Journal of KIIT. Vol. 17, No. 9, pp. 65-74, Sep. 30, 2019. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2019.17.9.65

SDN에서 효율적인 플로우 테이블 관리 방법

김동렬*, 조명원**, 이병대***

An Efficient Flow Table Management Scheme in SDN

Dongryeol Kim*, Myoung-Won Jo**, and Byoung-Dai Lee***

본 논문은 2019학년도 경기대학교 대학원 연구원장학생 장학금 지원에 의하여 수행되었음

요 약

SDN 스위치는 서로 규격이 다를 경우 플로우 테이블 오버 플로우 증가, 컨트롤러 오버헤드 증가, 사용자 QoS 저하 등의 문제가 발생한다. 본 논문에서는 이를 해결하기 위해 플로우의 특성을 고려한 플로우 테이블 교체 알고리즘을 제안한다. 제안하는 SFF(Short Flow First) 교체 알고리즘은 각각의 플로우 엔트리의 매칭 주기를 활용하여 플로우를 생존 기간이 짧은 숏(short) 플로우와 생존 기간이 긴 롱(long) 플로우로 구분하고 플로우 엔트리 삭제 시 숏 플로우를 먼저 삭제함으로써 플로우 엔트리 매칭 횟수를 증가시키고 제어기의 부하를 감소시킨다. 제안하는 알고리즘은 실험을 통해 기존의 대표적인 교체 알고리즘인 LRU 알고리즘에 비해 제어기 부하를 약 16% 줄임을 알 수 있었다

Abstract

If SDN switches have different specifications, problems such as flow table overflow, controller overhead, and user QoS degradation happen. In order to solve these problems, we propose a new flow table replacement algorithm that considers the characteristics of the flow in this paper. The proposed SFF (Short Flow First) replacement algorithm uses the matching cycle of each flow entry to classify the flow into short-lived short flows and long-lived long flows. When the flow table is full, the short flow is deleted first to increase the number of flow entry matching and to reduce the overhead of the controller. Experimental results show that the proposed algorithm reduces the controller overhead by about 16% compared to the conventional LRU replacement algorithm.

Keywords

flow entry replacement, flow table, flow management, SDN

* 현대 엠엔소프트
- ORCID: http://orcid.org/0000-0003-2616-2561
** 경기대학교 컴퓨터과학과
- ORCID: http://orcid.org/0000-0001-7593-3073
*** 경기대학교 컴퓨터공학부 교수(교신저자)
- ORCID: http://orcid.org/0000-0002-4028-6168

· Received: Aug. 12, 2019, Revised: Sep. 20, 2019, Accepted: Sep. 23, 2019

· Corresponding Author: Byung-Dai Lee
Division of Computer Science and Engineering, Kyonggi University, 154-42, Gwanggyosan-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do, Korea
Tel.: +82-31-249-9676, Email: blee@kyonggi.ac.kr

Ⅰ. 서 론

최근 IT 기기들의 성능이 높아지면서 멀티미디어 데이터가 폭발적으로 증가하였고 트래픽 패턴의 변화, 데이터 센터의 증가 등 네트워크 구조가 크게 변화하였다. 그로인해 네트워크 구조의 개방성이 요구되었고 변화한 구조에 대해 유연하게 대처할 수 있는 기술이 요구되었다. 또한 네트워크 구축, 관리, 운용에서 많은 비용이 발생하는 문제점과 복잡성을 해결하기 위한 방안이 필요해졌고 그에 따라 SDN (Software Defined Networking) 기술이 등장하였다[1].

SDN에서 패킷은 플로우 단위로 관리되며 스위치에서는 플로우 관련 정보를 엔트리로 만들어 플로우 테이블에 저장하고 관리한다. 스위치에 패킷이 들어오면 스위치는 플로우 테이블을 탐색하여 들어온 패킷과 매칭되는 플로우 엔트리를 찾고 그에 해당하는 동작을 취하거나 제어기와의 통신을 통해 패킷에 대한 플로우 엔트리를 새로 생성한다. 이 플로우 테이블은 스위치의 한정된 메모리 용량 범위 안에서 관리된다. 그러나 네트워크의 규모가 커질 경우 같은 메모리 용량을 가지는 스위치로 구성되기 어렵다. 그리고 다양한 회사의 스위치가 사용될 때, 패킷 전달 과정에서 상대적으로 낮은 성능을 가지는 스위치로 인해 플로우 테이블 교체 등의 처리 과정이 추가적으로 발생할 수 있다. 이 과정은 패킷 전송 지연 및 제어기 부하 증가 등의 문제를 야기할 수 있고 이것은 전체 네트워크 성능 저하의 원인이 된다[2]-[6]. 그러므로 플로우 엔트리 매칭율을 증가시켜 플로우 테이블의 교체 횟수와 제어기의 부하를 줄이고 네트워크 성능을 향상시키는 것은 SDN 운영 측면에 있어 매우 중요하다.

따라서 본 논문에서는 SDN을 구성하는 스위치에서 발생할 수 있는 플로우 테이블 오버 플로우, 제어기 부하 증가, 사용자 QoS 저하 등을 줄이기 위해 플로우의 특성을 고려한 플로우 엔트리 교체 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 제어기의 부하를 감소시키기 위한 기존의 연구에 대해서 알아본다. 3장에서는 SDN 환경을 구성하는 스위치 간 성능 차이로 인해 발생하는 문제점에 대해 설명한다. 그리고 플로우 특성을 고려한 새로운 플로우 엔트리 교체 알고리즘을 제안한다. 4장에서는 성능 평가를 위한 실험 환경, 트래픽 패턴 등을 설명하고 LRU 알고리즘과의 비교를 통해 제안하는 알고리즘의 성능을 평가한다. 마지막으로 5장에서는 결론을 맺고 향후 연구 과제에 대해 기술한다.

