J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

사용자 정의 네트워크를 위한 사용자 데이터그램 프로토콜 플로우 엔트리 관리 기법☆

UDP Flow Entry Management for Software-Defined Networking

최 한 힘 나 라[1] 샤이드 무하마드 라자[2] 김 문 성[4*] 추 현 승[3**]
Hanhimnara Choi Syed Muhammad Raza Moonseong Kim Hyunseung Choo

요 약

소프트웨어 정의 네트워킹(SDN)은 스위치의 데이터 플레인에서 컨트롤 기능을 분리해 프로그램 가능한 네트워크 관리법을 제시하는 차세대 네트워킹 기술이다. 하지만 SDN 스위치는 패킷 포워딩에 쓰이는 플로우 테이블의 부족한 용량으로 인해, 플로우 엔트리를 충분히 수용하지 못하는 문제가 있다. 이는 플로우 테이블의 오버플로우 등을 야기해 네트워크 성능을 저하시킬 수 있으므로, 본 논문은 비활성화된 플로우를 적시에 방출해 플로우 테이블 사용량을 최적으로 유지할 수 있는 정책을 제안한다. 특히, 사용자 데이터그램 프로토콜(UDP) 플로우 엔트리들의 정보를 주기적으로 샘플링하여 비활성화된 엔트리들이 조기에 방출될 수 있도록 한다. 트래픽에 기반한 실험을 통하여, 우리는 본 정책이 Random, FIFO 정책과 비교해 오버플로우 및 플로우 재설치 발생 횟수를 크게 감소시키는 것을 확인할 수 있었다.

☞ 주제어 : 소프트웨어 정의 네트워킹, 오픈플로우, 플로우 테이블 관리기법

ABSTRACT

Software-defined networking provides a programmable and flexible way to manage the network by separating the control plane from data plane. However, the limited switch memory restricts the number of flow entries in the flow table used to forward packets. This leads to flow table overflow and flow entry reinstallation, which severely degrade the network performance. Therefore, this paper proposes a comprehensive policy for timely eviction of inactive flow entries to optimally maintain flow tables usage. In particular, statistics of user datagram protocol flow entries are periodically sampled to enable the inactive entries to be evicted early. Through traffic-based experiments, we found that the proposed system reduces the number of overflow occurrences and flow entries reinstallation compared to the random and FIFO policies.

☞ keyword : Software-Defined Networking, OpenFlow, Flow table management

1. 서 론

소프트웨어 정의 네트워킹(SDN)은 기존 네트워크의 컨트롤 플레인과 데이터 플레인을 분리해, 컨트롤 기능을 논리적으로 중앙화된 SDN 컨트롤러에 집중시키는 차세대 네트워킹 기술이다[1]. 데이터 플레인에서 컨트롤 플레인을 분리시킴으로써 SDN은 다양한 네트워크 관리 애플리케이션을 통한 유연한 네트워크 관리를 가능케 한다. 이러한 네트워크 관리 방식은 컨트롤러와 데이터 플레인 간 표준화된 통신을 통해 이루어지며, 이 중 OpenFlow는 현재 SDN 기술의 기반이 되는 표준 통신 프로토콜이다. SDN에서는 스위치 내 정의된 플로우 테이블을 통하여 스위치가 네트워크로 접근하는 플로우의 패킷들을 포워딩한다. 플로우 테이블은 플로우 엔트리의 집합으로 플로우 엔트리는 플로우 패킷들에 대한 매칭 정보와 그 처리

1 Department of Artificial Intelligence, Sungkyunkwan University., Suwon, 440-330, Korea.
2 Department of Electrical and Electronic Engineering, Sungkyunkwan University, Suwon, 440-330, Korea.
3 College of Computing, Sungkyunkwan University, Suwon, 440-330, Korea.
4 Department of IT Convergence Software, Seoul Theological University, Bucheon, 422-742, Korea.
* Corresponding author (choo@skku.edu and moonseong@stu.ac.kr)
[Received 21 October 2020, Reviewed 3 November 2020(R2 1 January 2021), Accepted 16 February 2021]
☆ 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 ICT명품인재양성 사업 (IITP-2021-2020-0-01821), 글로벌핵심인재양성지원사업 (2019-0-01579), 2021년도 정부(과학기술정보통신부)의 지원(No.2019-0-00421, 인공지능대학원지원(성균관대학교))과 한국연구재단의 지원(NRF-2020R1A2C2008447)을 받아 수행된 연구임.
☆ 본 논문은 2020년도 한국인터넷정보학회 춘계학술발표대회 우수논문 추천에 따라 확장 및 수정된 논문임.

