패킷 분류를 위한 블룸 필터 이용 튜플 제거 알고리즘

(Tuple Pruning Using Bloom Filter for Packet Classification)

김 소 연 † 임 혜 숙 ††
(Soyeon Kim) (Hyesook Lim)

요 약 다양한 어플리케이션의 등장과 인터넷 사용자의 급속한 성장으로 인하여, 인터넷 라우터는 패킷이 입력되는 속도와 같은 속도로 패킷 분류작업을 수행하여 패킷의 클래스에 따른 품질 보장을 제공할 것이 요구되고 있다. 패킷 분류란 라우터에 입력된 패킷의 헤더가 가지고 있는 여러 개의 필드에 대해 다차원 검색을 수행하여, 미리 정의된 룰과 일치하는 결과 가운데 최우선순위를 갖는 룰을 찾아내는 과정을 말한다. 빠른 패킷 분류를 위하여 다양한 패킷 분류 알고리즘이 제안되어오고 있으며, 튜플 공간 제거(tuple space pruning) 알고리즘은 일치 가능한 룰을 갖는 튜플들만을 해싱을 사용하여 검색함으로 빠른 검색 성능을 제공한다. 블룸 필터(Bloom filter)는 특정 집합에 속하는 원소들의 멤버쉽에 관한 정보를 간단한 비트-벡터로 표현하는 데이터 구조로서, 특정 입력 값이 집합에 속한 원소인지를 알려주는 선-필터(pre-filter)로 사용된다. 본 논문에서는 블룸 필터를 이용하여 일치 가능성이 없는 튜플을 효율적으로 제거하는 새로운 튜플 제거 알고리즘을 제안한다. 실제 라우터에서 사용되는 룰 셋과 비슷한 특성을 갖는다고 알려진 데이터 베이스에 대한 성능 비교를 통하여, 본 논문에서 제안하는 구조가 패킷 분류 성능 및 메모리 사용량에 있어서 기존의 튜플공간 제거 알고리즘과 비교하여 월등히 우수함을 보았다.

키워드 : 패킷 분류, 튜플 제거, 블룸 필터, 해싱

Abstract Due to the emergence of new application programs and the fast growth of Internet users, Internet routers are required to provide the quality of services according to the class of input packets, which is identified by wire-speed packet classification. For a pre-defined rule set, by performing multi-dimensional search using various header fields of an input packet, packet classification determines the highest priority rule matching to the input packet. Efficient packet classification algorithms have been widely studied. Tuple pruning algorithm provides fast classification performance using hash-based search against the candidate tuples that may include matching rules. Bloom filter is an efficient data structure composed of a bit vector which represents the membership information of each element included in a given set. It is used as a pre-filter determining whether a specific input is a member of a set or not. This paper proposes new tuple pruning algorithms using Bloom filters, which effectively remove unnecessary tuples which do not include matching rules. Using the database known to be similar to actual rule sets used in Internet routers, simulation results show that the proposed tuple pruning algorithm provides faster packet classification as well as consumes smaller memory amount compared with the previous tuple pruning algorithm.

Key words : Packet classification, Tuple pruning, Bloom filter, Hashing

· This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (2010-0000483) and by the MKE(The Ministry of Knowledge Economy), Korea, under the HNRC-ITRC support program supervised by the NIPA (NIPA-2010-C1090-1011-0010)

† 학생회원 : 이화여자대학교 전자공학과
soykim@ewhain.net
†† 정 회 원 : 이화여자대학교 전자공학과 교수
hlim@ewha.ac.kr

논문접수 : 2009년 5월 14일
심사완료 : 2010년 2월 9일

정보과학회논문지: 정보통신 제37권 제3호(2010.6)

1. 서 론

인터넷 사용자 수의 급격한 증가뿐만 아니라, 다양한 어플리케이션의 등장과 이러한 어플리케이션에 대한 사용자들의 수요가 급격히 증가하고 있다. 이러한 어플리케이션들은 인터넷 프로토콜을 기반으로 하고 있으며, 서로 다른 품질 보장(quality of service, QoS) 수준을 요구한다. 그러므로 모든 패킷에 대하여 동일한 방식의 최선 지원(best-effort) 서비스를 제공하여 어플리케이션의 다양성에 따른 품질을 보장하지 못하던 기존의 라우터들은 필수불가결하게 다양한 레벨의 품질 보장 서비스를 제공할 수 있도록 진화하여야 한다. 다양한 어플리케이션에 대한 품질 보장을 위해서, 라우터는 입력 패킷의 헤더를 이용하여 입력 패킷이 속하는 클래스를 찾아, 각 클래스에 해당하는 적절한 서비스를 제공해 주어야 하는데, 입력 패킷이 속하는 클래스를 찾아주는 작업을 패킷 분류 (packet classification)라고 한다. 패킷 분류는 목적지 IP 주소, 근원지 IP 주소, 목적지 포트 넘버, 근원지 포트 넘버, 프로토콜 정보에 해당하는 5개의 필드를 사용하여 이루어진다. 패킷 분류를 위하여 라우터는 입력 패킷과 일치하는 룰들을 찾고, 그 중 최우선 순위를 갖는 룰을 찾아내는 다차원 검색을 수행하여야 한다[1].

패킷 분류는 다음과 같은 어려움을 가지고 있다. 첫째로, 각 필드에 따라 각각 다른 검색 방법을 사용하면서 이러한 다차원 검색을 동시에 진행하여 모든 필드에 대하여 일치하는 룰을 찾는 복잡한 검색을 요구한다는 점이다. 또한 검색을 통하여 일치하는 룰의 개수가 한 개 이상일 경우에, 일치하는 모든 룰을 검색하여 그 가운데서 최우선순위의 룰을 찾아내는 과정이 부가적으로 이루어져야 한다. 마지막으로 라우터는 모든 입력 패킷에 대하여 패킷이 입력되는 선속도(wire-speed)로 빠르게 처리해야 한다는 점에서 어려움을 가지고 있다[2].

패킷 분류에 사용되는 5개의 필드는 각각 완전 일치(exact match), 영역 일치(range match), 프리픽스 일치(prefix match)라는 각기 다른 방법을 이용하여 일치 여부를 판단하여야 하므로 검색이 복잡해지고 속도도 느려진다. 모든 필드를 검색하는 대신 일부 필드만 이용한 튜플(tuple) 구조를 정의하여 튜플 공간을 검색할 때 모든 필드를 검색 했을 경우와 성능이 동일하다면 튜플 공간을 이용하여 검색하는 것이 보다 효율적이며 이에 대한 연구가 활발히 진행되고 있다[3].

본 논문에서는 간단하면서 매우 효율적인 데이터 구조인 블룸 필터(Bloom filter)를 이용하여 메모리 접근 횟수와 메모리 요구량을 줄일 수 있는 튜플 제거 패킷 분류 알고리즘을 제안한다. 블룸 필터는 어떤 입력 값이 필터에 저장된 값들 중 일치하는 값이 있는지의 여부를 알려주는 간단한 비트-벡터(bit vector) 구조의 필터로서 불필요한 검색을 제한하여 보다 빠른 패킷 분류를 수행할 수 있도록 한다.

제안하는 첫 번째 알고리즘은 기존의 튜플 제거 알고리즘에 튜플 블룸 필터를 추가한 알고리즘이다. 기존 알고리즘에서와 같이 각 필드 별 검색으로 결정된 일치하는 프리픽스 길이들을 교차하여 튜플을 생성하면 프리픽스 값에 관한 정보는 사라지고 길이에 관한 정보만이 교차되므로, 불필요한 해시 메모리의 접근을 야기하는 튜플들이 남아있게 된다. 제안하는 알고리즘에서는 프리픽스 길이의 조합뿐만 아니라 프리픽스 값의 조합에 대한 정보를 가지는 튜플 블룸 필터를 통과시켜 불필요한 해시 테이블 검색을 제한한다.

제안하는 두 번째 알고리즘은 기존의 튜플 제거 알고리즘의 각 필드 별 검색과정을 프리픽스 정보를 저장한 블룸 필터와 해시 테이블로 대치한 알고리즘이다. 각 필드 별 검색과정에서, 먼저 블룸 필터는 길이에 대한 필터링 기능을 수행하는데, 주어진 인풋에 대하여 일치 가능성이 없는 길이 들을 제거한다. 블룸 필터를 통과한 일치 가능성이 있는 길이들에 대하여 해시 테이블에 접근하여 일치 길이를 확정한다.

