에서 오픈플로우 기반의 실질적인 광역 가상화 네트워크 구축 및 성능 분석에 대한 연구는 오픈플로우 망의 확대 적용 및 기존 기술과의 연동 등을 위해 반드시 요구된다.

3. KREONET 오픈플로우 망의 설계 및 구축

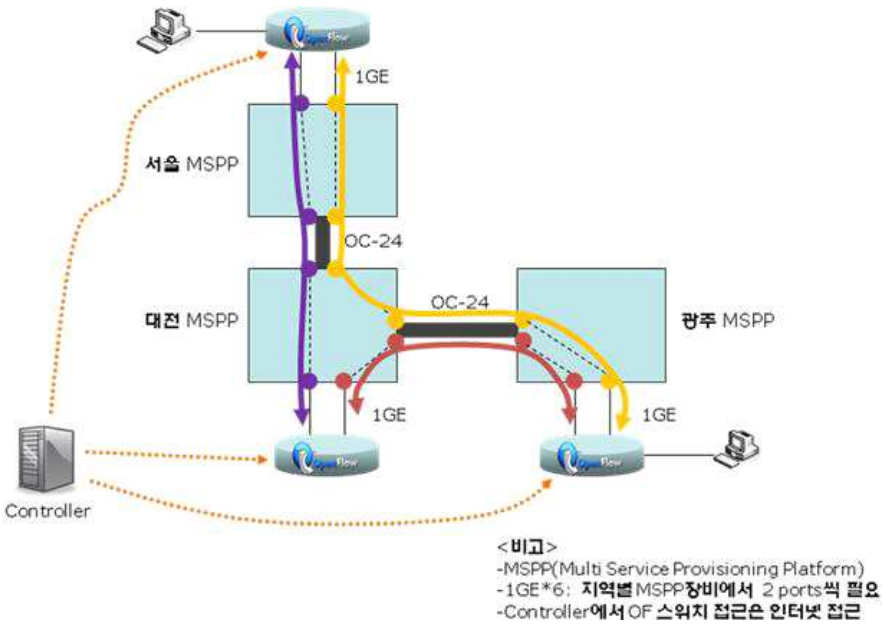
광역 규모의 오픈플로우 망을 구축하기 위해서는 일반 망과의 연동을 감안하여 전용 경로 또는 터널링 기술들을 고려해야 한다. 예를 들어, SONET/SDH 채널 기반의 Layer 1 전용 광 패스 설정, Layer 2 VLAN tagging 기법, Layer 3에서의 연동을 위해 MAC-in-IP 기능을 제공하는 Encapsulator의 적용, 캐리어 이더넷 백본 망을 구성하는 IEEE 802.1ah PBB 기법 등을 열거할 수 있다.

위의 기법들 중에서 Layer 2 VLAN tagging 기법, Layer 3 연동을 위한 MAC-in-IP Encapsulator 방법은 오픈플로우 망 구성을 위해 일반적으로 많이 이용되고 있다. Layer 2 VLAN tagging 기법은 망 관리자의 개입 하에 VLAN 경로를 구성하기 위해 VLAN 번호를 할당 받아 오픈플로우 스위치와, 연동되는 Layer 2 일반 스위치에 VLAN을 설정하여 양 스위치들을 연동하는 방법으로 일반적으로 많이 이용된다. 반면, Layer 3 연동을 위한 MAC-in-IP Encapsulator 방식은 터널링에 기반한다는 개념은 동일하지만 Layer 2 VLAN tagging 기법과는 달리 망 관리자의 개입 없이 일반 인터넷 망에서 조차도 오픈플로우 스위치와 Layer 3 라우터 사이에 MAC-in-IP Encapsulator를 위치시킴으로써 간략하게 터널링 기능을 구현할 수 있다. 하지만, Layer 3 네트워크의 특성상 성능과 관련하여 일정 수준 이상의 서비스 품질을 보장하기 어렵고 Encapsulator의 추가적인 설치 및 관리가 필요하므로 캠퍼스 망에서의 실험을 위한 용도 또는 단기적 테스트에 적합하다고 할 수 있다.

KREONET 백본 상에 오픈플로우 스위치 적용을 위해 SONET/SDH 채널 기반의 Layer 1 광 경로 설정 기법과 IEEE 802.1ah PBB 기법이 각각 검토되었다.

우선, SONET 기반의 우회 경로를 갖는 오픈플로우 망을 구성하기 위해서는 각 지역 망 센터에 있는 SONET/SDH 지원 장비에서 각각 최소 2개 이상의 신규 물리적 포트들을 요구하게 된다. 이러한 조건은 일반적으로 전송 장비가 보유하고 있는 적은 수의 물리적 인터페이스 한계로 인해 실질적인 적용을 어렵게 만든다. 즉, (그림 1)에서 볼 수 있듯이 SONET/SDH과 같은 전송 기술을 이용할 경우 SONET/SDH 장비들 간에는 같은 물리적 인터페이스를 채널링하여 논리적으로 나눌 수 있으나 결국 지역 망 센터에서 오픈플로우 스위치가 연동되는 지점에서는 전송 장비 종단에서의 "Drop"을 위해 물리적으로 독립된 인터페이스를 요구하게 된다.