방식에 대한 정보를 담고 있다[2]. 컨트롤러는 스위치와의 표준 프로토콜 메시지를 교환하며 플로우 테이블 내 플로우 엔트리를 추가하거나 삭제하는 등의 작업을 수행하며, 스위치에 접근하는 패킷은 해당하는 플로우 엔트리에 매칭되어 포워딩 처리된다.

스위치에는 주로 빠른 탐색속도를 가진 3 진 내용 주소화 기억장치(TCAM)가 사용되는데[3], 현재의 SDN 네트워크에서 사용되기에 TCAM은 그 용량에 한계가 있다. 이로 인해 스위치에 충분한 수의 플로우 엔트리를 수용하지 못하게 된다. 플로우 테이블이 가득 찼을 때, 스위치에 새로운 플로우가 접근하면 플로우 테이블 오버플로우가 발생할 수 있으며 플로우 테이블에 새로운 플로우를 위한 포워딩 룰을 설치할 수 없게 된다. 결과로, 스위치는 새로이 접근하는 플로우의 패킷을 매칭하는 데 실패하게 된다(테이블 미스). 테이블 미스가 일어나면 해당 패킷을 처리하기 위한 컨트롤러-스위치 간 통신이 필연적으로 발생하여 네트워크 컨트롤 오버헤드를 크게 증가한다. 더 나아가, 플로우 테이블 오버플로우는 네트워크에 추론 공격 등의 보안 공격을 가하는 데에 이용될 수 있다[4]. 따라서 네트워크의 성능 하락과 이러한 위협들을 예방하기 위해서는 비활성화된 플로우들을 적시에 방출해야 한다.

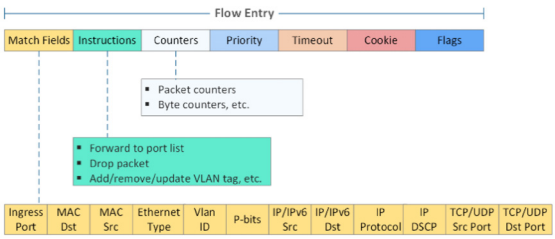
이에 플로우 테이블의 효율적인 사용을 위한 여러 플로우 엔트리 관리 기법들이 제안된 바 있다[5]. 대표적으로 FIFO, Random과 같은 전통적인 알고리즘을 사용해 방출의 우선순위를 정하는 연구가 제시된 바 있다[6]. 이외에도 다양한 알고리즘을 이용하거나 플로우의 프로토콜 특성을 이용하여 방출 전략을 구성하는 연구들도 제시되었다. 하지만 많은 연구들이 SDN에서 전송제어 프로토콜(TCP; Transmission Control Protocol)에 대한 이슈를 다루고 있는 반면[7], 사용자 데이터그램 프로토콜(UDP; User Datagram Protocol) 플로우를 위한 세밀한 플로우 테이블 관리 기법은 제시된 바 없다. 만약 UDP 플로우가 DNS resolution과 같이 네트워크 컨트롤을 위해 단발적인 패킷 전달에만 쓰이는 경우 이전의 정책들로도 충분하다. 하지만 미래의 네트워크에서 UDP 플로우는 다양한 쓰임새를 가질 수 있다. 특히, UDP 기반의 새로운 프로토콜인 QUIC이 구글에서 개발되었으며 이는 HTTP3 표준으로 채택되었으며[8] 이는 UDP 플로우의 패턴을 더욱 다양하게 만들 것이다. 그러므로 다양한 패턴을 지닌 UDP 플로우들을 관리할 수 있는 일반적이며 효율적인 플로우 엔트리 관리 정책이 필요하다.

따라서 본 논문은 UDP 플로우에 대해서도 비활성화된 플로우의 조기 방출을 실시하는 시스템을 제안한다. 제안 시스템은 UDP 플로우에 대한 정보를 주기적으로 업데이트해 이를 기반으로 비활성화된 플로우를 조기에 방출하여, 효율적인 플로우 테이블 사용을 목표로 한다.