Ⅱ. 관련 연구

SDN에서는 패킷 처리 과정이나 제어기의 스위치 관리 과정 등 여러 과정에서 부하가 커질 수 있는데 이 때 발생하는 부하를 줄이기 위한 연구가 다방면에서 진행 중이다. [7]은 플로우 기반 모니터링 기법을 활용한 프레임 워크이다. 와일드 카드 기법을 사용하여 스위치에서 제어기에 PacketIn 메시지를 보내지 않고 새로 도착한 패킷을 처리한다. 스위치에 도착한 패킷과 플로우가 일치할 때 플로우의 카운트 수가 증가하고 그 카운트 수가 설정해 두었던 임계값에 도달하면 Elephant 플로우로 구분되어 제어기에 보내서 플로우에 대한 처리를 요청한다. 이 처리 과정으로 인해 스위치에서 제어기로 보내는 PacketIn 메시지 전송 횟수와 부하를 감소시켰다. 그러나 와일드 카드를 사용하는 방법은 스위치의 한정된 메모리를 무분별하게 사용하게 될 수 있다는 단점이 있다.

[8]에서는 제어기의 부하를 줄이기 위해 부하가 큰 제어기가 관리하는 스위치를 다른 제어기로 이전시켜 부하의 균형을 이루도록 하였다. 그러나 스위치 단위로 제어기로 이전시키기 때문에 정확하고 세밀하게 부하를 분배하는 것이 불가능하고 스위치를 담당하는 제어기를 변경시키기 위해 추가적인 설정 과정이 요구되어 부하가 발생할 수 있다.

[9]에서는 Kandoo 프레임워크를 제안했다. 이것은 잦은 네트워크 이벤트로 인해 발생하는 제어기의 부하를 줄이기 위한 방법으로 제어면(Control plane)을 루트 제어기와 다수의 로컬 제어기 두 계층으로 나눠서 전체 네트워크 상태와 관련된 요청은 루트 제어기가 처리하고 나머지 이벤트는 로컬 제어기가 처리하여 제어기의 부하를 분산시켰다.

[10]에서는 NOX 제어기[11]에 어플리케이션을 적

용하여 논리적으로 중앙집중형 제어기이지만 실질적으로는 다수 분산 제어기로 구성된다. 분산 플랫폼을 지향하여 제어기를 분산적으로 배치하고 각 제어기가 지역적으로 네트워크를 관리하는 제어기 어플리케이션을 제안했다. 이러한 방법들은 제어기를 분산 배치하여 부하분산이 이뤄졌지만 여러 제어기 간 핸드오버 문제나 제어기 간 인터페이스인 east-west bound API의 표준이 정의되어 있지 않기 때문에 실용할 수 없는 상태이다.

[12]는 tag-in-tag 기법을 제안했다. 이것은 다수의 플로우를 특정 패스를 통해 라우팅시켜 플로우 테이블 크기를 줄이는 방법이다. 비슷한 방법으로 [13]에서는 레이블 스위칭을 통해 복잡한 코어 네트워크를 여유있게 만들고 TCAM에서 플로우 엔트리의 수를 줄였다.

III. 본 론

그림 1은 플로우 테이블 오버 플로우가 발생했을 때, 기존에 있던 불필요한 플로우 엔트리를 삭제하고 새로운 플로우 엔트리를 생성하는 과정을 나타낸다. 먼저, 스위치의 플로우 테이블이 가득 찬 상태에서 새로운 패킷이 들어오면 해당 패킷에 매칭되는 플로우 엔트리를 테이블에서 검색한다. 테이블에 플로우 엔트리가 존재하면 패킷은 그 정보대로 처리되지만 플로우 엔트리가 없으면 플로우 엔트리 생성을 위해 PacketIn 메시지를 제어기에 전송한다. 제어기는 경로를 계산하고 FlowModify 메시지를 스위치에 전송하여 테이블에 플로우 엔트리를 추가하도록 하고 PacketOut 메시지를 이용하여 해당 패킷의 수신지를 지정한다. 그러나 테이블이 가득 찬 상태이므로 플로우 엔트리를 추가할 수 없기 때문에 불필요한 플로우 엔트리를 삭제해야 한다. 그러므로 플로우 엔트리 삭제를 위해 스위치가 제어기에 FlowRemoved 메시지를 전송하여 제어기로부터 삭제할 플로우를 결정 받는다. 제어기는 네트워크 관리 정책에 따라 삭제할 플로우를 결정한 뒤 다시 스위치로 FlowModify 메시지를 전송하여 스위치의 플로우 테이블을 업데이트 한다.

이 과정에서 제어기와 스위치는 총 다섯 번의 메시지 교환을 통해 플로우 테이블을 업데이트하고 삭제할 플로우 엔트리를 결정하기 위해 제어기에서 플로우 테이블 교체 알고리즘을 실행하게 된다. 이 과정에서 SDN에 많은 부하가 발생하게 된다. 또한 더 큰 규모의 네트워크에서는 이런 과정이 더 많이 발생할 수 있고 이것은 제어기의 부하와 전체 네트워크 성능에 영향을 줄 수 있다.