본 논문의 구조는 다음과 같다. 먼저 2장에서는 기존의 패킷 분류 알고리즘과 튜플 공간을 이용한 패킷 분류 알고리즘 그리고 블룸 필터에 대하여 소개한다. 3장에서는 제안하는 알고리즘에 대해 설명하고, 4장에서는 제안한 알고리즘의 성능과 다른 패킷 분류 알고리즘과의 성능을 비교한다. 마지막으로 5장에서 간단한 결론을 맺는다.

2. 기존 연구

2.1 기존의 패킷 분류 알고리즘

패킷 분류를 효과적으로 하기 위해 다양한 알고리즘이 제안되고 있다. 그 가운데 근원지, 목적지 프리픽스를 이용한 트라이 구조가 많이 연구되고 있다. 일차원 프리픽스에 기초한 트라이 구조로 대표적인 계층 트라이[4]는 룰을 구성하는 필드들 중 근원지와 목적지 프리픽스 필드의 정보를 기반으로 트라이를 구성하고 근원지 프리픽스 트라이와 목적지 프리픽스 트라이를 계층적으로 연결한 구조로 가장 기본적인 구조이다. 상위 트라이의 검색 경로를 따라가다가 일치 가능한 노드에 도달하면 하위 트라이 포인터를 이용해 하위 트라이의 검색을 진행하고 일치하는 룰 정보가운데 최우선 룰을 찾아내는 구조이다. 계층 트라이는 하나의 필드 검색이 끝날 때마다 검색 영역이 확연히 줄어들기 때문에 매우 효율적인 구조이나 구조상 빈 노드의 존재가 불가피 하

기 때문에 불필요한 메모리 사용으로 검색 성능이 떨어지고 가장 높은 우선순위를 갖는 룰을 검색하기 위해 하위 트라이에서 상위 트라이로의 역추적으로 인해 메모리 접근 횟수가 증가하는 문제를 갖는다. 따라서 이 같은 역추적 문제를 해결하기 위해 셋프루닝(set-pruning) 트라이가 제안되었다[5]. 셋프루닝 트라이는 계층 트라이에서 역추적에 의한 검색의 비효율성을 제거함으로써 검색 성능을 높이기 위해 제안된 구조이다. 하지만 구조상 여전히 빈노드를 포함하고 있어 불필요한 메모리 접근으로 검색 성능이 저하되며 한 개의 룰이 여러 노드로 복사되기 때문에 메모리 요구량이 급격히 증가하는 문제점을 가지고 있다.

이차원 프리픽스에 기초한 트라이 구조 가운데 영역 분할 사분 트라이 (area-based quad trie, AQT)를 이용한 패킷 분류 방식은 근원지와 목적지 프리픽스 필드를 결합하여 만든 코드워드(code word)로 구성된 2차원 트라이를 이용한 패킷 분류 알고리즘이다[6]. 코드워드는 두 필드 중 길이가 짧은 필드의 프리픽스 길이만큼 생성되는데, 두 필드의 MSB(most significant bit) 부터 한 비트씩 조합하여 생성하고 해당하는 노드에 룰을 저장한다. 입력된 패킷에 대하여도 두 필드 프리픽스의 MSB부터 코드워드를 생성하여 일치하는 룰을 찾는 구조이다. AQT 구조는 코드워드가 길어 질수록 내부에 비어있는 노드로 인한 메모리 효율성이 떨어지므로 이를 해결하기 위해 우선순위에 기초한 사분 트리(priority-based quad trie, PQT)가 제안되었다[7]. PQT는 AQT를 생성 후 모든 노드에 우선 순위 정보를 저장하여 기존의 AQT 구조에 빈 노드가 존재한다면 그 노드의 서브 트라이 중 가장 우선 순위가 높은 룰을 빈 노드에 저장하고 원래의 노드는 제거하여 메모리의 효율성을 높인다.

2.2 튜플 공간 검색 알고리즘(Tuple Space Search Algorithm, TSS)

튜플 공간 검색을 기반으로 한 알고리즘은 기존의 알고리즘과 달리 튜플 공간을 검색하는 새로운 개념의 패킷 분류 알고리즘이다. 튜플 공간을 각 필드 원소의 비트 길이들의 조합으로 정의하고 이를 해쉬 키(hash key)로 이용하여 저장한다. 표 1에 근원지 프리픽스와 목적지 프리픽스로 이루어진 룰 셋의 예를 보였다. 본 논문에서는 표 1에 나타난 룰셋을 예로 알고리즘을 설명한다.

튜플 공간은 근원지 프리픽스와 목적지 프리픽스의 길이 조합을 이용하여 2차원 공간을 정의하고, 이를 (i, j)로 나타낸다. 여기서 i는 근원지 프리픽스 길이, j는 목적지 프리픽스 길이를 나타낸다. 두 필드만 이용하는 이유는 이 조합으로 이루어진 튜플 공간이 다른 필드의 조합보다 한 튜플에 포함되는 룰의 개수가 적어 룰들을

표 1 룰 셋의 예

Rule No.	Source prefix	Destination prefix	Tuple
R1	1*	1*	(1, 1)
R2	00*	11*	(2, 2)
R3	0*	100*	(1, 3)
R4	111*	0*	(3, 1)
R5	011*	111*	(3, 3)

다양하게 분포시킬 수 있기 때문이다. 프리픽스의 길이들의 조합으로 정의된 튜플은 33×33=1089 개의 튜플을 만들어 낼 수 있어, 최악의 경우 1089 개의 튜플을 모두 검색해야 하지만 대부분의 경우 일부 튜플에 룰 정보가 몰려 있으므로 룰이 존재하는 튜플만을 검색하면 된다. 튜플 공간을 효율적으로 검색하기 위하여 룰이 존재하는 튜플 중 주어진 입력과 일치하는 룰이 없는 튜플을 제거하는 튜플 제거(tuple pruning) 검색 방식으로 연구되고 있다.

튜플 제거 알고리즘에서는 각 필드에 대하여 하나의 프리픽스가 네스팅하고 있는 더 짧은 프리픽스의 길이들을 미리 계산하여 저장한다. 예를 들어 표 1의 목적지 프리픽스의 예에서 프리픽스 100*에는 길이 1이 미리 선 계산되어 저장된다. 이는 프리픽스 100*에는 길이가 1인 프리픽스인 1*이 네스팅되어 있다는 의미이다. 또한 룰 셋에 존재하는 근원지 프리픽스와 목적지 프리픽스가 각각 가지고 있는 길이에 관한 튜플 리스트를 미리 작성한다. 표 1의 예에서의 튜플 리스트를 표 1의 마지막 열에 보였다. 검색과정을 살펴보면 입력으로 들어온 패킷의 근원지 주소에 대하여 검색을 진행하여 최장길이 일치 프리픽스를 찾는다. 목적지 주소에 대해서도 동일한 과정을 병렬로 진행한다. 위의 과정에서 검색된 최장길이 프리픽스들에 대하여, 이 프리픽스에 네스팅되어 있는 길이들이 선계산되어 저장되어 있으므로, 네스팅되어 있는 길이들을 서로 교차하여 튜플을 생성한다. 이 튜플들 중 미리 작성된 튜플 리스트에 속하는 튜플들만을 뽑아내면, 패킷 분류를 위하여 검색되어야 할 튜플 공간이 결정한다. 교차하는 과정을 크로스-프로덕팅(cross-producting)이라 지칭한다. 크로스-프로덕팅 후 남은 튜플들에 대하여 선형검색을 적용하여 입력된 패킷의 헤더와 일치하는 룰을 찾아낸다.