2. 연구 배경 및 관련 연구

2.1 플로우 테이블 관리기법

SDN은 원칙적으로 제어 평면 및 데이터 평면을 분리하고, 정의된 API를 이용하여 프로그래밍 기반의 중앙집중화된 제어를 가능하게 한다. SDN의 평면 논리 구조는 어플리케이션들이 자리할 수 있는 응용 평면, SDN 제어기가 위치하는 제어 평면, 실제 데이터가 흐르고 처리되는 데이터 평면으로 나뉜다. 이때 SDN 제어기와 네트워킹 장치 간 통신의 대표적인 표준 프로토콜이 OpenFlow이다. OpenFlow 기반 네트워크에서는 스위치에 접근하는 플로우 패킷들이 스위치 내 정의되어있는 플로우 테이블을 통하여 포워딩된다[9]. 플로우 테이블은 패킷 흐름 처리 및 관리를 위한 기능을 명세하는 다수의 플로우 엔트리로 구성될 수 있다. 플로우 엔트리는 그림 1에서 볼 수 있듯이, 크게 match fields, actions, 그리고 counters(통계정보)로 구성된다. Match field에서는 매칭 정보를, action은 그에 맞는 패킷의 처리 방식을 정의하며 counters를 이용하여 플로우의 통계정보를 기록한다. OpenFlow에서는 컨트롤러의 요청에 따라 이러한 엔트리를 플로우 테이블에 설치하여 접근하는 패킷이 미리 정해진 방식에 따라 포워딩 된다.



(그림 1) 플로우 엔트리의 구조

(Figure 1) Structure of a flow entry

플로우 테이블 내에 패킷을 처리하기 위한 플로우 엔트리를 설치 및 관리하기 위해서 컨트롤러는 스위치에 요청 메시지를 보내게 된다. OpenFlow의 Flowmod 메시지를 통해 컨트롤러는 플로우 엔트리의 설치, 삭제 및 변경을 요청할 수 있다. 예로, 매칭이 되지 않는 패킷이 스위치에 도착하면 스위치는 이를 처리할 플로우 엔트리를 설치하기 위하여 컨트롤러에 해당 패킷을 packet_in 메시

지를 통해 전달하게 된다. 컨트롤러는 packet_in 메시지를 받은 후 스위치에게 해당 패킷을 처리할 수 있는 플로우 엔트리를 설치하라는 flowmod 메시지를 보낸다. 더 나아가 특정 플로우 엔트리를 삭제하는 경우에도 특정 파라미터가 추가된 flowmod 메시지를 전송하여 원하는 매칭 정보를 가진 플로우 엔트리를 삭제할 수도 있다.

한편, 한 번 설치된 플로우 엔트리는 여러 가지 방법으로 테이블 내에서 삭제될 수 있다. 그 중 한 방법은 플로우 엔트리를 설치할 때 timeout 값을 지정하여 플로우가 일정 시간이 지난 후 만료되도록 하는 것이다. 이 timeout 값들은 플로우 엔트리를 설치하기 위해 컨트롤러가 보내는 메시지에 파라미터로 전달될 수 있다. OpenFlow는 일정시간 이상 매칭이 일어나지 않은 플로우 엔트리를 테이블 내에서 만료시키기 위해 Hard timeout과 Idle timeout 기능을 제공한다. Hard timeout의 경우 플로우 엔트리가 설치되고 나서 단순히 지정된 시간만큼이 지났을 때 플로우 엔트리가 만료되도록 하며, Idle timeout의 경우는 플로우 엔트리에 매칭되는 패킷이 없는 idle 상태에서 일정 시간이 지났을 때 플로우 엔트리가 만료되도록 한다. 하지만 timeout 기능만으로는 오버플로우가 발생하거나 비활성화된 플로우 엔트리가 있을 때 능동적으로 플로우 엔트리를 방출할 수 없기 때문에 이런 경우에 대처할 수 있는 방출 정책이 따로 필요하다.

2.2 관련 연구

따라서 오버플로우의 발생 등으로 인해 플로우가 방출되어야 할 때 방출되기 적합한 플로우 엔트리를 선택하는 효율적 방법에 대한 연구들이 제안되어 왔다. 이 연구들은 크게 두 가지로 나눌 수 있다. 먼저 오버플로우가 발생한 상황에서 방출해야하는 플로우 엔트리의 우선순위를 정하는 방법에 대한 연구들이 있다. 예로, 캐시 교체 등의 관리 기법들에서 쓰이던 FIFO, Random 등의 알고리즘을 차용한 연구들이 있다[6]. 두 번째로 오버플로우에 대처하기 위하여, 테이블에 남은 공간이 사라지기 전에 플로우 엔트리를 조기 방출하는 방법들 또한 제안된 바 있다. 연구 [10]은 테이블 사용량이 일정 임계치를 넘어섰을 때 주어진 방출 알고리즘에 따라 플로우 방출을 실시할 것을 제안하였다. 이는 플로우를 방출할 때 무엇을 방출할지, 그리고 언제 방출할지 정하는 것 모두 네트워크 성능에 큰 영향을 줄 수 있음을 의미한다.

