길이에 대한 2차원 이진검색을 이용한 패킷분류 구조

준회원 문 주 형*, 정회원 임 혜 숙*

Packet Classification Using Two-Dimensional Binary Search on Length

Ju Hyoung Mun* ***Associate Member***, Hyesook Lim* ***Regular Member***

요 약

인터넷의 성장은 다양한 응용 프로그램들의 발달을 야기 시켰으며, 그로 인해 모든 패킷을 동일하게 처리하는 현재의 최선지원 서비스 보다 나은 서비스를 제공할 것을 요구하고 있다. 따라서 차세대 인터넷 라우터들은 다양한 레벨의 품질보장 서비스를 제공하여야 한다. 품질보장 서비스를 제공하기 위해서는 모든 입력 패킷을 미리 정의된 룰에 따라 구분하는 패킷 분류가 실시간으로 수행되어야 한다. 패킷분류는 패킷에 포함된 여러 헤더 필드에 대하여 다양한 종류의 검색을 수행하여야 하며, 일치하는 룰들 중에서 가장 높은 우선순위를 갖는 룰을 찾아야 하는 다차원 검색이다. 영역분할을 사용한 사분트라이 구조는 근원지와 목적지 프리픽스를 2차원 트라이 구조로 저장하여 검색을 진행하는 좋은 알고리즘이나, 길이에 대하여 선형검색을 하는 방법이므로 좋은 검색 성능을 보이지 못한다. 본 논문에서는 사분트라이 구조에서 길이에 대하여 이진검색을 진행하는 새로운 패킷분류 알고리즘을 제안한다. 또한 패킷이 여러 개의 룰과 일치하였을 경우 가장 높은 우선순위를 가지는 룰을 선택한다는 특성을 이용하여, 사분트라이를 만드는 과정에서 우선순위를 고려하여 검색 성능을 향상시킬 수 있는 방안을 제안한다.

Key Words : Packet classification, Hashing, Area-based quad-trie, Binary search on length, Best matching rule

ABSTRACT

The rapid growth of the Internet has stimulated the development of various new applications and services, and the service providers and the Internet users now require different levels of service qualities rather than current best-effort service which treats all incoming packet equally. Therefore, next generation routers should provide the various levels of services. In order to provide the quality of services, incoming packets should be classified into flows according to pre-defined rules, and this should be performed for all incoming packets in wire-speed. Packet classification not only involves multi-dimensional search but also finds the highest priority rule among all matching rules. Area-based quad-trie is a very good algorithm that constructs a two-dimensional trie using source and destination prefix fields. However, it performs the linear search for the prefix length, and hence it does not show very good search performance. In this paper, we propose to apply binary search on length to the area-based quad-trie algorithm. In improving the search performance, we also propose two new algorithms considering the priority of rules in building the trie.

Ⅰ. 서 론

인터넷은 성장을 거듭한 끝에 전 세계를 하나로 묶는 거대한 네트워크로 발전하였다. 인터넷의 성장은 호스트 수뿐만 아니라 다양한 응용 프로그램들의 발달을 야기 시켰으며, 그로 인해 사용자들은 다양한

※ This research was supported by the MIC, Korea, under HNRC-ITRC support program supervised by IITA.
* 이화여자대학교 정보통신학과 SoC설계연구실 (hlim@ewha.ac.kr)
논문번호 : KICS2007-03-094, 접수일자 : 2007년 3월 2일, 최종논문접수일자 : 2007년 7월 18일

레벨의 품질보장을 (quality of service, QoS) 지원할 것을 요구하게 되었다. 현재의 인터넷은 최선의 지원(best effort service)을 기본으로 하는 서비스이므로 인터넷 라우터는 들어오는 모든 패킷을 동일하게 처리하고 있다. 하지만 DiffServ나 IntServ와 같은 품질보장 방법을 지원하기 위해서는 패킷분류(packet classification)가 필수적으로 선행되어야 한다. 패킷분류는 인터넷 라우터에서 패킷을 룰 셋에 따라 클래스로 나누는 작업으로서, 입력된 패킷이 속한 클래스에 정의된 서비스를 제공함으로서 품질보장 지원을 가능하게 한다[1].

서비스의 품질을 결정하는 각각의 클래스는 패킷의 여러 헤더 필드들로 구성된 룰로 정의된다. 룰을 이루는 필드에는 주로 목적지 IP 주소, 근원지 IP 주소, 목적지 포트 넘버, 근원지 포트 넘버, 프로토콜 등 패킷의 헤더 정보들이 사용된다. 완전 일치하는 주소를 찾는 2계층 맥 주소 검색이나, 가장 길게 일치하는 프리픽스(longest matching prefix)를 찾는 3계층 인터넷 주소 검색과 달리 패킷분류를 위해서는 목적지 주소뿐만 아니라 여러 개의 필드들을 보고 분류하는 다차원 검색을 수행하여야 한다.

패킷분류는 들어오는 모든 패킷마다 룰을 구성하는 모든 필드에 대하여 검색을 수행하고, 일치하는 여러 룰 중에서 가장 우선순위가 높은 룰을 선택하는 방식으로 수행된다. 이 때 완전일치(exact match), 영역일치(range match), 프리픽스 일치(prefix match)와 같은 다양한 종류의 연산이 요구되며 패킷이 여러 룰에 일치할 수 있으므로 일치하는 모든 룰을 찾아 가장 우선순위가 높은 룰을 선택해야 하는 복잡성이 있다. 그럼에도 불구하고 패킷분류는 인터넷 주소 검색과 동일하게 라우터에 입력되는 모든 패킷에 대하여 실시간(wire-speed)으로 처리되어야 하는 어려움이 있다. 따라서 최근 라우터의 패킷분류를 위한 효율적인 알고리즘들이 널리 연구되고 있다. 주된 연구 방향은 효율적인 다차원 검색을 수행하는 것으로, 룰의 여러 필드 중에서도 근원지 IP 주소와 목적지 IP 주소를 사용한 검색에 그 초점이 맞추어져 왔다. 이것은 근원지 IP 주소와 목적지 IP 주소에 따른 플로우의 다양성이 다른 필드에 따르는 플로우의 다양성보다 월등히 크다는 분석에 기초하고 있다[2]. 패킷분류 방법을 위해 제안된 대부분의 알고리즘들은 다차원 검색이 아닌 일차원 검색의 순차적 연결에 그치고 있는 반면, 영역분할을 사용한 사분트라이(area-based quad-trie, AQT)는 근원지 IP 프리픽스와 목적지 IP 프리픽스를 사용하여 이차원 트라이로 확장시킨 것으로서 근원지 주소와 목적지 주소를 각각 한 비트씩 동시에 검사하여 검색하여야 할 영역을 재귀적으로 분할하며 검색을 수행한다. AQT는 영역으로 표현된 근원지와 목적지 프리픽스를 트라이 구조로 저장하여, 검색을 진행하는 좋은 알고리즘이나, 길이에 대하여 선형검색을 하는 방법이므로 좋은 검색 성능을 보이지 못한다는 것과 룰을 저장하지 않는 비어있는 노드들로 인하여 메모리가 낭비된다는 단점을 갖는다.

본 논문에서는 효율적인 패킷분류를 위하여 인터넷 주소 검색에서 매우 좋은 성능을 보이는 프리픽스 길이에 대한 이진검색 방법을 패킷분류에 적용하는 알고리즘을 제안한다. 또한 패킷분류는 패킷이 여러 개의 룰과 일치하였을 경우 가장 우선순위가 높은 룰을 찾아야 한다는 특성을 이용하여, 우선순위가 높은 룰을 먼저 검색하여 검색 성능을 향상시키는 몇 가지 방법을 제안한다.