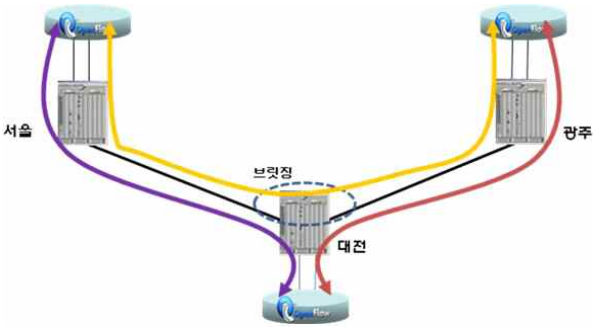
추가적으로 IEEE 802.1q VLAN 개념을 확장한 IEEE 802.1ah PBB 기술을 고려할 수 있다. PBB 기술은 캐리어 이더넷 백본을 구성하기 위한 기술로서 Ethernet 기반의 논리적 망 분리를 제공해 줄 수 있고 KREONET 과학기술자원융합망[7]의 백본에 구축된 사례가 있다. 과학기술자원융합망은 천문, 의료, 고에너지 물리 등의 대용량 데이터 전송 및 처리를 필요로 하는 첨단과학기술 응용연구 분야를 대상으로 고대역, 고품질의 네트워크 자원과 컴퓨팅 자원을 동



(그림 1) 오픈플로우 스위치와 SONET 장비 간 연결

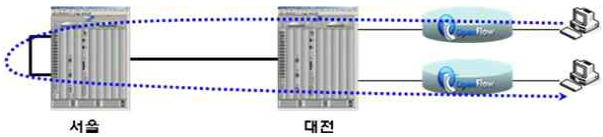
시에 제공해 주기 위한 진보된 개념의 연구망 인프라이다. 현재, 구축된 인프라의 일부가 PBB를 지원하는 캐리어 이더넷 장비를 통해 구축되어 있고 종단간 순수 Layer 2 전용의 서비스 제공이 가능하다.

(그림 2)는 PBB를 지원하는 캐리어 이더넷 장비로 구성된 과학기술자원융합망의 일부를 보여준다. (그림 2)에서 볼 수 있듯이 PBB망은 서울, 대전, 광주 지역에 구성되었고 오픈플로우 망을 일종의 오버레이 형태로 제공한다. 오픈플로우 망 관점에서 서울~광주 간의 우회 경로는 대전 PBB 장비에서 브리징 형태로 통과하는 망 구성을 한다. 실제로 PBB 장비 간에는 하나의 물리 회선을 PBB 프레임에서 사용되는 I-SID 필드를 통해 논리적으로 분리함으로써 독립된 경로 구성을 가능하게 한다[8].



(그림 2) 오픈플로우 스위치와 PBB 장비 간 연결

(그림 2)와 같은 네트워크 환경 구축에 앞서 서울~대전, 대전~광주 간의 PBB 장비들 간 연결 상태 및 오픈플로우 스위치와 PBB 장비 간 연결 상태를 사전에 점검하기 위해 (그림 3)과 같은 토폴로지 상에서 종단간 ping 프로그램을 이용하여 루프백 테스트를 수행하였다. 위의 과정은 PBB/VLAN으로 구성된 망에 오픈플로우 망이 오버레이 형

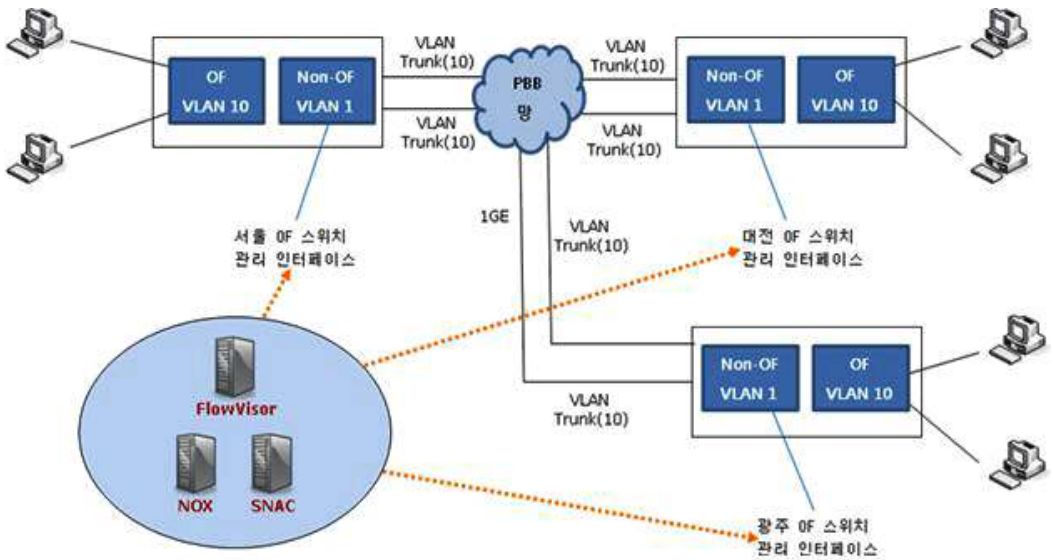


(그림 3) 장비 및 회선 점검을 위한 루프백 테스트

태로 적용되기 전 망의 연결성 및 상태를 점검하는 과정이므로 오픈플로우 기능이 비활성화된 상태에서 수행되었고 각각 RTT값이 약 6 ms로 측정되어 정상적인 망의 상태를 사전에 확인할 수 있었다.

루프백 테스트를 통해 연동될 망의 연결성을 확인한 후 PBB/VLAN 기술로 구성된 일반 망을 활용하여 광역 규모의 오픈플로우 망을 구축한다. 오픈플로우 기술의 특성을 반영하여 (그림 4)와 같이 데이터 평면과 제어 평면으로 나뉘게 되고 데이터 평면은Layer 2 Ethernet 망으로 구성되고, 제어 평면은 일반 인터넷 망을 통해 컨트롤러들과 오픈플로우 스위치의 관리 인터페이스를 연결함으로써 구성된다.

