패킷 분류를 위한 계층 이진 검색 트리

준회원 추 하 늘*, 정회원 임 혜 숙*

Hierarchical Binary Search Tree (HBST) for Packet Classification

Ha Neul Chu* *Associate Member*, Hyesook Lim* *Regular Member*

요 약

네트워크 상에서 정책 기반의 라우팅이나 품질보장(Quality of Service)과 같은 새로운 서비스들을 제공하기 위해서 인터넷 라우터는 패킷을 여러 개의 플로우로 분류하고 각 플로우에 대하여 서로 다른 처리를 해주어야 하는데, 이를 패킷 분류라 한다. 패킷 분류 기능은 초당 수백 기가 비트의 속도로 입력되는 모든 패킷에 대하여 선속도(wire-speed)로 처리되어야 하므로 인터넷 라우터 내에서 새로운 병목점으로 작용하고 있다. 따라서 빠른 속도의 패킷 분류 구조의 필요성이 대두되고 있는데 본 논문에서는 계층 트리를 이용한 패킷 분류 구조를 제안한다. 제안하는 구조는 빈 노드를 갖지 않는 이진 검색 트리를 계층적으로 연결하여 패킷 분류를 수행하는 구조로서, 메모리 효율성을 높이고 메모리 접근 횟수를 줄임으로써 검색 성능을 향상시킨 구조이다.

Key Words : Packet Classification, QoS, Hierarchical Trie, Binary Search Tree, HBST

ABSTRACT

In order to provide new value-added services such as a policy-based routing and the quality of services in next generation network, the Internet routers need to classify packets into flows for different treatments, and it is called a packet classification. Since the packet classification should be performed in wire-speed for every packet incoming in several hundred giga-bits per second, the packet classification becomes a bottleneck in the Internet routers. Therefore, high speed packet classification algorithms are required. In this paper, we propose an efficient packet classification architecture based on a hierarchical binary search tree. The proposed architecture hierarchically connects the binary search tree which does not have empty nodes, and hence the proposed architecture reduces the memory requirement and improves the search performance.

Ⅰ. 서 론

인터넷 기술의 급속한 발전과 새로운 응용 프로그램들의 빠른 확산으로 인하여 빠른 속도의 데이터 송수신뿐만 아니라, 입력된 패킷을 특정 클래스로 구분하여 품질 보증(Quality of Service, QoS)을 제공하는 서비스의 필요성이 대두되고 있다. 이러한 새로운 서비스들은 입력된 패킷을 미리 정의된 룰에 따라 플로우로 구분하고, 각 플로우에 해당하는 정책에 따라 각각 다른 서비스를 제공할 수 있도록 하는 패킷 분류(packet classification)를 통해 제공된다. 패킷 분류의 어려움에는 크게 두 가지 요인이 존재한다. 첫째로, 패킷 분류는 패킷 헤더의 한 필드만을 고려하는 것이 아니라 여러 필드를 동시에 고려하여 모든 필드와 일치하는지를 확인해야 하며, 일치 가능한 여러 개의 룰 가운데서 우선순위가 가장 높은(high

※ This research was partially supported by the MIC(Ministry of Information and Communication), Korea, under the HNRC-ITRC (Home Network Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).
* 이화여자대학교 정보통신학과 SoC설계연구실(hlim@ewha.ac.kr)
논문번호 : KICS2007-02-054, 접수일자 : 2007년 2월 9일, 최종논문접수일자 : 2007년 3월 13일

priority) 룰을 최종적으로 검색하여 룰에 정의된 정책(policy)에 따라 패킷을 처리해야 한다. 두 번째로, 패킷 분류는 초당 수백 기가 비트 이상의 속도로 입력되는 모든 패킷에 대하여 선속도(wire-speed)로 처리되어야 한다.