아래 그림 1은 표 1을 이용하여 패킷 분류를 위한 튜플 제거 알고리즘의 구조를 보여주고 있다. 실제로 라우터에 입력되는 근원지 프리픽스와 목적지 프리픽스의 길이는 32이지만, 본 논문의 예에서는 프리픽스 최대 길이가 8이라고 가정하였다. 입력 패킷이 (11101101, 00011001)란 튜플 정보를 가지고 있다면 원래의 튜플 공간 검색 구조에서는 존재하는 모든 튜플에 접근하여 최우선으로 일치하는 룰 정보를 찾아낸다. 튜플 제거 알고리즘을 적

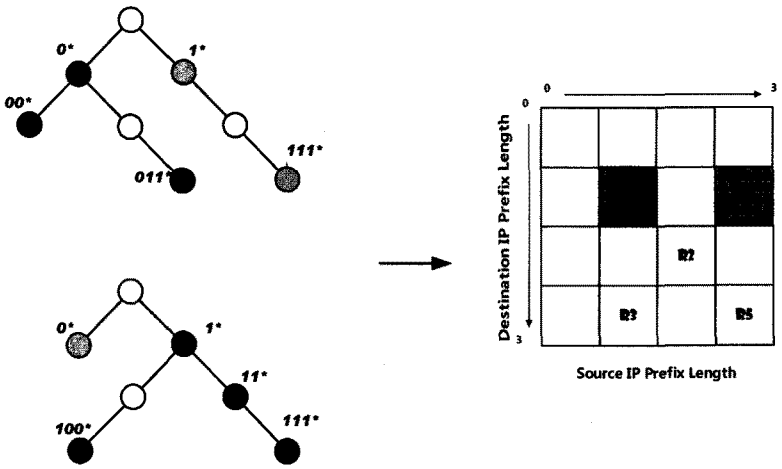

그림 1 튜플 제거 알고리즘의 구조

용하게 되면 각 필드 검색을 통하여 근원지 프리픽스 필드에서 길이 1과 3, 목적지 프리픽스 필드에서 길이 1에서 일치함을 얻어내고, 이 정보를 교차하여 (1,1), (3,1)에 해당하는 튜플만을 접근하여 일치하는 값을 찾아낸다.

2.3 블룸 필터 이론(Bloom filter theory)

블룸 필터는 필터에 저장된 정보와 입력 값을 비교하여 일치하는 값이 존재하지 여부를 간단히 알려주는 선-필터(pre-filter)로서 비트-벡터로 표현되는 간단하고 공간 효율적인 데이터 구조이다[8]. 블룸 필터는 초기에는 주로 데이터 어플리케이션 영역에서 사용되었지만 최근에 와서는 네트워크영역에서의 적용이 활발히 연구되고 있다[9].

블룸 필터의 동작은 데이터를 저장하는 프로그래밍(programming) 과정과 입력 값과 필터에 저장된 값을 비교하는 쿼리(query) 과정으로 나뉜다. 표 1에 나타난 목적지 주소 필드를 이용하여 블룸 필터의 동작을 설명하면 다음과 같다. 각 블룸 필터는 하나의 데이터 집합 밖에 표현하지 못하기 때문에 데이터 집합의 종류가 다수일 경우 모든 집합의 정보를 저장하기 위해서는 집합의 수만큼의 블룸 필터가 필요하다. 표 1의 목적지 주소 필드에는 길이가 1,2,3인 프리픽스들이 존재하므로 각 목적지 프리픽스의 길이 집합을 표현하기 위해서 3개의 블룸 필터가 필요하다. 프로그래밍에 앞서 이 예제에서는 각각의 블룸 필터의 크기를 4-비트라고 정의하고 모든 비트 값을 0으로 초기화시킨다.

블룸 필터의 프로그래밍 과정은 데이터 집합에 존재하는 각 원소들을 k개의 해싱 함수를 이용하여 블룸 필터 프로그래밍에 필요한 k개의 색인 값을 얻고, 해싱 색인에 대응되는 블룸 필터의 비트-벡터 값을 1로 프로그래밍한다. 예를 들어 요소 x를 블룸 필터에 프로그래밍하는 과정을 의사 코드(pseudo code)로 나타내면 다음과 같다.

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

예제에서는 임의의 해싱함수로 얻어진 두 개의 색인 값을 이용하여 각 블룸 필터를 프로그래밍 해 보았다. 예를 들어 각 프리픽스가 해싱함수를 통과하여 표 2에서 보인 것과 같은 인덱스를 얻었다고 할 때, 이러한 블룸 필터 인덱스를 사용하여 프로그래밍한 블룸 필터를 그림 2에 나타내었다.

쿼리(query) 과정은 입력된 값에 대하여 프로그래밍 시 사용한 동일한 과정을 이용하여 블룸 필터에 저장된 값과 일치하는 것이 있는지 찾아내는 과정이다. 입력 값

표 2 목적지 프리픽스의 CRC-8 출력 결과와 블룸 필터 색인

Len	Prefixes	BF indices
1	0*	1, 2
	1*	3, 3
2	11*	1, 2
3	100*	3, 3
	111*	2, 2

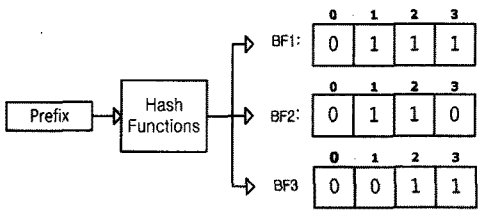

그림 2 목적지 프리픽스에 대하여 프로그래밍 된 블룸 필터

에 대한 해싱 색인 값을 이용하여 블룸 필터에 대응되는 비트-벡터의 위치로 가서 그 값이 0값을 갖는지 1값을 갖는지를 확인한다. 블룸 필터는 모든 색인 값들이 가리키는 비트의 값들이 모두 1이면 일치하는 정보가 존재한다고 선언(양성)하고 0이 하나라도 있으면 일치하지 않는다고 선언(음성)한다. 이는 0인 비트는 집합 내의 어떠한 원소 값에 의해서도 프로그래밍이 되지 않았다는 것을 의미하기 때문이다. 입력 x에 각 블룸 필터에 대한 쿼리의 과정을 의사 코드로 표현하면 다음과 같다.

BFQuery (x)

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

입력된 패킷의 목적지 주소 필드 값으로 "00010111"이란 주소 값이 들어오면 프로그래밍 과정에서 사용했던 동일한 해싱 함수들을 이용하여 해싱 색인 값을 출력한다. 예에서는 블룸 필터는 목적지 프리픽스 필드에 존재하는 길이 1,2,3에 해당하는 3개의 블룸 필터가 존재한다. 그러므로 주어진 입력 프리픽스에서 길이에 해당하는 '0', '00', '000'을 각각의 블룸 필터의 해싱 함수에 넣어 해싱 색인을 출력한 후 모든 색인 값들이 가리키는 비트의 값들을 확인한다. 길이 1인 블룸 필터의 쿼리 과정을 보면 '0'의 해싱 색인 값은 1과 2이고 블룸 필터의 해당 비트의 값을 확인하면 모두 1이므로 이 경우 양성의 결과가 나온다. 길이가 2인 블룸 필터의 경우에는 해싱 색인 값이 0과 3이므로 해당하는 비트를 확인해 보았을 때, 비트0이 0이므로 음성 값을 가진다. 길이가 3인 경우, 색인 값이 2, 3이고 양성 결과 값을 가지게 된다. 하지만 이 경우에는 '000'에 대하여 일치하는 값이 존재하지 않지만 존재한다고 선언 했으므로 이런 경우는 거짓-양성(false-positive) 오류의 경우에 속한다.