컨트롤러들로는 NOX, SNAC, FlowVisor들이 각각 독립된 서버에서 운영되고 있다. NOX[6]는 오픈플로우 망의 관리 및 제어용으로 활용되는 가장 보편적인 컨트롤러로서, 플로우 기반의 실험적인 네트워크 프로토콜들을 구현할 수 있는 공개된 다수의 컴포넌트들로 구성된 프레임워크이다. SNAC[13]은 웹 기반의 정책 관리자를 이용하여 오픈플로우 망의 보안 정책 및 관리 대상의 노드들에 대한 모니터링 정보를 제공해 주는 컨트롤러이다. 추가적으로, 오픈플로우 네트워크 상에서 다수의 네트워크 시나리오를 적용하기 위해서 FlowVisor 기반 슬라이싱 기법을 도입한다. FlowVisor[14]는 오픈플로우 망을 여러 컨트롤러들을 이용하여 제어할 수 있도록 지원하는 일종의 프락시 컨트롤러이다. FlowVisor는 네트워크 자원에 대한 슬라이스들을 생성하여 각각의 슬라이스마다 고유의 컨트롤러들로 제어를 위임하는 역할을 한다. 이러한 과정을 통해 특정 슬라이스가



(그림 4) PBB/VLAN 기반의 오픈플로우 망 구성도

다른 슬라이스의 트래픽에 영향을 주지 않도록 망 자원을 분리시킨다.

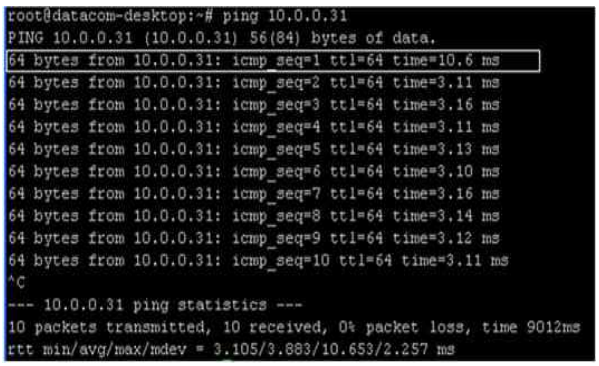
본 오픈플로우 망에서는 NOX의 “pyswitch” 컴포넌트와 SNAC의 망 모니터링을 위한 슬라이스들을 각각 생성하여 매핑시킨다. 그리고 오픈플로우 스위치마다 종단간 연결 상태 및 성능을 측정하기 위해 두 대의 서버를 각각 연결한다.

광역 규모 오픈플로우 망의 설계 및 구축 시 SONET/SDH 기반의 Layer 1 광 패스를 이용한 접근 방법은 망의 확장 및 운영 관점에서 유연성이 떨어지므로 망 제공자 측면에서 비효율적일 수 있다. 반면, PBB 망은 Layer 1 SONET/SDH 기반의 논리 망 적용 시 떨어지는 유연성을 Ethernet의 특징을 살려 보완할 수 있는 장점이 있고 추가적으로 다른 네트워크 커뮤니티를 위한 독립된 망을 보다 수월하게 제공해 줄 수 있다. 실제로 KREONET에는 PBB로 운영되는 구간이 있고 이러한 구간의 일부를 활용하여 오픈플로우 망의 적용을 위한 논리 망을 구성할 수 있었다. 결과적으로 KREONET 오픈플로우 망은 PBB 망 위에 오버레이 형태로 존재하는 구성을 따른다. 즉, 데이터 평면은 PBB/VLAN 망 위에 존재하고 제어 평면은 일반 망을 통해 오픈플로우 망을 제어하는 FlowVisor, NOX, SNAC 등의 컨트롤러들로 구성된다.

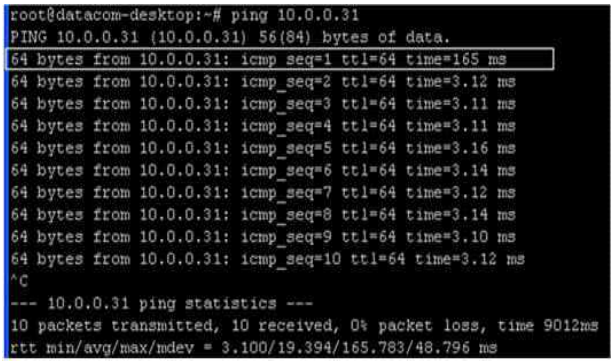
4. 실험 및 성능 분석

앞서 기술된 Layer 2 Ethernet 기반의 오픈플로우 망은 PBB/VLAN 상에서 동작한다. 오픈플로우 망은 일반 망과 달리 컨트롤러와의 통신을 통해 플로우 테이블을 내려 받아 동작하므로 오픈플로우 기능이 스위치에 적용된 경우와 그렇지 않은 경우로 분류하여 광역 규모 오픈플로우 망의 동작 및 성능을 비교, 검증해 볼 필요가 있다.