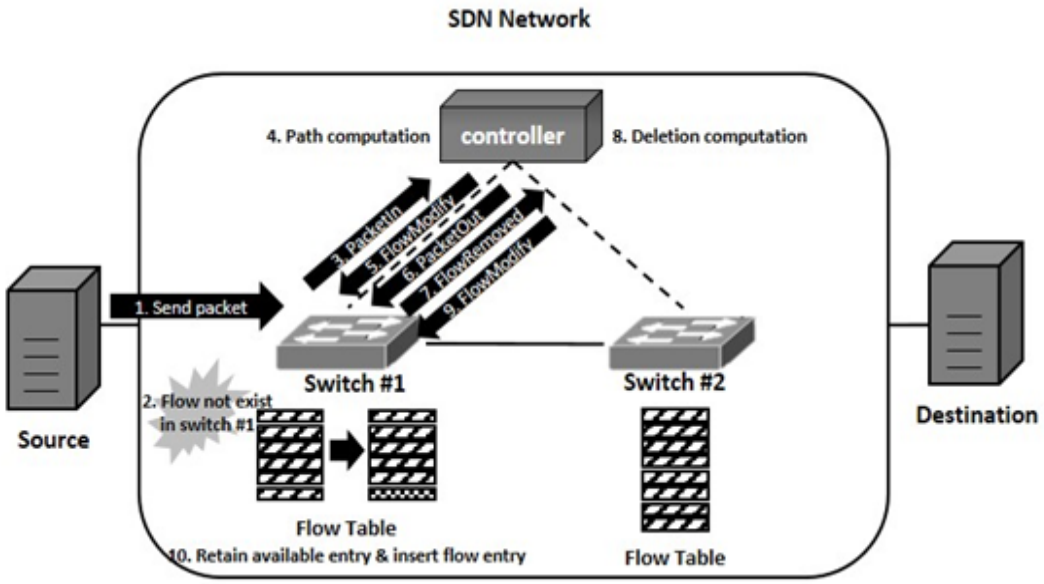

그림 1. 플로우 엔트리 교체 과정
Fig. 1. Flow entry replacement procedure

플로우 엔트리 교체 과정을 겪을 경우 SDN에서는 제어기 부하 및 패킷 전송 지연이 발생하게 된다. 따라서 효율적인 네트워크 운영을 위해서는 플로우 엔트리 교체 시 앞으로 사용되지 않을 엔트리를 가정 먼저 삭제하는 것이 효율적이다. 이를 위해 본 논문에서는 플로우의 존재시간을 예측하여 짧은 시간 동안 존재하는 플로우를 긴 시간동안 존재하는 플로우보다 우선적으로 삭제하는 SFF(Short Flow First) 교체 알고리즘을 제안한다. 제안하는 알고리즘은 웹 트래픽과 같이 짧은 시간 존재하는 플로우를 멀티미디어 트래픽과 같이 주기성을 가지면서 긴 시간 존재하는 플로우보다 먼저 삭제함으로써 전체적인 플로우 엔트리 교체 횟수를 줄인다.

기존 연구들은 플로우 엔트리의 매칭 시간 또는 매칭 횟수를 고려하여 플로우 엔트리를 교체한다. 이와 다르게 본 논문에서 제안하는 SFF 알고리즘은 플로우 엔트리가 매칭되는 주기를 활용하여 플로우의 특성을 크게 숏 플로우(Short flow)와 롱 플로우(Long flow)로 구분하여 교체될 플로우 엔트리를 결정한다. 숏 플로우는 일시적으로 발생한 패킷으로 인해 생긴 플로우 엔트리로 짧은 시간 동안만 매칭되고 일정 시간이 흐른 뒤에는 매칭되지 않는 플로우를 의미하고 웹 서비스 트래픽 등이 여기에 속한다. 이러한 숏 플로우는 일시적으로 사용된 후 시간이 지나면 사용되지 않기 때문에 플로우 테이블에서 빠르게 삭제되어도 네트워크 운영에 있어 큰 영향을 미치지 않는다.

반면, 롱 플로우는 반복적으로 발생하는 패킷으로 주기를 갖고 반복적으로 매칭되는 플로우로 멀티미디어 서비스 트래픽을 예로 들 수 있다. 롱 플로우의 경우 서비스가 종료될 때까지 플로우 테이블에서 유지되어야 플로우 엔트리를 삭제하거나 다시 생성하는 등의 부하가 발생하지 않는다. SFF 알고리즘에서는 이러한 플로우 엔트리의 특성에 따라 테이블을 관리한다. 플로우 테이블 오버플로우가 발생하여 플로우 엔트리 교체가 필요할 경우, 우선적으로 숏 플로우 특성을 가지는 엔트리를 교체 대상으로 설정하여 엔트리 교체를 수행한다. 그리고 롱 플로우 특성을 가지는 엔트리는 교체 대상에서 제외하여 플로우 테이블에서 유지될 수 있는 기회를 가지게 된다.

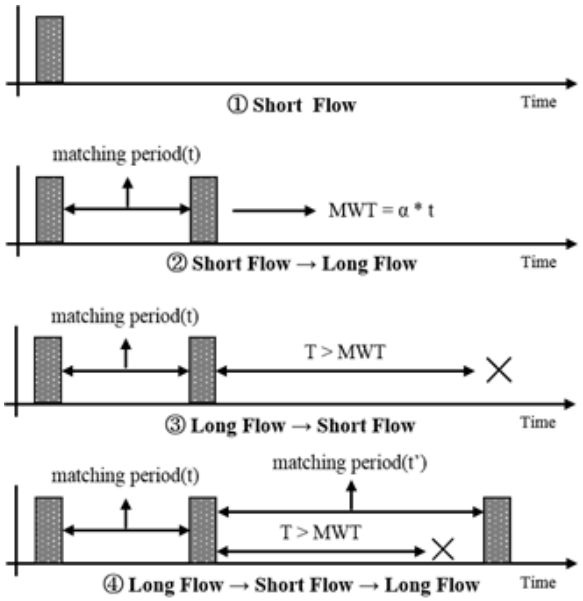

그림 2. SFF 알고리즘의 플로우 특성 구분 과정
Fig. 2. Classification of flow in SFF algorithm

SFF 알고리즘에서는 플로우의 특성을 파악하기 위해 플로우 엔트리의 매칭 주기를 활용한다. 그림 2는 생성된 플로우 매칭 주기를 기반으로 플로우를 숏 플로우나 롱 플로우로 구분하는 과정을 나타낸다. 그림에서 매칭 주기는 동일 플로우에 포함된 연속된 두 패킷 사이의 도착 시간의 차이를 나타낸다.

