SFV: 네트워크 가상화를 위한 플로우바이저의 지연 시간 감소 및 확장성 향상 연구

(SFV: Scalable FlowVisor Platform for Latency Reduction and Network Virtualization)

김 동 현 † 이 문 영 ††
(Donghyun Kim) (Munyoung Lee)
곽 명 철 †† 권 태 경 †††
(Myungchul Kwak) (Taekyoung Kwon)
최 양 희 †††
(Yanghee Choi)

요 약 현재 인터넷은 통신 환경의 급격한 변화와 새로운 서비스의 출현, 사용자의 요구사항 증대로 더욱 다양하고 혁신적인 서비스와 기능 및 자원들을 지원할 수 있는 기술이 요구 되고 있다. 네트워크 가상화는 네트워크의 물리적 자원을 가상화하는 동시에 여러 가상 네트워크들이 공존할 수 있도록 하는 역할을 하는 대표적인 기술이다. 본 논문에서는 네트워크 가상화를 위해 사용되는 플랫폼 중의 하나인 플로우바이저(FlowVisor)에서 발생하는 메시지 지연 문제의 원인을 분석한다. 또한 개선된 탐색 과정을 통해 지연 시간이 감소하고 엔트리 확장성이 증가된 SFV(Scalable Flowvisor)를 제안하고 성능을 검증한다.

키워드: 네트워크 가상화, SDN, 오픈플로우, 플로우 바이저

Abstract The rapid changes of communication environments and users' requirements need novel technologies for the current Internet in order to support various innovative services with limited network resources. A network virtualization is considered to be a promising solution that virtualizes the network resources to be shared among multiple virtual networks. In this paper, we study message latency of FlowVisor that is one of network virtualization platforms. Moreover, we propose a Scalable FlowVisor (SFV) scheme which reduces the latency of FlowVisor with enhanced entry look up process. It is, with measurement results with a network emulator, shown that our scheme can enhance the entry scalability of FlowVisor.

Keywords: Network Virtualization, SDN, OpenFlow, FlowVisor

1. 서 론

현재 인터넷에서 데이터가 목적지에 도착하기 위해서는 패킷이 여러 사업자의 라우터와 스위치를 거쳐서 목적지까지 전달되어야 한다. 그런데, 인터넷을 구성하는 라우터와 스위치는 장비를 생산하는 업체에 따라 사용 방식이나 인터페이스 설정 방법 등에 차이가 있기 때문에 네트워크 사업자들은 관리의 편의성을 위해 특정 회사 제품만 지속적으로 구입하거나 개방형 인터페이스를 갖춘 고가의 동기화 시스템을 구입해야 하는 경우가 많았다. 최근에는 네트워크 가상화 기술을 통해 네트워크 서비스를 네트워크 라우팅 기능과 하드웨어 기반의 스위치로 분리하여 이러한 문제점을 해결하려는 시도가 있다[1,2]. 가상화 기술을 적용하면 다수의 물리적 자원을 하나의 논리적 장치로 사용하거나 하나의 물리적 자원을 복수의 서로 다른 용도로 분할하여 사용하는 것이 가능하다는 특징이 있다.

이러한 가상화 기술의 특징을 기반으로 하여 SDN (Software Defined Networking)[3,4]이 제안되었다. SDN은 컨트롤 플래인의 분리 및 중앙 집중형 제어 방식으로의 패러다임 변화를 통해 현재 인터넷이 폭넓게 보장하지 못하고 있는 개방성과 관리 효율성을 지원하는 것이 목적이다. 통신 사업자나 네트워크 운영자 입장에서는 가상화 기술의 유연한 네트워킹 자원 할당 및 SDN의 중앙 집중형 제어 방식을 통해 전체 네트워크를 포괄하는 하향식 관점에서 보다 더 효율적으로 네트워

· 이 논문은 2013년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No.2013R1A2A2A01016562)
· 이 논문은 제40회 추계학술발표회에서 '네트워크 가상화를 위한 플로우 바이저 플랫폼 확장성 연구'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
dhkim@mmlab.snu.ac.kr
†† 비 회 원 : 서울대학교 컴퓨터공학부
mylee@mmlab.snu.ac.kr
mckwak@mmlab.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
tkkwon@snu.ac.kr
(Corresponding author임)
yhchoi@snu.ac.kr

논문접수 : 2014년 1월 22일
심사완료 : 2014년 3월 19일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제20권 제6호(2014.6)

크를 관리할 수 있다는 이점을 얻을 수 있다.