본 논문의 구조는 다음과 같다. 먼저 Ⅱ장에서는 기존에 나와 있는 이진검색 알고리즘을 소개한다. Ⅲ장에서는 제안하는 알고리즘에 대해 설명하고, 실제의 룰 셋의 특징을 살펴 제안하는 알고리즘의 성능을 향상시킬 수 있는 방법들을 설명한다. Ⅳ장에서는 제안하는 알고리즘의 성능 실험 결과 및 다른 이진검색 알고리즘과의 성능을 비교한다. 마지막으로 Ⅴ장에서 간단한 결론을 맺는다.

Ⅱ. 기존의 패킷분류 구조

2.1 이차원 이진 트라이

이차원 이진 트라이로는 area-based quad-trie (AQT)가 있다[1][3]. AQT는 룰의 근원지 프리픽스와 목적지 프리픽스만으로 구성된 트라이 구조를 사용하여 패킷분류를 수행한다. AQT에서의 룰들은 프리픽스 평면상의 직사각형 영역으로 표현되는데, 근원지 프리픽스와 목적지 프리픽스는 각각 최대 32 비트를 가질 수 있으므로, 프리픽스 길이에 따라 가로 2^{32}, 세로 2^{32}인 정사각형 평면에서의 직사각형 영역으로 표현된다. 룰의 근원지 프리픽스 길이가 l, 목적지 프리픽스 길이가 m이라면, 전체 평면에서 가로가 $2^{(32-l)}$, 세로 $2^{(32-m)}$인 직사각형으로 나타나게 된다.

AQT에서의 패킷분류는 입력된 패킷의 근원지 주소와 목적지 주소의 한 비트씩을 이용하여 검색범위를 1/4씩 줄어나가면서 패킷이 속하는 룰을 검색하게 된다. 검색의 매 단계에서 근원지 프리픽스와 목적지 프리픽스는 0, 1의 값으로 이등분되며, 두 프리

픽스로 표현된 정사각형 영역이 같은 크기의 00, 01, 10, 11의 4 영역으로 분할된다. 분할 과정이 n번 반복되면, 전체 평면은 4^n개의 영역으로 분할되고, 분할된 영역은 n-비트 길이의 프리픽스 쌍으로 표현된다. 룰은 근원지 프리픽스와 목적지 프리픽스의 길이가 다를 수 있으나, AQT에서의 영역 분할은 근원지 프리픽스와 목적지 프리픽스의 한 비트씩을 사용하여 이루어지므로 같은 길이의 프리픽스로만 표현되게 된다. AQT는 룰의 두 프리픽스 중에 하나라도 검색 영역이 나타내는 길이와 일치하여 룰의 영역이 완전히 검색 영역에 속하는 룰들을 그 영역의 crossing filter set(CFS)으로 정의하고, 검색시에는 이 CFS에 속한 룰들에 대하여 선형검색을 수행하게 된다.

표 1. 예시 룰 셋

Rule no	Src prefix	Dst prefix	Src port (start, end)	Dst port (start, end)	Prtl
R0	0000*	1010*	53, 53	443, 443	17
R1	000111*	11110*	53, 53	25, 25	6
R2	101011*	001101*	53, 53	25, 25	17
R3	00000*	10100*	67, 67	5632, 5632	6
R4	000*	1110*	1024, 65535	1024, 65535	6
R5	0011*	00*	53, 53	25, 25	4
R6	110*	01*	0, 65535	5632, 5632	6
R7	110010*	110110*	0, 65535	5632, 5632	6
R8	1010*	00110*	53, 53	25, 25	6

표 1에 2차원 룰 셋의 예를 보였다. 표 1과 같은 룰 셋을 AQT에서 정의한 대로 평면 상의 직사각형 영역으로 나타내면 그림 1과 같게 된다. 그림1의 프리픽스 평면을 근원지와 목적지 프리픽스를 한 비트씩 보며 재귀적으로 잘라가며 AQT 트라이를 만들면 그림 2와 같다. 만약 들어온 패킷의 근원지 주소가 **00001110**, 목적지 주소가 **10101000**일 때, AQT에서의 검색과정은 다음과 같다. 근원지 주소와 목적지 주소를 한 비트씩 보고 코드워드를 만들면 **10103220**이라는 코드워드가 생성된다. 루트 노드에 접근하여 루트 노드의 CFS를 검색한다. 이 때 R10과 일치하였으므로, R10을 현재까지 가장 잘 일치하는 룰 (best matching rule, BMR)로 저장하고 코드워드의 첫 번째 비트를 확인한다. 코드워드의 첫 번째 값이 **1**이므로, AQT 트라이에서 **1**로 표시된 에지를 따라 검색을 진행한다. 다음 노드가 룰을 저장하지 않는 내부 노드이므로 코드워드의 다음 값을 확인하여 해당하는 포인터를 따라가면서 검색을 계속 진행한다. 룰이 저장된 노드를 만나면 입력 패킷의 모든 필드에 대하여 CFS에 저장된 룰과 일치하는지 비교한다. R0와 일치하였고, R0가 현재의 BMR인 R10보다 우선순위가 높음으로 현재까지의 BMR 정보를 R0로 수정하고, 검색을 계속 진행하여야 하지만 더 이상 일치하는 노드가 없음으로 현재의 BMR을 출력하고 검색을 종료한다.

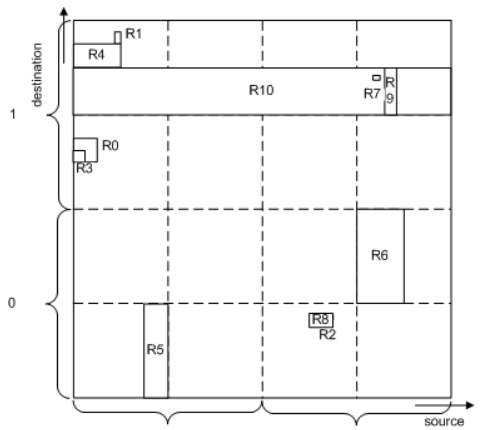

그림 1. 직사각형 영역으로 표현된 룰

AQT는 룰의 기하학적인 영역을 정의한 후, 각 영역을 사분-트라이를 이용하여 나타내었다. 하지만 프리픽스 길이가 최대 W라고 할 경우, AQT의 사분-트라이의 깊이는 대부분의 경우 최대 프리픽스 길이인 W가 되고, 해당 영역에 만족하는 CFS가 없을 경우 빈 내부 노드가 생기기 때문에 메모리가 낭비된다. 또한 AQT는 프리픽스 길이에 대해서 선형검색을 수행하는 구조이기 때문에, 메모리 접근 횟수가 많아 검색 시간이 길어지게 된다. 또한 룰 정보를 저장하지 않는 많은 내부 노드들로 인해 많은 메모리를 사용하게 된다는 단점이 있다.