거짓-양성의 오류는 블룸 필터의 주요한 특성 중 하나이다. 블룸 필터에서 거짓-양성의 오류가 나타나는 이유는 프로그래밍 할 때 k개 해싱 함수를 모든 집합 원소에 적용하여 각 k개의 비트들을 0에서 1로 프로그래밍을 하므로 1로 변경된 비트는 하나의 원소에 의해서가 아닌 집합의 여러 개의 다른 원소 값에 의해서도 1로 프로그래밍 될 수 있기 때문이다. 이는 역으로 블룸 필터에 거짓-음성(false negative)이 존재하지 않는 이유를 설명해준다. 블룸 필터의 거짓-양성은 다음의 식 (1)을 통하여 계산할 수 있다.

$$F = (1 - e^{(nk/m)})^k \quad (1)$$

식에 근거하여 해싱 함수의 개수 k, 블룸 필터 크기 m, 집합 원소의 개수 n을 적절히 조절하면 최적의 (optimal) 거짓-양성의 비율을 구할 수 있다는 것을 알 수 있다. 집합 원소의 개수 n과 블룸 필터 크기 m에 대해 거짓-양성의 비율을 최소화하는 최적의 해싱 함수의 수는 다음의 식에 의하여 주어짐이 연구된 바 있다[9].

$$k = (m/n)\ln 2 \quad (2)$$

2.4 통합 블룸 필터(all-length Bloom filter)

블룸 필터는 하나의 집합에 포함된 원소를 표현하는 데이터 구조이므로 여러 개의 집합의 데이터를 표현하기 위해서는 각각의 집합에 해당하는 블룸 필터를 각각 할당해야 한다. 만약에 블룸 필터를 이용하여 최장 길이 프리픽스 일치 검색(longest prefix match)을 한다면 라우팅 테이블에 포함된 특정 프리픽스의 길이마다 블룸 필터를 개별적으로 할당하여 해당 길이에 포함되는 프리픽스들의 정보를 저장한다. 이 경우에 여러 개의 블룸 필터를 각각 프로그래밍, 쿼리 작업을 수행하여야 하는 복잡성을 가지고 있다. 또한 프리픽스들이 길이에 따라 비슷한 개수로 고르게 분포하지 않는다면 블룸 필터들의 크기들이 불균형하며, 다수의 블룸 필터가 존재하므로 거짓-양성을 조절하는 것이 또한 쉽지 않다. 이를 보안하기 위하여 하나의 블룸 필터를 이용하여 여러 개의 프리픽스 길이의 집합을 표현할 수 있는 통합 블룸 필터가 제안되었다[10].

통합 블룸 필터는 일반 블룸 필터와 동작이 비슷하지만 일반 블룸 필터와 달리 룰셋 내의 프리픽스 길이 집합들을 구분하지 않고 모든 프리픽스에 동일한 해싱 함수를 적용하여 해싱 결과값에 해당하는 비트-벡터를 1로 프로그래밍하고 프로그래밍 때 사용된 프리픽스의 길이를 별도로 다른 공간에 저장한다. 그림 3에 통합 블룸 필터의 구조를 보였다. 쿼리 과정은 프리픽스가 입력되면 필터에 별도로 저장된 길이들의 집합에 존재하는 길이만큼 프리픽스를 추출하여 해싱 색인을 얻고 해당하는 비트-벡터 위치의 값을 확인하여 해당길이에 대하여 양성이나 음성의 결과 값을 얻는다. 이 과정은 저장된 프리픽스 길이의 집합의 모든 원소에 대하여 반복하여 입력된 주소가 어떤 길이의 프리픽스와 일치하는지 알 수 있다. 이를 통해 통합 블룸 필터를 이용해도 일반적인 블룸 필터를 사용했을 경우와 동일한 결과를 얻을 수 있음을 알 수 있다.

예제에서는 CRC-8을 해싱함수로 이용해 통합 블룸 필터 두 개의 색인 값을 얻었고 그 결과를 표 3에 정리하였다. CRC 생성기는 이진(binary) 구조에서 구현이 쉽고, 길이가 일정하지 않은 비트열 입력 값을 미리 정해놓은 함수로 랜덤하게 잘 섞은 후, 특정 길이의 다양한 난수 값을 출력할 수 있다. CRC-8의 경우에는 입력 값의 길이에 상관 없이 출력 값이 모두 8비트의 난수를 가지게 하므로 통합 블룸 필터의 해싱 함수로 사용되기에 적합하다. 그림 3에서는 해싱 함수 결과로 얻어진 색인 값으로 프로그래밍된 통합 블룸 필터를 나타내었다.

표 3 목적지 프리픽스의 CRC-8 출력 결과와 블룸 필터 색인

Len	Prefixes	CRC-8 code	BF indices
1	0*	01111110	1, 2
	1*	11001111	3, 3
2	11*	01010110	1, 2
3	100*	11110011	3, 3
	111*	10011010	2, 2

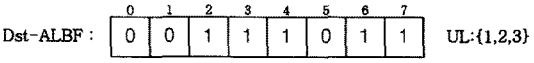

그림 3 목적지 프리픽스에 대하여 통합 블룸 필터 프로그래밍 결과

3. 블룸 필터를 이용한 튜플 제거 패킷 분류 알고리즘

본 논문에서는 블룸 필터를 이용하여 보다 효율적으로 검색에 불필요한 튜플들을 제거하는 패킷 분류 알고리즘을 제안한다. 제안하는 첫 번째 알고리즘은 기존의 튜플 제거 알고리즘에 튜플 블룸 필터를 추가한 알고리즘이다. 제안하는 두 번째 알고리즘은 기존의 튜플 제거 알고리즘의 각 필드 별 검색과정을 프리픽스 정보를 저장한 블룸 필터와 해시 테이블로 대치한 알고리즘이다.

3.1 제안하는 첫 번째 알고리즘

제안하는 첫 번째 알고리즘은 기존의 튜플공간 검색을 위한 튜플 프루닝 알고리즘에 해시 테이블 접근하기 전 튜플 블룸 필터를 적용하여 검색에 불필요한 튜플의 메모리 접근을 제한하는 구조이다. 기존의 튜플 프루닝 알고리즘은 각 필드의 최장길이 검색 후 얻어진 길이정보를 조합해 튜플을 생성하고 룰셋에 존재하지 않는 튜플들은 제거한 후 해시 테이블을 검색한다. 제안된 첫 번째 알고리즘은 기존의 튜플 제거 알고리즘에서 튜플을 구성하는 길이의 정보만을 이용하여 크로스-프로덕팅 후 남겨진 튜플들을 바로 해시 테이블에 접근하는 것이 아니라 크로스-프로덕팅 단계 후 튜플의 길이와 튜플을 구성하는 프리픽스의 값 정보를 이용하여 튜플 블룸 필터 검색을 수행한다. 적은 메모리 공간을 차지하는 비트-벡터로 이루어진 튜플 블룸 필터를 이용하여 길이정보뿐만 아니라 튜플의 값에 관한 정보까지 확인한 후 외부 해시 테이블 검색이 필요없는 튜플들을 제한할 수 있어 검색 속도를 향상시킬 수 있는 장점을 가진다.

3.1.1 프로그래밍 과정

표 1의 룰 셋에 대하여 프로그래밍된 제안하는 첫 번째 튜플 제거 패킷 분류 알고리즘을 그림 4에 보였다. 기존의 튜플 공간을 이용한 패킷 분류 알고리즘에 추가된 튜플 블룸 필터의 프로그래밍 과정은 다음과 같다. 튜플 블룸 필터를 프로그래밍 하기 위하여 CRC-16 생성기를 사용하였다. 각 근원지 프리픽스의 길이와 목적지 프리픽스 길이에 해당하는 프리픽스들을 추출하여 CRC-16 생성기를 통과시켜 튜플 블룸 필터의 색인 값을 얻어낸다. 튜플 블룸 필터의 색인 값에 해당하는 비트-벡터 값을 1로 프로그래밍을 한다.

해시 테이블은 튜플 블룸 필터와 같이 CRC-16 생성기를 이용하여 해시 테이블 색인 값을 얻어낸다. 해시 테이블 색인 값이 가르치는 곳에 해당하는 해시 테이블 위치에 입력 값의 모든 정보, 즉 룰 넘버, 근원지 프리픽스, 목적지 프리픽스, 근원지 프리픽스 길이, 목적지 프리픽스 길이, 프로토콜 타입, 근원지, 목적지 포트 범위 등을 저장하고, 튜플의 길이 종류에 대한 정보를 별도의 리스트에 저장한다. 표 4는 제안하는 첫 번째 알고리즘에서 사용되는 블룸 필터와 해시 테이블 색인 값을 정리하였다.

3.1.2 검색 과정

제안된 구조1의 검색 과정은 다음과 같다. 라우터에 패킷이 들어오면 주어진 입력에 대해 각 근원지 프리픽스 필드와 목적지 프리픽스 필드를 병렬로 최장길이 검색을 진행한다. 각 필드의 검색 결과를 크로스-프로덕팅 단계를 통과 후 남겨진 튜플들에 대하여 제안된 구조에서는 기존의 튜플 제거 알고리즘에 튜플 블룸 필터 검색을 추가적으로 진행하게 된다. 프로그래밍과정과 동일하게 튜플의 각 필드 길이에 해당하는 길이만큼의 프리픽스들을 이용하여 CRC-16에 입력하여 출력된 블룸 필터 색인 값을 이용하여 양성 결과의 튜플들을 구한다. 필터의 양성 결과의 튜플들에 한하여 외부 해시 테이블에 접근하여 해싱 색인이 가리키는 곳으로 접근하여 입력된 패킷 값과 비교하여 일치하는 최우선순위 룰을 찾아낸다.