먼저, 최초 패킷이 도착하여 새로운 플로우 엔트리가 생성되면 해당 플로우의 특성은 숏 플로우로 할당되어 유지된다. 숏 플로우로 할당된 엔트리가 플로우 테이블에서 삭제되기 전에 두 번째 패킷이 도착하여 엔트리에 매칭되면 이 엔트리는 롱 플로우로 변경된다. 그리고 첫 번째 패킷과 두 번째 패킷 사이의 시간이 해당 플로우의 매칭 주기(t)가 되고 해당 플로우의 MWT(Maximum Waiting Time)은 $\alpha \cdot t$로 설정된다. 그리고 롱 플로우로 변경된 엔트리의 패킷 간 도착 시간이 MWT보다 작으면 계속 롱 플로우로 지속된다.

그러나 롱 플로우의 특성을 갖는 엔트리에 해당하는 패킷이 MWT가 지나도 도착하지 않으면, 이 패킷은 더 이상 전송되지 않거나 다른 패킷들에 의해 전송이 늦어진 것일 가능성이 있다. 그러므로 해당 엔트리는 플로우 테이블에서 좀 더 유지될 필요가 있다. 따라서 해당 엔트리를 삭제하지 않고 숏 플로우로 변경한다. 이 엔트리는 롱 플로우의 특성

을 가졌던 엔트리이므로 다시 패킷이 전송될 가능성이 있기 때문에 플로우 테이블에서 유지될 수 있는 기회를 부여하는 것이다. 만약 숏 플로우로 변경된 엔트리가 플로우 테이블에서 삭제되기 전에 다시 매칭되면 패킷 간 도착 시간으로 새로운 매칭 주기가 생성되고 다시 롱 플로우로 전환된다. 이와 같은 방법으로 플로우 엔트리들의 특성이 파악되는데 모든 플로우 엔트리의 매칭 주기는 똑같지 않다. 각각의 플로우 엔트리들은 서로 다른 매칭 주기를 갖게 되고 MWT 또한 서로 다르다. 그리고 나서 플로우 테이블 오버 플로우 발생 시, 플로우 테이블에 있는 엔트리 중 숏 플로우로 파악된 엔트리만을 대상으로 플로우 테이블 교체 알고리즘을 적용한다. 숏 플로우가 존재하지 않을 경우, 롱 플로우를 대상으로 FIFO, LFU, LRU 등의 교체 알고리즘을 적용하여 플로우 테이블을 갱신한다.

그림 3은 SFF 알고리즘의 유사 코드이다. 유사 코드는 먼저, 패킷을 처리하기 전에 주기적으로 롱 플로우 엔트리의 매칭 주기를 확인한다. 롱 플로우로 할당되어 있는 플로우 엔트리 중 MWT동안 매칭되지 않은 엔트리는 다시 숏 플로우로 변경하고 해당 엔트리의 매칭 주기를 삭제한다(line 1-6).

```
 1: while period do
 2: foreach flow in long flowtable do
 3:    if T > MWT then
 4:       changeFlag(short, flow)
 5:    end
 6: end
 7:
 8: if packet is arrived then
 9:    flow = searchFlow(packet)
10:    if flow is existed then
11:       changeFlag(long, flow)
12:    else
13:       flow = generateFlow(packet)
14:       chageFlag(short, flow)
15:       if table is full then
16:          if short flow is existed then
17:             deleteFlow(short)
18:          else
19:             deleteFlow(long)
20:       insertFlow(flow)
21:    action(flow)
```

그림 3. SFF 알고리즘 유사 코드
Fig. 3. Pseudo code of SFF algorithm

다음으로. 스위치에 패킷이 도착하면 해당 패킷과 일치하는 플로우 엔트리를 탐색한다(line 8-9). 스위치에 들어온 패킷에 해당하는 플로우 엔트리가 존재하면 패킷 간의 도착 시간을 활용하여 매칭 주기를 생성하고 해당 플로우 엔트리를 롱 플로우로 변경한다(line 10-11). 만약, 플로우가 존재하지 않으면 패킷의 정보를 기반으로 플로우 엔트리를 생성한 뒤 숏 플로우 플래그를 할당한다(line 13-14). 그리고 생성한 플로우 엔트리를 플로우 테이블에 삽입하기 위해 테이블의 상태를 체크한다. 플로우 테이블이 가득 찬 상태일 경우, 새로운 엔트리를 삽입하면 오버 플로우가 발생하기 때문에 기존에 있던 엔트리를 삭제할 필요가 있다. 그 이유로 플로우 엔트리를 삭제하기 위해 숏 플로우가 존재하는지 체크한다(line 16). 테이블에 숏 플로우가 존재하면 숏 플로우 엔트리를 대상으로 교체 알고리즘을 적용하여 삭제할 엔트리를 숏 플로우 내에서 결정하고 삭제한다(line 17). 숏 플로우가 존재하지 않을 경우, 롱 플로우 엔트리를 대상으로 교체 알고리즘을 적용하여 삭제할 엔트리를 결정하고 삭제한다(line 19). 그리고 빈 공간에 새로운 플로우 엔트리를 삽입한다(line 20). 마지막으로 하고 입력된 패킷에 의해 수정된 플로우 엔트리에 해당하는 동작(패킷 전송, 수정, 폐기 등)을 수행한다(line 21).