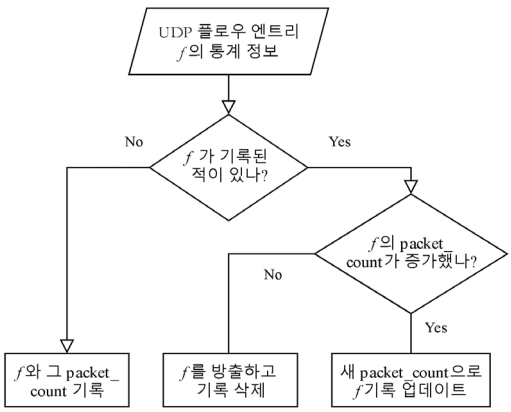
이외에 플로우가 가진 프로토콜별 특성을 고려한 방출 정책 또한 제시되었다. 연구 [11]은 TCP 플로우에서 명시적인 연결 종료를 나타내기 위해 전송되는 FIN 플래그를 이용해 비활성화된 TCP 플로우 엔트리를 조기에 플로우 테이블에서 제거하는 정책을 제시하였다. 하지만 UDP 플로우에 대해서는 임계치를 정하고 패킷이 그 임계치 보다 많은 횟수로 접근하는 경우에만 해당 플로우를 설치하는 단순한 방식만이 제시되었다. 따라서 본 논문은 주기적으로 플로우 테이블 내 정보를 수집하여 비활성화된 UDP 플로우 엔트리를 적시에 방출하는 방법을 제안한다.

3. 제안 시스템

본 논문에선 실시간으로 변하는 네트워크에서 플로우 엔트리의 통계 정보를 기반으로 비활성화된 UDP 플로우 엔트리의 조기 방출 전략을 제안한다. 더 나아가, [11]에서 제안된 기존의 TCP 플로우 조기 방출 정책과 제안 UDP 방출 정책을 결합하여 최종적으로 프로토콜에 기반한 종합적인 플로우 테이블 관리 시스템을 제안한다.

UDP 플로우는 연결 종료 등에 대한 명시적 신호를 전송하지 않으므로 플로우가 비활성화되는 시점에 그 플로우 엔트리를 방출하기가 어렵다. 하지만 SDN 네트워크에서 컨트롤러는 스위치에게 플로우 테이블 내에 설치된 플로우 엔트리의 통계 정보를 요청할 수 있으며, 이 통계 정보에는 플로우의 지속기간, idle 기간, 매칭된 패킷의 수(packet_count) 등이 있다. 본 시스템은 컨트롤러에서 해당 통계 정보들을 활용하여 UDP 플로우의 대략적 추세를 확인하여 비활성화된 것으로 판단되는 플로우가 있는 경우에 이를 조기에 방출시킨다. 이를 위해 컨트롤러는 주기적으로 스위치에 해당 통계 정보를 요청하는 메시지를 전송한다. 이 요청에 대한 회신으로, 스위치는 플로우 테이블 내 모든 UDP 플로우 엔트리의 packet_count 정보를 담은 메시지를 컨트롤러에 보내게 된다. 컨트롤러는 이 메시지를 수신했을 때 모든 UDP 플로우 엔트리의 정보를 순회하며 각 엔트리에 대한 작업을 수행한다.

스위치로부터 수신한 모든 플로우 엔트리에 대한 정보를 컨트롤러가 처리하는 방식은 그림 2와 같다. 먼저, 컨트롤러는 특정 플로우 엔트리의 packet_count가 이전에 샘플링되어 기록된 적이 있는 지 확인한다. 해당 엔트리의 정보가 이전에 측정된 적이 없을 경우, 그 플로우의 매칭 정보와 packet_count 값이 key-value 형태의 데이터에 기록된다. 만약 플로우 엔트리의 정보가 이전에 기록된 적이 있다면, 기존의 packet_count 값과 새로 갱신된 packet_count 값의 차를 취한다. 만약 이 차가 0이라면 해당 플로우 엔트리에는 일정 주기 동안 매칭된 패킷이 없었다는 뜻이다.



