Hashing을 사용한 Scalable Packet Classification 알고리즘 연구

허 재 성, 최 린
고려대학교 전자공학과
전화 : 02-3290-3896 / 핸드폰 : 011-9083-7400

Scalable Packet Classification Algorithm through Hashing

Jae Sung Heo, Lynn Choi
Dept. of Electronic Engineering, Korea University
E-mail : jsheo@korea.ac.kr

Abstract

It is required to network to make more intelligent packet processing and forwarding for increasing bandwidth and various services. Classification provides these 'intelligent' to network which is acquired by increasing number of rules in classification rule set. In this paper, we propose a classification algorithm efficient to scalable rule set ahead as well as present small rule set. This algorithm has competition to existing methods by performance and advantage that it is mixed with another algorithm because it does not change original shape of rule set.

I. 서론

인터넷 망의 발전은 양적으로 대역폭의 증가와 질적으로 다양한 서비스 제공으로 나타나고 있다. 인터넷 망이 보급된 이래 대역폭은 6개월에 두 배씩 증가하여 2002년 현재, 네트워크 중심부는 OC-768(40 Gbps)이 구현되고 네트워크의 주변부는 Gigabit 이더넷 속도가 보급되고 있다. 또한 인터넷 사용자가 증가함에 따라 제공되는 서비스도 다양해졌다. 이는 네트워크를 통해 교환되는 패킷들이 서로 다른 처리를 요구함을 뜻한다. 응용예로 Diff-Serv, 방화벽, VPN, load-balancing, policy-based 라우팅 등을 들 수 있다. 바로 이렇게 네트워크 망의 크기가 증가하며, 빠른 속도로 패킷을 분류하는 작업의 필요성이 커짐에 따라 패킷 classification 알고리즘의 효율이 중요해지고 있다.

현재의 패킷 classification은 payload 보다는 헤더를 기준으로 수행되고 있다. 일반적으로 3계층의 프로토콜과 IP 주소, 4계층의 포트 번호의 5개 필드를 참조하여 classification을 수행하며 이 5개 필드를 5-tuple이라 지칭한다. classification의 실제적인 방법은 아래 표 1과 같이 룰 셋을 정하고 이에 패킷을 매칭시켜 수행한다.

룰	source IP /Mask	destination IP /Mask	source port	dest port	proto -col	결 과
0	163.152.17.32 /255.255.255.255	221.17.3.0 /255.255.255.0	1024 ~65535	80	6	permit
1	163.130.3.0 /255.255.255.0	221.15.0.0 /255.255.0.0	23	1024 ~65536	6	deny
2	163.0.0.0 /255.0.0.0	223.25.31.5 /255.255.255.255	1024 ~65536	21	17	deny
3	123.12.25.46 /255.255.255.255	230.15.45.8 /255.255.255.255	2563	7777	6	permit
4	*	*	*	*	*	deny

표 1. classification 룰의 예시

패킷이 classification을 수행하는 게이트웨이, 또는 방화벽에 들어오게 되면 헤더의 5필드를 추출하여 위의 0번 룰부터 차례로 매칭을 확인하게 된다. 표 1에서 보듯 IP 는 CIDR방식에 의해 가변 길이의 마스크를 갖고, 포트 번호는 영역으로 표현된다. 이러한 특성은 classification을 더욱 어렵게 만드는 요인이 된다. 하나의 룰 셋 안에서 각개 룰들은 우선순위가 정해져 있어 처음 매칭되는 룰을 발견하면 더 이상 찾지 않고 그에 연관된 결과에 따라 패킷을 처리하게 된다. 일반적으로 룰 셋의 크기는 작은 기관, 단체에서 수백, 커다란 기관, 회사에서 수 천 개의 룰로 이루어져 있다. 서비스의 종류가 많아지고 룰들이 더욱 정교해지는 한 가지 이유와 여러 네트워크 망이 하나로 합쳐져 관리될 때 여러 룰 셋 역시 하나로 통합되는 이유에 따라 룰 셋 자체의 크기가 점점 더 커지고 있다.