제안된 첫 번째 알고리즘에 대한 검색의 예는 다음과

표 4 제안하는 알고리즘1에서 블룸 필터와 해시 테이블 색인 값

Rule	Tuple	CRC input	CRC code	Tuple-BF indices	Hash Table Index
R1	(1, 1)	11	0111100100110010	7, 2	2
R2	(2, 2)	0011	1000100111000010	8, 2	2
R3	(1, 3)	0100	1010111011001101	10,13	5
R4	(3, 1)	1110	0101110001001000	5, 8	0
R5	(3, 3)	011111	0111110000111101	7, 13	5

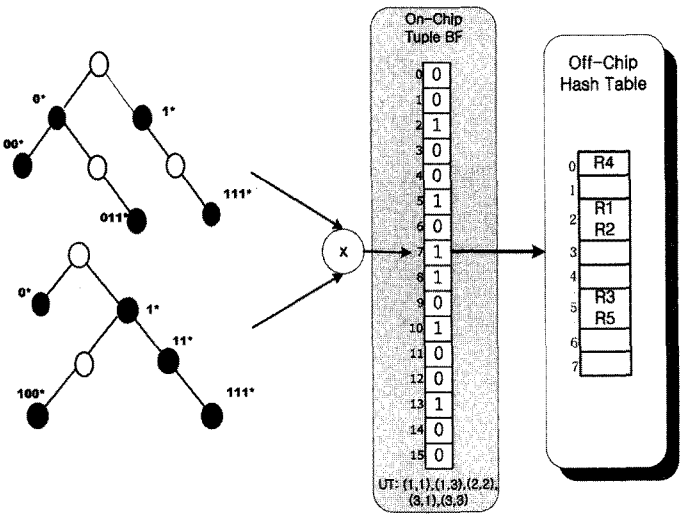

그림 4 제안하는 첫 번째 알고리즘의 구조 및 프로그래밍결과

같다. 표 1의 룰 셋에 대한 검색에서 (11101101, 00011001)란 패킷이 입력했다면 각 필드의 최장 길이 검색 결과들을 교차하면 (1,1), (3,1)의 튜플을 만들어 낼 수 있다. 해싱 함수의 입력으로 각 튜플의 근원지 프리픽스에 목적지 프리픽스를 연결한 값인 10, 1110를 이용한다. (1, 1)의 경우 CRC-16 결과 값으로 얻어 블룸 필터의 색인 값과 튜플 블룸 필터의 비트-벡터 값을 비교하면 색인이 가리키는 11,15 모두 '0'값을 가지므로 음성 결과를 가진다. (3, 1)의 경우 색인 5, 8 모두 '1'이므로 양성의 결과 값을 가진다. (3, 1)의 해시 테이블 색인 0에 해당하는 해시 테이블의 위치로 가서 입력 패킷 값과 엔트리에 저장된 값을 비교하면 입력 값과 일치하는 것을 확인 할 수 있고, 이때의 일치 결과는 R4이다.

메모리 접근 횟수를 계산하여 보면, 앞서 2.2에서 보인 기존의 튜플 제거 알고리즘을 사용한 경우 각 필드에서의 최장길이 일치 프리픽스를 검색하기 위한 메모리 접근 횟수에 더하여 추가적으로 튜플 (1, 1)과 (3, 1)에 대하여 해시 테이블에 접근이 이루어져야 한다. 그러나 제안하는 첫 번째 알고리즘은 간단한 튜플 블룸 필터를 사용하여 튜플 (1, 1)은 거짓 튜플 (false tuple)로 판단하고, 이 튜플의 해시 테이블로의 접근을 차단하여 해시 테이블로의 메모리 접근 횟수를 1회 줄인 것을 알 수 있다.

3.2 제안하는 두 번째 알고리즘

제안하는 두 번째 알고리즘은 그림 5에 보인 튜플 공간을 이용한 패킷 분류 알고리즘의 최장길이 일치 검색 과정에 블룸 필터와 해시 테이블을 적용하는 알고리즘이다. 기존의 튜플 제거 알고리즘을 위해서는 근원지, 목적지 프리픽스 필드에서 최장 길이 검색을 통하여 입력 값에 대하여 일치하는 모든 길이 정보를 검색해야 한다. 제안하는 두 번째 알고리즘에서는 근원지, 목적지 프리픽스 필드의 검색 시 발생하는 메모리 접근 횟수를 줄이기 위하여 각 필드의 최장길이 일치 과정을 블룸 필터와 해시테이블을 사용하여 대치하였다. 블룸 필터를 이용하여 음성 결과값에 대해서는 해시테이블을 접근하지 않으므로 메모리 접근 횟수를 감소시킬 수 있다. 또한 블룸 필터를 적용함으로써 존재하는 모든 프리픽스들에 대하여 네스팅 관계를 미리 선계산하여 저장할 필요가 없다는 장점을 가진다.

3.2.1 프로그래밍 과정

통합 블룸 필터를 사용한 프로그래밍 과정을 표 1의 예를 이용하여 설명한다. 프리픽스 필드의 블룸 필터를 프로그래밍하기 위하여 CRC-8을 해싱 함수로 사용하였다. 근원지 프리픽스 필드의 경우 표 1의 룰셋에 나타난 프리픽스들을 CRC-8 생성기에 입력하여 8-비트 코드 값을 얻어 낸다. CRC-8의 출력 값에서 블룸 필터 색인 값은 CRC 코드의 처음 3-비트와 마지막 3-비트를 사용하고 프리픽스 필드 해시 테이블 색인 값은 마지막 3-비트를 사용하였다. 표 5에는 각 근원지, 목적지 프리픽스에 대한 CRC 결과값과 블룸 필터, 해시 색인 값을 보였다. 이 색인 값들을 이용하여 블룸 필터와 해시 테이블 프로그래밍 결과를 그림 5에 보였다.

3.2.1 검색 과정

검색과정은 입력된 패킷에 대하여 근원지 주소필드와 목적지 주소 필드의 정보를 각각의 블룸 필터를 병렬적으로 통과시켜 블룸 필터에서 일치 가능성이 있는 양성 값의 길이 정보를 얻어낸 후 해시 테이블 검색을 수행하여 참-양성 길이 정보만을 얻어낸다. 각 필드의 참-

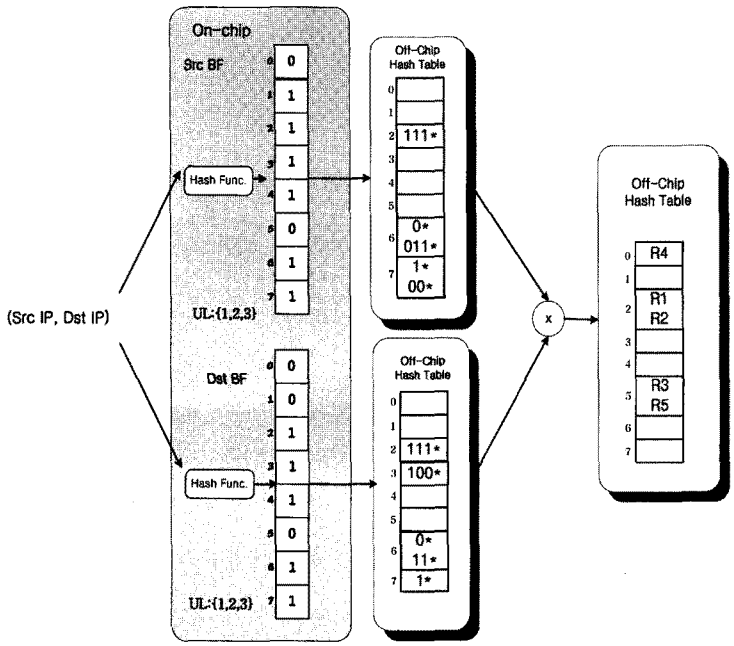

그림 5 제안하는 두 번째 알고리즘 구조 및 프로그래밍 결과

표 5 제안하는 두 번째 알고리즘에서 각 근원지, 목적지 필드 블룸 필터와 해시 테이블 색인 값

Distinct prefixes		CRC-8 code	BF indices	Hash Table Index
Source Field	1*	11001111	6, 7	7
	00*	00111111	1, 7	7
	0*	01111110	3, 6	6
	011*	11110110	7, 6	6
	111*	10011010	4, 2	2
Destination Field	1*	11001111	6, 7	7
	11*	01010110	2, 6	6
	100*	11110011	7, 3	3
	0*	01111110	3, 6	6
	111*	10011010	4, 2	2

양성 값을 이용하여 크로스 프로덕팅 단계를 거친 후 남은 튜플을 이용하여 해시 테이블 검색을 수행하여 최우선 순위의 룰을 찾아낸다.