(그림 2) 제안 방출 전략의 플로우차트

(Figure 2) Logic of proposed eviction strategy

따라서 그 UDP 플로우는 비활성화된 것으로 간주하여 그 플로우를 테이블 내에서 방출하고 컨트롤러 내에서 그 플로우의 기록 또한 삭제한다. 반대로 차가 0보다 클 경우에는 해당 엔트리에 패킷이 추가적으로 매칭되어 packet_count 값이 이전과 달라졌으므로 플로우 엔트리 기록을 새로 측정된 packet_count 값으로 업데이트해준다. 이러한 과정을 매 주기 반복하여 비활성화된 것으로 판단되는 UDP 플로우를 테이블에서 조기 방출할 수 있다.

더 나아가 제안시스템은 오버플로우가 일어났을 때 방출하는 플로우 엔트리의 우선순위 또한 컨트롤러에 의해 기록된 UDP 플로우 정보에 기반하여 정한다. 앞서 설명한 바와 같이, 컨트롤러는 UDP 플로우 정보를 기록할 때 특정 플로우 엔트리의 packet_count 증가량을 기록한다. 이를 통하여 제안 시스템은 오버플로우가 발생한 상황에서 새 플로우 엔트리를 설치해야할 때, packet_count 증가량이 기록된 UDP 플로우 중에서 가장 작은 packet_count 증가량을 가진 플로우 엔트리를 먼저 방출한다.

한편 샘플링을 통해 UDP 플로우의 정보를 주기적으로 업데이트 하기 위해서는 먼저 정보 요청 메시지의 전송 주기(sampling period)가 정해져야 한다. 만약 이 주기가 너무 짧은 경우 UDP 플로우가 추후 매칭될 패킷이 있음에도 불구하고 조기에 방출될 수 있다. 반면, 주기가 너무 긴 경우에는 이미 더 이상 매칭될 패킷이 없는 비활성화된 플로우가 오래 동안 플로우 테이블에 남아 공간을 차지하게 될 수 있다. 따라서 제안 시스템은 샘플링 주기를 최소 1초부터 최대 5초까지로 설정하고, 이를 동적으로 조절한다. 컨트롤러는 주기적으로 플로우 테이블의 통계정보를 확인하며, 플로우 테이블 사용량이 이전에 비하여 증가하였을 경우에는 샘플링 주기를 1초 감소시키고 이전에 비하여 감소하였을 경우에는 샘플링 주기를 1초 증가 시킨다. 테이블 사용량이 이전과 동일할 경우에는 샘플링 주기를 그대로 유지한다. 이와 같은 방식으로 제안 시스템은 샘플링 주기를 네트워크 환경에 따라 지속적으로 조절한다.

4. 시스템 구현 및 성능평가

4.1 시스템 구현

본 시스템의 성능평가를 위한 시뮬레이션 환경은 Mininet과 파이썬 기반 SDN 프레임워크인 Ryu를 사용해 구현되었다[12, 13]. 파이썬 스크립트를 통해 Ryu 컨트롤러에 기본적인 스위치 구동방식과 제안 로직이 정의되었다. 네트워크의 구성 요소들은 OpenFlow 메시지를 주고받으며 트래픽을 처리한다. 제안 시스템은 스위치가 컨트롤러로 전송하는 다음과 같은 네 가지의 메시지; 새로운 플로우의 패킷이 도착했을 때 발생하는 packet_in, 컨트롤러의 플로우 정보 요청에 대한 회신인 flow_stats_reply, 플로우가 삭제되었을 때 발생하는 flow_removed, 플로우 테이블 오버플로우가 일어났을 때 발생하는 error_tablefull을 통해 상호작용한다. 이 메시지들에 대응해 컨트롤러는 플로우 설치, 플로우 정보 업데이트, 플로우 방출을 실시한다.

본 실험에서 사용된 네트워크 토폴로지는 SDN 컨트롤러와 OpenFlow 스위치 하나, 스무 개의 호스트들로 구성되었다. 각 호스트들은 Mininet에서 구현되어 트래픽을 발생시키기 위해 서로가 전송자 및 수신자가 되어 패킷을 주고받는다. 실험 트래픽이 주어졌을 때, 과부하된 네트워크 트래픽 환경에서 제안 시스템의 성능을 확인할 수 있도록 플로우 테이블 사이즈를 50으로 설정하여 플로우 테이블에 충분한 수의 오버플로우를 발생시켰다. 표 1과 같이, 네트워크 환경에서 OVS 스위치 2.5.5 버전이 스위치로 사용되었으며 OpenFlow 1.3 버전이 사용되었다.