그림 4는 SFF 알고리즘을 적용했을 때 플로우 테이블이 업데이트되는 과정을 예로 나타낸 것이다. 그림에서 플래그는 플로우 엔트리의 특성을 나타낸다. S는 숏 플로우 엔트리를 나타내고 L은 롱 플로우 엔트리를 나타낸다. 플로우 테이블의 크기는 5로 설정하여 5개의 플로우 엔트리를 저장할 수 있고 t_0~t_5는 시간을 나타낸다. 그림에서 보면, 초기에 1, 2번 플로우가 도착 후 플래그가 S(숏 플로우)로 할당된다. t_2까지 3, 4, 5번 플로우가 순서대로 도착하면 이 플로우들도 플래그가 S로 할당된다. t_3에 1, 2번 플로우가 도착하면 해당 플로우는 주기를 갖고 반복적으로 매칭되는 플로우로 판단하여 플래그가 L(롱 플로우)로 변경된다. 해당 플로우의 주기는 t_3-t_0이 된다. 플로우 테이블이 가득 찬 상태에서, t_4에 플로우 6, 7, 8번이 도착하면 플래그가 L로 할당된 플로우를 제외하고 S로 할당된 숏 플로우 중에

서 삭제될 플로우가 결정된다. 그 결과 숏 플로우 중 3, 4, 5번 플로우 엔트리가 삭제되고 6, 7, 8번 플로우 엔트리가 삽입된다. t_5에서도 마찬가지로 S가 할당된 플로우 엔트리 중 6, 7번 플로우를 삭제한 후 9, 10번 플로우가 삽입된다.

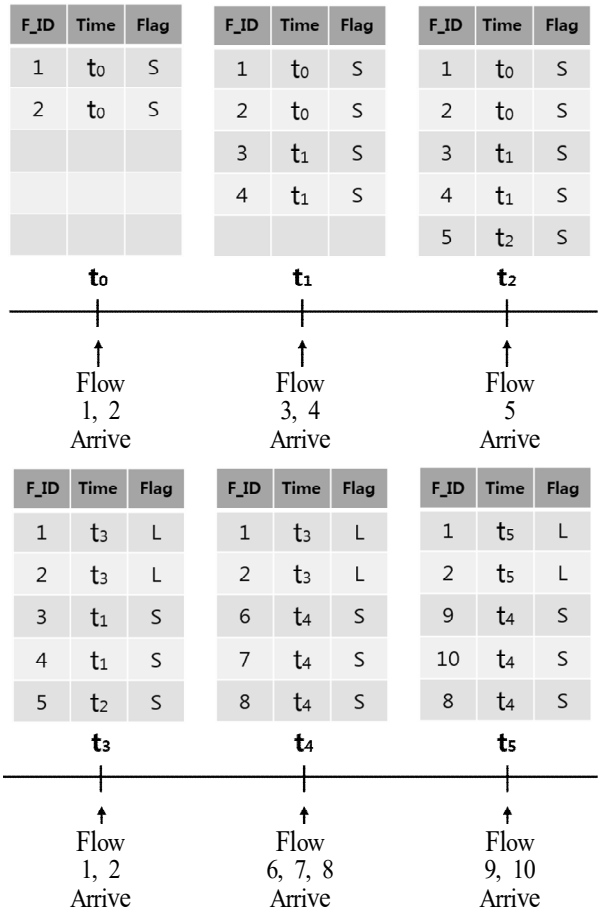

그림 4. SFF 알고리즘의 테이블 업데이트 과정 예
Fig. 4. Example of table update process of SFF algorithm

IV. 실 험

본 논문에서는 SFF 알고리즘의 성능 평가를 위해 Mininet Emulator[14]을 이용하여 실험을 수행하였다. Mininet Emulator는 linux에서 SDN기반 스위치, 링크, 제어기를 연결하여 네트워크를 구성할 수 있는 보편적인 도구로 SDN 분야에서 가장 널리 사용되고 있다. 실험에서는 그림 5와 같이 4개의 출발지와 40개의 목적지를 가진 SDN 환경을 구축하였다. 각 스위치의 플로우 테이블 크기는 18, 20, 22이며 출발지는 비디오 서버, 웹 서버의 역할, 목적지는 클라이언트 역할을 담당한다. SDN 제어기는 POX 제어기[15]를 사용하였고 제어기에 SFF 알고리즘을 적용하였다. 또한 알고리즘의 성능 평가를 기술하기 위해 플로우 테이블 교체 시 LRU 알고리즘을 적용하였고, 기존 알고리즘 중 LRU 알고리즘과의 성능 평가를 추가적으로 수행하고 성능 비교를 진행하였다. 그리고 제안한 기법의 성능 평가를 위해 Ostinato packet generator[16]를 사용하여 2000개의 패킷으로 구성된 데이터를 생성하였다. Ostinato는 Mininet 환경에서 패킷을 생성하는데 가장 널리 사용되는 도구이다.

스위치에 들어온 패킷과 플로우 테이블에 있는 플로우 엔트리가 일치하는지 비교하기 위한 매칭 필드로 패킷의 출발지 IP, MAC 주소, 목적지 IP, MAC주소를 사용하였고 플로우 엔트리의 매칭 주기를 체크하기 위한 주기 정보와 플로우 특성을 체크하기 위한 플래그 정보를 추가하였다. 각각의 데이터 집합은 tcpreplay 패킷 재생 도구[17]을 사용하여 전송하였다.