패킷 분류를 위해 정의된 클래스는 패킷 헤더에 포함된 몇 개의 필드로 구성된 룰로 이루어진다. 이 룰은 대체로 패킷 헤더의 근원지 IP 주소, 목적지 IP 주소, 근원지 포트 번호, 목적지 포트 번호, 프로토콜 타입의 5개의 필드로 이루어진다. 패킷 헤더의 각 필드의 값은 크게 다음 세 가지의 방법을 통해 일치 여부를 결정하게 된다. 첫 번째는, 확정 일치(exact match)로 인풋 헤더의 필드가 같은 길이의 특정한 룰의 필드와 완벽히 일치하는 경우이다. 두 번째는 프리픽스 일치(prefix match)로서 이것은 패킷의 필드가 특정 룰에 정의된 프리픽스 길이에 해당하는 만큼만 일치하는 방법이다. 마지막으로, 영역 일치(range search)가 있는데 이것은 패킷 헤더 필드의 값이 룰에 정의된 특정 영역 내에 속하는지 확인함으로써 일치 여부를 판단한다.

본 논문에서는 프리픽스로 표현되는 두 필드에 대해 프리픽스 일치 방법을 통하여 일차적으로 일치 가능한 룰을 결정하고, 이들 가운데서 나머지 필드의 값과도 일치하며 가장 우선순위가 높은 룰을 최종적으로 결정하는 이차원 검색에 기초한 계층 이진 트리 구조를 제안한다.

본 논문의 구성은 다음과 같다. II 장에서 기존의 패킷 분류에 대한 연구들을 간단히 살피고, III장에서는 제안하는 이차원 계층 이진 트리에 대해 설명한다. IV장에서는 제안하는 계층 이진 트리의 성능 평가 결과를 밝히고, 기존의 패킷 분류 구조와 제안하는 계층 이진 트리의 성능을 비교한다. 마지막으로 V장에서 간단한 결론을 맺는다.

Ⅱ. 기존의 구조

2.1 계층 트라이(Hierarchical Trie[1])

이 구조는, 룰을 구성하는 필드들 중에서 프리픽스로 표현된 근원지 주소와 목적지 주소를 이용해 각각의 필드들에 대해서 별도의 트라이를 구성하고 이것을 근원지 주소에서 목적지 주소로 계층적으로 연결하여 패킷 분류를 가능하게 한 구조이다. 첫 번째 필드인 근원지 주소로 이루어진 트라이에서 일치되는 모든 엔트리에 대해 계층적으로 연결된 하위 필드 트라이로 검색을 계속 진행하는 방식이다. 이 구조에서는 일치하는 모든 룰을 찾기 위한 백-트랙킹(back-tracking)이 요구된다. 이것은 패킷 분류가 주어진 패킷의 여러 필드에 대해 일치 가능한 모든 룰 중에서 우선순위가 가장 높은(high priority) 룰에 따라 클래스를 결정하기 때문이다. 이러한 백-트랙킹을 피하기 위하여 룰의 복사나 스위치 포인터를 이용하여 검색 시간의 복잡도를 단축시킨 셋 프루닝 트라이(set-pruning trie[2])와 그리드 트라이(grid-of-trie[3])가 제안되었다. 그러나 위 두 구조의 경우, 검색 시간을 줄인 대신에 룰의 복사나 스위치 포인터의 저장으로 인하여 메모리 요구량이 증대 되었고, 선-처리(pre-processing)로 인하여 새로운 룰의 부가적

표 1. 분류테이블의 예

Rule Number	근원지 주소	목적지 주소	근원지 포트	목적지 포트	프로토콜
0	1000*	*	53, 53	443, 443	17
1	111*	101*	53, 53	25, 25	6
2	*	10*	53, 53	25, 25	17
3	101*	11*	67, 67	5632, 5632	6
4	01*	0011*	1024, 65535	1024, 65535	6
5	101*	11*	53, 53	25, 25	4
6	11*	0100*	0, 65535	5632, 5632	6
7	101*	1011*	0, 65535	5632, 5632	6
8	*	10*	53, 53	25, 25	6
9	11*	11*	0, 15576	2783, 2783	4
10	010*	00*	53, 53	443, 443	17
11	11*	00*	53, 53	25, 25	6
12	101*	0*	0, 65535	5632, 5632	6
13	01*	1*	53, 53	443, 443	17
14	01*	0*	0, 65535	5632, 5632	6