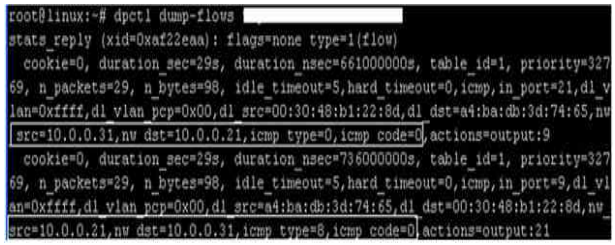
본 실험에서는 앞 절에서 소개된 광역 규모의 1Gbps 오픈플로우 네트워크 토폴로지 중 일부인 대전 ~ 광주 구간에서 ping 프로그램을 이용한 RTT 값, iperf[15] 기반의 TCP/UDP 트래픽에 대한 동작 및 성능을 각각 일반 망과 오픈플로우 망에서 측정하여 비교 분석한다. 각 지역에는 오픈플로우 스위치 스펙 1.0[16]을 지원하는 K.14.63o 버전



(그림 5) 일반 망에서의 ping 프로그램 실행



(그림 6) 오픈플로우 망에서의 ping 프로그램 실행

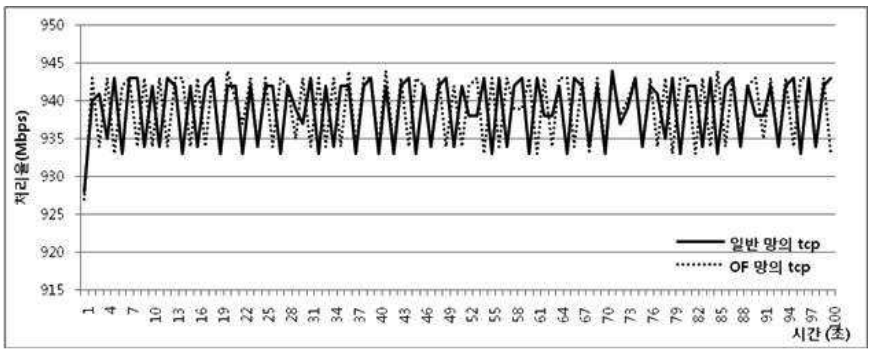


(그림 7) ICMP 프로토콜에 대한 플로우 테이블

의 오픈플로우 firmware가 적용된 HP Procurve 3500yl 스위치[17]가 설치되었고 종단간 성능 측정을 위한 도구로는 iperf 2.0.4를 이용한다. 오픈플로우 망을 관장하는 컨트롤러로는 공개 소프트웨어인 NOX 0.8.0을 이용하고 기본적인 스위치 기능을 구현한 “pyswitch” 컴포넌트를 로딩시켜 오픈플로우 망과 연동한다. 일반 망의 RTT 값 및 성능을 측정할 때는 VLAN 상에서 활성화된 오픈플로우 기능을 비활성화하여 컨트롤러와의 통신을 차단한다.

4.1 RTT 측정 및 분석

오픈플로우 스위치에 연결된 서버에서 ping 프로그램을 이용하여 종단간 RTT 값을 각각 측정해 보면 (그림 5), (그림 6)과 같이 오픈플로우 기능이 비활성화된 일반 망과 활성화된 망 모두 약 3ms에 근접하는 값을 얻게 된다. 다만, 첫 패킷의 경우 일반 망과 오픈플로우 망이 각각 약 10ms, 165ms의 차이를 보인다. 이것은 일반 망의 경우 측정 대상 호스트의 IP 주소에 대한 MAC 주소 학습을 위한 ARP 프로토콜 처리 및ICMP 프로토콜의 처리 과정에서 소요되는 시간이다. 반면, 오픈플로우 망의 경우도 동일한 과정이 요구되지만, ARP/ICMP 프로토콜 처리를 위한 일련의 이러한 과정이 플로우 기반으로 처리되기 때문에 관련 플로우 테이블을 컨트롤러로부터 내려 받는 데 걸리는 시간이 추가적으로 포함되어 차이가 발생한다. 하지만, 플로우 테이블을 내려 받은 이후에는 일반 망의 경우처럼 비슷한 RTT 값을 얻게 된다. 참고로, (그림 7)에서는 대전에 있는 오픈플로우 스위치에서 컨트롤러와의 통신을 통해 내려 받은 ICMP 프로토콜 처리를 위한 플로우 테이블을 캡쳐하여 보여준다.



(그림 8) 일반 망과 오픈플로우 망의 TCP처리율 비교

4.2 TCP 성능 측정 및 분석

(그림 8)에서는 일반 망과 오픈플로우 망에서의 TCP 트래픽에 대한 성능을 비교한다. 각각 종단 호스트로 "iperf -c 10.0.0.31 -i 1 -t 100"을 실행하여 100초간 TCP 트래픽을 발생시켰고 일반 망과 오픈플로우 망에서 약 940Mbps의 비슷한 TCP 처리율을 얻을 수 있었다. 또한, (그림 9)에서 볼 수 있듯이 오픈플로우 망에서는 iperf에서 발생되는 TCP 트래픽을 처리하기 위한 플로우 테이블이 생성된 것을 확인할 수 있다.

```
root@linux:~# dpctl dump-flows
stats_reply (xid=0x58e9bede): flags=none type=1(flow)
  cookie=0, duration_sec=65s, duration_nsec=762000000s, table_id=1, priority=327
68, n_packets=5229996, n_bytes=74, idle_timeout=5,hard_timeout=0,tcp,in_port=9,d
l_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=a4:ba:db:3d:74:65,dl_dst=00:30:48:b1:22:8d
,nw_src=10.0.0.21,nw_dst=10.0.0.31,tp_src=53060,tp_dst=5001,actions=output:21
  cookie=0, duration_sec=65s, duration_nsec=742000000s, table_id=1, priority=327
68, n_packets=800779, n_bytes=74, idle_timeout=5,hard_timeout=0,tcp,in_port=21,d
l_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=00:30:48:b1:22:8d,dl_dst=a4:ba:db:3d:74:65
,nw_src=10.0.0.31,nw_dst=10.0.0.21,tp_src=5001,tp_dst=53060,actions=output:9
```