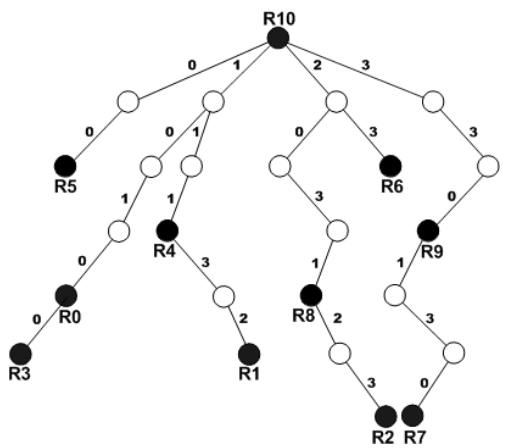

그림 2. 영역기반 사분-트라이 (AQT)

2.2 프리픽스 길이에 대한 이진검색

프리픽스 길이에 대한 이진검색(binary search on length, BSL) 구조[4]는 프리픽스 길이에 대하여는 이진검색을, 프리픽스 값에 대하여는 해싱을 사용하는 구조로서 빠른 검색 속도를 장점으로 한다. 이 구조에서는 이진 트라이를 레벨 별로 나누고 각 레벨에 존재하는 모든 프리픽스들을 하나의 해쉬 테이블에 저장한다. 입력된 어드레스에 대해 각각의 레벨에 따르는 해쉬 값을 구하여 해당 해쉬 테이블을 접근하였을 때, 프리픽스와 일치하는 경우에는 더 긴 프리픽스가 존재할 것을 가정하여, 더 긴 레벨로 진행하고, 프리픽스와 일치하지 않는 경우에는 현재 레벨보다 긴 프리픽스가 존재하지 않는 것을 가정하여, 짧은 레벨로 진행함으로서, 프리픽스 길이에 대하여 이진검색을 수행하는 구조이다. 그러나 해당 해쉬 테이블의 프리픽스와 일치하였다고 하여도, 더 긴 레벨의 프리픽스가 존재하지 않을 수 있고, 해당 해쉬 테이블의 프리픽스와 일치하지 않았다고 하여도, 더 긴 레벨의 프리픽스와 일치할 수 있다. 따라서 길이에 대한 이진검색을 위한 이 구조의 기본적인 가정에는 모순이 존재한다.

이러한 모순을 해결하기 위하여 이 구조에서는 선계산 (pre-computation) 기법을 이용하였다. 비어있는 내부 노드에 마커를 미리 계산, 저장하여, 해당 해쉬 엔트리에 일치하는 프리픽스가 존재하지 않는 경우에도 더 긴 프리픽스가 존재함을 표시한다. 그러나 마커가 모든 종류의 프리픽스를 커버할 수 없다는 문제점이 있다. 마커가 있었기 때문에, 아래 레벨에 대해서 이진검색을 진행하였으나, 더 길게 일치하는 프리픽스가 존재하지 않은 경우, 해당 마커의 위 쪽 레벨에 대하여 다시 검색을 수행하여야 하는 백-트랙킹(back-tracking) 문제가 발생한다. 이 경우 검색 시간을 증가시키기 때문에 바람직하지 않은데, 이 구조에서는 모든 마커마다 현재까지 가장 잘 일치하는 프리픽스 (best matching prefix, BMP)를 미리 계산하여 저장하는 방법으로 해결하였다. 마커보다 더 긴 레벨을 검색하였으나 실패한 경우 현재까지의 BMP를 기억하고 있으므로, 그것이 입력된 인터넷 주소와 가장 길게 일치하는 프리픽스가 되는 것이다. 마커와 BMP는 모든 비어있는 내부 노드에 저장될 필요가 없으며, 이진검색시 먼저 접근이 되는 레벨의 내부 노드에만 미리 계산하여 저장하면 된다. 이 구조의 경우 프리픽스 길이에 대하여 이진검색을 수행하므로, 하나의 어드레스 검색을 위한 최대 메모리 접근 횟수가 $O(\log_2 W)$ 가 된다는 장점을 갖는다.

그림 3은 **0*, 000*, 0010*, 10*, 1010*, 1110*** 6개의 프리픽스를 예로 길이에 대한 이진 트라이를 만든 것이다. 그림 3을 예로 검색 과정을 설명하면, 먼저 레벨 2인 프리픽스들 중에서 검색을 수행하고, 그 다음으로 레벨 1이나, 레벨 3을 검색하게 된다. 레벨 2 (혹은 레벨 1, 3)와 같이 우선적으로 선택되는 레벨의 내부 노드들에 마커와 현재까지의 BMP를 저장해 두어야 한다. 예를 들어 **001100**이라는 어드레스가 입력된 경우, 처음 접근되는 레벨 2의 **00*** 노드에는 프리픽스가 저장되어 있지 않지만, 마커가 있으므로, 현재까지의 BMP인 P1을 기억한 후 더 긴 레벨로 검색을 진행한다. 레벨 3을 접근하였을 때, 또한 마커가 있으므로, 현재까지의 BMP를 기억한 후, 더 긴 레벨인 레벨 4로 진행한다. 레벨 4에서는 노드가 존재하지 않으므로, 짧은 레벨로 진행하여야 하나 레벨 3과 레벨 4사이에는 다른 레벨은 존재하지 않으므로, 기억된 BMP인 P1이 입력 어드레스의 가장 길게 매치하는 프리픽스로 출력된다. 또 다른 예로 입력된 어드레스가 **110000** 이라면, 레벨 2인 노드 **11*** 가 검색되고, 마커가 저장되어 있으므로, 더 긴 레벨로 검색을 진행하여 레벨 3의 **110***가 검색되나 검색된 엔트리에는 노드가 존재하지 않아 짧은 레벨로 진행하여야 하나 레벨 2와 레벨 3 사이에는 다른 레벨이 존재하지 않으므로, 일치하는 프리픽스 없이 검색이 종료된다. 이 경우에는 기본 (default) 라우팅 정보가 사용된다.

다른 내부 노드들에 대해서도 마커와 BMP를 선계산 해준 결과를 그림 3에 표시하였다. 이 구조는 IPv4의 경우 가장 짧은 프리픽스 길이가 8임을 가정했을 때 최악의 경우 5번의 메모리 접근으로 하나의 인터넷 주소 검색이 가능하다는 장점을 갖으나, 마커와 해당 마커에 대한 BMP를 미리 계산해주어야 하며, 이러한 선계산 때문에 프리픽스의 부가적 추가 (incremental update)가 가능하지 않다는 단점을 지닌다.

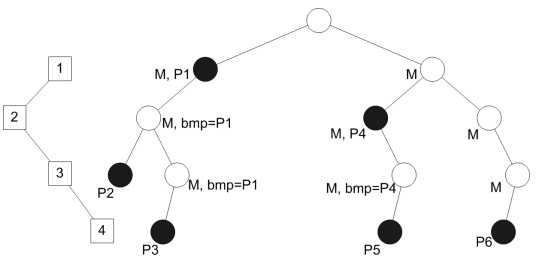

그림 3. 길이에 대한 이진검색 트라이

프리픽스의 부가적인 추가가 가능하면서도 길이에 대한 이진검색 트라이의 좋은 성능을 유지하기 위해서 최장 길이 우선에 기초한 프리픽스 길이에 따른 이진검색 구조(binary search on length, BSL)[5]가 제안되었다. 최장 길이 우선에 기초한 길이에 대한 이진검색 구조를 만드는 과정은 다음과 같다. 라우팅 데이터에서 상대적으로 가장 긴 길이를 갖는 리프 프리픽스들만을 가지고 BSL 트라이를 만든다. 이 리프 프리픽스들 간에는 어떠한 네스팅 관계도 없게 되므로 선계산 없이도 길이에 대한 이진검색이 가능하게 된다. 다시 남아있는 라우팅 데이터에서 리프 프리픽스들을 가지고 두 번째 BSL 트라이를 만든다. 이러한 과정을 라우팅 데이터의 모든 프리픽스가 BSL 트라이에 속할 때까지 반복한다. 최장 길이 우선에 기초한 BSL 트라이는 프리픽스 계층구조의 상대적인 레벨에 따라 여러 개의 트라이를 생성한 후, 상대적으로 긴 프리픽스로 구성된 트라이를 먼저 검색함으로서 평균 메모리 접근 회수를 줄일 수 있다. 하지만 여러 개의 BSL 트라이를 사용하므로 최대 메모리 접근 회수가 나빠질 수 있는 단점이 있다.