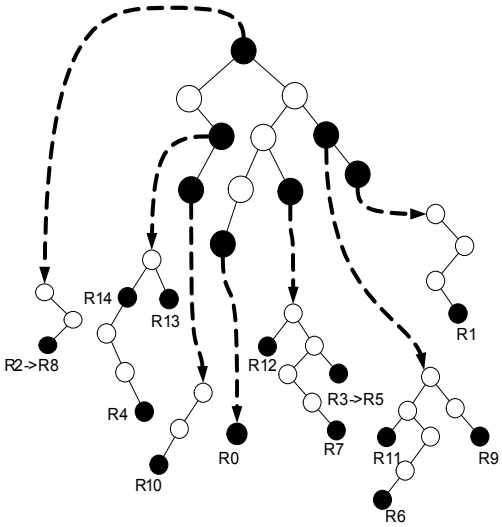


그림 1. 표 1의 분류테이블을 이용해 구성된 이차원 계층 이진 트라이의 예

인 추가가 어렵다는 단점이 있다.

표 1의 패킷 분류테이블은 실제 라우터에서의 룰 특성을 반영하는 클래스벤치(classbench[4],[5])를 통해 구성된 분류테이블에서 15개의 룰을 택해서 구성한 예시이다. 분류테이블에서 룰 번호가 작을수록 높은 우선 순위를 갖는 것으로 가정한다. 그림 1은 표 1의 분류테이블을 사용하여 이차원 계층 트라이를 구성한 예이다. 그림에서 흰 색 노드는 내부의 빈 노드(empty node)를, 검은 색 노드는 유효한 프리픽스 노드를, 점선은 두 번째 필드 트라이를 가리키는 다음 트라이 포인터를 나타낸다. 그림 1에서 보여지듯이, 계층 트라이 구조는 근원지 주소로 구성된 첫 번째 필드 트라이에서 빈 노드가 존재하므로 메모리 효율성이 떨어지게 되며, 이로 인한 메모리 접근 횟수도 증가하게 된다. 목적지 주소로 구성된 두 번째 필드 트라이는 첫 번째 필드의 유효한 노드마다 별도의 이진 트라이로서 존재하게 되고, 각각의 노드에는 같은 근원지 주소와 목적지 주소를 갖는 룰들이 저장되게 된다. 두 번째 필드 이진 트라이 역시 빈 노드가 존재하므로 전체적인 메모리 효율성은 더욱 떨어지게 된다.

2.2 비트 벡터(Bit Vector[6])

이 구조는 프리픽스 형태로 표현되는 근원지 주소와 목적지 주소를 이용해서 독립적인 이진 트라이를 구성하고, 각각의 필드 트라이의 프리픽스 노드마다 주어진 룰 셋(rule set)의 사이즈에 해당하는 비트 벡터를 갖는 구조이다. 각 필드 트라이의 유효한 프리

