블룸 필터를 사용한 길이에 대한 2차원 이진검색 패킷 분류 알고리즘

준회원 최 영 주*, 종신회원 임 혜 숙*

Two-Dimensional Binary Search on Length Using Bloom Filter for Packet Classification

Youngju Choe* *Associate Member*, Hyesook Lim* *Lifelong Member*

요 약

패킷 분류는 인터넷 라우터가 수행하는 가장 중요한 기능 중 하나로써 들어오는 모든 패킷을 선 속도로 처리하기를 요구한다. 영역분할을 사용한 사분트라이 구조에 길이 별 이진 검색을 적용한 알고리즘은 2차원 필드를 동시에 검색하면서 검색영역을 반으로 줄여나갈 수 있으므로 매우 효율적인 구조이다. 하지만 트라이의 레벨에 노드가 없는 경우에도 해시 테이블에 접근하는 문제점이 존재한다. 따라서 본 논문에서는 해시 메모리로의 불필요한 접근을 줄이기 위해서 영역분할을 사용한 사분 트라이의 길이별 이진 검색에 블룸 필터를 적용하는 패킷 분류 구조를 제안한다. 현재 사용되는 ACL, FW, IPC 룰 타입의 1000, 5000, 10000개의 룰 셋으로 실험한 결과, 블룸 필터를 적용함으로써 검색 성능이 21~33%까지 향상되는 결과를 얻었다.

Key Words : Packet classification, Bloom filter, binary search on trie levels, best matching rule

ABSTRACT

As one of the most challenging tasks in designing the Internet routers, packet classification is required to achieve the wire-speed processing for every incoming packet. Packet classification algorithm which applies binary search on trie levels to the area-based quad-trie is an efficient algorithm. However, it has a problem of unnecessary access to a hash table, even when there is no node in the corresponding level of the trie. In order to avoid the unnecessary off-chip memory access, we proposed an algorithm using Bloom filters along with the binary search on levels to multiple disjoint tries. For ACL, FW, IPC sets with about 1000, 5000, and 10000 rules, performance evaluation result shows that the search performance is improved by 21 to 33 percent by adding Bloom filters.

Ⅰ. 서 론

인터넷 서비스를 이용할 수 있는 어플리케이션의 종류가 다양해지고 이를 응용한 콘텐츠들의 개발과 보급이 더욱 활발해지면서 인터넷 프로토콜을 기반으로 처리해야 하는 데이터 량은 기하급수적으로 늘어나고 있다. 이러한 상황에서 사용자들에게 양질의 서비스를 제공하기 위해서는 데이터를 최종 목

※ This work was supported by the Mid-Career Researcher Program through NRF grant funded by the MEST (2011-0000232).
※ This work was also supported by the MKE (The Ministry of Knowledge Economy), Korea, under the HNRC-ITRC support program supervised by the NIPA (2011-C1090-1111-0010).
* 이화여자대학교정보통신학과 SoC Design 연구실(hlim@ewha.ac.kr)
논문번호 : KICS2012-01-012, 접수일자 : 2012년 1월 10일, 최종논문접수일자2012년 4월 17일

적지로 전송시켜주는 라우터의 성능이 중요하다. 어플리케이션에서 사용자들이 보내고 받는 데이터는 인터넷 기반에서 전송되기 위해 패킷이라는 단위로 바뀌게 되는데 해당 어플리케이션의 품질을 보장하기 위해 각 패킷을 특정 클래스로 분류해서 취급하게 된다. 라우터에서는 들어오는 패킷을 분류하기 위해 패킷 헤더에 표시되어 있는 목적지 IP 주소, 근원지 IP 주소, 목적지 포트 넘버, 근원지 포트 넘버, 프로토콜 정보를 검색한다. 라우터에 미리 저장된 룰 테이블에서 입력 패킷 헤더의 모든 필드와 일치하는 룰들을 찾고 그 중에서 최우선순위를 갖는 룰의 포트로 입력 패킷을 전달하게 되는데 이러한 과정을 패킷 분류라고 한다[1].

패킷 분류는 패킷 헤더의 5개의 필드에 대해 완전 일치(exact match), 영역 일치(range match), 프리픽스 일치(prefix match)라는 다른 방법을 적용하여 일치 여부를 판단한다. 라우터에서는 이러한 과정을 패킷이 입력되는 속도에 대응되도록 빠르게 처리해야 하기 때문에 패킷 분류의 성능을 개선시키기 위한 알고리즘 연구가 오랫동안 활발히 진행되고 있다[2].

빠른 검색 속도를 얻기 위해 각 필드를 선형 검색이 아닌 다차원 필드로 검색하는 패킷 분류 알고리즘이 많이 연구되었다. 패킷 헤더의 5개의 필드 중에서 플로우의 다양성이 가장 큰 근원지 IP주소와 목적지 IP주소를 다차원 검색하는 것이 가장 효율적이라는 연구 결과가 나왔고, 이를 바탕으로 영역분할을 사용한 사분할 트라이, 즉 AQT(area-based quad-trie)는 근원지 IP 프리픽스와 목적지 IP 프리픽스를 조합하여 이차원 트라이를 만들어 검색을 함으로써 두 필드를 동시에 검색하는 효율적인 알고리즘이다[3].

길이에 대한 이진 검색(binary search)은 검색 영역을 반씩 줄여나가면서 검색을 진행하기 때문에 IP 주소 검색에 있어서 좋은 성능을 보인다는 것이 알려져 있다[4]. 이를 패킷 분류에 적용시키려는 목적의 일환으로 영역분할을 사용한 사분할 트라이(AQT)에 길이에 대한 이진 검색을 시도하였다. 이 아이디어를 구현하기 위해서는 프리픽스의 네스팅 관계를 고려하면서 최우선 순위의 룰을 검색하기 위해 추가적인 장치가 필요했다. 기존에 제안된 알고리즘에서는 AQT 트라이의 네스팅 관계에 있는 룰들을 다른 트라이의 상대적인 레벨로 분류하고 각 분리된 트라이에 길이에 대한 이진 검색을 수행함으로써 일치하는 최우선 순위 룰을 찾았다[5]. 하지만 길이에 대한 이진검색의 특성상 트라이에 존재하지 않는 노드에 대해서도 해시 테이블로 접근하는 비효율적인 부분이 존재했다.

이에 따라 본 논문에서는 AQT 트라이를 길이별로 이진 검색하는 알고리즘을 기반으로 각 분리된 트라이에 블룸 필터를 추가하는 알고리즘을 제안한다. 블룸필터는 외부 메모리에 일치하는 룰이 있다고 판단되는 경우에만 메모리로의 접근을 진행하므로 제안하는 알고리즘은 불필요한 외부 메모리로의 접근을 줄여 검색성능을 향상시킬 수 있다. 표 1에 본 논문에서 사용할 룰 셋의 예시를 보였다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 영역 분할 패킷 분류 알고리즘과 이차원 트라이에 길이에 대한 이진 검색을 적용한 알고리즘, 블룸필터 이론과 CRC 해싱함수에 대해서 설명한다. 3장에서는 제안하는 알고리즘에 대한 빌드 과정과 검색 과정을 설명하고 4장에서는 제안하는 알고리즘과 기존의알 고리즘들 간의 성능을 비교한 후 5장에서 결론을 맺는다.