현재까지 제안된 classification 방법으로는 다음과 같은 방법들이 있다. TCAM을 사용하여 하드웨어적으로 처리하는 방법, Recursive Flow Classification (RFC)을 사용하여 패킷이 룰에 매칭되는 패턴을 간략화 시켜 나가는 방법[1], 룰 셋을 비트 벡터(Bit Vector)로 표현한 뒤 이를 분할하여 매칭된 룰을 찾는 방법(Aggregated Bit Vector) [2], 룰 셋을 나무구조로 표현한 후 자주 매칭되는 룰을 찾는 깊이를 좀 더 짧게 만들어주는 방법[3] 등이 효용성을 인정받은 알고리즘들이다. 이중 TCAM을 사용한 방법이나, RFC 방법 등 대부분의 방법이 룰의 재조직 과정에서 저장공간과 전처리 시간에 제약이 심하다는 단점을 갖고 있

다. 때문에 이러한 알고리즘들은 5000룰을 제한점으로 그 보다 큰 룰 셋은 처리하기 힘들다. ABV는 제안된 방법들 중, 만 단위 룰 셋을 처리할 수 있는 거의 유일한 알고리즘이다. 하지만 효용은 많이 떨어져, 2만 룰 셋에서 평균 140번의 메모리 참조를 보이고 있다. 또한 기존의 방법들이 성능 상에 제한을 보이는 단점 외에 대부분 전처리 방법으로 룰 셋을 재조직하는 과정이 필요하기에 둘 이상의 알고리즘을 동시에 적용하기 힘들다는 점을 지적할 수 있다. 이상에서 볼 때, 요구되는 알고리즘은 **i) 적은 메모리 사용으로 더욱 커다란 룰 셋을 다룰 수 있어야** 하며, **ii) 네트워크 망의 커다란 대역폭을 지원할 수 있는 빠른 참조 속도**를 지녀야 한다.

본 논문에서는, '룰 들이 패킷에 매칭될 가능성'이라는 연관관계를 기준으로 커다란 룰 셋을 작은 버킷으로 쪼갠 후, 패킷 처리 시엔 매칭 가능성이 있는 버킷을 해싱으로 찾아내고 그 안에서만 최종 룰을 찾는 방법을 제안한다.

이는 전처리 단계에서 기존의 방법과 비교해 볼 때, 룰 셋 자체를 변형시키지 않고 단지 커다란 룰 셋을 작은 룰 셋들로 분할할 뿐이므로 다음의 두 가지 장점을 갖는다.

i) 최종 생성된 버킷은 그 자체로 작은 룰 셋이므로 버킷 안에서 기존에 제안된 대부분의 classification 방법을 적용할 수 있다.

ii) 단지 룰 들을 버킷별로 구획지울 뿐이므로 요구되는 메모리 용량이 적다.

또한 패킷 매칭 단계에서 기존 방법에 비해 갖는 장점은, 총 104비트의 5-tuple 중 그 일부만을 참조하여 두 단계 해싱을 거쳐 해당 버킷을 찾으므로 기존의 방법에 비해 비약적인 매칭 속도를 보여준다.

II. Multi-Level Hashing Classification 알고리즘

제안하는 알고리즘을 살펴보기 이전에 classification의 대상이 되는 룰의 특성에 대해 개괄해 보겠다.

2.1 룰의 특성

룰 셋의 기본적인 모양새는 표 1에서 제시하였다. 5-tuple 개개의 특성을 살펴보면 아래와 같다.