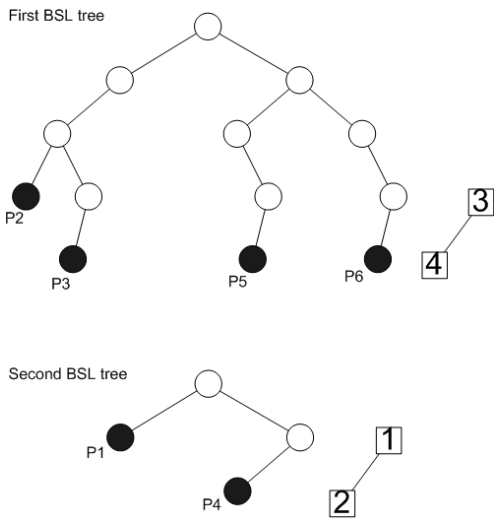

그림 4. 최장 길이 우선 검색에 기초한 길이에 대한 이진검색 구조

그림 4는 앞에서와 동일한 예를 사용하여 최장 길이 우선 BSL 트라이를 만든 것이다. 입력된 패킷의 주소가 **001000** 인 경우, 첫 번째 BSL 트라이를 검색한다. 레벨 3의 **001***의 내부 노드와 일치하였으므로 더 긴 레벨 4로 검색을 진행한다. 레벨 4의 **0010***인 P3와 일치하였으므로 다른 BSL 트라이를 검색하지 않고 검색을 즉시 종료한다. 입력된 어드레스가 **110000** 인 경우, 첫 번째 BSL 트라이의 레벨 3에서 **110***을 찾는다. **110***에 해당하는 노드가 없으므로 더 짧은 쪽으로 검색을 진행하여야 하지만, 레벨 3가 가장 짧은 프리픽스 길이이므로 일치하는 정보 없이 검색이 종료된다.

Ⅲ. 제안하는 검색 알고리즘

3.1 Algorithm 1

본 논문에서는 인터넷 주소검색에서 탁월한 성능을 보이는 프리픽스 길이에 대한 이진검색 구조를 패킷분류에 적용하는 알고리즘을 제안한다. 제안하는 알고리즘의 전체적인 구성은 그림 5와 같다. 패킷분류의 룰은 인터넷 주소검색과 달리 근원지 프리픽스, 목적지 프리픽스, 근원지 포트 범위, 목적지 포트 범위, 프로토콜 타입과 같은 많은 필드들을 포함한다. 제안하는 알고리즘은 먼저 근원지 주소와 목적지 주소만을 이용하여 길이에 대한 이진검색을 수행한다. 그 뒤 근원지 프리픽스와 목적지 프리픽스 쌍에 대하여 일치한 룰들만을 가지고 모든 필드들을 비교하여 검색을 수행한다. 실제 사용되고 있는 룰 셋을 분석한 결과, 근원지 주소 혹은 목적지 주소 중의 하나의 필드만으로 일치 가능한 룰들을 추려보면 매우 많은 수의 룰과 일치하는 것으로 나타나지만, 두 개 필드를 이용하여 일치 가능한 룰들을 추려보면 5% 정도의 입력만이 20여개의 룰과 일치하고 나머지 95%의 입력은 5개 미만의 룰과 일치하는 것으로 알려져 있다[2]. 이러한 특성 때문에 효율적인 이차원 검색 알고리즘을 다차원 패킷분류에 적용할 수 있다. 제안하는 알고리즘은 이러한 특성을 사용하여 근원지 프리픽스와 목적지 프리픽스의 두 개의 필드만을 사용하여 검색을 수행하여 입력된 패킷과 일치 가능한 룰들을 거른 후에, 걸러진 룰들에 대해 룰의 모든 필드를 사용하여 검색을 수행한다.

인터넷 주소검색을 위한 프리픽스 길이에 대한 이진검색 구조는 이진 트라이를 길이 별로 쪼개어 각각의 길이에 해당하는 해쉬 테이블에 저장하여 길이에 대한 이진검색을 수행하는 구조이다. 제안하는 알고리즘은 널리 알려진 패킷분류 방법인 AQT 트라이를 길이 별로 쪼개어 해당 길이에 해당하는 해쉬 테이블을 저장한 후에 길이에 대한 이진검색을 수행한다. 하지만 가장 긴 길이의 프리픽스가 가장 높은 우선순위를 갖는 인터넷 주소 검색과는 달리 패킷분류에서는 길이와 관계없이 룰의 우선순위에 의해 검

색이 이루어져야 한다. 패킷분류는 일치하는 모든 룰들 중에서 가장 우선순위가 높은 룰을 찾아야 하므로 입력된 패킷과 일치하는 모든 룰을 찾아야 한다. 따라서 가장 길게 매치하는 프리픽스(BMP)를 찾기 위해 선계산을 하여 저장해 놓은 마커와 마커의 BMP는 매치하는 모든 룰을 검색하기에 적합하지 않다. 그러나 마커와 마커의 BMP를 제거할 경우, 검색이 올바르게 수행되지 않는다. 이러한 문제를 해결하기 위해 본 논문에서는 최장 길이 우선에 기초한 프리픽스 길이에 따른 이진검색 구조를 사용한다.