표 1. 예시 룰 셋
Table 1. Simple rule set

Rule No.	Source prefix	Destination prefix	Source Port	Destination Port	Protocol
R_0	00010*	11001*	161,161	1526,1526	17
R_1	0110*	1001*	123,123	1704,1704	2
R_2	00*	0*	119,119	0,65535	2
R_3	0*	110*	123,123	5542,5542	2
R_4	101*	1101*	53,53	0,512	1
R_5	100*	010*	80,80	21,65535	2
R_6	100*	01*	0,65535	0,65535	0
R_7	*	010*	53,53	0,65535	2

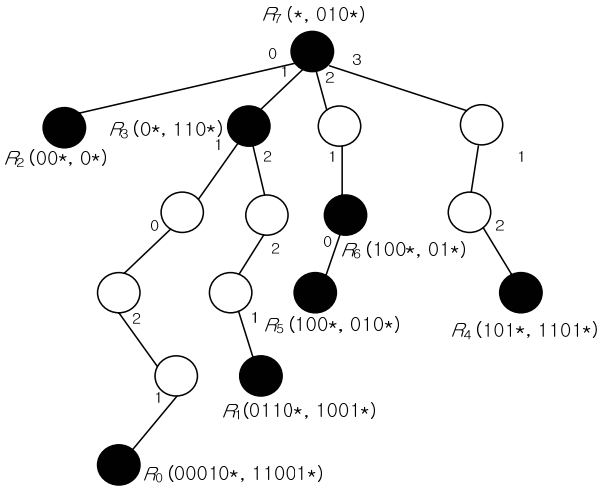

그림 1. AQT 트라이
Fig. 1. AQT (area-based quad-trie)

Ⅱ . 기존의 연구

2.1. 영역분할을 이용한 이차원 트라이

AQT(area-based quad-trie)는 룰의 근원지 프리픽스와 목적지 프리픽스만으로 구성된 트라이 구조를 사용하여 패킷 분류를 수행한다[3].

표 1의 룰 셋으로 영역분할을 이용한 이차원 트라이를 그림 1에 보였다. 트라이의 루트 노드는 근원지 프리픽스와 목적지 프리픽스를 각각 x, y-축으로 갖는 정사각형 영역에 해당되고, 트라이의 레벨이 깊어질수록 같은 크기의 정사각형 영역으로 사분할된다.

각각의 룰은 두 프리픽스 필드 중 짧은 길이의 프리픽스에 해당하는 레벨의 노드에 저장되는데, 프리픽스의 길이가 길어질수록 룰은 작은 정사각형 영역에 매핑되어 저장된다.

표 1에 제시된 룰 셋 중 R6(100*, 01*)를 예를 들어 설명하면 두 프리픽스 중에서 더 짧은 길이인 목적지 프리픽스의 길이 2만큼 번갈아 가며 근원지 프리픽스와 목적지 프리픽스 비트를 조합함으로써 10, 01의 코드워드를 얻게 된다. 이것을 이차원 AQT 트라이에서 본다면 루트 노드부터 시작해서 2→1 패스에 위치하는 노드에 R6이 저장되는 것이고 이를 검정노드로 표현한다. 이와 같은 방법으로 모든 룰에 관해서 AQT 트라이를 구성하는데 이때 같은 노드에 위치하게 되는 룰들은 링크드 리스트로 우선순위를 고려하여 연결한다.

AQT 트라이에서의 검색과정은 입력된 패킷의 근원지 주소와 목적지 주소의 한 비트씩을 이용하여 검색범위를 1/4씩 줄여나가면서 패킷과 일치하는 룰을 찾게 된다[6]. 검색의 매 단계에서 입력 패킷의 근원지 프리픽스와 목적지 프리픽스는 0, 1중 하나의 값을 가지게 되고, 두 프리픽스를 한 비트씩 조합한 경우의 수는 00, 01, 10, 11이므로 한 비트씩 검색할 때마다 검색영역이 1/4로 줄어드는 것이다[6].

2.2. 이차원 트라이에 길이에 대한 이진 검색을 적용한 알고리즘

패킷 분류를 위한 길이에 대한 이차원 이진 검색 알고리즘은 AQT 트라이에서 네스팅 관계에 있는 룰들을 상대적인 레벨에 따라 여러 개의 트라이로 분리하고 각 분리된 트라이에 대해서 길이에 대한 이진검색을 수행하는 구조이다[5]. 그림 2에 길이에 대한 이진 검색을 수행하는 AQT 트라이 구조를 보였다.

AQT 트라이의 한 패스상에 존재하는 여러 룰들을 각기 다른 트라이에 존재하도록 트라이를 분리하는 과정은 다음과 같다.

AQT 트라이를 만든 후, 트라이의 마지막 레벨에서 루트 노드로 접근하면서 각 패스에서 검정노드를 만나는 순서대로 다른 트라이에 저장한다. AQT 트라이에서 처음 만나는 리프노드들로 새로운 트라이를 구성한 후 AQT 트라이에서 해당 리프노드들과 다음 룰이 저장된 검정노드를 만나기 전까지의 빈노드들을 제거한다. 그러면 각 패스상에서 두 번

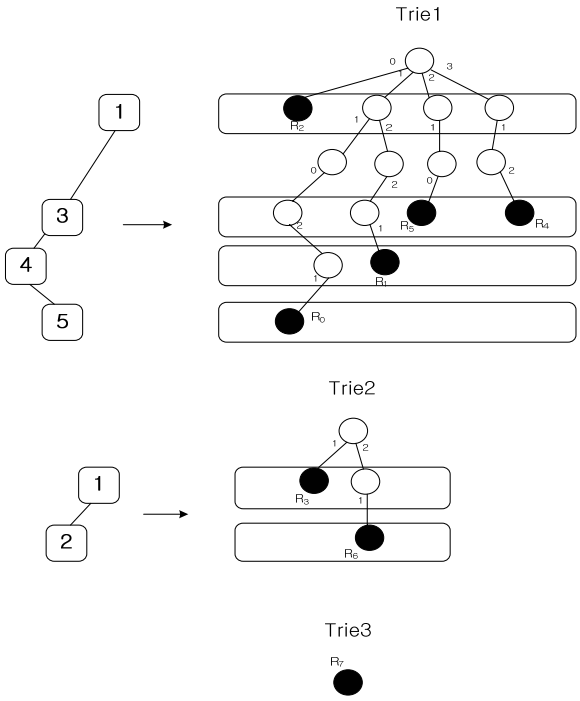

그림 2. 길이에 대한 이진 검색을 위한 패킷 분류트라이 구조
Fig. 2. Two-Dimensional Binary Search on levels trie

번째 프노드를 만나게 되고 앞서 설명한 과정과 동일하게 수행한다. 이와 같은 과정을 AQT 트라이의 모든 노드들이 없어질 때까지 진행한다.

이 때 각 새로이 만들어진 트라이는 룰 간의 네스팅 관계를 분리한 트라이를 의미하고 길이에 대한 이진 검색을 위한 검정노드가 있는 레벨의 해시 테이블은 각 분리된 트라이마다 따로 존재한다. 분리된 트라이의 최우선 순위를 고려한 검색을 하기 위해 각 분리된 트라이에 저장된 룰들 중 우선순위가 가장 높은 룰을 기억하는 과정이 추가로 진행된다. 그림 2의 분리된 트라이를 예로 본다면 트라이1은 R0를, 트라이2는 R3을, 트라이3은 R7을 기억하는 것이다.

앞선 연구에서 프리픽스가 긴 룰들이 우선순위가 높은 경향이 있다는 결과가 나온바 있다[7,8]. 이 때 가장 먼저 만들어진 트라이에는 상대적으로 프리픽스 길이가 긴 룰들이 저장되고 따라서 트라이가 만들어진 순서대로 검색을 진행한다면 가장 긴 길이를 갖는 룰 셋의 집합으로 이루어진 트라이를 우선적으로 검색함으로써 우선순위가 상대적으로 높은 룰을 먼저 검색하게 된다. 검색하려는 노드의 룰보다 현재까지 찾은 BMR(best matching rule)의 우선순위가 더 높다면 입력과 저장된 룰과의 비교를 위한 외부 메모리 접근을 하지 않는다. 대체적으로 낮은 우선순위를 가지며 많은 선형 검색을 요구하는 와일드카드 트라이를 가장 마지막에 검색함으로써, 많은 양의 선형 검색을 피할 수 있다.