> **i)** 프로토콜은 8비트로 이루어져 있으며 정해진 하나의 값을 갖는다. 실제로 룰에 나타나는 프로토콜은 5~20가지 이하이다.
>
> **ii)** IP는 32비트로 구성되며 가변 길이의 마스크를 갖는다.
>
> **iii)** 포트는 16비트이며 고정된 하나의 번호 또는 구역을 갖는다. 특히 방화벽을 기준으로 볼 때, 출발지와 목적지 중 한 곳은 서비스 포트로서 고정된 하나의 번호를 갖고 다른 한쪽은 클라이언트가 되어 1024부터 65535까지의 영역을 임시 포트로 갖는다.

이러한 5-tuple의 특성을 참고로 우리는 특정 필드, 또는 특정 비트들만을 인덱스 삼아 해싱을 통해 룰 셋을 찢어 버킷을 생성한다. 선택하는 비트와 해싱의 단계는 어떤 것이든 될 수 있으며 경험적으로 프로토콜과 포트를 첫째 단계로, IP주소를 둘째 단계로 해싱하는 구조를 제안한다.

2.2. 해싱의 2단계 구성

(가) 1 단계 해싱: 포트와 프로토콜

첫 번째 단계에서 선택한 포트와 프로토콜은 모두 40비트이다. 이 모든 비트를 인덱스로 삼아 2^{40}개의 버킷을 만들 수는 없으므로 적절한 해싱 비트를 선정할 필요가 있다. 우리는 포트 번호 중 1023 이하의 서비스 포트만을 사용하였으며 서비스 포트가 출발지와 목적지중 어느 쪽에 있는지를 나타내는 한 비트의 방향 비트를 추가하였다. 이로써 32비트의 포트 번호를 11비트로 줄일 수 있게 된다.

프로토콜의 경우 실제 룰 셋에 나타나는 종류가 5~20가지뿐임을 감안할 때, 이 프로토콜들을 구분해 줄 수 있는 비트만을 선택하는 방법으로 사용 비트 수를 좀 더 다이어트해줄 수 있다. 하지만 그렇지 못하고 8비트를 그대로 쓴다고 하더라도 2^{19}개의 버킷은 충분히 현실적인 수치이다. 본 논문에서 실험에 사용한 룰 셋에는 10개의 프로토콜이 있고 이를 5비트로 줄여 해싱하였다. 이렇게 첫 번째 단계 해싱에서는 모두 합쳐 16비트가 사용되었으며 2^{16}개의 버킷을 생성하여 룰을 해싱하였다(아래 그림 1).

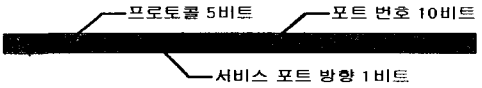

그림 1. 1단계 해싱에 사용된 비트

첫 번째 단계의 해싱만을 수행하였을 때, 룰 셋에 출현빈도가 적은 포트와 프로토콜을 갖는 룰들은 작은 크기의 버킷을 형성하지만, 6번 프로토콜에 80번 포트(http 서비스)와 같이 룰 셋 중 많은 부분을 차지하는 룰들로 만들어진 버킷은 상대적으로 커지게 된다. 실험적으로 첫 번째 단계의 해싱만을 거쳤을 때, 약 700여개의 버킷이 생성되며 이중 가장 커다란 '6번 프로토콜—80번 포트—목적지에 서비스 포트 존재' 버킷에는 750여개의 룰들이 들어가게 되었다. 이렇게 첫 번째 단계의 해싱만으로 충분하게 나누어지지 않은 버킷에 대해선 IP를 사용한 두 번째 단계의 해싱에 들어가게 된다. 본 논문에서는 두 번째 단계 해싱에 들어가는 문턱치 룰 개수로서 16을 선택하였다.

(나) 2단계 해싱: IP 주소

첫 번째 단계 해싱에서 16개가 넘는 룰을 가진 버킷에 대해서는 IP를 사용한 두 번째 단계 해싱에 들어간다. IP 주소는 출발지와 목적지를 합쳐 64비트로 구성되며 역시 적절한 수의 비트를 선택할 필요가 있다. 우리는 출발지와 목적지를 합쳐 16개 정도의 비트를 선택하여 해싱하였다.