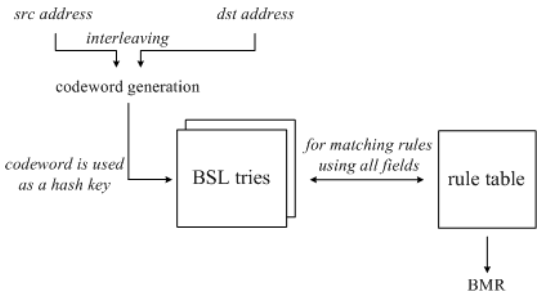

그림 5. 제안하는 알고리즘

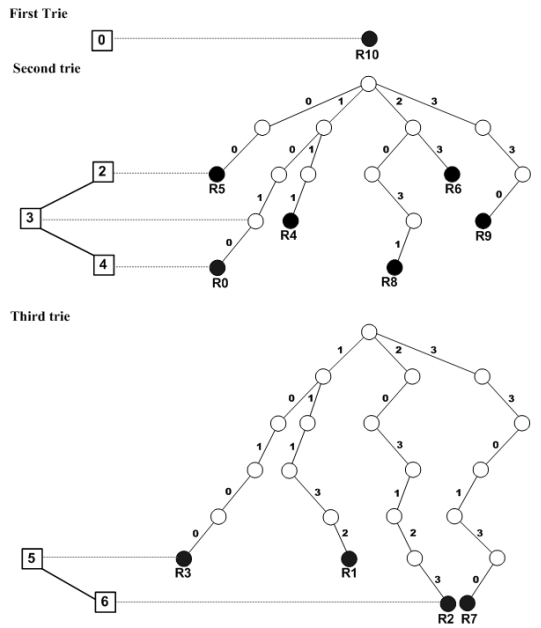

그림 6. 제안하는 BSL 트라이 구조

본 논문에서는 제안하는 구조는 AQT 트라이에서 룰들의 상대적인 레벨에 따라 여러 개의 트라이로 분리하여 길이에 대한 이진검색을 수행하는 구조이다. 즉 루트 노드로부터 출발하여 모든 패스에서 가장 먼저 만나는 룰들을 AQT 트라이에서 제거하여 첫 번째 트라이를 만들고, 모든 패스에서 두 번째로 만나는 룰들을 제거하여 두 번째 트라이를 만든다. 모든 룰이 트라이에 포함될 때까지 남은 룰들에 대하여 같은 방법으로 트라이를 만드는 과정을 반복한다. 이렇게 해서 만들어진 여러 개의 트라이들은 어떠한 네스팅 관계도 갖지 않게 된다. 제안하는 구조는 이렇게 여러 개로 나누어진 별도의 BSL 트라이에 대하여 길이에 대한 이진검색을 수행한다. IPv4를 사용할 때 룰들의 네스팅 관계는 최대 32개까지 나타날 수 있다. 하지만 패킷분류의 룰들은 최대 5-6개 정도의 네스팅 관계를 갖고 있는 것으로 알려져 있으며, 뒤의 실험 결과에서 보이려는 바와 같이 실제의 룰 셋과 동일한 특성의 룰 셋을 제공하는 클레스벤치[6]를 이용하여 수행한 시뮬레이션 결과 역시 최대 6개의 네스팅 관계를 갖는 것으로 확인할 수 있었다. 그림 6은 표 1의 룰 셋을 사용하여 제안하는 BSL 트라이 구조를 나타낸 것이다. 그림 2의 AQT 구조에서 확인할 수 있듯이 주어진 룰 셋에는 3개의 네스팅 관계가 존재하므로 3개의 BSL 트라이가 만들어 지게 된다.

3.1.1 빌드

완전 해쉬 함수를 가정하였을 때 본 논문에서 제안하는 패킷분류 구조를 만드는 과정은 다음과 같다. 먼저 근원지 프리픽스와 목적지 프리픽스를 인터리빙하여 코드워드를 만든 후, 만들어진 코드워드를 사용하여 AQT 트라이를 만든다. 이때 같은 노드에 위치하게 되는 룰들은 링크드-리스트 (linked-list)로 연결하되 높은 우선순위에서 낮은 우선순위로 우선순위에 따라 연결하여 저장한다. 네스팅 관계에 있는 룰들이 서로 다른 트라이에 위치하도록 AQT 트라이의 루트 노드로부터 출발하여 모든 패스에 대하여 가장 처음 만나는 노드들을 길이의 대한 이진검색을 위한 해쉬 테이블로 옮긴다. 위의 과정을 AQT 트라이에 룰 노드가 더 이상 존재하지 않을 때까지 반복한다.

3.1.2 검색

제안하는 알고리즘의 검색 과정은 패킷이 들어오면 먼저 근원지 주소와 목적지 주소를 사용하여 코드워드(codeword)를 만들고, 그 코드워드를 해쉬 키로 하여 길이에 대한 이진검색을 수행하되 첫 번째 BSL 트라이에서 마지막 BSL 트라이로 진행한다. 길이에 대한 이진검색은 하나 이상의 룰이 저장된

레벨에 대해서만 수행된다. 하나의 노드에 저장된 모든 룰들은 룰의 우선순위에 따라 링크드-리스트로 연결되어 있으므로, 현재까지의 BMR이 노드에 저장된 룰보다 우선순위가 높은 경우, 현재 노드에서의 검색은 즉시 종료되고 다음 레벨로 진행된다. 검색중인 트라이에서 더 이상 진행할 레벨이 없는 경우, 검색은 다음 트라이로 이동한다.

3.1.3 룰 업데이트

룰 업데이트에는 삭제와 추가가 있다. 기존의 룰을 삭제하고자 하는 경우, 먼저 검색을 수행하여 해당하는 노드를 해쉬 테이블에서 찾은 뒤에 룰 테이블 그리고 룰 테이블의 링크드-리스트 (linked-list)에서 삭제한다. 룰을 추가하고 싶은 경우에도 먼저 검색을 수행하여야 한다. 만약 현재 검색중인 트라이에 일치하는 노드가 없다면, 추가되는 룰은 기존의 룰과 네스팅 관계가 없는 룰이므로, 해당 해쉬 테이블에 해쉬 값을 넣고, 룰 테이블에 추가하고자 하는 룰을 저장한다. 일치하는 엔트리가 있고 추가하고자 하는 룰의 코드워드 길이가 더 짧은 경우는 추가하려는 룰이 AQT 트라이 상에서 기존 엔트리의 상위에 위치하는 룰이므로 기존 엔트리를 대치하여 저장한다. 다음 트라이에 대하여 자리를 빼앗긴 엔트리를 위한 위의 과정을 반복한다.

3.2 Algorithm 2
(낮은 우선순위를 나중에 검색하는 알고리즘)

패킷분류는 인터넷 주소 검색과 다르게 룰들의 우선순위를 고려해가며 검색을 수행하여야 한다. 따라서 여러 개의 BSL 트라이 중에서 어떤 트라이를 먼저 검색하느냐에 따라서 검색의 성능이 결정된다. 만약 낮은 우선순위를 갖는 룰이 많은 트라이를 먼저 검색하였다면 검색 성능이 저하될 것이고, 높은 우선순위를 갖는 룰이 많은 트라이를 먼저 검색하였다면 검색 성능이 향상될 것이다. 이 같이 패킷분류가 룰의 우선순위에 따른다는 성질을 이용하여, 여러 개의 BSL 트라이 중 어떤 트라이를 먼저 검색하는 것이 좋은 검색 성능을 보일지를 고려하여 알고리즘 2와 알고리즘 3를 제안한다.

실제 사용되고 있는 룰 셋과 유사한 특성의 룰 셋을 제공하는 클레스벤치의 룰들의 분포를 살펴 본 결과 제안하는 알고리즘 1이 상대적으로 낮은 우선순위를 갖는 와일드카드를 먼저 검색하는 것을 알 수 있었다. 실험을 위해 사용된 룰 셋에서 트라이의 루트 노드에 저장되어야 하는 와일드카드의 수는 10개 이하로 적은 수를 갖는 것도 있었지만, 많게는 600여 개를 갖는 룰 셋도 있었으며, 대부분의 룰들은 최하위 우선순위에 주로 몰려있었다. 제안하는 알고리즘 1은 낮은 우선순위를 갖는 와일드카드를 제일 먼저 선형검색을 하게 되므로 검색 성능이 나빠질 수 밖에 없는 것으로 나타났다.