2.3. 블룸 필터 이론

블룸 필터는 주어진 집합에 속하는 원소들의 존재를 비트-벡터 형식으로 나타내는 데이터 구조로서, 입력이 주어지면 입력과 일치하는 원소가 주어진 집합 내에 존재하는지의 여부를 판단하는 장치이다. 블룸 필터는 작은 메모리로 검색 성능을 향상시킬 수 있는 효율적인 데이터 구조이다[9,10].

블룸 필터가 선-필터(pre-filter)역할을 하기 위해서는 주어진 집합의 데이터를 비트-벡터로 저장하는 프로그래밍 과정을 먼저 수행해야 한다[9]. 블룸 필터 프로그래밍 과정은 다음과 같다.

BFAdd (x)
1) for (i = 1 to k)
2) Vector[$h_i(x)$] ← 1

우선 블룸 필터의 모든 색인의 비트-벡터 값은 0으로 초기화 된다. 그 다음 주어진 데이터 집합에 존재하는 각 원소 x들의 정보를 해시 함수의 입력으로 넣으면 출력으로 블룸 필터를 프로그래밍할 위치를 나타내는 색인 hi(x)를 얻게 된다. 집합 내의 모든 원소에 대해서, 해당 색인에 대응되는 비트-벡터 값을 1로 지정해주는 작업을 수행하면 블룸 필터 프로그래밍이 완료된다. 블룸 필터프로그래밍에 있어 미리 프로그래밍된 원소와 동일한 색인이 얻어지는 경우 해당 색인의 비트-벡터 값을 1로 유지한다. 이 때 사용하는 해시 함수의 개수 k는 블룸 필터의 크기에 따라서 정해지게 된다. 만약 원소 하나당 k개의 해시 함수를 사용했다면 프로그래밍 해야 할 위치 색인을 k개 얻는 것이다. 블룸 필터에 저장되어야 하는 전체 노드의 개수를 N 이라고 할 때 N 보다 큰 2의 배수를 블룸 필터의 기본크기로 정하면, 블룸 필터의 기본크기 N'는 다음과 같다.

$$N' = 2^{\lceil \log_2 N \rceil}$$

기본 블룸 필터의 크기에 확장배수를 곱하여 크기가 M인 블룸 필터를 사용할 경우, 프로그래밍 되는 색인의 개수도 늘어나야 올바른 성능을 얻을 수 있다. 프로그래밍 해야 할 색인은 블룸 필터의 크기에 비례하여 많아진다. 블룸 필터의 해시 함수 개수 k를 정하는 식은 다음과 같다.

$$k = \frac{M}{N'} \times \ln 2$$

프로그래밍된 블룸 필터를 검색과정에서 사용할 때는 입력 값과 필터에 저장된 값을 비교하는 쿼리과정을 진행한다[9,10]. 쿼리과정은 다음과 같다.

BFQuery (x)
1) for (i = 1 to k)
2) if (Vector[h_i (x)] = 0) return negative
3) return positive

쿼리를 수행하기 위해서는 우선적으로 입력 값의 블룸 필터 색인을 알아야한다. 이를 구하기 위해서 입력 정보를 블룸 필터 프로그래밍 과정에서 사용했던 해시 함수와 동일한 함수의 입력으로 넣어 입력의 블룸 필터 색인을 얻는다. 이때 입력의 블룸 필터 색인의 개수는 블룸 필터가 프로그래밍 될 때의 색인 개수와 같아야 하고 입력으로부터 얻은 모든 블룸 필터 색인의 위치에 해당하는 비트-벡터 값

을 확인한다. 하나의 입력에 대해서 구해진 모든 색인의 비트-벡터 값이 1인 결과를 양성이라 하고 양성의 경우에만 블룸 필터를 통과하게 된다. 하나의 입력에 대한 색인들 중에서 하나라도 비트-벡터 값이 0을 나타나게 되면 블룸 필터를 통과하지 못하고 이 경우를 음성이라 한다. 실제 집합에 존재하는 원소가 아닌데 우연히 입력의 색인이 집합에 존재하는 원소들의 색인을 조합한 것과 일치하는 경우가 있을 수 있는데 이런 경우를 거짓-양성이라고 한다. 거짓-양성일 때에는 입력이 블룸 필터를 통과하지만 집합에서 입력과 일치하는 원소를 찾을 수 없다. 반대로 양성의 결과로 블룸 필터를 통과하고 실제로 입력과 일치하는 정보가 주어진 집합에 존재하는 경우는 참-양성이라 한다.

2.4. CRC 해싱 함수

CRC는 정해진 크기의 키 값인 생성 다항식으로 데이터 블록의 값을 나눈 나머지를 누적시키는 해시 함수이다[11]. CRC 연산은 데이터를 메시지 다항식으로 변환해서 생성 다항식으로 나눗셈을 하는 방식이고 연산도중 발생하는 캐리는 사용하지 않는다. 이러한 연산의 결과로 얻어지는 나머지를 CRC 코드라고 한다. CRC는 주로 데이터 오류 검출을 하는데 사용되어 왔고, 생성 다항식을 어떤 값으로 하느냐에 따라 데이터 내용 검증에 대한 성능이 영향을 받는다.

본 논문에서 제안하는 구조에서는 블룸 필터의 색인을 정하기 위한 해시 함수로 CRC 생성기를 사용한다. CRC 생성기를 해시 하드웨어로 사용하면 다양한 길이의 프리픽스에 대해 동일한 길이의 비트가 잘 섞인 CRC 코드 값을 얻을 수 있다. 이렇게 얻은 CRC 코드에서 원하는 비트를 추출하면 다양한 크기의 해시 색인을 원하는 수만큼 선택하여 사용할 수 있다는 장점이 있다. 각 룰에 대한 블룸 필터 색인을 정하는 문제에서 프로그래밍 되는 색인 값이 군집화 되지 않고 전체 색인 값에 걸쳐 골고루 분포되어야 블룸 필터가 좋은 성능을 보일 수 있다. 이러한 점을 고려했을 때 CRC 생성기는 입력되는 비트 열의 0과 1을 비슷한 비율로 잘 섞어 주어 0과 1이 잘 분산된 CRC 코드 값을 얻을 수 있기 때문에 색인을 추출하기 위해 사용하는 해시 하드웨어로 매우 적합하다고 볼 수 있다. 또한 CRC 해시 함수를 하드웨어로 구현함에 있어서 생성 다항식에 존재하는 항들을 레지스터에 미리 저장하고 나머지 계산은 시프트 레지스터와 EX-OR을 사용하여 구현하면 되기 때문에 구조가 간단하다. 그림 3은 생성 다항식이 x^8+x^2+x+1 일 때 8비트-CRC 하드웨어 구조를 보이고 있다.

표 2. 표 1의 목적지 프리픽스에 대한 CRC코드
Table 2. CRC code for destination prefix of Table I

Destination prefix	CRC code
11001*	10011100
1001*	10000111
0*	00000000
110*	10110100
1101*	11101011
010*	11011000
01*	10110001

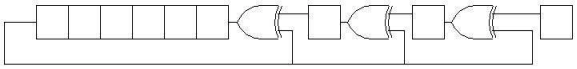

그림 3. 8bit-CRC 하드웨어 구조
Fig. 3. 8bit-CRC Hardware Structure

CRC내의 모든 플립-플랍(flip-flop)은 0으로 초기화되어 있으며, 입력 값이 한 비트씩 들어오면서 플립-플랍에 저장된 값과 XOR 된다. 입력 비트열의 마지막 비트까지 들어오면 CRC 생성기의 연산 수행을 종료하고 플립-플랍에 저장되어 있는 값을 CRC 코드로 출력한다. 예를 들어, 표 1의 목적지 프리픽스에 대하여 그림 3의 8비트-CRC 하드웨어를 거치면 표 2의 CRC 코드가 생성된다.