해싱을 위해 선택한 16개의 비트는 최종적인 결과에 직접적인 영향을 미치기에 우리는 최선의 해싱 비트 패턴을 찾기 위해 다양한 방법으로 해싱 마스크를 만들어 실험하였다(아래 표 2).

해싱 마스크(출발지/목적지)	설 명
0xff000000/0xff000000 : 16bit	최상위 8비트
0xd1010001/0xd1010001 : 12bit	상대 숫수 거리 패턴
0x92224420/0x92224420 : 16bit	마스크 분포에서 유추

표 2. 해싱 마스크의 종류와 설명

i) 0xff000000/0xff000000

출발지와 목적지 IP에서 최상위 8비트씩 뽑아 해싱하는 방법이다. 다른 해싱 마스크와의 성능비교에 기준이 된다.

ii) 0xd1010001/0xd1010001

최상위 비트로부터 비트간의 거리가 2배씩이 되도록 선택한 마스크이다. 1, 2, 4, 8, 16, 32번째 비트로 구성된다.

iii) 룰 셋의 IP 마스크 패턴의 분포를 보고 아래의 방법으로 찾아낸 마스크이다. 32비트 IP 마스크의 각 비트 단위로 IP 마스크가 1로 셋된 룰의 수를 구한다(그림 2-(a)참조). MSB에서 LSB로 가며 각 자리의 셋 비트 수를 누적한다(그림 2-(b)참조). 최대값의 $\frac{n}{8}$ 이 되는 셋 비트 수와 가장 가까운 비트를 n째 비트로 선택한다.

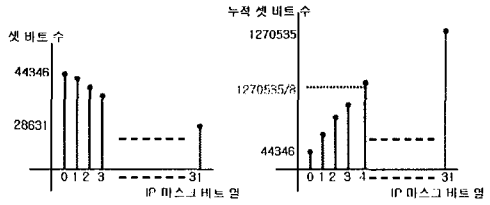

그림 2(a) IP 마스크 비트의 셋 수 (b) 비트 셋의 누적

이와 같은 방법을 사용하여 주어진 룰 셋에 대해 각기 다른 출발지와 목적지 해싱 마스크를 구할 수 있다. 실제의 실험에서 본 방법 3으로 선택한 해싱 마스크가 항상 가장 좋은 성능을 보였다.

III. 실 험

실험은 실제 PUBNET[4]에서 채집된 1,311,770개의 패킷 트레이스와 이에서 역으로 합성해 낸 두 개의 룰 셋으로 수행하였다. 룰 셋은 5천과 5만 룰 크기로 현재의 룰 셋 크기와 추후 요구되는 룰 셋 크기에 대해 본 알고리즘의 효용성을 살펴보았다.

3.1. 해싱의 결과

(가) 5천 개 룰 셋의 실험 결과

5천개의 룰 셋을 사용하여 첫 번째 해싱을 수행하였을 때, 결과는 다음과 같이 나왔다(표 3).

사용필드	비트수	버킷수	버킷당 평균 룰 수	가장 큰 버킷의 룰 수
포트	10	565	8.85	2173
포트+프로토콜	15	583	8.58	1318
방향+포트+프로토콜	16	648	7.71	746

표 3. 5천 개 룰 셋의 첫 단계 해싱 결과

예측할 수 있듯, 사용 비트수가 늘어남에 따라 버킷당 평균 룰 수와 가장 큰 버킷의 룰 수가 줄어듦을 관찰할 수 있다. 첫 번째 단계 해싱이 최종적으로 끝났을 때, 16개 이상의 룰이 들어있는 버킷은 25개가 만들어졌다. 이 25개의 버킷에 대해서 각각 두 번째 단계 해싱을 적용한 후 그 결과를 도시한 것이 아래 그림 3이다.