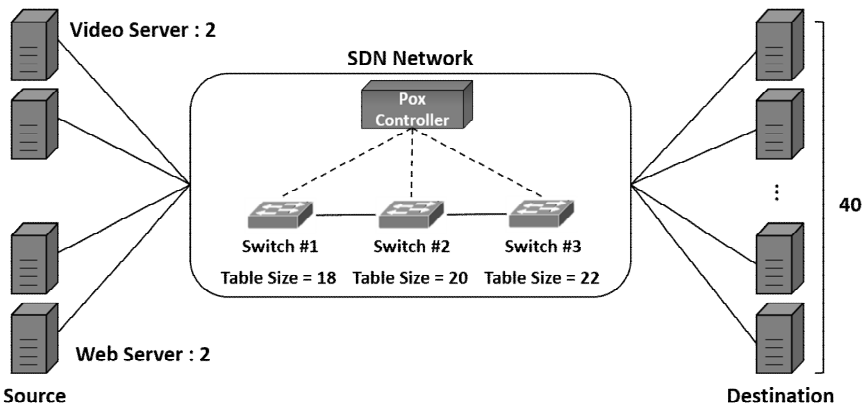

그림 5. 실험 환경 구성
Fig. 5. Experimental environment configuration

주기적으로 사용되지 않는 롱 플로우를 숏 플로우로 변경하기 위한 플로우 테이블 업데이트 주기 및 MWT를 결정하기 위한 파라메터 α에 대해서 다양한 값을 이용하여 실험을 수행하였으며, 본 논문에서는 그 중 수행된 실험 환경에서 가장 우수한 성능을 보인 업데이트 주기 60초, α 1.0인 경우에 대한 실험 결과를 보여 준다.

그림 6은 앞서 설명한 트래픽 패턴에 맞게 제작한 데이터 집합을 이용하여 LRU 알고리즘과 SFF 알고리즘의 플로우 엔트리 매칭 횟수를 비교한 결과이다. x축은 트래픽 양의 차이를 나타낸다.

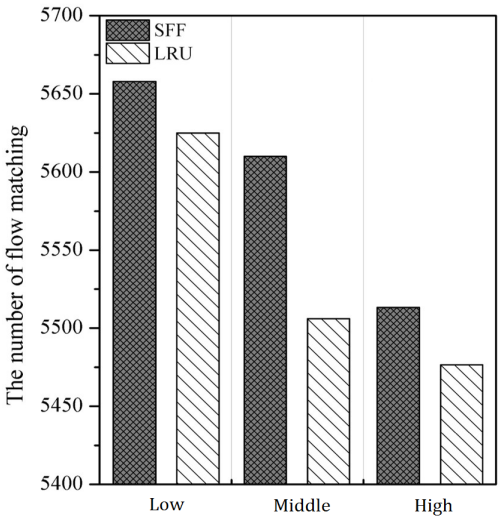

그림 6. 플로우 매칭 횟수
Fig. 6. Number of flow matching

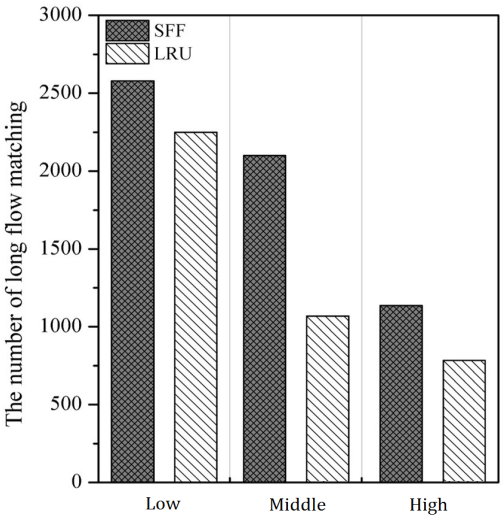

그림 7. 롱 플로우 매칭 횟수
Fig. 7. Number of long flow matching

그림 6에서 알 수 있듯이 LRU 알고리즘은 숏 플로우 또는 롱 플로우 등 플로우가 가지는 특성을 고려하지 않고 단순히 오랫동안 매칭되지 않은 엔트리를 삭제한다. 그러므로 LRU 알고리즘을 적용할 경우 반복적으로 매칭될 가능성이 있는 롱 플로우 엔트리라도 오랫동안 매칭되지 않으면 삭제될 수 있다. 그러나 SFF 알고리즘은 플로우의 특성을 고려하여 숏 플로우 엔트리로 판단된 엔트리 중에서 엔트리를 삭제한다. 그리고 반복적으로 매칭될 가능성이 있는 롱 플로우 엔트리를 삭제 대상에서 제외시켜 플로우 테이블에서 오랫동안 유지될 수 있게 한다. 그렇기 때문에 테이블에 오랫동안 유지시키지 않아도 되는 숏 플로우 엔트리는 빠르게 교체되고 롱 플로우 엔트리는 테이블에서 오랫동안 유지되어 매칭 횟수가 증가하게 된다.