Ⅲ . 제안하는 구조

3.1. 블룸 필터를 적용한 AQT 트라이의 길이에 대한 이진 검색

본 논문에서 제안하는 알고리즘은 길이에 대한 이진 검색을 패킷 분류에 적용하기 위해 기존의 AQT 트라이를 상대적인 레벨을 가지는 트라이로 분리시키고 각 분리된 트라이에 블룸 필터를 추가하여 길이에 대한 이진 검색의 성능을 향상시키는 알고리즘이다.

패킷 분류는 입력 패킷과 일치하는 룰들 중에서 최우선 순위 룰을 선택해야 하므로 우선 순위가 높은 룰을 우선적으로 검색해서 일치하는 룰을 찾는다면 더 낮은 우선순위의 룰 검색을 제외할 수 있어 해시 테이블로의 메모리 접근을 줄일 수 있다. 따라서 AQT 트라이를 리프에서 루트노드 방향으로 분리한 트라이 순서로 검색을 진행함으로써 상대적으로 프리픽스의 길이가 긴, 우선순위가 높은 룰들

표 3. 제안하는 알고리즘의 그림 2의 분리된 트라이1에 대한 CRC코드와 블룸 필터 색인 값
Table 3. CRC code for discomposed trie1 of proposed algorithm and Index of Blomm Filter

Level	Node Type	key	CRC code	BF Index
Level 1	R_2	0000001	01100110	6
	Internal node	0100001	01110000	7
	Internal node	1000001	01101101	6
	Internal node	1100001	01111011	7
Level 3	Internal node	01010000011	01110011	7
	Internal node	01101000011	10111111	11
	R_5	10010000011	00000010	0
	R_4	11011000011	10101000	10
Level 4	Internal node	0101001000100	01001110	4
	R_1	0110100100100	01100000	6
Level 5	R_0	0101001001001 01	00010010	1

을 먼저 검색하게 한다.

블룸 필터를 적용하는 방법은 다음과 같다. 루트 노드를 제외한 각 분리된 트라이 마다 독립적으로 블룸 필터를 만들고 해당 트라이의 룰이 존재하는 유효 레벨의 모든 노드들을 블룸 필터에 프로그래밍 한다.

각 분리된 트라이의 길이 별 이진 검색을 수행하는 과정에서 접근해야 하는 해시테이블의 레벨이 정해지면 메모리에 접근하기에 앞서 해당 트라이의 프로그래밍된 블룸 필터에 쿼리를 한다. 블룸 필터의 특성상 해시테이블에 해당 노드가 저장되어 있다면 블룸 필터를 반드시 통과할 것이고 접근하려는 해시테이블에 일치하는 노드가 존재하지 않는다면 높은 확률로 블룸 필터를 통과하지 못할 것이다. 이 때 블룸 필터를 통과한 경우에만 해당 레벨의 해시 테이블로 접근하여 일치하는 룰을 찾은 과정을 계속해서 진행하게 된다. 동일 검색과정을 분리된 모든 트라이에 대해서 수행한다.

3.1.1. 블룸 필터 프로그래밍

각 분리된 트라이를 위한 블룸 필터를 프로그래밍 하는 과정은 다음과 같다. 하나의 트라이에 유효 레벨에 존재하는 노드들을 프로그래밍 한다. 유효 레벨이란 룰이 하나라도 존재하는 레벨을 말한다. 표 3은 그림 2의 첫 번째 트라이의 유효 레벨 해시 테이블에 저장된 노드에 해당하는 코드워드, CRC 코드, 블룸 필터 색인 값을 정리한 것이다. 첫 번째 트라이의 유효 레벨 1을 보면 왼쪽부터 차례로 4개의 노드가 있다. 이를 표 3의 Level1에 위부터 순서대로 나타내었다. Node Type에 룰 번호가 적힌 노드는 검정노드 이고 internal node는 빈 노드를 의미한다. 블룸 필터의 크기는 기본크기를 사용한 것이다.

예를 들어 그림 2의 첫 번째 트라이의 경우 레벨 1, 3, 4, 5가 유효 레벨이다. 하나의 룰을 블룸 필터에 프로그래밍 하기 위해서는 해당 룰이 저장된 분리된 트라이가 가지는 유효 레벨 마다 코드워드를 만들어 해시 함수에 입력한다. R0(00010*, 11001*)를 예로 들면 첫 번째 분리된 트라이의 유효 레벨이 1, 3, 4, 5이므로 R0의 코드워드는 1, 110, 1102, 11021이다. 이어서 각 코드워드를 레벨을 나타내는 5 비트의 값으로 패딩한다. 표 3의 마지막 행에, 룰 R0의 코드워드에 레벨 값이 패딩된 key값을 볼 수 있다. 레벨 값을 패딩하면 다른 유효 레벨에 존재하는 노드들의 색인 조합으로 거짓-양성의 결과가 나오는 문제점을 개선할 수 있다. 결과로 얻은 CRC코드에서 블룸 필터 색인으로 사용할 값을 선택하고 해당 색인을 주소로 가지는 부분의 블룸 필터 비트 값을 1로 저장한다.

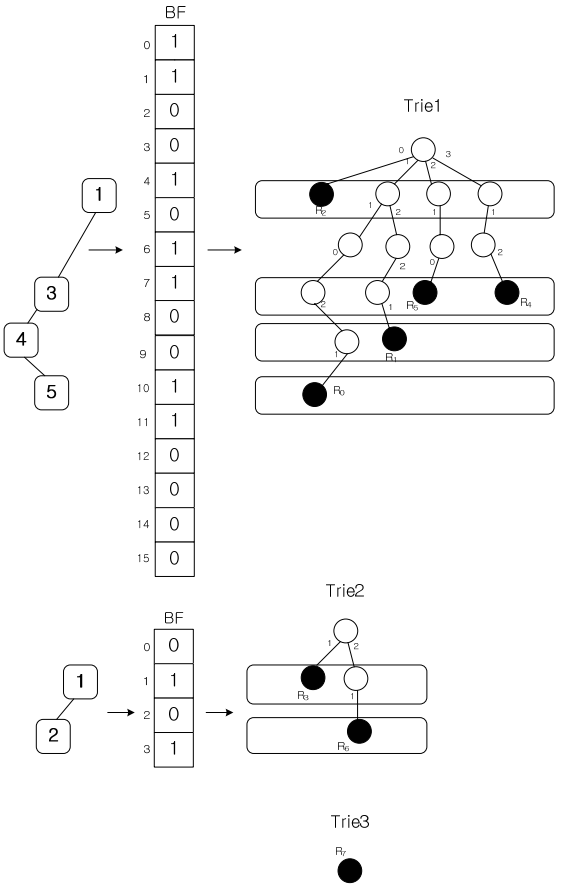

그림 4. 제안하는 알고리즘 구조
Fig. 4. Proposed algorithm

표 3의 블룸 필터 색인 위치의 비트 벡터 값이 1로 프로그래밍 된 결과를 그림 4의 Trie1에서 확인할 수 있다. 그림 2의 트라이1의 유효 레벨에 존재하는 노드의 개수가 11개 이므로 블룸 필터의 기본크기는 16 비트이다.

본 논문에서 예시를 들어 설명하기 위해 사용한 CRC코드는 표기를 용이하게 하기 위해서 CRC-64를 거쳐 나온 코드의 앞 8 비트만을 CRC 코드로 사용했다. 따라서 표 3의 CRC 코드가 8 비트로 표현된 것이다. 마찬가지로 검색 과정에서 인풋에 대한 블룸 필터 색인을 얻을 때도 64비트의 CRC 코드에서 앞의 8비트만을 CRC 코드로 사용했다.

3.1.2. 검색 과정

검색과정은 입력된 패킷에 대하여 분리된 트라이가 만들어진 순서대로 모든 트라이에 걸쳐 진행한다. 그림 5에 검색 과정을 플로우 차트로 나타내었다.