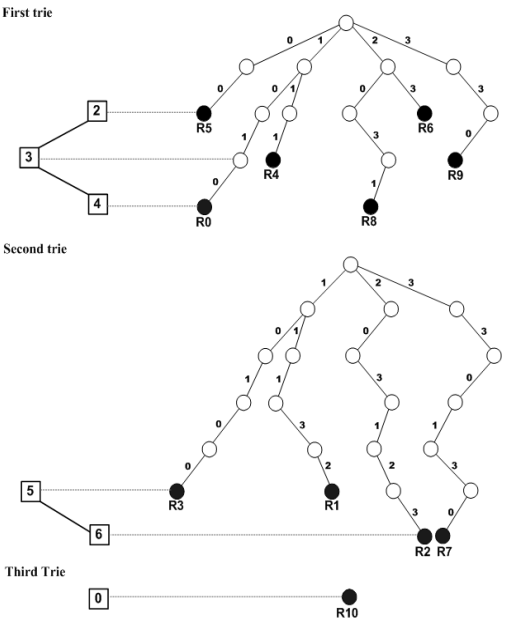

그림 7. 알고리즘 2

제안하는 알고리즘 2는 낮은 우선순위를 갖는 와일드카드를 가장 먼저 검색하였던 알고리즘 1과 달리, 와일드카드를 가장 나중에 검색하는 방법이다. 그림 7에 알고리즘 2의 트라이의 구조를 보였는데, 검색할 때 와일드카드 트라이를 가장 마지막에 검색한다는 것 외에 알고리즘 1과 다른 것이 없다. 하지만 많은 수의 선형 검색이 처음에 이루어졌던 것에 비해, 알고리즘 2에서는 상대적으로 높은 우선순위를 갖는 룰을 검색하여 일찍 검색을 종료할 수 있기 때문에 선계산이나 추가적인 메모리 소요 없이 성능을 향상시킬 수 있다. 앞서 언급된 바와 같이 하나의 노드에 저장된 복수의 룰들은 가장 우선순위가 높은 룰부터 낮은 룰의 순으로 우선순위에 따라 링크드-리스트로 연결되어 저장함을 가정한다. 알고리즘 2를 이용해 구성된 트라이를 예로 들어 설명하면, 두 번째 트라이까지의 검색 결과인 현재까지의 BMR이 와일드카드 트라이가 갖는 최우선순위에 해당하는 값보다 더 높은 우선순위를 갖는다면 검색은 바로

종료될 수 있다. 혹은 와일드카드 트라이에서 현재까지의 BMR 보다 낮은 우선순위를 갖는 룰과 만나게 되었을 때 바로 종료할 수 있다. 따라서, 많은 수의 선형 검색을 피할 수 없었던 알고리즘 1 보다 와일드카드의 선형 검색을 가장 나중에 하는 알고리즘 2를 이용하여 성능을 향상 시킬 수 있다.

3.3. Algorithm 3
(높은 우선순위를 먼저 검색하는 알고리즘)

패킷분류는 우선순위를 고려해야 하는 검색이므로, 상대적으로 높은 우선순위를 갖는 룰을 먼저 검색한다면 검색의 성능이 더욱 향상될 것으로 기대된다. 따라서 룰 셋의 우선순위 분포를 분석하여 좀 더 높은 우선순위를 갖는 룰들을 먼저 검색하는 알고리즘 3을 제안한다.

룰 셋을 분석해본 결과, 근원지와 목적지 프리픽스의 길이가 짧으면 낮은 우선순위를, 프리픽스의 길이가 길면 높은 우선순위를 갖는 경향이 있음을 알아낼 수 있었다. 따라서 상대적으로 긴 길이를 갖는 AQT 트라이의 리프들로 이루어진 BSL 트라이를 먼저 검색할 경우 성능 향상을 기대할 수 있다.

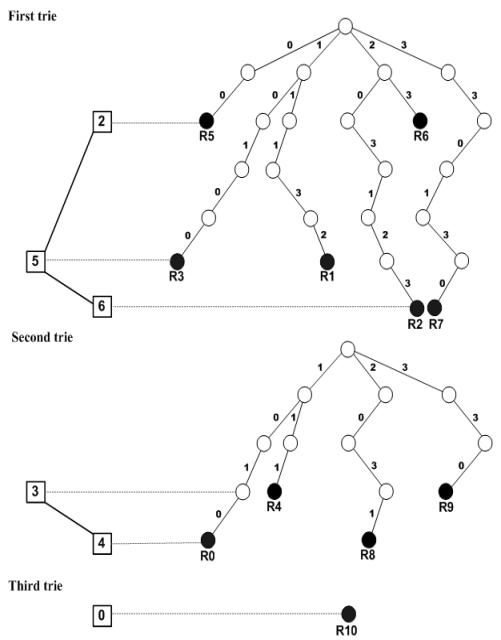

그림 8. 알고리즘 3

그림 8은 최장 길이 검색을 우선하기 위하여 제안하는 알고리즘 3을 나타낸 것이다. 알고리즘 1이 AQT 트라이 상에서 루트로부터 출발하여 가장 먼저 만나는 룰들로 첫 번째 트라이를 구성한 반면, 알고리즘 3은 AQT 트라이의 리프들로 첫번째 트라이를 구성하고, 리프가 제거된 AQT 트라이에서의 또 다른 리프들로 두 번째 트라이를 구성하는 과정을 반복한다. 알고리즘 3에서는 우선순위가 상대적으로 높은 룰을 먼저 검색하기 때문에 선계산이나 추가적인 메모리 소요 없이도 성능이 향상되며, 가장 긴 길이를 갖는 룰 셋의 집합으로 이루어진 BSL 트라이를 우선적으로 검색하여 보다 빠른 검색 속도를 얻는다. 또한, 알고리즘 2와 마찬가지로 대체적으로 낮은 우선순위를 가지며 많은 선형 검색을 요구하는 와일드카드 트라이를 가장 마지막에 검색함으로써, 많은 양의 선형 검색을 피하고, 이를 통해 가장 빠른 검색 성능을 보일 수 있다.

Ⅳ. 실험 결과

본 논문은 인터넷 주소 검색에서 매우 좋은 성능을 보이는 길이에 대한 이진검색을 패킷분류에 적용하여 길이에 대한 이차원 이진검색 알고리즘을 제안하였다. 실제 사용되고 있는 룰들과 매우 유사한 특성을 갖는 룰 셋을 제공하는 클래스벤치를 사용하여 제안하는 알고리즘의 성능을 평가하였으며, 기존의 알고리즘들과 성능을 비교하였다. 성능 평가에 사용된 룰 셋은 클래스벤치가 제공하는 ACL (access control lists), FW (firewalls), IPC (IP chains)의 특성에 따라 각각 약 1000개에서 약 5000개의 룰을 포함한다.

제안하는 3개의 알고리즘의 실험 결과를 표 2 ~ 표 4에 나타내었다. 각각의 표들은 제안하는 3개의 알고리즘에 대하여 룰의 개수 (N), 만들어진 BSL 트라이의 개수(N_t), 입력된 패킷을 분류하는 데 소요되는 평균 메모리 접근 회수(T_{avg}), 최대 메모리 접근 회수(T_{max}), BSL 트라이를 저장하는데 소요되는 메모리 크기(M_{tree})와 룰을 저장하는데 소요되는 메모리 크기(M_{filter})를 보인다.

표 2는 알고리즘1의 성능을 나타내었다. 알고리즘 1의 메모리 접근 회수가 기대했던 것보다 크게 나타났는데, 이는 프리픽스의 길이가 0인 와일드카드 룰들에 대한 선형 검색부터 진행하기 때문이다. 룰의 특성을 살펴 본 결과 근원지 프리픽스 혹은 목적지 프리픽스에 와일드 카드를 갖는 많은 룰들이 존재하지만 상대적으로 낮은 우선순위를 갖기 때문에 불필요한 메모리 접근을 야기시킴을 알 수 있었다.

표 2. 알고리즘 1의 성능

	N	N_t	T_{max}	T_{avg}	M_{tree}	M_{filter} (byte)
acl 1k	958	5	47	21.33	23000	20118
acl 5k	4659	6	78	34.67	84250	97860
fw 1k	870	3	293	192.66	3500	18291
fw 5k	4343	3	1002	560.67	4200	91371
ipc 1k	988	5	76	63.23	32340	20748
ipc 5k	4467	5	263	182.25	25530	93828

클레스벤치가 제공하는 룰의 특성을 살펴보았을 때, 와일드카드는 상대적으로 낮은 우선순위를 갖고 근원지 및 목적지 프리픽스의 길이가 긴 룰들이 상대적으로 높은 프리픽스를 갖는다는 것을 확인할 수 있었다. 표 3은 알고리즘 2의 결과로서 낮은 우선순위를 갖는 와일드카드를 제일 마지막으로 검색하여 검색의 효율을 높인 방법이다. 검색 성능이 알고리즘 1에 비하여 더 우수하게 나타나는 것을 확인할 수 있다.