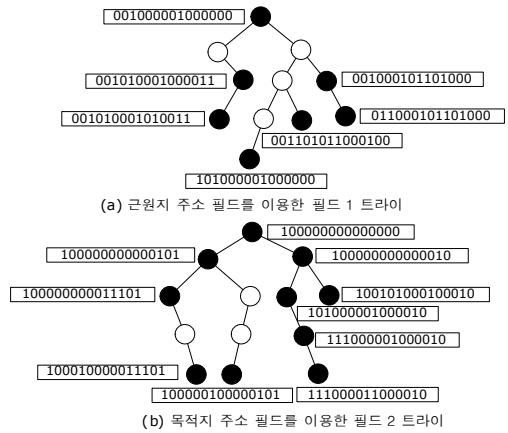


그림 2. 표 1의 분류기를 이용하여 구성한 비트 벡터 알고리즘의 예

픽스 노드의 비트 벡터는 그 노드에 일치 가능한 프리픽스를 갖는 룰의 존재 유무에 해당하는 정보를 나타낸다. 비트 벡터의 값은 유효한 프리픽스 노드에 일치 가능한 룰의 번호에 해당하는 인덱스인 경우에는 1로, 아닌 경우는 0으로 결정된다. 프리픽스들 사이에는 네스팅 관계가 존재하므로, 트라이의 리프에 존재할수록 비트 벡터의 값이 1인 인덱스의 수가 늘어나게 된다. 주어진 인풋 헤더의 두 필드의 주소의 값을 각각의 해당하는 트라이에서 한 비트씩 비교하면서 최장 길이 일치(longest-match)에 해당하는 노드를 찾아 그 노드에 저장되어 있는 비트 벡터를 구한다. 그 후에, 두 비트 벡터를 AND 연산을 통해서 새로운 비트 벡터를 계산한다. 이렇게 새로 계산된 비트 벡터에서 1로 셋 된 모든 인덱스에 해당하는 룰과 인풋 헤더의 나머지 필드들을 순차적으로 비교하면서 최종적으로 가장 높은 우선순위를 갖는 룰을 검색한다. 이 구조 역시 트라이에 기초한 구조이므로 각 트라이의 내부에 빈 노드가 존재하게 되는 것을 피할 수 없으며, 이로 인해 메모리의 효율성이 떨어지게 된다. 또한, 각 노드에 저장되는 비트 벡터의 크기는 룰 셋(rule set)의 크기에 비례하게 된다. 따라서 룰 셋이 큰 경우에 대해서는 비트 벡터의 크기가 매우 커지게 되므로 메모리의 효율성이 더욱 현저히 감소하게 된다. 이러한 점을 보완하기 위해 여러 비트를 한 번에 묶어서 표현하는 비트 벡터를 이용하여 비트 벡터의 크기를 줄인 집합 비트 벡터(aggregated bit vector[7]) 구조가 제안되었다. 또한, 룰 셋의 그룹핑을 통해 비트 벡터의 크기를 줄이는 그룹 비트 벡터(group bit vector[8]) 구조가 제안되기도 하였다. 그림 2는 표 1에서 주어진 분류테이블의

에를 이용해 구성한 비트 벡터 구조를 나타낸다.

2.3 영역분할 사분 트리(Area-based Quad Tree, AQT[9])

영역분할 사분 트리는 이차원 이진 트라이 구조로써 각 필터가 갖는 프리픽스 형태로 표현된 근원지 주소와 목적지 주소의 이차원 프리픽스를 하나의 평면상의 직사각형 영역으로 정의한다. 이 때 직사각형의 위치와 크기는 근원지 주소와 목적지 주소의 값과 길이에 따라 결정된다. 검색의 매 단계에서 근원지 주소와 목적지 주소는 각각의 영역을 0, 1의 값으로 이등분하기 때문에, 두 프리픽스에 의해 표현된 정사각형 영역이 00, 01, 10, 11의 4영역으로 분할된다. AQT에서는 각 주소 필드의 한 비트씩을 가지고 영역을 분할하므로, 같은 길이의 프리픽스로만 표현되게 된다. 하지만, 표 1의 분류 테이블에서 볼 수 있듯이 룰을 구성하는 두 프리픽스는 서로 다른 길이를 가질 수 있다. 이 문제를 해결하기 위해 AQT에서는 크로싱 필터 셋(crossing filter set, CFS)이라는 새로운 개념을 제안하였다. 이것은 평면에서 직사각형으로 정의된 룰 영역의 한 필드의 길이가 평면 영역의 필드 길이와 일치하면, 직사각형으로 정의된 룰을 그 영역에 해당하는 최적의 필터로 정의하는 것이다. 이렇게 정의된 CFS는 AQT의 각 노드에 저장된다. AQT의 경우도 트라이 구조에 기반을 두고 있기 때문에 내부에 빈 노드가 반드시 존재하게 되며, 프리픽스의 길이가 길어질수록 메모리의 비효율성은 심화된다. 또한, 트라이 내부의 빈 노드로 인해 검색을 위한 메모리 접근이 많아진다. AQT는 검색하는 매 단계마다 프리픽스로 표현된 두 필드를 동시에 살피므로, 전체 검색 범위를 1/4로 줄여가며 패킷이 해당하는 영역 내의 필터들을 검색하게 된다. 또한 프리픽스 길이에 대해서는 선형적인 검색이 이루어지게 되므로 최대 프리픽스 길이만큼의 검색 시간이 요구된다. 그림 3은 표 1의 분류테이블에 대하여 영역분할 사분 트리(AQT)를 구성한 예이다.

