SDN 스위치의 효율적인 TCAM 사용을 위한 플로우 엔트리 클러스터링 기법

이용승, 염상길, 김동수, 추현승
성균관대학교 정보통신대학
{ys.lee, sanggil12, dskim61, choo}@skku.edu

Flow Entry Clustering for Space-Efficient TCAM utilization in SDN Switches

Yongseung Lee, Sanggil Yeoum, Dongsoo Kim, Hyunseung Choo
College of Information and Communication Engineering,
Sungkyunkwan University

요 약

최근 차세대 네트워크 패러다임으로 주목받는 소프트웨어 정의 네트워킹(SDN)에서는 네트워크를 컨트롤 플레인과 데이터 플레인으로 나누고 중앙집중형 제어를 통해 효과적이고 유연한 네트워크 관리를 가능하게 한다. 하지만 잦은 컨트롤 이벤트 발생으로 인한 컨트롤러 및 컨트롤 채널의 부하와 거대한 플로우 엔트리 크기로 인한 스위치 내 TCAM(Ternary Content Addressable Memory) 메모리 부족 문제 등의 본질적인 문제로 실제 네트워크 적용 시 확장성 문제가 야기된다. 이러한 문제를 해결하기 위해 기존의 연구들은 컨트롤러의 연산능력을 향상시키거나, 컨트롤 이벤트의 발생을 줄이는데 초점이 맞춰져 왔으며, 한정적인 TCAM 공간의 효율적인 사용에 대한 연구는 부족한 상황이다. 따라서 본 논문에서는 효율적인 TCAM 자원 활용을 위한 플로우테이블 관리 기법을 제안한다. 제안 기법은 플로우 엔트리의 클러스터링을 통해 플로우 엔트리를 특성에 따라 그룹화하고 사용빈도를 기준으로 분할 및 병합을 수행함으로써 스위치 내의 가용한 플로우 수를 최대화한다.

1. 서론

최근 차세대 네트워크 패러다임으로 주목 받고 있는 소프트웨어 정의 네트워크(SDN)는 기존의 네트워크 구조를 컨트롤 플레인과 데이터 플레인으로 분리하고 컨트롤 플레인에서 소프트웨어 정의에 의한 네트워크 관리가 이루어지는 네트워크 아키텍처를 의미한다[1]. OpenFlow는 이러한 SDN의 개념을 구현하는 대표적인 기술로 SDN을 위한 인터페이스 표준기술로 정의된다[2]. OpenFlow를 통해 SDN의 개념을 실현 시킴으로써 네트워크 관리자는 좀더 효과적이고 유연하게 네트워크를 관리할 수 있다.

고성능 네트워크에서 사용되는 네트워크 스위칭 장치는 대규모 데이터에 대한 빠른 조회 및 분류를 수행해야 한다. TCAM(Ternary Content Addressable Memory)은 단일 사이클에 라우팅 테이블 조회를 가능한 메모리 장치로 네트워크 스위칭 장치에 필수적으로 사용된다. 하지만 다른 메모리 장치들에 비해 현저히 높은 가격으로 인해 스위칭 장치 가격에 직접적인 영향을 주어 적은 양의 메모리를 극히 제한적으로 사용한다[3]. SDN스위칭 장치 역시 TCAM 공간에 플로우 테이블을 구성함으로써 빠른 데이터 처리를 수행한다. 하지만 OpenFlow 플로우 엔트리의 거대한 매치필드 크기로 인해 상대적으로 적게 저장할 수 밖에 없는 고질적인 문제가 발생한다[4]. 이는 곧 스위치에서 처리할 수 없는 플로우의 수가 많음을 의미하며 해당 플로우 처리를 위해 스위치가 컨트롤러에게 문의하는 상황을 발생시키게 되어 컨트롤러의 부하를 가중 시키는 원인이 된다.

OpenFlow의 컨트롤러 부하 이슈는 실제 네트워크 적용 시 확장성 문제로 귀결된다. 기존의 확장성 문제 해결을 위한 연구들은 컨트롤러를 분산처리 함으로써 연산 능력을 향상 시키거나[5,6] 데이터 플레인의 수정을 통해 컨트롤 이벤트 발생 빈도를 줄이는데[4,7] 초점이 맞춰져 있을 뿐 TCAM의 효율적인 관리를 위한 연구는 부족한 실정이다.