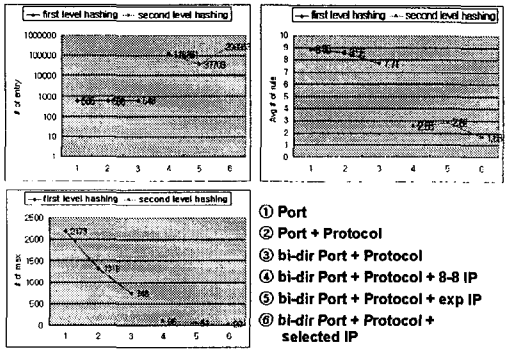

그림 3. 5천개 룰 셋에서의 해싱 결과(시계방향으로 버킷의 수, 버킷 당 평균 룰 수, 가장 큰 버킷의 룰 수)

5천개의 룰 셋에서 가장 큰 버킷의 룰이 20개로, 충분히 효과 있는 룰 셋 분할이 되었다. 이는 패킷이 들어왔을 때, 5-tuple중 일부를 보고 테이블을 참조하는 것만으로, '5천개에서 해당 룰 찾기 문제'가 '20개중 해당 룰 찾기 문제'로 축약되었음을 뜻한다.

(나) 5만 개 룰 셋의 실험 결과

다음은 5만개 룰 셋에 대한 실험 결과이다(그림 4).

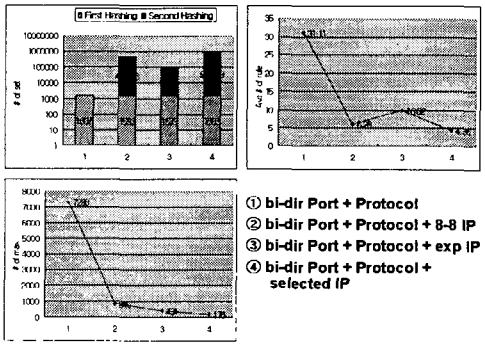

그림 4. 5만개 룰 셋에서의 해싱 결과(시계방향으로 버킷의 수, 버킷 당 평균 룰 수, 가장 큰 버킷의 룰 수)

첫 번째 해싱의 결과 모두 1,607개의 버킷이 만들어졌고 그중 가장 큰 버킷에는 7,286개의 룰이 들어있었다. 1,607개의 버킷 중 16개 이상의 룰이 들어있는 버킷은 74개였다. 이에 대해 두 번째 해싱을 적용했을 때, 최종적으로 얻을 수 있는 가장 좋은 효율은,

1,000,872개의 버킷수와, 가장 큰 버킷의 룰 수는 176개였다.

3.2 메모리 요구량

아래 그림 5는 2단계 해싱테이블의 전체 구조와 메모리 소요량을 도시하고 있다.

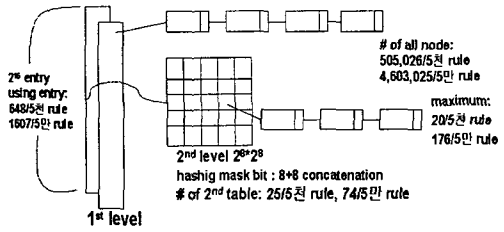

그림 5. 해싱 테이블 구조와 메모리 소요량

첫 번째 단계 해싱에서 각 룰들은 linked-list로 연결되어있고 각 노드에는 하나의 룰이 들어있다. 하나의 노드에는 IP와 마스크, 다음 노드의 포인터가 들어있으므로 6word의 메모리 공간을 요구한다. 이중 16개 이상의 노드가 들어있는 버킷에 대해선 두 번째 단계의 해싱을 적용한다. 두 번째 단계는 2^{16}크기의 테이블로 나타낼 수 있으며 마찬가지로 테이블의 엔트리를 버킷으로 보아 나누어진 룰들이 linked-list로 연결된다. 테이블의 엔트리는 linked-list의 노드를 가리키는 포인터이므로 1word로 계산한다. 최종적인 메모리 소요량은 아래과 같다.