그림 7은 LRU 알고리즘과 SFF 알고리즘의 롱 플로우 엔트리 매칭 횟수를 비교한 결과이다. 롱 플로우의 특성을 가지는 플로우 엔트리가 매칭된 횟수를 측정한 것이다. LRU 알고리즘은 플로우 엔트리의 특성에 상관없이 플로우 엔트리가 오랫동안 매칭되지 않으면 삭제한다. 그렇기 때문에 플로우 테이블이 가득 찬 시점에서 새로운 플로우 엔트리를 필요로 하는 패킷이 들어왔을 때, LRU 알고리즘의 경우 특정 주기를 갖고 반복적으로 매칭되는 롱 플로우도 오랫동안 사용되지 않은 것으로 판단되면 테이블에서 삭제될 수 있다. 반면 SFF 알고리즘은 플로우 엔트리 삭제 시 숏 플로우의 특성을 갖는 엔트리 중에서만 삭제할 엔트리가 결정된다. 또 롱 플로우의 특성을 갖는 엔트리가 일정 시간 동안 매칭되지 않아도 바로 삭제하는 것이 아니라 다시 숏 플로우로 할당되어 테이블에서 상주할 수 있는 기회를 부여 받는다. 그러므로 롱 플로우의 특성을 갖는 엔트리는 숏 플로우 엔트리 보다 플로우 테이블에 오래 유지될 수 있고 시간이 지나도 반복적으로 매칭될 가능성이 있다. 따라서 SFF 알고리즘의 롱 플로우 엔트리의 매칭율이 더 높게 나타난다.

그림 8은 LRU 알고리즘과 SFF 알고리즘을 적용했을 때, 스위치와 제어기 간 메시지 교환 횟수를 비교한 결과이다. 메시지 교환은 새로운 플로우 엔트리를 삽입하거나 플로우 테이블이 모두 채워져 엔트리를 삭제할 때 이뤄진다. 3장에서 설명한 바와

같이 플로우 테이블 오버 플로우가 발생하면 플로우 엔트리 삭제 및 생성을 위해 스위치와 제어기 간 5개의 메시지가 교환되고, 삭제 과정 없이 새로운 플로우 엔트리를 생성하여 테이블에 삽입하기 위해서는 3개의 메시지가 교환된다. 메시지 교환이 많이 발생한다는 것은 스위치에 들어온 패킷과 매칭되는 엔트리가 플로우 테이블에 없어 새로운 플로우 엔트리를 생성하거나 테이블에서 기존에 있던 엔트리를 삭제하고 새로운 엔트리를 생성하는 상황이 많이 발생한다는 것이다.

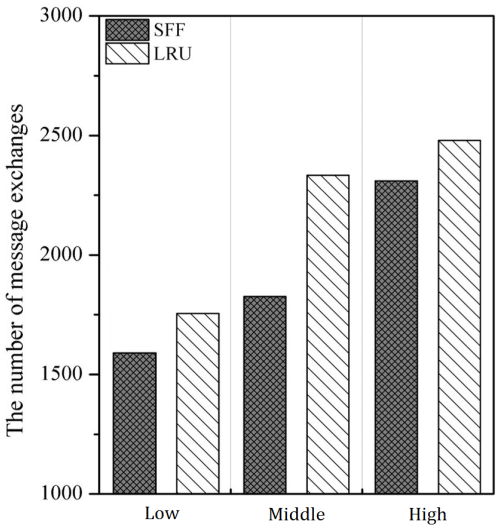

그림 8. 메시지 교환 횟수
Fig. 8. Number of message exchange

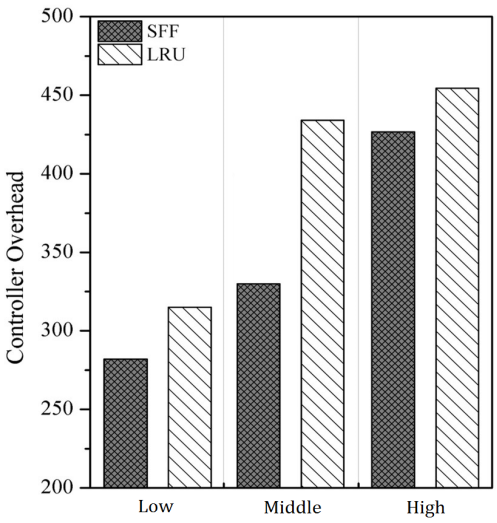

그림 9. 제어기 부하
Fig. 9. Controller overhead

플로우 엔트리 매칭 횟수를 비교한 그래프를 보면, SFF 알고리즘 보다 LRU 알고리즘의 매칭 횟수가 낮다. 이것은 LRU 알고리즘을 적용했을 때보다 SFF 알고리즘을 적용했을 때, 스위치에 들어온 패킷과 매칭되는 플로우 엔트리가 더 많았다는 것을 의미한다. 그러므로 LRU 알고리즘에서 더 많은 메시지 교환이 발생하였다.

그림 9는 LRU 알고리즘과 SFF 알고리즘을 적용했을 때, 제어기의 부하를 비교한 결과이다. 제어기 부하는 플로우 테이블에서 삭제할 엔트리를 결정하기 위해 제어기가 계산하는 횟수를 의미한다. 스위치의 플로우 테이블이 가득 찬 상황에서 새로운 플로우 엔트리를 필요로 하는 패킷이 들어오면 스위치는 제어기에 메시지를 전송하여 제어기에서 삭제할 엔트리를 계산한다. 숏 플로우의 특성을 가지는 엔트리는 일시적으로 사용되기 때문에 테이블에서 빠르게 삭제되는 것이 효율적인 반면 롱 플로우의 특성을 가지는 엔트리는 지속적으로 매칭되기 때문에 테이블에서 오래 유지되는 것이 효율적이다. 플로우 엔트리 삭제 시 LRU 알고리즘을 적용하는 경우 롱 플로우의 특성을 가지는 엔트리가 오랫동안 매칭되지 않은 경우 삭제될 수 있다. 그러나 해당 엔트리를 필요로 하는 패킷이 다시 들어오면 다른 플로우 엔트리를 삭제하고 새로운 엔트리를 생성해야한다. 그 결과 SFF 알고리즘에 비해 LRU 알고리즘이 제어기의 부하가 더 많이 발생하였다.