패킷이 들어오면 우선 첫 번째로 만들어진 트라이에 존재하는 해시 테이블의 레벨에 대해 길이에 대한 이진 검색을 한다. 접근해야 하는 트라이의 중간 레벨이 정해지면 해당 레벨의 길이만큼 근원지 프리픽스와 목적지 프리픽스를 조합하여 코드워드를 만들고 프로그래밍에 사용한 해시함수와 동일한 CRC-64에 입력하여 64비트 출력 값을 얻는다. CRC 출력 값에서 블룸 필터 프로그래밍 했을 때와 동일한 방법으로 색인 값을 선택하고 해당 색인 위치에 블룸 필터 쿼리를 수행한다. 접근하는 모든 색인의 블룸 필터 비트값이 1이라면 검색해야 하는 레벨의 해시 테이블로 접근을 하고 하나라도 0이 존재한다면 해당 레벨의 해시테이블로 접근하지 않고 상위 레벨로 이진 검색을 진행하게 된다.

블룸 필터 쿼리 결과가 양성인 경우 현재까지 일치하는 룰이 없었을 때는 무조건 해시테이블에 접근해서 해당 노드가 있는지를 확인한다. 노드가 존재하면 참-양성인 경우이고 존재하는 노드가 빈 노드인 경우에는 하위 레벨로 이진 검색을 진행하게 되고 룰이 저장된 노드인 경우에는 해당 룰의 모든 필드와 인풋의 필드가 일치하는지 확인한다. 만약 모든 필드가 일치한다면 그 룰을 BMR (best matching rule)로 정하고 바로 다음 트라이로 검색을 진행한다. BMR이 존재하는 경우에는 블룸 필터 쿼리에 앞서 우선적으로 검색을 진행하려는 트라이의 최우선순위를 확인한다. 트라이의 최우선순위가 현재까지 일치한 룰의 우선순위보다 더 낮은 경우에는 현재 트라이의 검색을 하지 않고 바로 다음 트라이를 검색한다. 반대의 경우에는 블룸 필터 쿼리를 하고 계속해서 블룸 필터 쿼리 결과가 양성이 나온 경우에는 해당 노드가 가리키는 룰의 우선순위와 BMR의 우선순위를 비교한다. 접근하려는 노드의 룰의 우선순위가 더 높은 경우에 룰과 인풋의 일치 여부를 판단하고, 일치하는 경우 BMR을 업데이트한 후 다음 트라이로 검색을 진행한다. 일치하지 않는 경우는 현재 BMR을 유지하고 다음 트라이로 검색을 진행한다. 접근하려는 노드의 룰의 우선순위보다 현재 BMR의 우선 순위가 더 높은 경우에도 현재 BMR을 유지하고 다음 트라이로 검색을 진행한다. 블룸 필터를 통과했는데 해시 테이블에 노드가 존재하지 않았을 경우에는 거짓-양성으로 하위 레벨에는 인풋과 일치하는 룰이 없을 것이기 때문에 상위 레벨로 이진 검색을 진행하게 되고 계속해서 이진 검색을 진행하다. 현재 트라이의 마지막 검색 레벨에 도달하면 다음 분리된 트라이로 검색을 계속하고 더 이상 분리된 트라이가 없을 때까지 검색을 진행한다.

블룸 필터 쿼리 결과가 음성인 경우에도 거짓-양성일 때와 마찬가지로 하위 레벨에는 인풋과 일치하는 룰이 없을 것을 보장하기 때문에 상위 레벨로 검색을 진행한다. 마찬가지로 모든 분리된 트라이에 대해서 길이에 대한 이진 검색을 수행한다. BMR이 존재하는 경우에는 블룸 필터 쿼리에 앞서 우선순위를 고려하는 검색을 하는데 블룸 필터 쿼리 결과가 양성인 경우에 설명한 방법과 동일하다.

제안하는 구조의 검색과정을 표 1에 주어진 룰 셋으로 인풋(00101, 00111)을 예로 들어 설명하고자 한다. 설명과정에서 사용하는 값들을 표 4에 정리했다. 예시에서는 CRC-64의 CRC코드 중에서 앞의 8 비트만을 CRC코드로 사용하고 블룸 필터 색인은 CRC코드의 앞 4 비트로 추출하였다. 우선 첫 번째 트라이의 유효레벨의 중간 레벨인 레벨4로 접근해야 한다. 따라서 인풋의 4 비트씩 조합한 결과(00001101)에 길이4를 5 비트로 표현한 (00100)패딩을 붙여 CRC에 입력할 키(0000110100100)를 구한다. 이 키 값을 CRC에 넣어 얻은 CRC코드의 값에서 앞의 4 비트를 선택하여 블룸 필터 색인을 추출하면 13이다.

표 4. 인풋 (00101, 00111)의 키, CRC코드, 블룸 필터 색인
Table 4. CRC Code of Input(00101, 00111) and Index of Bloom Filter

Trie	level	Key	CRC-8 value	BF index
Trie1	Level4	0000110100100	11011101	13
	Level3	00001100011	01001010	4
	Level1	0000001	01100110	6

앞서 Trie1의 블룸 필터 색인 13의 비트 값은 1로 프로그래밍 되어 있으므로 양성이고 해시 테이블에 접근한다. 하지만 해시 테이블에 해당 노드가 존재하지 않으므로 거짓-양성이 된다. 노드가 없다는 것은 해당 인풋과 일치하는 룰이 하위레벨에는 없을 것이란 것을 보장하기 때문에 상위레벨로 검색을 진행한다. 상위레벨의 중간 레벨인 3에 접근하기 위해 인풋의 근원지 프리픽스와 목적지 프리픽스를 3 비트씩 조합한 후, 길이 3을 패딩한 결과를 붙인 키로 CRC 코드를 얻는다. CRC 코드에서 앞의 4비트를 선택한 블룸 필터의 색인은 4가 되고 트라이1의 블룸 필터에 색인4는 0으로 프로그래밍 되어 있으므로 음성이 되어 해시 테이블로 접근을 하지 않는다.

계속해서 상위레벨 중에서 레벨1을 검색 하게 되고 이때는 블룸 필터가 양성 반응이므로 해시테이블로 접근하는데 해당 노드가 존재하기 때문에 참-양성이 된다. 검정노드로 저장된 룰과 인풋이 일치하는지를 검사하고 그 결과 모든 필드가 일치하므로 BMR로 R2를 기억한 후 다음 트라이로 검색을 진행한다. 트라이2의 우선순위는 3으로 BMR인 R2보다 우선순위가 낮으므로 검색을 하지 않고 트라이3도 마찬가지 이유에서 검색하지 않는다. R2를 출력하고 검색을 종료한다.

Ⅳ. 성능 평가

본 논문에서는 실제 백 본 라우터에서 사용되는 라우팅 테이블을 사용하여 제안하는 알고리즘의 성능을 평가하였다. 성능 평가에 사용된 룰 셋은 클래스벤치가 제공하는 룰 셋 중에서 ACL(access control list), FW(firewall), IPC(IP chain) 타입의 룰을 약 1000개, 5000개, 10000개의 룰을 포함하는 룰 셋으로 성능을 실험하였다[12,13]. 본 논문에서는 1000개, 5000개, 10000개를 1k, 5k, 10k로 나타내었고 실제 사용된 룰 테이블의 룰 개수는 표 5에 나타내었다.

각 룰 타입은 서로 다른 룰 특성을 보인다. 그림 6에 각 룰 타입 별로 분리된 트라이에 존재하는 유효레벨과 그 레벨에 저장된 룰의 개수를 보였다. ACL의 경우 분리된 트라이가 5개로 룰들의 최대 네스팅 관계가 5번인 것을 알 수 있다. 첫 번째 리프노드로 구성된 트라이에 가장 많은 룰이 저장되었고 총 레벨의 가지 수는 14개로 길이가 긴 레벨에 많은 룰이 저장되는 것을 볼 수 있다. 다음으로 분리되는 트라이 일수록 저장되는 룰의 수가 줄어들고 마지막 트라이 인 루트노드에는 3개의 룰 만이 저장된다. FW는 분리된 트라이의 수가 3개로 네스팅 관계가 ACL, IPC보다 적은 것을 알 수 있다.
첫 번째 분리된 트라이에 레벨의 가지 수는 27로 매우 다양하고 각 레벨에 저장되는 룰의 개수를 살펴보면 길이가 작은 레벨에 많이 저장된 것을 알 수 있다. 마지막 루트노드에 저장되는 룰이 162개로 ACL과 IPC와 비교했을 때 월등히 많은 것을 알 수 있다.