5000 룰: 4,229,066 word
50000 룰: 27,930,326 word

1word를 4byte로 본다면 5천 룰에서 16Mbyte, 5만 룰에서 110Mbyte정도로 현재의 개인 PC 메모리 공간에서도 충분히 받아들일 수 있는 수준이다.

3.3. 패킷 트레이싱

2단계 해싱을 거쳐 최종적으로 만들어진 룰 셋에 대해 실제 1,311,770의 패킷을 흘려보내 그 매칭 결과를 시뮬레이션 해보았다. 서론에서 언급했다시피 본 알고리즘은 해싱을 통해 커다란 룰 셋을 작게 쪼개는 방법이다. 곧 작게 쪼개진 룰 셋의 재조직 방법 역시 성능 향상에 도움을 줄 수 있다. 하지만 본 실험에서는 제안하는 알고리즘의 효용을 보이기 위해, 별도의 효율적인 재조직 알고리즘을 사용하지 않고 순차찾기(linear searching) 방법을 사용하였다. 아래 그림 6은 한 패킷 당 평균 룰 참조 회수를 도시한 것이다.

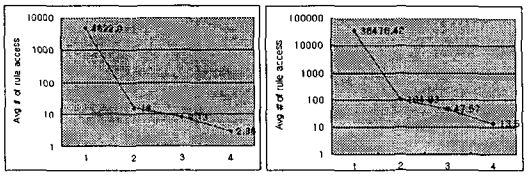

그림 6. 평균 룰 참조 수. (a)5천 룰과 (b)5만 룰에 대한 패킷 트레이싱 결과(x축은 그림 4 참조)

5천 룰에서 평균 2.86번의 룰 참조수를 얻었는데, 이는 하나의 패킷이 해당 룰을 찾기 위해 평균 2.86개의 룰을 메모리에서 가져와 5-tuple과 매칭 시켜보았다는 의미이다. 이 결과는 현재까지 나온 알고리즘 중 수천개 이하의 룰에서 가장 나은 성능을 보이는 RFC의 메모리 참조 회수 13에 대해 뒤지지 않음을 보이고 있다. 또한 5만개의 룰 셋에서의 13.5번이라는 결과 역시, 수 만개의 룰에서 가장 나은 성능을 보이는 ABV 알고리즘의 100회가 넘는 메모리 참조 수에 비해 두 배 이상 나은 성능을 보임을 관찰할 수 있다.

IV. 향후 계획 및 결론

우리는 2단계 해싱을 통해 커다란 룰 셋을 잘게 찢는 알고리즘을 제안하였다. 본 방법은 그 자체로 메모리용량과 효율면에서 현재의 수천 개 룰 셋과 앞으로 나타날 수 만개 룰 셋의 classification에서 충분히 효용을 가짐을 실험으로 제시하였다. 또한 이 방법은 작게 나누어진 룰 셋에 대해 기존에 제안된 대부분의 알고리즘을 적용할 수 있다는 장점을 갖고 있다.

향후 본 알고리즘의 개선책으로서 최선의 해싱 마스크를 찾는 노력과 다양한 포트 영역을 갖는 룰에 대해 적용하는 방법 등이 모색되어야 할 것이다.

참고문헌(또는 Reference)

[1] P. Gupta and N. McKeown, "Packet classification on multiple fields," ACM SIGCOMM 1999.

[2] F. Baboescu and G. Varghese, "Scalable packet classifiation," ACM SIGCOMM 2001.

[3] T. Woo, "A modular approach to packet classification: algorithms and results," IEEE *Infocom* 2000.

[4] 한국인터넷 정보센터, http://www.nic.or.kr/