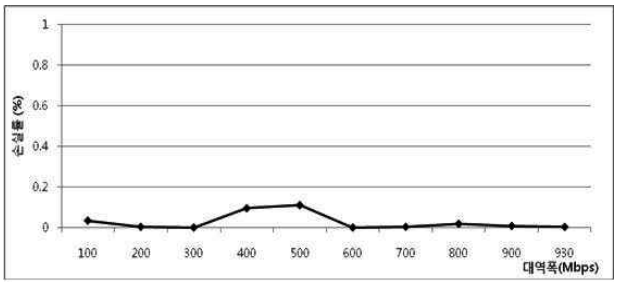
(그림 9) iperf의 TCP 트래픽에 대한 플로우 테이블

4.3 UDP 성능 측정 및 분석

UDP 트래픽에 대한 성능 분석은 대역폭 파라미터를 증가시키면서 일반 망과 오픈플로우 망에서의 성능을 각각 측정하였다.

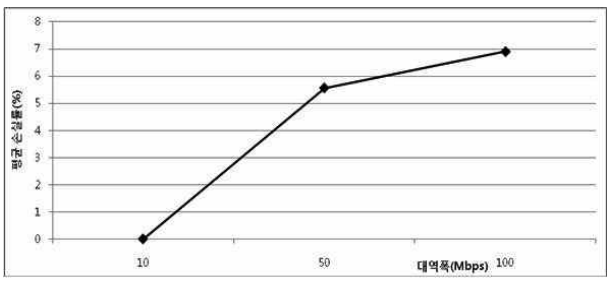
일반 망의 경우 800Mbps까지는 "iperf -c 10.0.0.31 -i 1 -t 100 -b bandwidth"을 실행하여 100초간 UDP 트래픽을 발생시켰고 800Mbps를 초과하는 트래픽에 대해서는 UDP 버퍼 크기 및 송수신되는 패킷의 크기를 각각 "-w", "-l" 옵션을 이용하여 조절하였다. 본 실험에서는 900Mbps와 930Mbps로 전송 시 256Kbyte의 UDP 버퍼 크기와 1700Byte의 패킷 크기를 추가적으로 설정하여 실험하였다. 전체적인 실험 결과는 (그림 10)처럼 각 대역폭을 기준으로 0.1% 미만의 패킷 손실률을 기록하였다.

한편, 오픈플로우 망에서의 UDP 트래픽의 경우 일반 망에서의 UDP 트래픽과는 다른 패턴을 보였다. 일반 망과 달리 오픈플로우 망에서의 UDP 트래픽에 대해서는 상당한 패킷 손실이 발생되었다. 이러한 점은 TCP 트래픽의 경우 일반 망과 오픈플로우 망에서 비슷한 성능을 얻을 수 있는 점과 대비된다.



(그림 10) 일반 망의 UDP 패킷 손실률

(그림 11)에서 볼 수 있듯이 오픈플로우 망에서의 UDP 트래픽은 대역폭이 증가함에 따라 패킷 손실률이 급격히 증가한다. 높은 패킷 손실률을 감안하여 오픈플로우 망에서는 일반 망에서처럼 100Mbps 단위로 대역폭을 조절하지 않고 10Mbps, 50Mbps, 100Mbps 등으로 측정 단위를 좀 더 세분화하여 각 대역폭마다 3회의 동일 실험을 반복한 후 패킷 손실률의 평균값을 얻었다. (그림 12)에서는 오픈플로우 망에서 iperf를 통해 발생되는 UDP 트래픽들을 처리하기 위한 플로우 테이블을 보여준다. TCP와는 달리 단일 플로우 테이블만이 생성되는 것을 확인할 수 있다.



(그림 11) 오픈플로우 망의 UDP 패킷 평균 손실률

```
root@linux:~# dpctl dump-flows
stats_reply (xid=0x7f1379d7): flags=none type=1(flow)
  cookie=0, duration_sec=28s, duration_nsec=402000000s, table_id=1, priority=327
68, n_packets=23610, n_bytes=7560, idle_timeout=5,hard_timeout=0,udp,in_port=9,d
l_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=a4:ba:db:3d:74:65,dl_dst=00:30:48:b1:22:8d
,nw_src=10.0.0.21,nw_dst=10.0.0.31,tp_src=56852,tp_dst=5001,actions=output:21
```

(그림 12) iperf 의 UDP 트래픽에 대한 플로우 테이블

```
[root@localhost ~]# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.31 port 5001 connected with 10.0.0.21 port 45138
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  1.51 MBytes  12.7 Mbits/sec  1.958 ms 1529/ 2607 (59%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  1.0- 2.0 sec  1.61 MBytes  13.5 Mbits/sec  2.186 ms 3568/ 4713 (76%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  2.0- 3.0 sec  1.67 MBytes  14.0 Mbits/sec  0.841 ms 2947/ 4137 (71%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  3.0- 4.0 sec  1.64 MBytes  13.7 Mbits/sec  1.141 ms 3046/ 4214 (72%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  4.0- 5.0 sec  1.69 MBytes  14.1 Mbits/sec  1.175 ms 3077/ 4280 (72%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  5.0- 6.0 sec  4.04 MBytes  33.9 Mbits/sec  0.077 ms 2742/ 5624 (49%)
[  3]  5.0- 6.0 sec  497 datagrams received out-of-order
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  6.0- 7.0 sec  5.97 MBytes  50.0 Mbits/sec  0.081 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  7.0- 8.0 sec  5.97 MBytes  50.0 Mbits/sec  0.091 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  8.0- 9.0 sec  5.97 MBytes  50.1 Mbits/sec  0.085 ms    0/ 4256 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  9.0-10.0 sec  5.97 MBytes  50.0 Mbits/sec  0.088 ms    0/ 4255 (0%)
```