표 3. 알고리즘 2의 성능

	N	N_t	T_{max}	T_{avg}	M_{tree}	M_{filter} (byte)
acl 1k	958	5	43	17.34	22990	20118
acl 5k	4659	6	59	20.10	84240	97860
fw 1k	870	3	286	115.95	3490	18291
fw 5k	4343	3	971	392.54	4200	91371
ipc 1k	988	5	71	33.38	32330	20748
ipc 5k	4467	5	233	65.18	25520	93828

패킷분류는 우선순위가 가장 높은 룰을 찾아야 하기 때문에, 상대적으로 더 높은 우선순위를 갖는 룰을 먼저 검색하는 것이 검색 성능을 향상시키게 된다. 이러한 특성을 이용하여 최장길이를 갖는 룰들부터 검색을 수행하는 최장길이 우선 검색을 수행하는 알고리즘 3에 대하여도 같은 실험을 수행하였다. 표 4는 알고리즘 3의 결과로서 상대적으로 높은 우선순위를 갖는 근원지 및 목적지 프리픽스가 긴 룰들을 먼저 검색하여 검색의 효율을 높인 방법이다. 표 4에서 확인 할 수 있듯이 상대적으로 우선순위가 높은 룰들을 먼저 검색하였기 때문에 알고리즘 1이나 2보다 좋은 성능을 보이는 것을 확인할 수 있다.

표 4. 알고리즘 3의 성능

	N	N_t	T_{max}	T_{avg}	M_{tree}	M_{filter} (byte)
acl 1k	958	5	21	12.54	23220	20118
acl 5k	4659	6	39	17.91	93830	97860
fw 1k	870	3	378	19.31	3780	18291
fw 5k	4343	3	804	67.02	4540	91371
ipc 1k	988	5	69	21.89	33880	20748
ipc 5k	4467	5	234	32.30	24530	93828

다음은 기존의 알고리즘들과 제안하는 알고리즘의 성능을 비교하는 실험을 수행하였다. 표 5는 약 1000개와 5000개의 룰을 가지는 ACL 타입의 룰 셋에 대하여 패킷을 분류하는 데 소요되는 평균 메모리 접근 회수(T_{avg}), 최대 메모리 접근 회수(T_{max}), 데이터 구조와 룰을 저장하는데 소요되는 총 메모리 크기(M)에 대한 성능을 보여주고 있다. 그림 9는 이러한 룰 셋에 대하여 기존의 알고리즘들과 제안하는 알고리즘의 평균 메모리 접근 횟수를 그래프로 비교하여 나타낸 것이다. 표와 그림에서 볼 수 있는 것처럼 제안하는 알고리즘이 메모리 소요량과 메모리 접근 회수 모두 좋은 성능을 보이는 것을 확인할 수 있다. 특히 와일드카드의 수가 적고 근원지와 목적지 프리픽스의 길이가 긴 룰이 많은 ACL의 특성 때문에 알고리즘 3의 성능이 다른 알고리즘들에 비해 매우 좋게 나타나는 것을 확인할 수 있었다. 또한 PQT 다음으로 적은 메모리 사용량을 나타내는 것을 확인할 수 있다.

표 5. ACL 타입에 대한 기존 알고리즘과 제안하는 알고리즘의 성능 비교

	acl 1k			acl 5k		
	T_{avg}	T_{max}	M (Kbyte)	T_{avg}	T_{max}	M (Kbyte)
H-trie [7]	77.17	124	82.91	84.01	177	401.48
Bit Vector [8]	66.03	68	153.29	64.1	76	2793.23
AQT [3]	38.59	64	56.38	50.11	94	200.22
PQT [9]	35.62	75	29.94	59.61	113	145.63
Propose d1	21.33	47	42.10	34.67	78	177.84
Propose d2	17.34	43	42.10	20.10	59	177.84
Propose d3	12.54	21	42.32	17.91	39	187.20

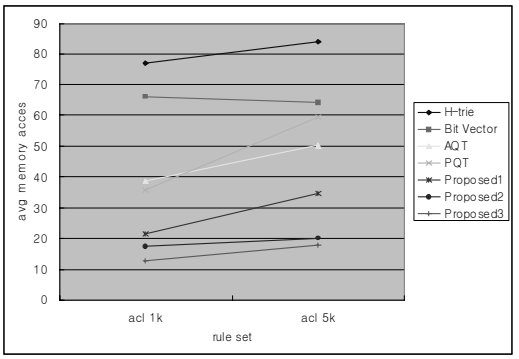

그림 9. ACL 타입에 대한 기존의 알고리즘과 제안하는 알고리즘의 성능 비교

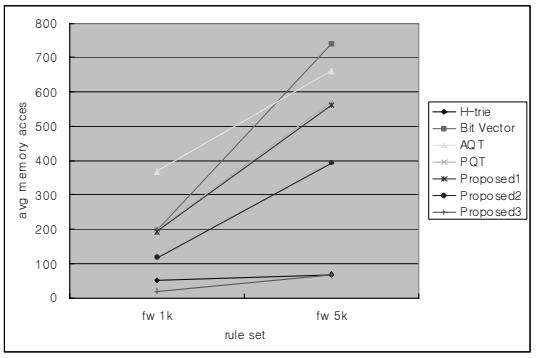

그림 10. FW 타입에 대한 기존의 알고리즘과 제안하는 알고리즘의 성능 비교

표 6은 약 1000개와 5000개의 룰을 가지는 FW 타입의 룰 셋에 대하여 패킷을 분류하는 데 소요되는 평균 메모리 접근 회수(T_{avg}), 최대 메모리 접근 회수(T_{max}), 데이터 구조와 룰을 저장하는데 소요되는 총 메모리 크기(M)에 대한 성능을 보여주고 있다. 그림 10은 이러한 룰 셋에 대하여 기존의 알고리즘들과 제안하는 알고리즘의의 평균 메모리 접근 횟수를 비교하여 나타낸 것이다. 표와 그림에서 나타난 것처럼 제안하는 알고리즘이 Bit-Vector, AQT 그리고 PQT보다 좋은 성능을 보이는 것을 확인할 수 있다. 하지만 H-trie의 성능이 다른 검색 알고리즘에 비해 월등히 좋게 나타난 것을 확인할 수 있다. 이것은 FW의 룰 특성이 긴 근원지 프리픽스와 짧은 목적지 프리픽스를 갖는 룰이 매우 많기 때문이다. 제안하는 알고리즘은 짧은 쪽까지 사용하여 만든 코드워드를 이용하여 검색을 수행하기 때문에 선형 검색의 양이 증가되어 제안하는 알고리즘의 성능이 저하되었다. 하지만 FW 타입에 대해서 제안하는 알고리즘이 가장 적은 메모리를 사용하는 것을 확인할 수 있다.