본 논문에서는 컨트롤러 부하의 근본적인 문제 해결을 위해 TCAM 영역 내 플로우 테이블의 효과적인 관리 방법을 제안한다. 플로우 엔트리의 클러스터링을 통해 각 엔트리들을 특성에 따라 그룹화하고 사용빈도를 기준으로 엔트리들간의 병합 및 분할 동작을 통해 컨트롤러의 개입을 최소화하면서 스위치 내의 한정적인 TCAM공간에 가용한 플로우 수를 최대화한다. 이를 통해 컨트롤러의 부하를 감소시키고 결과적으로 OpenFlow 확장성에 기여한다.

본 논문의 구성은 다음과 같다. 2장은 SDN의 확장성 이슈를 해결하기 위한 기존의 연구들을 소개한다. 3장에서는 효과적인 TCAM 사용을 위한 플로우 엔트리 클러스터링 기법을 설명하고 마지막으로 본 논문의 결론을 4장에 기술한다.

2. 관련연구

OpenFlow의 중앙집중형 제어 방식은 잦은 컨트롤 이벤트를 발생시키고 이는 컨트롤러 및 컨트롤 채널의 부하를 불러온다. 100개의 에지 스위치를 가진 데이터센터의 경우 최대 초당 1천만개의 새로운 플로우가 발생하는 데 비해 상용 머신들은 초당 5만개의 플로우만을 처리할 수 있어 처리량이 현저히 부족하다[8]. 이러한 확장성 문제로 인해 실제 네트워크 적용에는 많은 어려움이 따른다. OpenFlow의 확장성 문제를 해결하기 위해 현재까지 많은 연구가 진행됐으며, 크게 두 가지 접근법이 있다.

먼저 HyperFlow[6], Kandoo[5]와 같은 컨트롤러의 분산처리기법은 빈번히 발생하는 컨트롤 이벤트를 OpenFlow의 본질적인 오버헤드로 보고 컨트롤러가 처리할 수 있는 한계점을 설정하여 그 안에서 동작하도록 강제한다. 컨트롤 플레인을 물리적으로 분산시키고 논리적으로 중앙화함으로써 컨트롤 플레인의 부족한 연산 성능을 증대시킨다. 하지만 여전히 컨트롤러 및 컨트롤 채널의 부하를 줄이지 못하는 한계를 가진다.

OpenFlow의 확장성 문제를 해결하기 위한 또 다른 접근 방법으로는 스위치 수정을 통해 새로운 메커니즘을 적용하여 컨트롤러로 향하는 부하를 줄이는 것이다. 대표적으로 DIFANE[8], DevoFlow[4] 기법들이 존재한다. DevoFlow에서는 과감한 Wild card 사용과 규칙 복제로 Packet IN 이벤트 없이 새로운 플로우를 처리한다. 복제된 규칙들은 플로우가 처리됨에 따라 카운트 값이 증가하고 임계점에 도달하면 Elephant 플로우로 구분 짓고 컨트롤러에 트리거하여 플로우 처리를 요청한다. 따라서 컨트롤 플레인에 훨씬 적은 이벤트를 전달하게 되며 컨트롤러와 컨트롤 채널의 부하를 줄여준다.

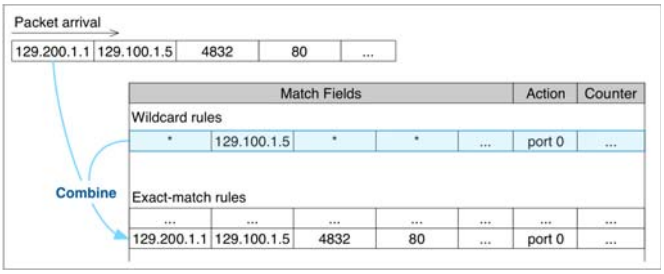

그림 1 DevoFlow 의 규칙복제 메커니즘

OpenFlow의 잦은 컨트롤 이벤트 발생은 플로우 테이블의 Miss 상황에서 발생하게 된다. 따라서 플로우 테이블에서 처리가능한 플로우 수가 많아진다면 컨트롤 이벤트 발생은 자연스럽게 줄어들 것이다. 하지만 기존에 제안된 기법들은 플로우 테이블을 효율적으로 관리하기보다 연산능력을 향상시키거나 컨트롤 이벤트 발생을 줄이는 방식을 취하고 있어 플로우 테이블의 효율적인 관리에 대한 대처가 미약하다. 특히 DevoFlow에서는 Elephant 플로우 감지를 위해 TCAM 공간을 무분별하게 사용함으로써, 한정적인 TCAM 공간을 비효율적으로 사용하게 되는 문제가 발생한다. 중요하지 않은 플로우들이 규칙 복제 메커니즘에 의하여 플로우 테이블을 차지하면서, 유효한 플로우 엔트리를 위한 공간이 잠식되는 치명적인 문제가 예상된다.