검색과정은 예는 다음과 같다. 근원지 프리픽스와 목적지 프리픽스가 (11101101, 00011001)란 정보를 가지는 패킷이 입력으로 들어온다면, 각각의 프리픽스에 대하여 근원지 블룸 필터와 목적지 블룸 필터는 동시에 병렬로 검색을 수행한다. 근원지 블룸 필터의 경우, 미리 저장된 프리픽스의 특정 길이는 {3,2,1}이므로 길이에 대하여 순차적 검색한다. 이 경우 모든 길이에서 양성 결과를 얻게 되고, 양성결과 값에 대하여 해시 테이블 검색을 진행하여 길이 1과 3 만이 참-양성 결과값임을 확인할 수 있다. 목적지 필드의 경우 블룸 필터의 양성 값인 길이 1과 3에 대하여 해시 테이블 검색을 진행하여 길이 1 만이 참-양성 결과 값임을 알 수 있다.

각각의 근원지와 목적지 프리픽스 블룸 필터와 해시 테이블을 통과하여 얻어진 근원지, 목적지 프리픽스의 참-양성의 결과 값을 가지고 크로스-프로덕팅을 진행하고 남은 튜플들에만 해시 테이블 검색을 수행한다. 예제에서 (1, 1), (3, 1) 튜플이 생성되고 튜플 리스트에 존재하는 값이므로 CRC-16을 통과시켜 얻어진 해싱 색인을 가지고 해시 테이블에 접근하면 입력된 패킷과 R4와 일치한다는 것을 알 수 있다. 이때의 메모리 접근 횟수는 각 프리픽스 필드에서 각각 3회, 2회 수행하였고 마지막 튜플에 대하여 2번 접근하였으므로 총 7회의 메모리 접근을 하였다. 제안하는 구조 1과 2를 결합한 구조도 가능하나 이는 새로운 구조는 아니므로 본 논문에서는 이에 대한 실험은 생략하였다.

4. 시뮬레이션 결과

4.1 룰셋의 특성

본 논문에서는 실제 백 본 라우터에서 사용되는 다양한 크기의 라우팅 테이블 가운데 5000개 룰을 갖는 ACL (access control list), FW(firewall), IPC(IP chain) 셋을 사용하여 성능을 실험하였다[11,12]. 각 룰셋은 서로 다른 특징을 가지고 있다. 표 6은 룰셋에 저장되어 있는 룰의 개수, 유니크 프리픽스의 개수, 유니크 길이 정보

표 6 룰 셋의 유니크한 프리픽스 길이, 튜플의 개수 정보

Rule Type	No. of Rules	Source		Destination		Tuple	
		prefix	Len	prefix	Len	prefix	Len
ACL	4660	641	14	898	30	2443	102
IPC	4468	121	33	446	33	2876	680
FW	4351	64	33	120	33	1144	579

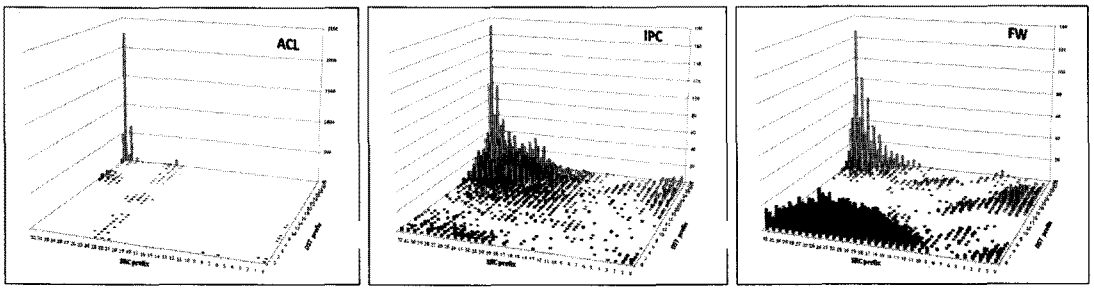

그림 6 ACL, IPC, FW의 프리픽스와 튜플 분포

의 개수, 유니크 튜플의 개수 정보를 나타내고 있다. 그림 6은 각 룰셋의 프리픽스 분포를 나타내었다.

그림 6에 룰 셋 별로 프리픽스 길이에 따른 룰의 개수에 관한 분포 특성을 보였다. 각 룰셋의 특성을 살펴보면 ACL의 경우 근원지 프리픽스의 길이의 종류가 14개로 다양하지 않고 대부분의 프리픽스들이 길이 32에 몰려있는 특성을 가진다. 목적지 프리픽스의 경우도 30개의 길이 종류를 가지지만 대부분의 프리픽스들은 길이 32에 몰려있는 특성을 가진다. 두 필드의 프리픽스의 종류를 보면 길이의 종류가 적은 것에 반해 다양한 유니크 프리픽스들이 존재하는 것을 볼 수 있다. ACL의 튜플의 분포 특성은 길이 (32, 32) 튜플에 50% 이상의 룰이 모여있다.

IPC와 FW의 경우에는 프리픽스의 모든 길이가 룰 셋에 나타나는 것을 알 수 있다. 프리픽스들의 길이 분포를 보면 길이가 짧은 길이에서 프리픽스들이 많이 나타나고 FW같은 경우 목적지 프리픽스 필드에 길이 0인 프리픽스들이 특히 많은 편이다. IPC, FW의 경우에는 ACL에 비해 다양한 튜플들이 존재하고 다양하게 분포되어있지만 길이가 8이하의 짧은 길이나 24이상의 긴 길이로 조합된 튜플들에 많이 몰려있는 것을 확인할 수 있다.

서로 다른 룰 셋의 특성 때문에 제안하는 구조의 블룸 필터와 해시 테이블 메모리의 크기, 블룸 필터의 검색 성능이 조금씩 차이를 나타낸다. 그 이유는 길이 블룸 필터 검색 시 길이가 짧은 프리픽스가 많으면 한 개의 입력 값에 대하여 양성의 결과 개수가 증가하기 때문이다.

4.2 제안하는 첫 번째 알고리즘의 성능평가

제안된 첫 번째 알고리즘은 기존의 튜플 공간 검색 알고리즘에 튜플 블룸 필터를 추가한 구조이므로 튜플 블룸 필터의 성능이 제안하는 구조의 성능에 가장 큰 영향을 끼치게 된다. 제안된 알고리즘은 블룸 필터를 이용하여 음성 결과 값에 대해서는 검색을 더 이상 진행하지 않아 검색에 불필요한 값들을 제거하여 메모리 접근 횟수를 줄이는데 주 목적을 가지고 있다. 여기서 음성 결과를 갖는 튜플이란 앞서 예에서 보인 false tuple인 경우로서, 룰셋에 존재하는 튜플이지만, 주어진 입력의 프리픽스 조합과는 상관이 없는 튜플로서 제안하는 튜플 블룸 필터에 의하여 걸러지는 튜플을 말한다. 또한 블룸 필터의 양성의 결과 값에 대해서는 검색을 계속 진행하므로 양성 결과 가운데 거짓-양성의 비율이 작을수록 불필요한 검색을 제한 할 수 있다. 여기서 거짓-양성의 결과를 갖는 튜플이란 룰셋에 존재하는 튜플이지만, 주어진 입력의 프리픽스 조합과는 상관이 없는 튜플이나, 블룸 필터 고유의 특성으로 인하여 걸러지지 않은 튜플을 말한다. 그러므로 블룸 필터의 음성 비율과 거짓-양성 비율은 블룸 필터의 성능을 평가하는데 주요한 요인이 된다.