(그림 13) 50Mbps 대역폭으로 전송 시 UDP 서버 문제

(그림 13)은 오픈플로우 망의 송신측에서 “iperf -c 10.0.0.31 -i 1 -t 100 -b 50M”을 실행할 때 수신측에서 얻는 결과를 보여준다. (그림 13)에서 볼 수 있듯이 초반의 약 6초간 패킷 손실이 연속적으로 발생하는 것을 볼 수 있다. 반복된 실험을 통해 (그림 11)의 그래프에서 발생한 UDP 패킷 손실은 (그림 13)에서처럼 초반에 집중적으로 발생하는 것을 확인할 수 있었다. 더불어 대역폭이 커짐에 따라 초반에 발생되는 패킷 손실 시간은 점점 늘어나는 것으로 관찰되었다.

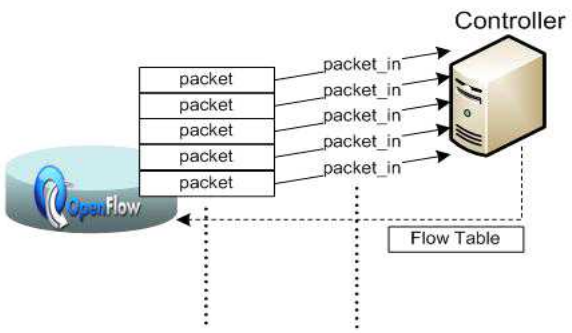
오픈플로우 망에서의 이러한 UDP 트래픽 성능 저하와 관련한 현상은 오픈플로우 망의 동작 특성에서 기인한다고 예상할 수 있다. 즉, 오픈플로우 망에서의 동작은 플로우 테이블에 근거하여 패킷을 전달하게 되는데, (그림 13)에서 패킷 손실이 집중적으로 발생하는 약 6초간은 오픈플로우 스위치에 UDP 트래픽과 관련된 플로우 테이블이 생성되지 않았고 결과적으로 초반 지속적인 패킷 손실이 발생된다고 추정할 수 있다.

5. 제안하는 컨트롤러 수정 및 성능 분석

본 절에서는 앞서 실험하였던 오픈플로우 망에서의UDP 트래픽 성능 분석과 관련하여 초반 지속적으로 발생하는 패킷 손실에 대한 원인을 분석하고, 패킷 손실률을 줄일 수 있는 방법을 제시한다.

오픈플로우 망에서의 UDP성능 저하와 관련된 상황을 (그림 14)와 같이 표현할 수 있다. 즉, iperf에서 생성되는 UDP 트래픽은 UDP 프로토콜의 특성 상 송신된 데이터에 대한 수신측의 “Ack”없이 연속적으로 데이터를 전송하게 된다. 이러한 상황에서 UDP 트래픽 처리를 위한 플로우 테이블이 오픈플로우 스위치에 도달하기 전까지 오픈플로우 스위치에는 UDP트래픽을 처리할 수 있는 플로우 테이블이 존재하지 않게 된다. 그러므로 오픈플로우 스위치는 “Packet_in” 메시지를 통해 동일한 UDP 패킷을 반복하여 컨트롤러로 보내게 된다.

한편, 컨트롤러에서는 연속하여 수신되는 동일한 UDP 패킷 마다 이벤트가 발생되고 동일한 이벤트 처리 과정을 의미 없이 반복함으로써 컨트롤러 상의 부하를 초래하게 된다. 이러한 과정에 의해 플로우 테이블의 전송은 일반적인 상황에서의 전송 시 보다 지연 시간이 길어지게 된다. 결과적으로 종단 시스템에서는 이러한 원인으로 인해 초반 상당 수준의 패킷 손실이 발생하게 된다. 더욱이, UDP 전송 대역폭이 커지면 이러한 현상은 더욱 심해지는 것을 실험을 통해 확인할 수 있었다.



(그림 14) UDP 패킷 손실의 원인

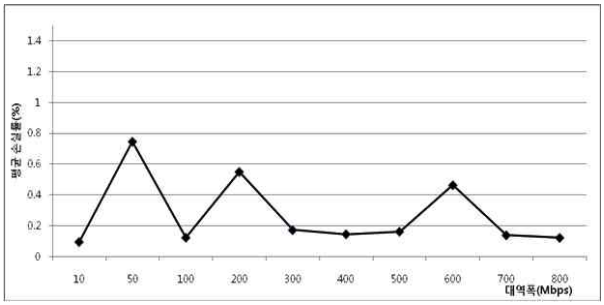
```
# from a modified Packet_in event handler
…
packet_in_list = { 'time':[], 'flow':[], 'count':[]}
current_flow = extract_flow(current_packet)

for last_time in packet_in_list['time']:
   if last_time + TIMEOUT < current _time():
      packet_in_list['time'].pop(lpointer)
      packet_in_list['flow'].pop(lpointer)
      packet_in_list['count'].pop(lpointer)
   else:
      lpointer = lpointer + 1

if current_flow in packet_in_list['flow']:
   idx = packet_in_list['flow'].index(current_flow)
   packet_in_list['time'][idx] = current_time()
   packet_in_list['count'][idx] = packet_in_list['count'][idx] + 1
   if packet_in_list['count'][idx] > MAX_COUNT:
      print 'SKIP the same flow processing'
      return CONTINUE
else:
   packet_in_list['time'] = packet_in_list['time'] +
[current_time()]
   packet_in_list['flow'] = packet_in_list['flow'] + [current_flow]
   packet_in_list['count'] = packet_in_list['count'] + [1]
…
```