본 시뮬레이션에서는 각 호스트에서 iperf를 사용하여 생성한 두 가지의 플로우 기반 트래픽을 사용했으며 두 트래픽 모두 총 1,500 개의 플로우로 구성된다. 트래픽은 TCP 및 UDP 플로우로 구성되며, 생성된 두 가지 트래픽은 각 트래픽 내에서 TCP 플로우가 차지하는 비율에 따라 T40과 T60으로 명명하였다. 예를 들어, T40의 경우 40%의 TCP 플로우로 구성된다. 실험 트래픽에서 발생시킨 전체 플로우들은 지수분포를 따르는 일정 간격들을 가지고 하나씩 생성되도록 설정되었다. 트래픽 내 플로우들은 지속시간이 길고 페이로드가 큰 elephant 플로우와 그 반대 성질을

(표 1) 시뮬레이션 환경
(Table 1) Simulation environment

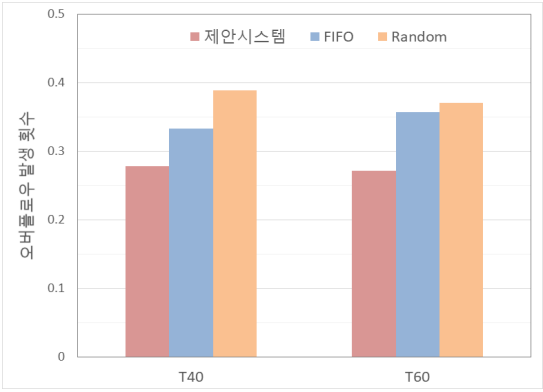
컨트롤러	시뮬레이터	오픈플로우 버전	스위치
Ryu	Mininet	1.3	OpenVSwitch 2.5.5

가진 mice 플로우로 분류될 수 있으며 그 비율은 5%와 95%로 설정하였다. Elephant 플로우의 사이즈 평균은 약 30 MB 이며 mice 플로우의 사이즈 평균은 약 300KB 로 설정하였다. 한편, 재설치를 계산하기 위하여 엔드포인트가 같은 플로우는 한 시점에 하나만 존재하도록 구현하였다.

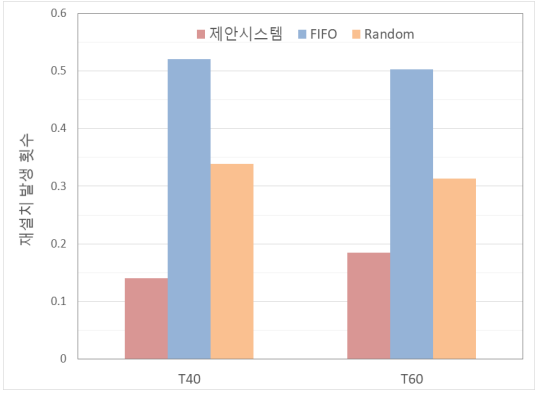
4.2 성능 평가

제안 시스템의 성능을 평가하기 위해 실험 트래픽에서 제안 아이디어와 비교 대상들의 시뮬레이션을 진행하였다. 총 두 가지 성능 지표를 측정 및 비교하였다. 첫 번째는 오버플로우 발생 횟수이며 두 번째는 플로우의 재설치 횟수이다. 플로우의 재설치란 이전에 설치되었던 플로우가 방출되었으나 다시 설치되는 경우를 의미한다. 이는 아직 전달해야할 패킷이 남아 있는 플로우의 엔트리가 플로우 테이블에서 방출되었을 때 발생할 수 있다. 이 지표를 통해 추후 전달할 패킷이 남아있음에도 방출되는 플로우 엔트리가 얼마나 많이 존재하는 지, 즉 플로우 방출 정책이 얼마나 적합한 선택을 했는지 알 수 있다. 제안 시스템과 성능을 비교하기 위한 정책들로는 Random과 FIFO 두 가지 알고리즘 기반의 방출 정책들을 사용했다. 한편, 제안 아이디어는 플로우 프로토콜에 따라 그 엔트리를 조기 방출하기 때문에 비교 정책 또한 조기 방출을 가능케 해 더욱 유의미한 비교를 할 수 있다. 따라서 비교 정책들에도 능동적 조기 방출을 적용, 플로우 테이블 사용량이 일정 임계값을 넘어설 때부터 플로우 엔트리의 조기 방출이 실시되도록 하였다. 이 정책들과의 비교를 통해 제안 아이디어가 적합한 방출 플로우를 선택하는 지, 플로우 테이블 사용량을 최적으로 유지하는 지 확인할 수 있다.