네트워크를 가상화하기 위해 개발된 플로우바이저 플랫폼(FlowVisor Platform)[5]은 SDN 패러다임 하에서 컨트롤 플래인 상의 다수의 콘트롤러들과 물리적인 네트워킹 자원들로 구성되어 있는 포워딩 플래인 사이에 위치하여 동작한다[6]. 플로우 바이저의 역할은 가상화된 네트워크를 슬라이스라는 논리적 단위로 나누고, 전달된 패킷이 어떤 슬라이스에서 요청된 것인지 판단하여 해당 슬라이스를 관리하는 컨트롤러로 요청을 보내 전달 경로를 설정하는 것이다.

플로우바이저 플랫폼의 대표적인 기술적 요구사항은 엔트리 확장성이다. 즉, 가상화된 네트워크가 많거나 등록된 플로우 엔트리가 많을 때에도 요청된 패킷들을 빠르게 처리하는 것이 필수적이다. 하지만 플로우바이저에 다수의 슬라이스와 플로우 엔트리를 삽입하고 패킷 처리 시간을 측정해 본 결과, 엔트리 숫자가 늘어날수록 포워딩 플래인과 컨트롤 플래인 사이에서 교환되는 '메시지의 지연 현상'이 발생하는 것을 알 수 있었다. 이는 플로우바이저가 하나의 패킷 플로우를 처리할 때 각 슬라이스와 등록된 플로우 엔트리를 모두 탐색하는 방식으로 동작하기 때문이다. 따라서 현재의 플로우바이저는 지연시간이 길어질 수 밖에 없는 한계성을 내포한다. 본 논문에서는, 플로우바이저의 엔트리 확장성을 높히기 위해 탐색 시간 단축 알고리즘을 적용한 SFV(Scalable Flow-visor)를 제시한다.

본 논문은 총 6절로 구성되어 있다. 2절에서는 네트워크 가상화에 대한 설명과 관련 연구, 오픈플로우, 플로우바이저 플랫폼에 대해 기술한다. 3절에서는 메시지 지연 발생 문제에 대한 현상을 분석하고, 4절에서는 탐색 시간을 줄이기 위한 개선 기법을 제안한다. 5절에서는 제안한 기법을 적용한 실험과 그 결과를 기술하고, 마지막 6절에서 결론을 맺고 향후 연구 과제에 대해서 검토한다.

2. 관련 연구

2.1 네트워크 가상화(Network Virtualization)

스마트폰, 랩탑, 태블릿 등 개인 모바일 기기의 빠른 보급은 네트워크에서 발생하는 트래픽을 급증시키고 있다. 이에 따라 통신 사업자들은 원활한 서비스 제공을 위해 네트워크 용량을 지속적으로 확장해 왔다. 최근에는 사업자들이 단순히 네트워크 용량을 확장하기 보다 네트워크 가상화 기술을 도입하여 이 문제를 근본적으로 해결하려 노력하고 있다. 또한 기업 내에서 퍼블릭 클라우드 혹은 프라이빗 클라우드를 이용한 솔루션의 도입이 늘어나면서 네트워크 자원 공유에 대한 수요 또한 증가하고 있다.