3. OpenFlow 매치필드 확장

초기 OpenFlow의 매치필드는 Ingress port, VLAN ID, Ethernet src/dst address, Ethernet type, IPv4 src/dst address, IP protocol, TCP/UDP 프로토콜의 src/dst port등10개의 튜플(288bit)을 사용하였다[2]. 이후 ONF에 의해 표준화가 진행됨에 따라 OpenFlow1.0에서는 VLAN Priority, IP ToS bits 필드가 추가되어 12개의 튜플로 확장하였으며, OpenFlow1.1에서는 Metadata, MPLS label, MPLS traffic class 필드가 추가되어 15개의 튜플로 확장하였다.

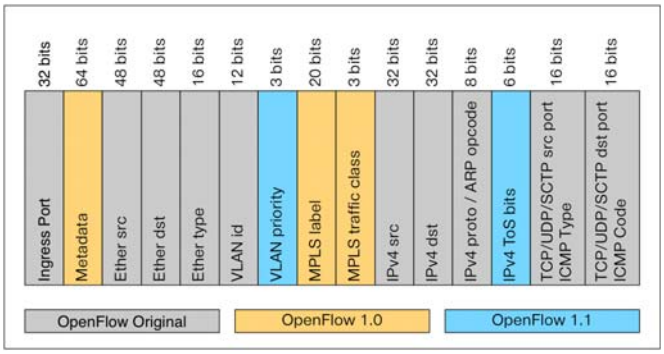

그림 2 OpenFlow 1.1 까지의 Match Fields 변화

최근 OpenFlow 1.4에 도달해서는 13개의 필수 튜플과 28개의 옵션 튜플로 구성되어 최대 41 튜플의 매치필드를 갖게 되었다[9]. 이러한 플로우 엔트리 매치필드의 크기 증가는 TCAM 메모리 부족현상을 더욱 가중시킨다.

4. 제안기법

본 논문에서는 OpenFlow 스위치에서 효율적으로 플로우테이블을 관리하는 방법을 제안한다. 플로우 테이블에 각 엔트리의 Phase를 저장하기위한 2개의 비트와 레퍼런스 여부를 판단하는 플래그 비트를 추가로 삽입한다. 특정 주기로 래퍼런스 플래그를 확인하여 해당 엔트리의 참조 여부에 따라 4개의 그룹으로 클러스터링하게 된다.

Match Fields	Action	Counter	Phase		Ref	
ingress port == 2	...	...	0	1	0	→ phase 3
IP_addr == 129.80.1.1	...	...	1	1	1	→ phase 3
Eth_addr == 00:12:34:..	...	...	1	1	0	→ expire
ingress port == 4	...	...	1	0	1	→ phase 2
Eth_tpye == ARP	...	...	0	1	1	→ phase 1
...	...	...	0	1	0	→ phase 3
...	...	...	0	1	0	→ phase 3

그림 3 플로우 엔트리 클러스터링을 위한 테이블 구조

각 Phase별로 일정 기간마다 래퍼런스 플래그를 확인하여 플로우 엔트리가 기간 내 이슈가 된 경우 상위 Phase로 이동하고 이슈 되지 않은 경우 하위 Phase로 이동시켜 플로우 엔트리 사용률에 따른 클러스터링을 수행한다. 플로우 엔트리를 사용률에 따라 Hot/Cold 플로우로 나누었을 때, Phase 1에는 빈번하게 이슈 되는 Hot 플로우에 대한 엔트리가 모이게 되며, Phase4에는 잘 사용되지 않는 Cold 플로우의 엔트리가 모이게 될 것이다. Phase 2, 3은 각 플로우 엔트리가 바로 Hot에서 Cold, Cold에서 Hot으로 급격하게 변하지 않도록 완충작용을 한다. 래퍼런스 플래그 확인은 각 Phase마다 다른 주기로 수행되며 Phase 1로 갈수록 은 빈번하게 수행한다.

새로 생성된 플로우 엔트리는 Phase 1에 추가하며, Phase 4에서도 이슈 되지 않는 플로우 엔트리는 테이블에서 버려지게 되어 플로우 테이블 내 가용한 플로우 수를 최대화한다.