(그림 15) 연속된 동일 패킷에 대한 "Packet_in" 이벤트 처리 알고리즘

앞서 지적한 동일한 "Packet_in" 메시지에 대한 처리 과정을 상황에 맞게 수정할 필요가 있다. 즉, 짧은 일정 시간에 발생되는 반복된 동일 "Packet_in" 이벤트에 대해서는 수신 패킷에 대한 MAC 주소 학습, 플로우 테이블 전송 및 플러딩 등 일반적으로 적용되는 이벤트 처리 과정 전에 해당 패킷에 대한 플로우 테이블의 유효 시간(time)과 반복되는 동일 "Packet_in" 이벤트의 누적 횟수(count)를 고려한 처리 과정을 거치도록 한다.



(그림 16) 수정 구현된 컨트롤러를 적용한 오픈플로우 망의UDP 패킷 평균 손실률

(그림 15)는 "Packet_in" 이벤트 핸들러에서 이러한 파라미터들(time, count)을 고려하여 작성된 처리 과정을 보여준다. packet_in_list는 일정시간(TIMEOUT) 동안 "Packet_in" 이벤트를 통해 전달된 패킷들에 대해 플로우 정보(flow), 마지막으로 수신된 시간(time), 누적 횟수(count)를 유지하는 리스트이고 current_flow는 "Packet_in" 이벤트를 통해 현재 수신된 패킷에 대한 플로우 정보를 유지한다.

주요 처리 과정은 "Packet_in" 이벤트의 발생 시, 1) 일정시간(TIMEOUT) 동안 재 수신되지 않는 패킷에 대한 플로우 정보는 packet_in_list에서 삭제하고 2) 현재 수신된 패킷에 대한 플로우 정보(current_flow)가 packet_in_list에 존재하면 해당 플로우의 유효 시간(time)을 현재의 시간(current_time())으로 변경시키고 누적 횟수를 증가시킨다. 3) 만약, 해당 플로우의 누적 횟수가 특정 임계치(MAX_COUNT)를 넘기면 해당 패킷에 대한 남겨진 플로우 처리 과정은 생략한다.

```
[root@localhost ~]# iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:   107 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.31 port 5001 connected with 10.0.0.21 port 35881
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  6.22 MBytes  52.2 Mbits/sec  0.019 ms 2314/ 6748 (34%)
[  3]  0.0- 1.0 sec  179 datagrams received out-of-order
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  1.0- 2.0 sec  5.97 MBytes  50.1 Mbits/sec  0.017 ms    0/ 4256 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  2.0- 3.0 sec  5.97 MBytes  50.1 Mbits/sec  0.025 ms    0/ 4256 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  3.0- 4.0 sec  5.97 MBytes  50.0 Mbits/sec  0.019 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  4.0- 5.0 sec  5.97 MBytes  50.0 Mbits/sec  0.017 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  5.0- 6.0 sec  5.97 MBytes  50.1 Mbits/sec  0.023 ms    0/ 4256 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  6.0- 7.0 sec  5.97 MBytes  50.0 Mbits/sec  0.019 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  7.0- 8.0 sec  5.97 MBytes  50.1 Mbits/sec  0.021 ms    0/ 4256 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  8.0- 9.0 sec  5.97 MBytes  50.0 Mbits/sec  0.018 ms    0/ 4255 (0%)
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  9.0-10.0 sec  5.97 MBytes  50.1 Mbits/sec  0.018 ms    0/ 4256 (0%)
```

(그림 17) 50Mbps 대역폭으로 전송 시 UDP 서버 로그

(그림 16)은 위의 알고리즘을 기반으로 구현된 컨트롤러를 이용하여 앞서 수행된 같은 조건의 실험 환경에서 iperf 테스트를 통해 얻은 UDP 패킷의 손실률을 대역폭별로 보여준다. 실험에서는 TIMEOUT값으로 1초, MAX_COUNT 값으로 3회가 지정되었고 각 대역폭마다 3회의 동일 실험을 반복하여 패킷 손실률의 평균값을 얻었다. 기존의 "pyswitch" 컨트롤러를 기반으로 실험했을 때 발생했던 초반의 UDP 패킷 손실률은 본 실험에서 평균 0.3% 미만으로 낮아졌고 이러한 수치는 일반 망에서의 UDP 패킷 손실률과 비교할 때 비슷한 수준을 유지하는 것으로 볼 수 있다.

(그림 17)은 앞선 실험과 동일하게 오픈플로우 망의 송신측에서 "iperf -c 10.0.0.31 -i 1 -t 100 -b 50M"을 실행할 때 수신측에서 얻는 결과를 보여준다. 기존의 "pyswitch"를 이용한 실험과는 달리 처음 약 1초 간에만 패킷 손실이 발생하고 이후에는 안정적으로 UDP 패킷이 전달되는 것을 확인할 수 있다. 또한, 전송 대역폭이 증가해도 패킷이 손실되는 시간은 약 1초 간에만 발생하는 것으로 관찰되었다. 이것은 컨트롤러로부터 관련된 플로우 테이블을 전달받아 적용되기까지의 시간이 약 1초 미만이 걸리는 것을 의미한다. 이 짧은 시간 동안 발생되는 패킷 손실은 플로우 테이블이 스위치에 적용되기 전에 처리되지 못한 패킷들에 의해 발생되는 것이고 컨트롤러를 통해 스위치의 동작이 결정되는 오픈플로우 망 동작의 구조적 특성에 기인한다고 할 수 있다.