그림 7에 보여준 실험 결과를 살펴보면 블룸 필터의 이론에서 볼 수 있듯이 튜플 블룸 필터의 크기가 커짐에 따라 음성의 결과가 증가하고 거짓 양성이 줄어듦으로, 양성의 개수가 줄어듦을 볼 수 있다. 그림 7에는 전체 튜플 블룸 필터의 입력 값에 대한 양성과 음성 결과 값의 비율과 양성 결과 값에 대한 거짓 양성의 비율을 나타내었다. 제안된 첫 번째 알고리즘에서는 튜플 블룸 필터의 양성결과 값에 대해서만 해시 테이블 검색을 수행하기 때문에 필터 사이즈의 크기가 증가함에 따라 양성 결과 개수가 줄어들어 해시 테이블 검색 횟수가 줄어드는 것을 알 수 있다. 여기서 블룸 필터의 크기를 나타내는 N'은 유니크 튜플 개수보다 크거나 같으면서 가장 작은 2의 제곱수를 나타낸다. ACL의 경우 룰 셋에

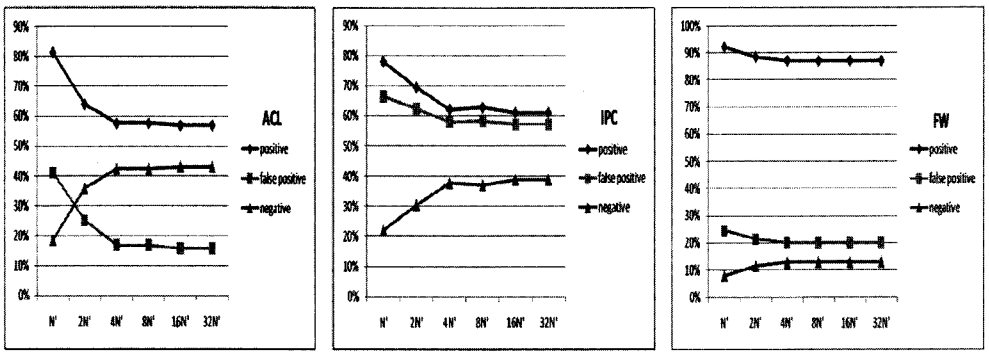

그림 7 ACL, IPC, FW에서 튜플 블룸 필터 사이즈 변화에 따른 성능 변화

유니크 튜플 개수가 2451이므로 튜플 블룸 필터의 크기 $N'=2^{\lceil \log_2 2451 \rceil}=4096$ 비트가 된다.

이는 제안하는 구조에서 크로스-프로덕팅 단계에서 튜플의 길이 정보만을 이용하여 검색에 불필요한 튜플들을 제거 후, 튜플 블룸 필터를 이용하여 튜플의 길이 정보뿐만 아니라 튜플을 구성하는 프리픽스 값의 정보를 이용하여 거짓-튜플을 필터링 하므로 해시 테이블에 접근하는 튜플의 개수를 더 줄일 수 있기 때문이다. 그래프에서 블룸 필터의 크기가 $8N'$ 이상으로 증가하여도 블룸 필터의 성능이 더 이상 좋아지지 않는 까닭은 식 (2)에서 주어진 바와 같이 블룸 필터의 크기가 커짐에 따라 해싱 함수의 수도 함께 증가되어야 하나, 본 논문에서는 구현의 단순화를 위하여 해싱 함수의 수를 2로 고정하였기 때문이다.

4.3 제안하는 두 번째 알고리즘의 성능평가

4.3.1 근원지, 목적지 프리픽스 블룸 필터 성능 평가

제안하는 두 번째 알고리즘에서는 기존의 1차원 최장 길이 검색을 블룸 필터로 대체하여 튜플 공간 검색을 진행한 구조이다. 그러므로 근원지 블룸 필터와 목적지 블룸 필터 성능이 검색에 큰 영향을 미칠 수 있으므로 각 블룸 필터의 성능을 평가해 보았다. 원래 블룸 필터가 가지고 있는 성질로서 메모리의 사이즈가 커질수록 거짓-양성의 비율은 줄어들고 음성 결과 비율은 증가하여 해시 메모리에 접근하는 양성 결과 수가 줄어들어 검색 성능을 향상시킬 수 있다. 그림 8은 세 룰 셋 가운데 대표로 ACL를 이용하여 각 필드의 블룸 필터의 성능을 나타내었다. 그림 8에서도 전체 튜플 블룸 필터의 입력 값에 대한 양성과 음성 결과 값의 비율과 양성 결과 값에 대한 거짓 양성의 비율을 나타내었다. 다른 룰 셋의 경우도 ACL과 동일하게 블룸 필터 사이즈가 커질수록 음성결과는 증가하고 거짓-양성 결과는 줄어드는 것을 확인하였다. 블룸 필터의 크기를 나타내는 N'은 각 프리픽스 필드의 유니크 프리픽스 개수보다 크거나 같으면서 가장 작은 2의 제곱수를 나타내었다. ACL의 경우 룰 셋에 근원지 프리픽스의 유니크 프리픽스의 개수가 653이므로 튜플 블룸 필터의 크기 $N'=2^{\lceil \log_2 653 \rceil}=1024$ 비트이다.

4.3.2 전체 구조의 성능

제안된 알고리즘의 경우에는 근원지 블룸 필터를 통과 후 양성 결과 가운데 거짓-양성 결과를 제거하기 위

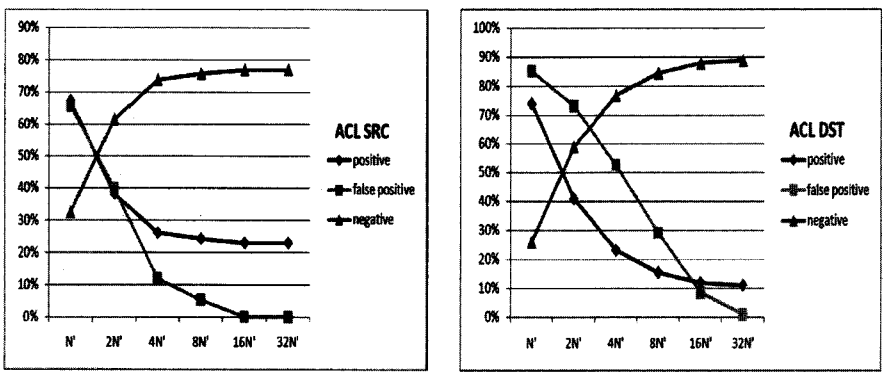

그림 8 ACL에 대한 제안하는 두 번째 알고리즘의 근원지, 목적지 프리픽스 블룸 필터의 사이즈에 따른 블룸 필터 성능

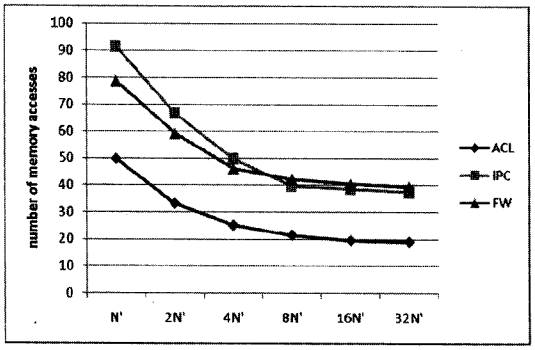

그림 9 제안하는 두 번째 알고리즘의 근원지, 목적지 프리픽스 블룸 필터의 사이즈에 따른 전체 해사 메모리 평균 접근 횟수

해 근원지 해시 테이블에 접근한다. 이와 마찬가지로 목적지 블룸 필터를 통과 후 목적지 해시 테이블에 접근하여 거짓-양성 결과를 제거한다. 근원지 해시 테이블과 목적지 해시 테이블을 통과한 결과값을 교차 하여 생성된 튜플들 가운데 튜플 리스트에 존재하는 튜플들만을 가지고서 튜플 해시 테이블에 접근하여 일치하는 최우선 순위 룰을 찾아낸다. 그러므로 제안된 알고리즘에서의 한 개의 입력된 패킷 검색을 위한 총 메모리 접근 횟수는 근원지 해시 테이블, 목적지 해시 테이블, 튜플 해시 테이블의 접근 횟수를 합한 값과 같다.