네트워크 가상화는 가상 자원 공유를 위해 물리적 인프라와 인터넷 서비스 제공자를 분리할 수 있게 해준다. 즉, 네트워크 가상화는 물리적인 네트워크 장비를 여러 대 설치하는 대신 하나의 네트워크 장비가 여러 네트워크에 포함될 수 있도록 컨트롤 플래인과 하드웨어 포워딩 플래인을 분리한다. 이 기술을 이용하면 하나의 물리적인 네트워크 위에서 다양한 가상 네트워크가 존재하여 독립적으로 특정 목적의 기능을 수행할 수 있고, 필요에 따라 상호 연동도 가능하다. 따라서 네트워크 사업자가 특정 회사 제품에 한정되어 네트워크를 운영하거나 값비싼 동기화 시스템을 구축할 필요가 없다. 또한, 중앙 집중형 제어 방식을 통해 쉽게 모든 네트워크 장비에 접근 및 관리할 수 있다.

2.2 오픈플로우(OpenFlow)

오픈플로우는 NOX[7], Floodlight[8], Maestro[9] 등을 이용해 구현된 중앙 컨트롤러가 제어 기능이 분리된 스위치에 흐르는 네트워크 플로우들을 제어하고 쉽게 모니터링할 수 있도록 네트워크 스위치(혹은 라우터)와 컨트롤러 간의 통신을 위한 프로토콜을 제공한다. 그림 1은 오픈플로우 기반 네트워크의 제어 기능 중 가장 대표적인 플로우 셋업 과정을 나타낸 것이다. 어떤 플로우가 스위치에 도착했을 때, 플로우 엔트리 테이블에 해당 경로 정보가 존재할 경우에는 정해진 경로로 패킷을 전송한다. 스위치의 플로우 엔트리 테이블은 컨트롤러에 의해 설정된다.

만약 테이블에 해당 플로우에 매칭되는 플로우 엔트리가 없는 경우, 스위치는 독자적으로 라우팅 등의 네트워크 제어 연산을 수행할 수 없기 때문에 Secure Channel을 통해 해당 플로우에 대한 정보를 컨트롤러로 전송하고, 컨트롤러는 해당 플로우에 대한 경로를 계산하여 플로우 엔트리를 스위치로 전송하는 과정을 거치게 된다.

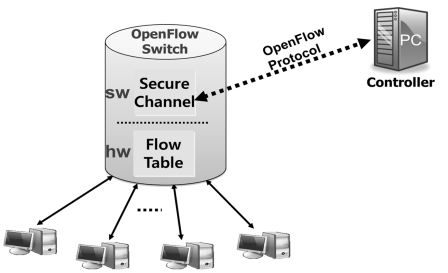

그림 1 오픈플로우 스위치의 플로우 셋업 과정
Fig. 1 Flow setup process of Openflow switch

2.3 플로우바이저(FlowVisor)

플로우바이저[5]는 오픈플로우 기반의 대표적인 스위치 가상화 기술[6]로, 같은 하드웨어 포워딩 플래인을 다수의 논리적인 포워딩 로직이 공유하고 컨트롤할 수

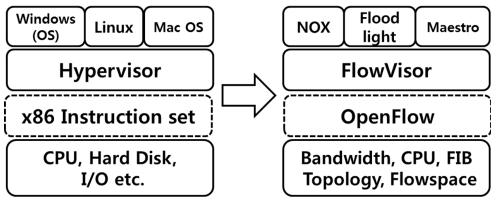

그림 2 컴퓨터 가상화 및 스위치 가상화 계층도
Fig. 2 Computing/switch virtualization architecture

있도록 하는 플랫폼이다.

그림 2는 플로우바이저의 가상화 방식을 컴퓨터 가상화 방식과 비교한 계층도이다. 먼저 왼쪽 그림은 독립적인 운영 체제를 갖는 여러 대의 가상 머신(Virtual Machine)들이 물리적인 자원 즉, CPU, 메모리, 하드디스크 등을 하이퍼바이저를 통해 분할해서 사용하는 컴퓨터 가상화 기술을 의미한다. 그림 2의 오른쪽 그림은 플로우바이저의 네트워크 가상화 방식을 나타낸 것으로 플로우바이저는 왼쪽 그림의 컴퓨팅 자원 가상화 기술과 유사하게 물리계층인 스위치 하드웨어와 이를 컨트롤하는 독립적인 개체인 컨트롤러 및 컨트롤 어플리케이션 사이에 위치하며, 오픈플로우 프로토콜을 통해 다수의 가상 네트워크가 물리적 스위치를 공유하도록 중재하는 역할을 한다.