6. 결 론

본 논문에서는 KREONET 백본 기반의 광역 규모 오픈플로우 망에 대해 설계, 망의 동작 및 성능을 분석하였다. 광역 규모의 오픈플로우 망은 연구망의 다양한 네트워크 응용들을 고려하여 확장성과 안정된 성능을 지원하기 위해 PBB/VLAN 기반의 순수 Layer 2 에서 설계 및 구축되었다. 그리고 PBB/VLAN 망에 오픈플로우 기술이 적용되기 전과 적용된 후의 망 성능을 RTT, TCP/UDP 성능 테스트를 통해 측정 및 비교 분석하였다. 분석 결과, 컨트롤러로부터 오픈플로우 스위치로 플로우 테이블을 내려 받기까지의 초반을 제외하고는 오픈플로우 기술이 적용된 망의 성능이 일반 망의 성능과 대등한 수준이었다. 다만, UDP 성능 테스트의 경우는 초반 지속적인 패킷 손실이 발생하여 망의 성능이 저하되었지만 오픈플로우의 기술적인 특징인 망에 대한 프로그래밍의 용이성을 이용하여 반복 유입되는 "Packet_in" 이벤트에 대한 처리를 컨트롤러 상에 추가함으로써 초반 패킷 손실로 인한 망의 성능 저하 현상을 개선시킬 수 있었다.

결과적으로, 광역 규모의 KREONET오픈플로우 망은 이미 적용된 PBB/VLAN의 일반 망과 비교할 때 대등한 수준의 망 성능을 제공할 수 있었고 컨트롤러를 통한 망의 제어 기능을 기반으로 오픈플로우 망의 운영에 대한 유연성을 확보할 수 있었다. 최근에는 KREONET 오픈플로우 망의 확장을 통한 캠퍼스 오픈플로우 망의 연동이 요구되고 있고 이러한 요구사항을 반영하기 위해서 망 자원의 공동 이용에 대한 정책 및 스케줄링 기법들이 추가적으로 연구되어야 할 것이다.

참 고 문 헌

[1] GENI, "http://www.geni.net"

[2] FIRE, "http://cordis.europa.eu/fp7/ict/fire"

[3] FIF, "http://fif.kr"

[4] 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 Computer Communication Review, April, 2008

[5] OpenFlow, "http://www.openflow.org"

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks", ACM SIGCOMM Computer Communication Review, July, 2008

[7] KREONET, "http://kreonet.net"

[8] D. Allan, N. Bragg, A. McGuire and A. Reid, "Ethernet as Carrier Transport Infrastructure" IEEE Communications Magazine, Vol.44, No.2, pp.134-140, Feb., 2006

[9] IEEE 802.ah standard, "http://www.ieee802.org/1/pages/802.1ah.html"

[10] E. Grasa, S. Figuerola, A. Lopez, G. Junyent and M. Savoie, "UCLPv2: A network virtualization framework built on web services", Communications Magazine, Web Services in Telecommunications, part II, pp.126 - 134, Mar., 2008

[11] J. Wu, H. Zhang, S. Campbell, M. Savoie, G. Bochmann, and B. St Arnaud, "A grid oriented lightpath provisioning system", GLOBECOM04, pp.395-399, 2004

[12] Matt Kolon, "Intelligent logical router service", White Paper, Juniper, 2004

[13] SNAC, "http://www.openflow.org/wp/snac"

[14] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzellery, Martin Casado, Nick McKeowny, and Guru Parulkary, "FlowVisor: A Network Virtualization Layer", Technical Report, 2009

[15] iperf, "http://en.wikipedia.org/wiki/Iperf"

[16] OpenFlow switch specification version 1.0, "http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf "

[17] HP procurve 3500yl switch manual, "http://www.hp.com/rnd/support/manuals/3500yl_6200yl.htm"

홍 원 택

e-mail : wthong@kisti.re.kr
1998년 성균관대학교 정보공학과(학사)
2000년 성균관대학교 전기전자 및 컴퓨터 공학과(석사)
2008년～현 재 성균관대학교 전자전기 컴퓨터공학과 박사과정
2000년～2002년 (주)콤텍시스템 기술연구소 연구원
2002년～현 재 한국과학기술정보연구원 선임연구원
관심분야 : 망 관리, 트래픽 분석, 프로그래머블 네트워크

공 정 욱

e-mail : kju@kisti.re.kr
1993년 한국과학기술원 전기 및 전자공학과(학사)
1998년 포항공과대학교 정보통신학과 (공학석사)
2008년 충남대학교 정보통신공학과 박사수료
1993년～2001년 (주)데이콤 중앙연구소 선임연구원
2001년～2002년 (주)맥스웨이브 책임연구원
2002년～현 재 한국과학기술정보연구원 선임연구원
관심분야 : 망 성능 분석, 망 자원 관리 등

정 진 욱

e-mail : jwchung@skku.edu
1974년 성균관대학교 전기공학과(학사)
1979년 성균관대학교 전자공학과(석사)
1991년 서울대학교 전자계산학과(박사)
1973년～1985년 한국과학기술연구소 실장
1992년～1993년 미국 Maryland 대학교 객원교수
1997년～1998년 컴퓨터 침해사고 대응팀 협의회 운영위원장
1985년～현 재 성균관대학교 정보통신공학부 교수
2007년～2008년 성균관대학교 일반대학원장
2006년～현 재 IT 전문가협회 부회장
2009년～현 재 인터넷윤리 실천협의회 공동회장
관심분야 : 컴퓨터 네트워크, 네트워크 프로토콜, 인터넷 윤리