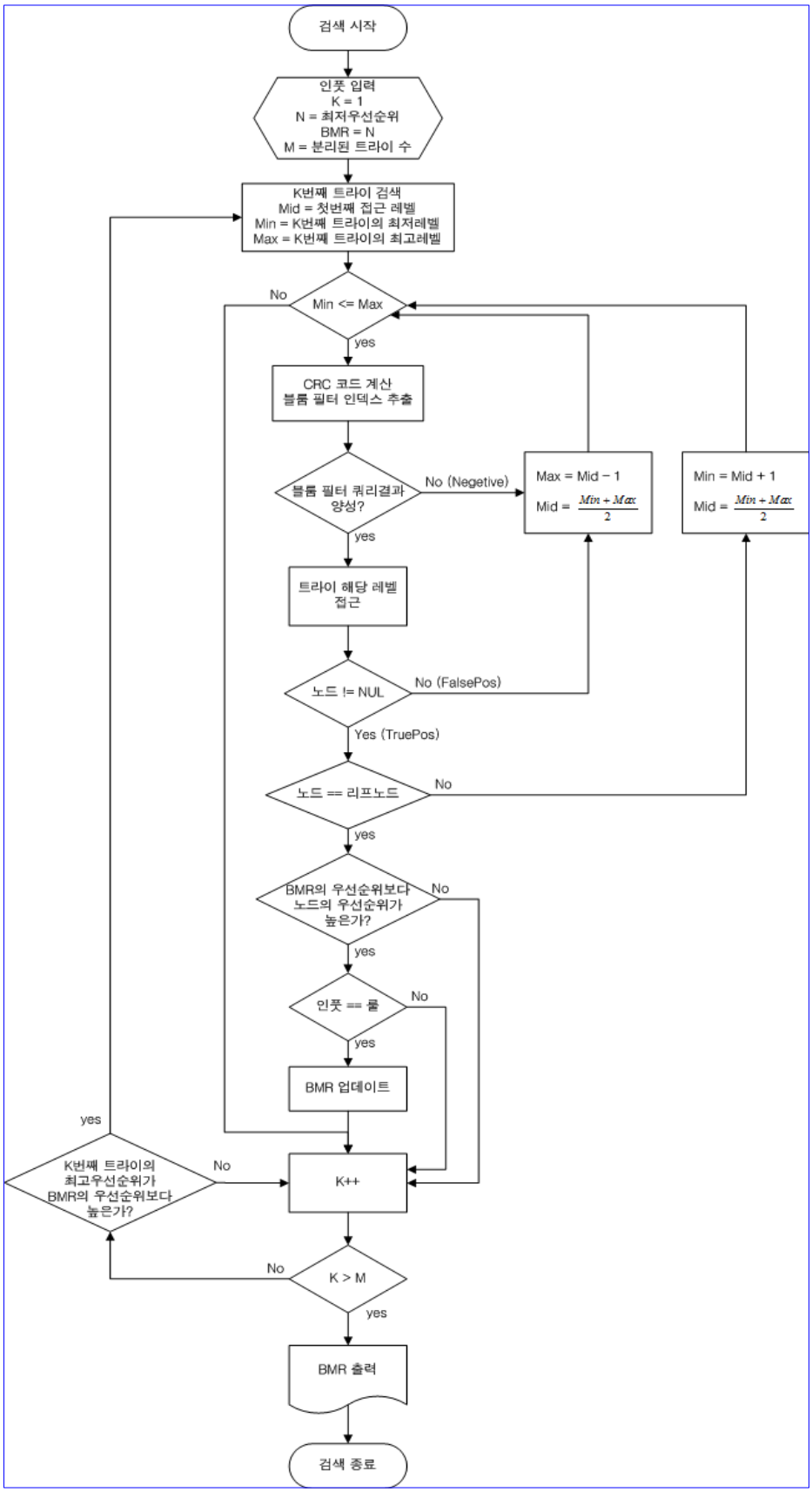

그림 5. 제안하는 알고리즘 검색 과정
Fig. 5. Search flow of proposed algorithm

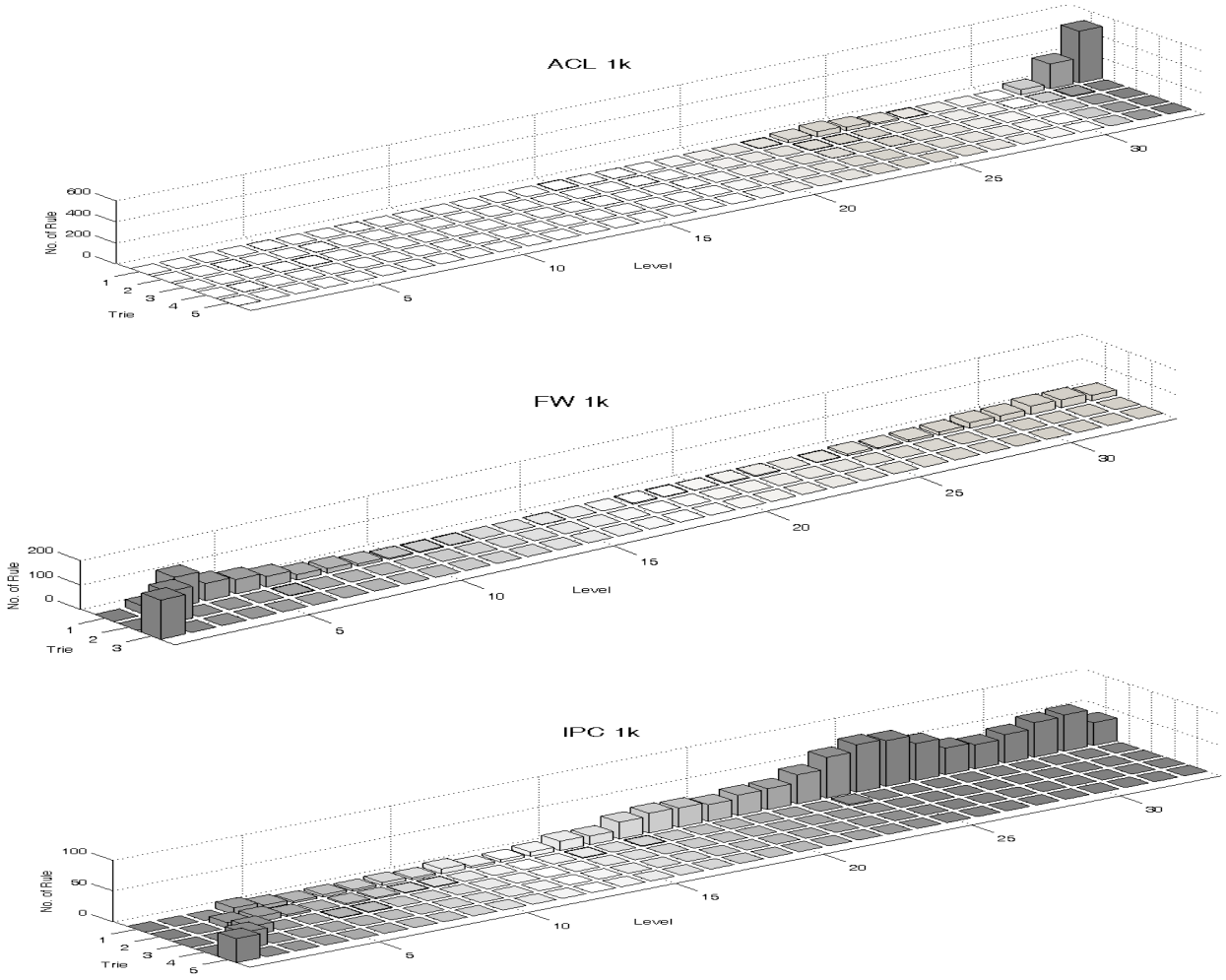

그림 6. Acl, Fw, Ipc 룰 타입에 따른 분리된 트라이의 레벨 별 저장되는 룰의 개수
Fig. 6. Number of rules for each decomposed trie for Acl, Fw, Ipc