3. 플로우바이저의 메시지 교환 지연 문제

플로우바이저는 네트워크의 스위치와 링크 등의 물리적인 자원을 슬라이스라는 단위로 나누어 다수의 컨트롤러에게 할당한다. 또한 스위치에서 보내는 오픈플로우 메시지를 해당 슬라이스에 속한 컨트롤러로 전송하여 스위치의 동작을 제어하는 역할을 한다. 이를 위해 플로우바이저는 컨트롤러마다 해당되는 슬라이스와 플로우 스페이스 테이블을 유지하게 된다. 각 슬라이스는 네트워크 트래픽의 일부를 컨트롤할 수 있으며, 각 트래픽 플로우를 구별하기 위해 패킷 헤더에 존재하는 TCP 포트, MAC 주소와 IP주소를 이용해 플로우 스페이스를 구성하게 된다. 하지만 슬라이스의 숫자가 증가하고 등록된 플로우스페이스가 많아질수록 스위치에서 컨트롤러로 경로 정보를 요청하고 받는 '메시지 교환 지연 현상'이 발생한다는 문제가 존재하였다. 이 현상을 분석하기 위해 플로우 엔트리를 요청하고 받는 메시지 교환 과정을 살펴보자면, 그림 3과 같다. (1) 클라이언트가 패킷을 전송했을 때 스위치에 플로우 엔트리가 존재하지 않는다면, (2) 스위치는 컨트롤러로 경로 정보를 요청하게 된다. (3) 이 때 플로우바이저는 스위치와 컨트롤러 사이에서 이 메시지를 가로챈 후 패킷에 대한 플로우 스페이스를 판별하여 해당 슬라이스의 컨트롤러로 메시지를 전송해준다. (4) 컨트롤러에서 패킷 처리에 대한 룰 정보가 담긴 플로우 엔트리를 스위치로 전송할 경우, (5) 플로우바이저는 이 룰이 담긴 메시지를 전달 받아서 해당 슬라이스 자원에 적합한 룰인지를 판별하여 요청한 스위치로 전송한다.

이와 같은 메시지 교환 과정에서 플로우바이저는 스위치에서 컨트롤러로 플로우 엔트리 요청메시지를 전달할 때와 컨트롤러에서 스위치로 플로우 엔트리 설정 메시지를 전송할 때 두 번의 플로우 스페이스 탐색을 수행한다. 그런데 이 탐색을 수행할 때, 플로우바이저에서는 각 슬라이스 마다 할당된 플로우 스페이스 엔트리를 순차 검색하도록 구현이 되어있는 것을 발견할 수 있었다. 즉, 하나의 슬라이스의 플로우 스페이스를 찾더라도 전체 엔트리를 탐색하도록 구현이 되어 있기 때문에 엔트리 수가 증가함에 따라 불필요하게 지연시간이 증가하는 문제를 확인할 수 있었다.

실제로 Mininet[9]을 이용하여 가상의 토폴로지를 생성하고, FlowVisior자바 오픈 소스와 C++ 기반의 NOX컨트롤러를 연동한 후(자세한 실험환경은 5절의 내용과 동일하다), 오픈플로우의 메시지 지연 시간을 측정해보았을 때 등록된 플로우 엔트리의 수가 많아질수록 지연시간이 증가하는 것을 확인할 수 있었다. 즉, 그림 4와 같이 1개의 플로우 스페이스 엔트리를 보유할 경우 평균 약 1.9ms의 지연시간이 발생하지만 1,000개의 엔트리가 등록되어 있을 경우에는 평균 약 2.4ms로 지연시간이 발생하는 것을 알 수 있었다. 또한, 3,000, 6,000엔트리의 경우 약 3.0ms, 4.0ms으로 지연시간이 계속 증가하는 것으로 나타났다. 이는 플로우바이저가 일치하는 플로우스페이스 엔트리를 찾기 위해 다른 슬라이스의 중복된 엔트리까지 검색하도록 구현되어 있기 때문에, 엔트리 수가 증가함에 따라 탐색해야 하는 테이블의 정보가 많아져 지연 시간이 증가하는 것으로 추측되었다. 이는 곧 네트워크 인프라를 여러 사업자가 공유할 경우, 엔트리 증가로 인한 플로우 바이저의 확장성 문제로 나타날 수 있다.