첫 번째로 오버플로우의 발생 횟수를 통해 성능을 비교했다. 각 트래픽 상에서 제안 아이디어 및 비교 대상들의 오버플로우 발생 횟수를 측정했다. 그 후 측정된 각 정책의 측정값을 해당 트래픽에서 실험한 세 가지 정책에서 발생한 오버플로우 횟수의 합으로 나누어 정규화하였다. 즉, 한 트래픽에서 각 방출 정책들의 측정값들을 모두 합하면 1 이 된다. 그림 3과 같이, 모든 트래픽에서 제안 시스템이 오버플로우를 가장 적게 발생 시킨 것을 확인할 수 있다.



(그림 3) 각 정책에서 오버플로우 발생 횟수
(Figure 3) Overflow occurrence comparison



(그림 4) 각 정책에서 플로우 재설치 발생 횟수
(Figure 4) Flow entries reinstallation comparison

제안 시스템이 가장 적은 오버플로우를 발생시켰으며 FIFO, Random 정책이 각각 그 뒤를 따르는 경향을 모든 트래픽에서 확인할 수 있다. T60 트래픽에서는 FIFO와 Random 간의 격차가 다소 줄어들었다. 한편, Random 정책은 무작위로 방출 엔트리를 결정했기 때문에 모든 트래픽에서 가장 많은 오버플로우를 기록한 것으로 볼 수 있다. FIFO 정책 또한 먼저 설치된 엔트리를 방출시키므로 제안 시스템보다 많은 오버플로우를 발생시킨 것으로 보인다.

다음으로는 정책들 간의 재설치 발생횟수를 비교하였다. 재설치 횟수 또한 오버플로우 발생 횟수와 같이 하나의 트래픽에서 발생한 각 정책들의 측정값을 정규화하여 그림 4에 나타냈다. 재설치의 경우, 제안 아이디어에서의 발생 횟수가 다른 정책에 비해 크게 적으며 전체적인 추세는 두 트래픽 모두 동일함을 확인할 수 있다. 이는 플로우 샘플링과 플로우 테이블 내 TCP 플래그 매칭을 이용해 비활성화

된 플로우를 조기에 방출하는 제안 시스템의 조기 방출 선택이 적합했음을 의미한다. 재설치의 경우 제안 시스템, Random, FIFO 순으로 적은 재설치 횟수를 보여주었다. FIFO의 경우, 지속적인 연결을 필요로 하는 플로우의 엔트리를 지속적으로 방출해 잦은 재설치를 발생시킨 것으로 보인다. 결론적으로, 오버플로우와 재설치 횟수 모두 고려하였을 때 제안 아이디어가 비교 정책 두 가지에 비하여 더욱 효율적인 플로우 엔트리 방출 정책임을 확인할 수 있다.

5. 결 론

본 논문에서 우리는 플로우의 프로토콜에 따라 다른 플로우 엔트리 방출 정책을 적용한 플로우 테이블 관리 시스템을 제시하였으며, 그 중 특히 UDP 플로우를 조기에 방출하는 방법에 집중하였다. 우리는 스위치에서 수집한 UDP 플로우의 정보에 기반해 주기적으로 비활성화된 UDP 플로우를 제거하고 해당 주기를 테이블 사용량에 기반해 조절하는 시스템을 제안하였다. 우리는 트래픽에 기반한 시뮬레이션을 통해 우리의 제안 시스템이 다른 정책들과 비교하여 더 적은 플로우 테이블 오버플로우 및 재설치 횟수를 보여주는 것을 확인할 수 있었다. 최종적으로, 향후 제안 시스템이 샘플링 주기를 더욱 정교하게 설정하고 네트워크 환경과 더욱 긴밀히 상호작용할 수 있도록 하여 제안 시스템의 범용성을 보완할 것이다.