V. 결 론

본 논문에서는 SDN 네트워크가 성능이 크게 다른 스위치로 구성되어 있을 때 발생할 수 있는 문제점에 대해 설명하고 이러한 문제점을 줄이기 위해 SFF 알고리즘을 제안하였다. 제안한 SFF 알고리즘의 성능을 평가하기 위해 기존 알고리즘 중 연구를 통해 가장 성능이 우수한 것으로 증명된 LRU 알고리즘과의 성능 비교를 수행하였다. 성능 평가 결과 SFF 알고리즘이 LRU 알고리즘에 비해 전체 플로우 엔트리 매칭 횟수가 증가하였고, 롱 플로우의 특성을 가지는 플로우 엔트리의 매칭 횟수가 증가하였다. 또 스위치와 제어기 간 메시지 교환 횟수

가 감소하였고, 제어기의 부하가 감소하였다. 이를 통해 LRU 알고리즘에 비해 SFF 알고리즘이 더 효율적임을 알 수 있었다. 또 어떤 알고리즘을 사용하여 플로우 테이블을 관리할 때 플로우의 특성 파악 여부에 따라 전체 네트워크의 성능이 크게 좌우됨을 알 수 있었다.

향후 연구 과제로 패킷 처리 시간을 감소시키기 위한 다양한 연구가 진행되어야 할 필요가 있으며 스위치 메모리 사용의 효율성을 높이기 위한 연구가 필요하다. 그리고 더욱 다양한 패턴을 가지는 트래픽과 실제 인터넷 트래픽을 이용하여 각 상황에서 효율적인 알고리즘을 선택하여 사용할 수 있는 기법에 대한 연구가 진행되어야 한다. 또한 시뮬레이션 환경이 아닌 실제 SDN 네트워크를 구성하여 연구 및 성능 비교를 수행할 필요가 있다.

References

[1] N. McKeown, T. Anderson, H. Balakrishnanet, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Tuner, "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review, Vol. 38, No. 2, pp. 69-74, Apr. 2008.

[2] Q. He, X. Wang, and M. Huang, "OpenFlow-based low-overhead and high-accuracy SDN measurement framework", Transactions on Emerging Telecommunications Technologies, Vol. 29, No. 1, e3263, Jan. 2018.

[3] Y. C. Wang and S. Y. You, "An efficient route management framework for load balance and overhead reduction in SDN-based data center networks", IEEE Transactions on Network and Service Management Vol. 15, No. 4, pp. 1422-1434, Dec. 2018.

[4] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks", In ICCCN'11, Maui, HI, USA, pp. 1-6, Aug. 2011.

[5] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis", in IMC'09, Chicago, Illinois, USA, pp. 202-208, Nov. 2009.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild", in IMC'10, Melbourne, Australia, pp. 267-280, Nov. 2010.

[7] A. R. Curtis, J. Tourrilhens, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks", in ACM SIGCOMM Computer Communication Review, Vol. 41, No. 4, pp. 254-265, Aug. 2011.

[8] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller", ACM SIGCOMM Computer Communication Review, Vol. 43, No. 4, pp. 7-12, Oct. 2013.

[9] S. H. Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications", in HotSDN'12, Helsinki, Finland, pp. 19-24, Aug. 2012.

[10] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow", in 2010 Internet network management conference on research on enterprise networking, San Jose, CA, pp. 3-3, Apr. 2010.

[11] 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, Vol. 38, No. 3, pp. 105-110, Jul. 2008.

[12] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow table management in SDN switches", in CNSM'14, Rio de Janeiro, Brazil, pp. 109-117, Nov. 2014.

[13] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow macs: Scalable label-switching for commodity ethernet", in HotSDN'14, Chicago, Illinois, USA, pp. 157-162, Aug. 2014.

[14] B. Lantz, B. Heller, and N. McKeown, "A network in a Laptop: Rapid Prototyping for

Software-Defined Networks", Monterey, California, USA, in HotNets, Article No. 19, Oct 2010.
[15] POX, Python Network Controller, http://www.noxrepo.org/pox/about-pox/ [accessed: Jun. 22. 2019]
[16] Ostinato, Packet/traffic generator and analyser, https://ostinato.org [accessed: Jul. 8. 2019]
[17] tcpreplay, Replay captured network traffic, http://tcpreplay.appneta.com/ [accessed: Jul. 08. 2019]

저자소개

김 동 렬 (Dongryeol Kim)

2015년 2월 : 경기대학교 컴퓨터과학과(공학사)
2017년 2월 : 경기대학교 컴퓨터과학과(공학석사)
2017년 3월 ~ 현재 : 현대 엠엔소프트
관심분야 : 컴퓨터 네트워크

조 명 원 (Myoung-Won Jo)

2011년 2월 : 경기대학교 컴퓨터과학과(공학사)
2011년 3월 ~ 현재 : 경기대학교 컴퓨터과학과 석사과정
관심분야 : 컴퓨터 네트워크

이 병 대 (Byoung-Dai Lee)

2003년 9월 : 미네소타 주립대학교 컴퓨터공학과 (공학박사)
2003년 10월 ~ 20010년 2월 : 삼성전자 DMC연구소 책임연구원
2010년 3월 ~ 현재 : 경기대학교 컴퓨터공학부 교수
관심분야 : 네트워크 시스템