그림 9에서 두 프리픽스 필드의 블룸 필터의 사이즈를 동시에 증가시키며 근원지, 목적지, 튜플 해시메모리 접근 횟수의 합이 어떻게 변화하는지 실험해 보았다. 실험에서 블룸 필터의 메모리 사이즈가 커짐에 따라 거짓-양성의 결과는 줄어들고 음성 결과 값이 증가하는 특징을 다시 한번 확인할 수 있다. 이러한 결과로 프리픽스 필드 검색에 필요한 해시 테이블 접근 횟수가 줄어들게 되어 제안하는 구조의 전체 검색 속도를 향상시킬 수 있었다.

4.4 제안하는 구조와 기존의 패킷 분류 구조와의 알고리즘의 성능 비교

아래의 표 7과 8은 제안된 알고리즘의 성능과 앞서 2절에서 소개한 기존의 패킷 분류 구조인 H-trie, set-pruning trie, AQT, PQT, TSS의 성능과 평균 메모리 접근 횟수와 메모리 요구량에 있어 비교해 놓은 것이다. 기존의 튜플 공간 검색 알고리즘의 경우 근원지 프리픽스 필드와 목적지 프리픽스 필드에서 메모리 접근 횟수가 발생하게 된다. 그러므로 프리픽스의 최장 길이 일치 검색 알고리즘 가운데 가장 성능이 좋은 이진 레벨 검색(binary search on length, BSL) 구조를 적용할 때의 메모리 접근 횟수를 이용하였다. BSL 구조의 경우 한번 외부 메모리에 접근할 때마다 검색 영역이 1/2로 줄어들어 트라이 기반 프리픽스 최장길이 검색구조에서 적은 메모리 접근 횟수로 빠른 검색 성능을 보여주는 구조이다.

표에 제시한 성능은 제안하는 알고리즘들에 사용된 모든 블룸 필터의 크기가 $8N'$인 경우인데, 이는 앞서 설명한 바와 같이 블룸 필터 사이즈가 $8N'$일 때를 비교한 이유는 $8N'$를 기점으로 블룸 필터의 성능이 완화되기 때문이다. 기존의 트라이 구조를 기반으로 하는 패킷 분류 알고리즘들과 비교하였을 때 대체적으로 튜플 공간을 이용한 구조들이 더 적은 메모리 사용량을 차지하면서 검색성능이 좋은 것을 알 수 있다.

제안하는 첫 번째 알고리즘의 경우 기존의 알고리즘에 평균 2~4kbytes 정도 메모리를 차지하는 튜플 블룸 필터만을 추가 함에도 불구하고 메모리 접근 횟수가 줄어드는 것을 알 수 있다. 이는 제안하는 구조에서 크로스-프로덕팅 단계에서 튜플 블룸 필터를 이용하여 튜플의 길이 정보뿐만 아니라 튜플을 구성하는 프리픽스 값의 정보를 이용하여 튜플을 필터링하므로 해시 테이블에 접근하는 튜플의 개수를 더 줄일 수 있기 때문이다.

제안하는 두 번째 알고리즘과 기존의 튜플 공간 검색 구조를 비교해보면 제안하는 알고리즘은 메모리 접근 횟수 측면에서 조금 더 적거나 비슷한 메모리 접근 횟수를 보이는 것을 알 수 있다. 이는 프리픽스 간 네스팅 관계로 인하여 블룸 필터에서 양성 결과들이 영향을 받기 때문이다. 하지만 메모리 접근 횟수를 비교할 때 프리픽스 필드 검색 구조 중 가장 성능이 좋은 BSL 구조를 적용하여 비교해서 한 결과이며, 기존의 튜플 공간 검색 구조보다 훨씬 적은 메모리 공간을 차지하면서 복잡한 선-계산이 필요 없기 때문에 기존의 구조보다 장점을 가지고 있다.

표 7 제안하는 알고리즘과 기존 패킷 분류 알고리즘과의 메모리 접근 횟수 비교

Rule Type	Prop.1	Prop.2	TSS	H-trie	set-pruning	AQT	PQT
ACL	15.92	21.43	19.21	84	67.6	50.1	59.6
IPC	29.06	42.21	36.24	85.6	59.8	344.8	202.1
FW	42.81	39.69	44.19	67.5	52.2	660.5	571.1

표 8 제안하는 알고리즘과 기존 패킷 분류 알고리즘과의 메모리 요구량(kBytes) 비교

Rule Type	Prop.1	Prop.2	TSS	H-trie	set-pruning	AQT	PQT
ACL	272.4	174.3	268.2	509.0	990.9	200.2	145.6
IPC	184.6	164.5	179.2	246.8	620.4	234.3	139.9
FW	176.5	161.8	167.1	109.9	491.8	479.8	136

5. 결 론

본 논문에서는 인터넷 라우터 내에서 빠르게 입력되는 패킷을 선 속도로 처리할 수 있는 패킷 분류 알고리즘을 제안하였다. 제안된 알고리즘은 패킷 분류를 위한 다차원 검색을 튜플 공간이라는 이차원 공간을 이용하여 검색하는 튜플 공간 패킷 분류 알고리즘에 블룸 필터를 적용하여 불필요한 튜플들을 제거하여 적은 메모리 공간을 차지하면서 적은 메모리 접근 횟수로 패킷을 분류 할 수 있는 알고리즘이다. 칩 내부에 저장될 수 있는 적은 메모리 공간을 요구하는 블룸 필터는 필터의 입력 값이 필터에 미리 저장된 정보인지 아닌지를 빠르게 판단하여 불필요한 검색들을 걸러내는 역할을 한다. 이러한 블룸 필터의 특성을 이용하여 룰셋의 근원지와 목적지 프리픽스의 정보를 블룸 필터에 저장하고 패킷이 입력 되었을 때 각 목적지, 근원지 블룸 필터를 병렬로 통과시켜 입력 패킷에 대한 칩 외부의 해시 테이블 접근 횟수를 효과적으로 줄일 수 있도록 하였다. 실제 라우터에서 사용되는 라우터 테이블을 이용하여 제안된 알고리즘을 메모리 사용량과 접근 횟수로 비교한 결과 모두 기존의 패킷 분류 알고리즘과의 성능평가에서 타 알고리즘에 비해 메모리 사용량과 검색 속도 면에서 우수함을 보였다.

참 고 문 헌

[1] H. Jonathan Chao, "Next Generation Routers," *Proceedings of the IEEE JPROC.2002.802001*, 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] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *in Proc. of ACM SIGCOMM Computer Communication Review*, vol.29, no.4, pp.135-146, 1999.

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

[5] G. Vaghese, "Network Algorithmics," Morgan Kaufmann, 2005.

[6] M. M. Buddhikot, S. Suri, and M. Waldvogel, "Space Decomposition Techniques for Fast Layer-4 Switching," *in Proc. Conf. Protocols for High Speed Networks*, pp.25-41, Aug. 1999.

[7] 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.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *in Communications of the ACM*, 13(7): pp.422-426, July 1970.

[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] Alagukonar Ganapathy Alagupriya, "Packet Classification Algorithms Using Bloom Filters," 이화여자대학교 대학원 2008년도 석사학위 청구논문.

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

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

[13] 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.

[14] Sarang Dharmapurikar, Praveen Krishnamurthy, David E. Talyor, "Longest Prefix Matching Using Bloom Filters," *SIGCOMM*, Aug. 2003.

[15] F. Baboescu, S. Singh, G. Varghese, "Packet classification for core router: is there an alternative to CAMs?," *IEEE INFOCOM 2003*, vol.1, pp.53-63, Mar. 2003.

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

[17] H. Song, J. Turner, and S. Dharmapurikar, "Packet Classification using Coarse-grained tuple spaces," *ANCS*, 2006.

[18] M. Ahmadi and S. Wong, "Modified Collision Packet Classification Using Counting Bloom Filter in Tuple Space," *in Proc., PDCN 2007*, pp.70-76, 2007.

임 혜 숙

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 부교수. 관심분야는 라우터나 스위치 등의 네트워크 관련 알고리듬 및 SoC 설계, TCP/IP 관련 하드웨어 설계

김 소 연

2008년 이화여자대학교 정보통신학과 졸업(학사). 2010년 이화여자대학교 전자공학과 졸업(석사). 현재 삼성전자 연구원. 관심분야는 라우터나 스위치 등의 네트워크 관련 알고리듬 및 SoC 설계, TCP/IP 관련 하드웨어 설계