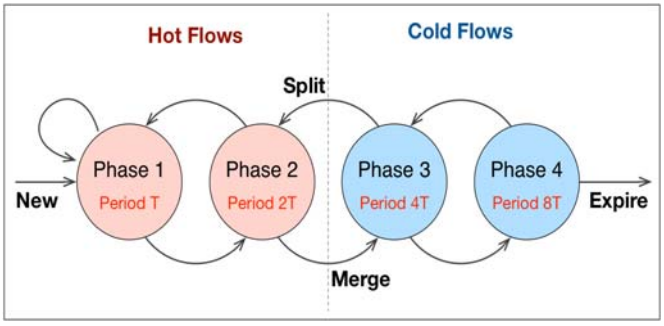

그림 4 플로우 엔트리 클러스터링

플로우 엔트리의 사용률에 따른 클러스터링이 동작함과 동시에 플로우 테이블을 효과적으로 사용하기 위한 플로우 엔트리간 Merge, Split 메커니즘을 추가한다.

플로우 엔트리의 사용률이 떨어져 Phase 2(Hot)에서 Phase 3(Cold)로 이동이 이루어질 때, 유사한 플로우들의 매치필드를 Merge하여 플로우 엔트리 수를 줄인다. 플로우 엔트리간 Merge 동작은 Phase 3으로 이동하는 플로우 엔트리의 매치 필드를 조사하여 유사한 매치필드를 갖는 엔트리 들을 Wildcard로 묶음으로써 통합한다. 엔트리 사용률이 낮은 여러 엔트리를 통합하기 때문에 포트의 과부하를 예방할 수 있다. 반대로 Phase 3(Cold)에서 Phase 2(Hot)으로 이동하게 되는 플로우 엔트리는 컨트롤러에게 새로이 질의를 통해 다시 Split 동작을 수행하게 된다.

결과적으로 스위치가 플로우 엔트리의 사용률에 따라 자율적으로 엔트리를 통, 폐합을 수행함으로써 컨트롤러에 문의 없이도 효율적으로 플로우 테이블 관리가 가능해진다. 이를 통해 스위치는 플로우 테이블 제약상황에서 좀 더 많은 플로우 처리 규칙을 가질 수 있게 되고, 플로우 테이블 관리를 자율적으로 수행함에 따라 컨트롤러에서 발생하는 부하도 줄일 수 있다.

4. 결론

OpenFlow 플로우 엔트리의 거대한 매치필드 크기로 인해 플로우 처리를 위해 기존 L2 스위치보다 최소 5배 이상의 공간을 차지하게 된다. 따라서 스위치 내부에 존재하는 플로우 엔트리 수가 현저히 줄어들게 되고 이는 곧 컨트롤 플레인의 부하를 일으킨다. 특히 OpenFlow 1.4에 이르러서는 최대 41개의 튜플의 거대한 매치필드를 지원함에 따라 한정적인 TCAM 공간에서 효율적인 플로우 테이블 관리의 필요성이 야기되었다. 본 논문의 제안기법은 이러한 문제점 해결을 위해 플로우 엔트리의 사용률에 따라 4개의 그룹으로 클러스터링을 수행하여 플로우 테이블 내 가용한 엔트리 수를 극대화 시킨다. 이와 동시에 사용률이 낮은 엔트리를 Merge하고, 다시 사용률이 높아진 엔트리를 Split함에 따라 최소한의 컨트롤러 개입으로도 효율적인 플로우 테이블 관리를 제공한다.

ACKNOWLEDGMENT

본 연구는 미래창조과학부 (한국산업기술평가관리원)의 산업융합 원천기술개발사업 (정보통신) [10041244, 스마트TV2.0 소프트웨어 플랫폼], 미래창조과학부의 차세대정보·컴퓨팅기술개발사업 (2010-0020727) 및 미래창조과학부 및 정보통신산업진흥원의 브레인스카우팅사업 (NIPA-HB603-12-1002)의 연구 결과로 수행되었음.

참고문헌

[1] Feamster, Nick, Jennifer Rexford, and Ellen Zegura., "The Road to SDN," *ACM Queue*, vol. 11, no. 12, December, 2013.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM CCR*, Vol.38, No.2, pp.69-74, 2008.

[3] K. Kannan and S. Banerjee "Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN," *ICDCN*, pp439-444, 2013

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," *SIGCOMM*, pp.254-265, 2011.

[5] S. H. Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," *SIGCOMM HotSDN*, pp.19-24, 2012.

[6] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," *INM/WREN*, 2010.

[7] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," *SIGCOMM*, 2010.

[8] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," *IMC*, 2010.

[9] Open Network Foundation, "Openflow switch specification (version 1.4.0)," Technical report, *Open Network Foundation*, October 2013.