한 노드에 링크드 리스트로 저장된 룰이 많기 때문에 FW 같은 경우는 선형검색에 영향을 많이 받을 것이라는 것을 예상할 수 있다. IPC의 경우는 분리된 트라이가 5개이고 첫 번째 리프노드로 구성된 트라이에 존재하는 레벨의 가지 수가 30개로 제시된 룰 타입 중에서 가장 많고 저장된 룰도 가장 많은 것을 알 수 있다. 블룸 필터는 트라이 당 하나씩 적용되는 상황에서 첫 번째 트라이에 있는 레벨이 다양하므로 다른 레벨에 존재하는 노드들의 색인 조합과 우연히 일치해 일치하는 노드가 없는 인풋이 블룸 필터를 통과하게 되는 거짓-양성의 기회가 많아질 가능성이 크다. 이러한 문제점은 CRC 코드워드를 만들 때 레벨, 즉 길이를 값으로 패딩을 함으로써 대부분 해결된다.

패킷 분류 문제에서 off-chip 메모리로의 접근이 검색 속도에 큰 영향을 끼친다. 따라서 그림 7에서는 $8N'$ 크기의 블룸 필터를 길이에 대한 이차원 이진검색에 적용했을 때와 적용하지 않았을 때의 평균 메모리 접근 횟수를 비교하여 나타내었다.

그림 7 보면 전체적으로 평균 2~3 번의 메모리 접근 횟수가 줄어든 것을 알 수 있다. ACL의 경우에는 근원지와 목적지 프리픽스의 길이가 긴 룰이 많고 한

표 5. 실제 룰 테이블의 룰 개수
Table 5. Number of rules for performance evaluation

	acl	fw	ipc
1k	958	871	988
5k	4660	3067	4468
10k	9735	4351	8112

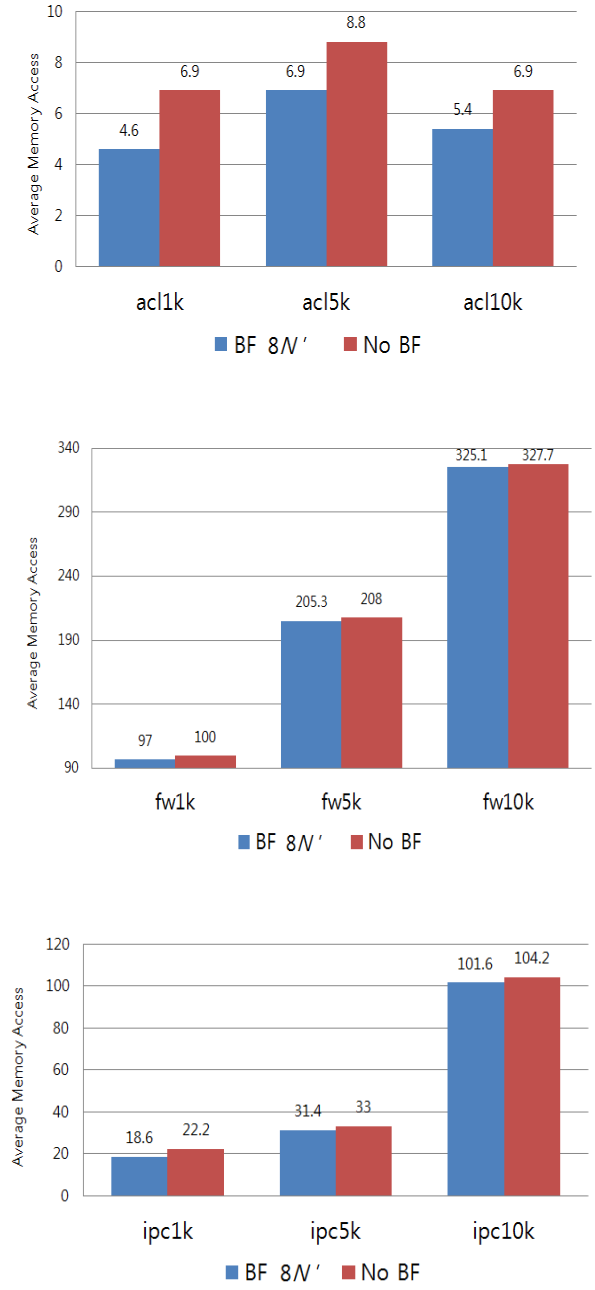

그림 7. ACL, IPC, FW에서 8N' 블룸 필터 적용에 따른 성능 변화
Fig. 7. Performance comparison of Using Bloom filters and Not Using Bloom filters

노드에 링크드 리스트로 연결된 룰의 수가 적기 때문에 대부분의 메모리 접근은 길이에 대한 이진 검색의 성능을 따라가게 된다. 블룸 필터를 적용했을 때는 적용하지 않았을 때 보다 2번 정도 적은 접근으로 일치하는 룰을 찾을 수 있는데 이는 없는 노드에 대한 접근을 블룸 필터가 차단해 주기 때문에 얻은 성능향상으로서, 비율로 보면 최대 33%의 메모리 접근을 감소시키는 효과라 할 수 있다.

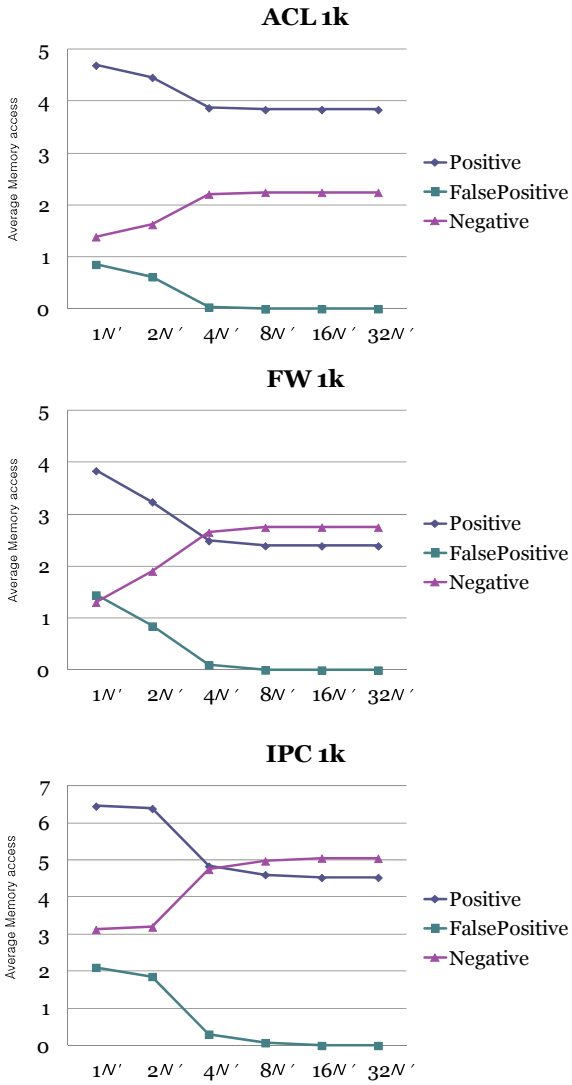

그림 8. ACL, IPC, FW에서 블룸 필터 사이즈 변화에 따른 성능 변화
Fig. 8. Performance depending on size of the Bloom filter for Acl1k, Fw1k, Ipc1k