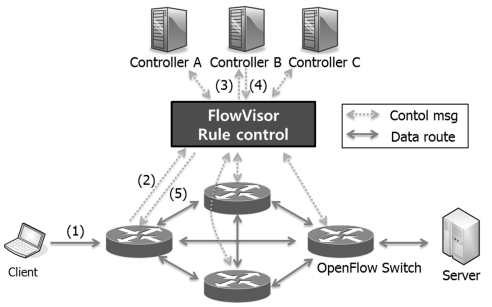

그림 3 플로우바이저 메시지 교환 과정
Fig. 3 Message exchange process of FlowVisor

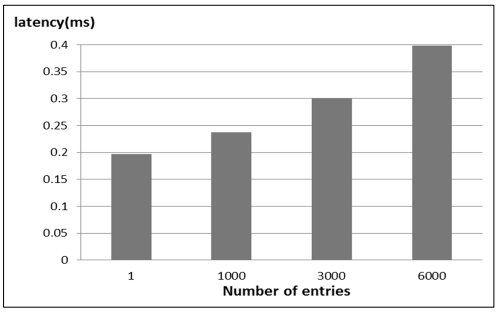

그림 4 플로우스페이스 엔트리 개수에 따른 메시지 교환 지연시간

Fig. 4 Message exchange latencies according to number of flowspace entries

4. Scalable FlowVIsor(SFV)

본 연구에서는 플로우바이저의 엔트리 확장성 및 메시지 지연 발생 문제 해결을 위해 컨트롤러에서 스위치 방향으로 룰이 전송될 때의 탐색 과정을 개선한 Scalable FlowVisor (SFV)를 제안한다. 스위치에서 컨트롤러로 올라가는 상향 탐색 과정은 메시지가 속하는 컨트롤러를 판단하는 과정이지만 하향 탐색 과정의 경우, 룰 메시지를 전송하는 컨트롤러의 이름을 알 수 있기 때문에 플로우바이저 내부에서 해당 컨트롤러의 플로우 스페이스 엔트리만 검색하여 탐색 시간을 단축할 수 있다.

```
{*
ruleset : flowspace entries of all controllers
rule.get(int i) : function that gets i-th rule in the ruleset
slice_name_list : list of slice name on flowvisor
MatchType : user-defined data set {NONE, EQUAL}
Match(int a, int b) : comparison function, if a and b are matched,
                      return EQUAL, else return NONE
}*
TOP: for each entry i in the ruleset
        a ← rule.get(i)
        b ← rule message (received from controller)

        for each entry j in the slice_name_list
            if
              j equals to name of slice of controller that sends b,
              check = true
            else
              check = false
            endif
        endfor

        if(check == false)
          goto TOP
        endif

        MatchType = Match(dest address of a, dest address of b)
        if
            MatchType == NONE, then goto TOP
        endif
        MatchType = Match(src address of a, src address of b)
        if
            MatchType == NONE, then goto TOP
        endif
    endfor
```