참고문헌(Reference)

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks”, SIGCOMM Computer Communication Review (CCR), 2008. https://doi.org/10.1145/1355734.1355746

[2] Open Networking Foundation, “OpenFlow Switch Specification Version 1.3.0”, 2012. https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[3] “TCAMs and Openflow: What every practitioner must know”, 2012. https://www.sdxcentral.com/articles/contributed/sdn-openflow-tcam-need-to-know/2012/07/

[4] Leng, Junyuan, et al., ″An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network″, arXiv preprint arXiv:1504.03095, 2015. https://arxiv.org/abs/1504.03095

[5] J. Su, R. Xu, S. Yu, B. Wang and J. Wang, ″Redundant rule Detection for Software-Defined Networking,″ KSII Transactions on Internet and Information Systems, vol. 14, no. 6, pp. 2735-2751, 2020. https://doi.org/10.3837/tiis.2020.06.022

[6] A. Zarek, Y. Ganjali, and D. Lie, “Openflow timeouts demystified”, Univ. of Toronto, 2012. https://security.csl.toronto.edu/papers/zarek_mscthesis.pdf

[7] Z. Shah, ″Mitigating TCP Incast Issue in Cloud Data Centres using Software-Defined Networking (SDN): A Survey,″ KSII Transactions on Internet and Information Systems, vol. 12, no. 11, pp. 5179-5202, 2018. https://doi.org/10.3837/tiis.2018.11.001

[8] Bishop and Mike, ″Hypertext transfer protocol version 3 (HTTP/3)″, Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-20, 2019. https://tools.ietf.org/html/draft-ietf-quic-http-24

[9] P. Kasabai, K. Djemame and S. Puangpronpitag, ″Priority-based Scheduling Policy for OpenFlow Control Plane,″ KSII Transactions on Internet and Information Systems, vol. 13, no. 2, pp. 733-750, 2019. https://doi.org/10.3837/tiis.2019.02.014

[10] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, “Effective switch memory management in openflow networks,” in Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, ser. DEBS ’14. New York, NY, USA: ACM, pp. 177–188, 2014. http://doi.acm.org/10.1145/2611286.2611301

[11] S. Shirali-Shahreza and Y. Ganjali, ″Delayed Installation and Expedited Eviction: An Alternative Approach to Reduce Flow Table Occupancy in SDN Switches,″ IEEE/ACM Transactions on Networking, vol. 26, no. 4, pp. 1547-1561, 2018. https://doi.org/10.1109/TNET.2018.2841397

[12] Mininet – an instant virtual network on your laptop (or other PC), February 2017. http://www.mininet.org/

[13] Ryu SDN Framework. https://github.com/osrg/ryu

◐ 저 자 소 개 ◑

최 한 힘 나 라(Hanhimnara Choi)
2019년 성균관대학교 수학과(이학사)
2021년 성균관대학교 일반대학원 인공지능학과(공학석사)
관심분야 : 소프트웨어 정의 네트워킹, 네트워크 기능 가상화 등
E-mail : hanhimy@skku.edu

샤이드 무하마드 라자 (Syed Muhammad Raza)
2006년 파키스탄 PIEAS 대학교 컴퓨터공학부(공학사)
2010년 스웨덴 룬드 대학교 무선통신공학부(공학석사)
2018년 성균관대학교 일반대학원 전자전기컴퓨터공학과(공학박사)
2020년~현재 성균관대학 전자전기컴퓨터공학과 연구교수
관심분야 : 소프트웨어 정의 네트워킹, 네트워크 기능 가상화 등
E-mail : s.moh.raza@skku.edu

김 문 성(Moonseong Kim)
2002년 성균관대학교 일반대학원 수학과(이학석사)
2007년 성균관대학교 일반대학원 전기전자및컴퓨터공학부(공학박사)
2007년~2009년 미국 미시간주립대학교 컴퓨터과학공학과 연구원
2009년~2018년 특허청 사무관
2018년~현재 서울신학대학교 IT융합소프트웨어학과 조교수
관심분야 : 유무선 네트워크, 모바일 컴퓨팅, 머신러닝, 지식재산권 등
E-mail : moonseong@stu.ac.kr

추 현 승(Hyunseung Choo)
1990년 댈러스 텍사스 대학 컴퓨터공학과(공학석사)
1996년 알링턴 텍사스 대학 컴퓨터공학과(공학박사)
1997년~1998년 특허청 사무관
1998년~현재 성균관대학교 소프트웨어대학 교수
관심분야 : 모바일 센서 네트워크, 소프트웨어 정의 네트워킹, 지능형 모바일 컴퓨팅,
멀티 액세스 엣지 컴퓨팅, 머신러닝 및 인공지능 등
E-mail : choo@skku.edu