FW의 경우는 근원지나 목적지 프리픽스의 길이 가 짧은 룰이 많다. AQT의 기본원리는 근원지나 목적지의 프리픽스 중에서 짧은 길이의 프리픽스를 기준으로 한 레벨에 룰이 저장되는 것이기 때문에 본 논문에서 제안하는 알고리즘에 이 원리가 똑같이 적용되어 FW는 한 노드에 저장되는 룰이 많다. 같은 노드에 저장되는 룰들은 링크드 리스트로 연결되어 있기 때문에 선형검색을 해야 하고 FW는 이러한 링크드 리스트로 인한 선형검색이 많기 때문에 평균 메모리 접근 횟수가 다른 룰 타입들에 비해서 많다. 블룸 필터를 사용하면 기존의 방식보다 일치하는 룰을 찾기 위한 메모리 접근 횟수를 3번 적게 한다. FW의 경우 선형검색 때문에 블룸필터로 인한 상대

적인 검색성능의 향상은 크지 않음을 알 수 있다. IPC의 경우에는 1000개의 룰 셋에 대해서 블룸 필터를 적용했을 때 일치하는 룰을 찾기 위한 메모리 접근 횟수를 기존의 알고리즘보다 최대 5번 감소시킨다. IPC 타입의 룰들은 프리픽스의 값과 길이가 다양하기 때문에 접근하는 해시 테이블에 일치하는 노드가 없을 경우, 블룸 필터가 접근을 차단하는 기능을 더 잘 수행할 수 있기 때문이다.

제안하는 구조의 성능은 블룸 필터의 메모리 사이즈에 따라서도 달라진다. 일반적으로 블룸 필터는 메모리의 사이즈가 커질수록 음성의 비율이 늘어나고 거짓-양성의 비율이 줄어들기 때문에 결과적으로 메모리 접근 횟수가 줄어들게 된다. 그림 8은 ACL, FWIPC의 1000개의 룰에 대하여 블룸 필터의 크기를 기본크기(N')의 2배, 4배, 8배, 16배, 32배로 늘려가면서 입력 패킷 당 음성, 양성, 거짓-양성의 평균횟수에 대한 성능 평가를 한 결과를 나타내고 있다.

결과를 살펴보면 각 룰 타입이 공통적으로 블룸 필터의 크기를 증가시켰을 때 양성과 거짓 양성의 횟수는 줄어들게 되고 음성의 횟수는 증가하는 것을 알 수 있다. ACL의 경우 8N'의 크기를 가질 때 거짓-양성이 0이 되는 최적화된 성능을 얻을 수 있고 음성의 횟수는 인풋 당 1~3번 일어나는 것을 나타내고 있다. 그림 7에 보여졌던 평균 메모리 접근 횟수는 4.6이고 그림 8에 보여지는 양성의 수가 평균 3.9번 인 것으로 봐서 링크드 리스트로 인한 작은 개수의 선형접근이 있었음을 알 수 있다. FW의 경우에는 16N'일 때 거짓-양성이 0이 되는 결과를 얻을 수 있고 이 때 양성횟수가 평균 2.5번인 것에 반해 그림 7의 평균 메모리 접근 횟수는 97번으로 링크드 리스트에 의한 선형 검색이 매우 많이 일어남을 보여준다. 음성은 인풋 당 3번이 못 미치게 일어난다. IPC의 경우에는 16N'일 때 거짓-양성이 0이 되는 결과를 얻을 수 있고 이 때 음성이 평균 5번 일어난다.

V. 결 론

인터넷에서의 품질보장 요구가 많아짐에 따라 라우터에서의 패킷분류 기능이 점점 중요해지고 있다. 따라서 본 논문에서는 길이에 대한 이진검색으로 패킷 분류를 하는 알고리즘에 블룸 필터를 추가하여 검색 성능을 향상시켰다.

기존의 길이에 대한 이진검색에 블룸 필터를 적용하면 존재하지 않는 노드에 대한 해시 테이블의 접근을 제한하기 때문에 메모리 접근 횟수를 줄여 검색성능을 향상시킬 수 있다. 제안하는 구조는 룰 들의 네스팅 관계를 없애기 위하여 여러 개의 트라이를 만드는데, 클레스벤치를 사용한 실험 결과 최대 경우 6개, 평균적으로 5개 이하의 트라이가 만들어지고 블룸 필터는 각 만들어진 트라이에 독립적으로 적용한다. 제안하는 알고리즘에 대한 성능평가의 결과로는 실제 사용하는 룰들의 타입에 대해 블룸 필터를 적용했을 때 최대 33% 메모리 접근 횟수를 줄여 검색 성능을 향상시키는 것을 알 수 있었다.

참 고 문 헌

[1] H. Jonathan Chao, "Next Generation Routers," Proceedings of the IEEE, vol. 90, Iss. 9, pp.1518-1588, Sept. 2002.

[2] M. de Berg, M. Van Kreveld, M.Overmars, and O. Schwarzkopf, "Computational Geometry: Algorithms and Applications," Springer-Verlag, 2000.

[3] M. M. Buddhikot, S. Suri, and M.Waldvogel, "Space decomposition techniques for fast layer-4switching," in Proc. Conf. Protocols for High Speed Networks, pp.25-41 Aug.1999.

[4] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," Proc. ACM SIGCOMM, pp.25-35, 1997.

[5] Hyesook Lim and Ju Hyoung Mun, "High-Speed Packet Classification Using Binary Search on Length," IEEE/ACM ANCS, 2007.

[6] P. Gupta and N. Mckeown, "Algorithm for packet classification," IEEE Network, vol.15, no.2, pp.24-32, Mar./Apr. 2001.

[7] S. Dharmapurikar, H. Song, J. Turner, J. Lockwood,"Fast Packet Classification Using Bloom filters," in ANCS, 2006.

[8] Hyesook Lim, Min Young Kang, and Changhoon Yim, "Two-dimensional packet classification algorithm using a quad-tree," Computer Communications, Elsevier Science, vol.30, no.6, pp.1396-1405, Mar. 2007.

[9] Andrei Broder and Michael Mitzenmacher, "Network Applications of Bloom filters: A

Survey," Internet Mathematics, vol.I, no.4, pp.485-509, May, 2004.

[10] Soyeon Kim and Hyesook Lim, "Tuple Pruning Using Bloom Filters for Packet Classification," ITC-CSCC 2009.

[11] Raj Jain, "Comparison of Hashing Schemes for Address Lookup in Computer Networks," in IEEE Transactions on Communications, vol.40, no.10, pp.1570-1573, 1992.

[12] D. E. Taylor and J. S. Turner, "Classbench: A packet classification benchmark," in IEEE INFOCOM, 2005.

[13] D. E.Taylor, J. S. Turner. The Source Code of Packet Classification Bench, http://www.arl.wustl.edu/~det3/ClassBench.index.htm

최 영 주 (Youngju Choe) 준회원

2011년 2월 이화여자대학교 전자공학과 졸업(학사).

2011년 3월~현재 이화여자대학교 전자공학과(석사).

<관심분야> Router나 switch 등의 Network 관련 SoC설계, TCP/IP 관련 하드웨어 설계

임 혜 숙 (Hyesook Lim) 종신회원

1986년 서울대학교 제어계측공학과 졸업(학사).

1986년 8월~1986년 2월 삼성휴렛 팩커드 연구원.

1991년 서울대학교 제어계측공학과 졸업(석사).

1996년 The University of Texas at Austin, Electrical and Computer Engineering 졸업(박사). 1996년 11월~2000년 7월 Lucent Technologies Member of Technical Staff. 2000년 7월~2002년 2월 Cisco Systems, Hardware Engineer. 2002년 3월~이화여자대학교 공과대학 전자공학과 정교수. 관심분야는 라우터나 스위치 등의 네트워크 관련 알고리듬 및 SoC 설계, TCP/IP 관련 하드웨어 설계