그림 5 SFV의 하향 탐색 과정 pseudo code

Fig. 5 Pseudo code of downward look-up process of SFV

그림 5는 플로우바이저 오픈 소스의 탐색 과정을 개선한 SFV의 핵심 부분을 pseudo code로 나타낸 것이다. 기존 플로우바이저는 컨트롤러에서 경로 정보가 담긴 오픈플로우 메시지를 전달받은 후, 등록되어 있는 모든 플로우 스페이스 엔트리를 순차 탐색하지만, SFV는 플로우 스페이스 엔트리 탐색과정을 진행하기 이전에 메시지의 출처를 파악하는 매칭 작업을 선행한다. 이 작업은 저장된 슬라이스 네임을 리스트를 만들어 저장하는 것으로부터 시작이 된다. 이 후, 하향 룰 메시지를 전송한 컨트롤러의 슬라이스 이름과 리스트에 등록된 슬라이스에서 동일한 이름을 가진 슬라이스의 플로우 스페이스 엔트리를 탐색하는 과정이 이어진다. 이 과정에서 해당되지 않는 슬라이스의 플로우 스페이스 엔트리는 걸러지며, 적정 슬라이스만을 탐색한 결과를 오픈플로우 룰메시지로 스위치에게 전송한다. 따라서 SFV는 중복된 엔트리에 대한 검색을 줄이고 해당 슬라이스의 자원만을 검색하도록 하여 검색의 효율성을 향상 시킬 수 있다.

5. 성능 평가

5.1 평가 실험 환경

성능 분석을 위해 Intel Core i3 2100, 2GB의 메모리 사양을 갖는 호스트 PC에 Mininet을 이용하여, 그림 6과 같이 노드 2개(클라이언트, 서버), 스위치를 포함하는 1Gbps 대역폭의 네트워크를 구성하였다. Mininet은 가상 네트워크 생성을 지원하는 SDN에뮬레이터로 호스트, 스위치, 라우터, 링크를 하나의 커널 상에서 생성하고 테스트 할 수 있다. 생성된 네트워크 위에 플로우 바이저를 연동하고 Intel Core i3 2100, 4GB 메모리 사양을 갖는 NOX 컨트롤러를 플로우바이저와 연결하여 실험 환경을 구성하였다.

클라이언트는 50 packets/sec의 속도로 스위치로 패킷을 전송하도록 설정하였고, 스위치에서 각 패킷들에 대해 지속적으로 경로 정보 요청이 발생하도록 컨트롤러에서 실제 경로 정보가 아닌 더미(dummy) 경로를 반복 생성하여 전송하도록 설정하였다.

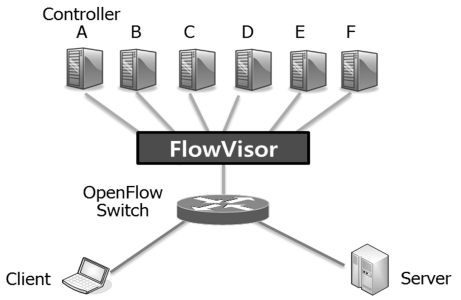

그림 6 실험 환경 및 시나리오

Fig. 6 Experimental settings and scenario

플로우바이저에는 컨트롤러 A, B, C, D, C, E, F를 차례대로 등록하고 1000개씩의 플로우스페이스 엔트리를 각각 삽입하였으며, 컨트롤러 하나씩 플로우바이저에 추가하면서 실험을 진행하였다. SFV는 컨트롤러 A의 슬라이스 내 플로우스페이스 엔트리만을 탐색하게 되며, 성능 분석을 위해 클라이언트에서 패킷을 전송한 시간과 SFV가 엔트리 탐색을 끝낸 후, 룰메시지를 스위치로 전송하는 시간 차이를 측정하였다.

5.2 성능 개선 기법 (SFV) 실험 결과

개선된 SFV의 지연 시간을 기존의 플로우바이저의 지연 시간과 비교해본 결과, 그림 7과 같이 개선된 성능을 확인할 수 있었다. 결과를 자세히 살펴보면 표 1과 같이 엔트리가 2,000개인 경우 기존 플로우바이저에 비해 약 8% (평균:2.63ms → 2.43ms), 엔트리가 3,000개인 경우에는 약 17% (평균: 3.01ms → 2.51ms), 엔트리 수가 가장 많은 6,000개의 경우에는 약 30% (3.98ms → 2.77ms)의 성능 향상을 보임을 확인할 수 있었다. 이는 기존 플로우바이저에서는 전체 컨트롤러의 플로우스페이스 엔트리를 탐색하는 과정을 거치면서 전체 엔트리 개수에 따라 탐색 시간이 선형 증가 양상을 보이는 문제가 있지만, SFV에서는 특정 컨트롤러에게 할당된 엔트리만 탐색함으로써 탐색 성능을 개선할 수 있었기 때문이다.