표 6. FW 타입에 대한 기존의 알고리즘과 제안하는 알고리즘의 성능 비교

	fw 1k			fw 5k		
	T_{avg}	T_{max}	M (Kbyte)	T_{avg}	T_{max}	M (Kbyte)
H-trie [7]	52.14	117	39.44	69.20	162	119.10
Bit Vector [8]	196.55	318	111.93	738.75	1044	2.34M
AQT [3]	369.34	444	35.25	660.52	1193	479.77
PQT [9]	197.92	293	27.22	571.06	999	135.97
Proposed1	192.66	293	21.28	560.67	1002	93.33
Proposed2	115.95	286	21.28	392.54	971	93.33
Proposed3	19.31	378	21.55	67.02	804	93.66

표 7은 약 1000개와 5000개의 룰을 가지는 IPC 타입의 룰 셋에 대하여 패킷을 분류하는 데 소요되는 평균 메모리 접근 회수(T_{avg}), 최대 메모리 접근 회수(T_{max}), 데이터 구조와 룰을 저장하는데 소요되는 총 메모리 크기(M)에 대한 결과를 보여주고 있다. 그림 11은 이러한 룰 셋에 대하여 기존의 알고리즘들과 제안하는 알고리즘의 평균 메모리 접근 횟수를 비교하여 나타낸 것이다. IPC의 특성은 ACL과 FW의 중간 정도의 성격을 갖는다. 따라서 제안하는 알고리즘의 성능 역시 ACL과 FW 타입의 중간 정도의 성능을 나타내는 것을 확인할 수 있었다. 룰 셋이 그리 크지 않았을 때는 가장 좋은 검색 성능을 나타내었으나, 룰 셋이 커졌을 때는 선형 검색의 양이 급격하게 증가하여 검색 성능이 다소 저하되어 최대 메모리 접근 횟수의 경우 H-trie에 이어 두 번째로 좋은 성능을 나타내었다. 룰의 크기가 작을 때는 두 번째로 적은 메모리 사용량을, 룰의 크기가 클 때는 가장 작은 메모리 사용량을 보였다.

여러 특성을 가지는 다양한 크기의 룰 셋을 이용하여 제안하는 알고리즘과 기존의 알고리즘들을 실험하였다. 기존의 알고리즘들의 룰의 특성에 따라 검색성능과 메모리 사용량이 많은 차이를 나타내지만 제안하는 알고리즘의 경우 모든 특성의 룰 셋에 대해 좋은 검색 성능을 보이며 메모리를 적게 소요하는 것을 확인할 수 있었다.

표 7. IPC 타입에 대한 기존의 알고리즘과 제안하는 알고리즘의 성능 비교

	Ipc 1k			Ipc 10k		
	T_{avg}	T_{max}	M (Kbyte)	T_{avg}	T_{max}	M (Kbyte)
H-trie [7]	71.91	128	121.57	85.64	192	224.70
Bit Vector [8]	63.58	80	154.32	151.86	230	2531.48M
AQT [3]	94.49	119	71.19	344.79	415	234.26
PQT [9]	73.58	106	30.88	202.07	295	139.96
Proposed1	63.23	76	54.22	182.257	263	116.56
Proposed2	33.38	71	54.21	65.18	233	116.56
Proposed3	21.891	69	56.66	32.30	234	115.58

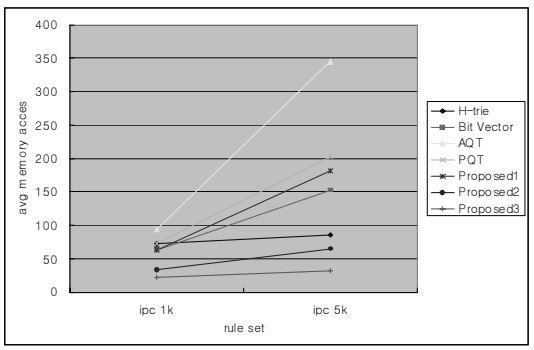

그림 11. IPC 타입에 대한 기존 알고리즘과 제안하는 알고리즘의 성능 비교

V. 결 론

인터넷에서의 품질보장 요구가 많아짐에 따라 라우터에서의 패킷분류 기능이 점점 중요해지고 있다. 효율적인 패킷분류를 위한 여러 가지 방법들이 활발히 연구되어 왔지만, 빠른 검색 속도와 작은 메모리 사용량을 모두 만족시키지 못하고 있다. 본 논문에서는 인터넷 주소 검색에서 매우 좋은 성능을 보이는 길이에 대한 이진검색을 패킷분류에 적용하는 길이에 대한 2차원 이진 패킷분류 구조를 제안하였다. 제안하는 방법은 길이에 대한 이진검색을 사용하여 메모리 접근 횟수를 줄였으며, 가장 높은 우선순위를 갖는 룰을 찾아야 하는 패킷분류의 특성을 고려하여 검색을 더욱 효율적으로 진행할 수 있게 하였다. 제안하는 구조는 룰들의 네스팅 관계를 없애기 위하여 여러 개의 트라이를 만드는데, 클래스벤치를 사용한 실험 결과 최대 경우 6개, 평균적으로 5개 이하의 트라이가 만들어지는 것을 확인하였다. 제안하는 알고리즘은 5000여개의 룰을 갖는 ACL 룰 셋의 경우 평균 메모리 접근이 17.91로 기존의 알고리즘들 보다 매우 뛰어난 성능을 보였으며, 메모리 사용량은 187.20KB로 매우 적은 양의 메모리를 사용하는 것으로 나타났다. 또한 룰의 특성에 따라 검색 성능이 달라지는 기존의 알고리즘들과 달리 제안하는 알고리즘은 다양한 특성의 룰 셋에 대해 좋은 검색 성능을 보이면서도 메모리를 적게 소요하는 것을 확인할 수 있었다.

참 고 문 헌

〔1〕 H. Jonathan Chao, "Next generation routers," Proceedings of the IEEE, Volume 90, Issue 9, pp. 1518 1558, Sept. 2002.

〔2〕 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.

〔3〕 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.

〔4〕 M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in Proc. ACM SIGCOMM1997, Cannes, France, pp. 25-35, 1997.

〔5〕 추하늘, 임혜숙, "최장길이 우선검색에 기초한 프리픽스 길이에 따른 이진 IP 검색구조," 한국통신학회 논문지, vol .31, no.8B, pp.691-700, Aug. 2006.

〔6〕 D.E. Taylor and J.S. Turner, "ClassBench: a packet classification benchmark,", INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE Volume 3, vol.3, pp. 2068-2079, 13-17 March 2005.

〔7〕 P. Gupta and N. Mckeown, "Algorithms for packet classification," IEEE Network, vol.15, no. 2, pp.24-32, Mar./Apr. 2001.

〔8〕 T.V. Lakshman and D. Stildialis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *in Proc. ACM SIGCOMM1998*, Sep. 1998.

〔9〕 Hyesook Lim, Min Young Kang, and Changhoon Yim, "Two-dimensional Packet Classification Algorithm Using a Quad-Tree," Computer Communications, vol. 30, no.6 pp. 1396-1405, March 2007

문 주 형 (Ju Hyoung Mun) 준회원

2005년 2월 이화여자대학교 정보통신학과, 학사

2005년 3월~현재 이화여자대학교 정보통신학과, 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계

임 혜 숙 (Hyesook Lim) 정회원

1986년 2월 서울대학교 제어계측공학과, 학사

1986년 8월~1989년 2월 삼성 휴렛 팩커드, 연구원

1991년 2월 서울대학교 제어계측공학과, 석사

1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering, Ph.D.

1996년 11월~2000년 7월 Lucent Technologies, Member of Technical Staff

2000년 7월~2002년 2월 Cisco Systems, Hardware Engineer

2002년 3월~현재 이화여자대학교 공과대학 정보통신학과 부교수

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