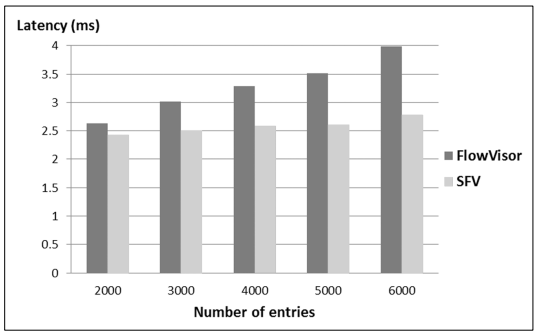

그림 7 SFV와 플로우바이저의 지연시간 측정 비교
Fig. 7 Latency Comparison of FlowVisor and SFV

표 1 플로우스페이스 엔트리 개수에 따른 메시지 교환 지연시간
Table 1 Message exchange latencies according to number of entries

Number of entries	FlowVisor (ms)	SFV (ms)	Reduction ratio(%)
2000	2.63	2.43	7.59
3000	3.01	2.51	16.61
4000	3.29	2.59	21.17
5000	3.51	2.60	25.80
6000	3.98	2.77	30.36

6. 결 론

본 논문에서는 플로우바이저에서 발생하는 메시지 교환 지연 문제의 원인을 분석하여, 불필요한 엔트리를 탐색하는 기존 플로우바이저의 구현 상의 비효율적인 부분을 확인하였다. 그리고 메시지 교환 지연 문제 해결 및 플로우바이저의 엔트리 확장성을 위해 플로우에 해당하는 컨트롤러의 슬라이스 자원만을 탐색하는 방식으로 개선한 Scalable FlowVisor(SFV)를 제안하였다. 성능 분석 결과, 제안된 SFV는 기존의 플로우바이저 플랫폼의 순차 검색으로 인한 지연시간을 감소시키고, 플로우바이저의 엔트리 확장성을 높일 수 있음을 확인할 수 있었다. 향후에는 실제 SDN 테스트베드에서 노드와 스위치, 컨트롤러 숫자를 추가하여 보다 다양한 시나리오에 대한 SFV의 성능을 분석하고, 상향 탐색 과정에 대한 지연시간을 줄이는 방안에 대해 추가 연구를 진행할 계획이다.

References

[1] Chowdhury, et al., "Network virtualization: state of the art and research challenges," *IEEE Communications Magazine*, vol.47, 2009.
[2] Rob Sherwood et al., "Can the Production Network Be the Testbed?," *Proc. USENIX OSDI*, 2010.
[3] Binyoung Yoon et al., "Future network technique," 2012 Electronics and Telecommunications Trend
[4] Jaehyong Yoo et al., "SDN/OpenFlow trend and prospect," *KNOW Review*, vol.15.
[5] R. Sherwood et al., "FlowVisor: A Network Virtualization Layer," *Proc. USENIX OSDI*, 2010.
[6] Youngwha Kim et al., "Network Virtualization technique of future internet," *Electronics and Telecommunications Trend*, vol.25 1th, 2010. 2, pp.132-147.
[7] http://www.noxrepo.org/_/nox-classic-doxygen/index.html
[8] http://docs.projectfloodlight.org/display/floodlight-controller/The+Controller
[9] Zheng Cai et al., "Maestro: A System for Scalable OpenFlow Control," Tech. Rep. TR10-08, Rice University, 2010.
[10] Mininet, http://mininet.org