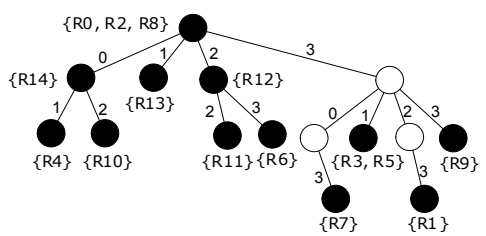


그림 3. 표 1의 분류테이블을 이용하여 구성한 영역분할 사분 트리(AQT)의 예

앞서 소개된 검색 방법들은 트라이 구조의 최대 단점인 비어있는 노드를 저장하는 데서 오는 메모리의 비효율성의 한계를 극복하지 못하고 있다. 본 논문에서는 패킷 분류를 위하여 두 프리픽스 필드를 계층적으로 연결한 이차원 검색 구조를 제안한다. 제안하는 구조는 중간에 빈 노드를 포함하는 트라이 구조 대신 빈 노드를 포함하지 않는 트리 구조를 채택함으로써 이차원 검색을 위한 메모리 효율을 극대화하면서 이진 검색이 가능한 계층 이진 트리 구조이다.

Ⅲ. 제안하는 구조

기존의 계층 이진 트라이의 경우, 각 필드로 구성된 트라이 내부에 빈 노드가 많이 존재하게 되므로 메모리의 효율성이 떨어진다. 본 논문에서 제안하는 구조는 이러한 단점을 프리픽스로 이루어진 두 필드의 이진 트라이에서 내부 빈 노드를 제거하여 메모리의 효율성을 높인 이진 트리로 구성하여 해결하였다. 제안하는 구조를 설명하기에 앞서, 일차원 이진 트리에 대해 간단히 살펴보기로 한다.

3.1 일차원 이진 트리 (One-dimensional Binary Search Tree, BST[10])

이 구조는, 일차원 이진 트라이의 내부에 존재하는 빈 노드를 제거하고, 프리픽스들 사이의 이진 검색을 가능하게 한 구조이다. 프리픽스간의 이진 검색이 어려웠던 것은, 고정된 길이를 갖는 프리픽스들을 비교하는 것이 아니라, 서로 다른 길이를 갖는 프리픽스의 크기 비교가 어렵기 때문이다. 또한, 프리픽스 길이의 가변성으로 인해 프리픽스 사이에 네스팅 관계가 형성됨으로 인하여 이진 검색이 불가능하였다. 따라서 이 구조에서는 다른 길이를 갖는 프리픽스들 사이의 크기 비교를 다음과 같이 정의하고, 네스팅 관계에 있는 프리픽스를 정의함으로써 이진 검색이 가능하게 하였다.

두 개의 서로 다른 길이를 갖는 프리픽스가 있을 때, 짧은 쪽 프리픽스 길이까지의 값을 비교하여, 프리픽스 값이 더 큰 쪽이 큰 프리픽스로 정의된다. 짧은 쪽 프리픽스 길이까지의 값이 같으면, 긴 프리픽스의 다음 비트의 값이 0인 경우에는 짧은 프리픽스가 큰 프리픽스로, 1인 경우에는 긴 프리픽스가 큰 프리픽스로 정의된다. 예를 들어, 100*과 0101*을 비교하여 보면, 짧은 쪽 프리픽스 길이인 처음 세 비트를 비교하여 100*이 0101*보다 큰 프리픽스로 정의된다. 100*과 10001*을 비교해 보면, 짧은

쪽 프리픽스 길이까지의 값이 같으므로, 긴 프리픽스의 다음 비트를 보고 크고 작음을 결정한다. 이 경우, 긴 프리픽스의 다음 비트가 0이므로, 100*이 10001*보다 큰 프리픽스이다. 100*과 10010*을 비교해보면, 짧은 쪽 프리픽스 길이까지의 값이 같으므로 긴 프리픽스의 다음 비트를 확인하고, 이 값이 1이므로 긴 프리픽스인 10010*이 큰 프리픽스로 정의된다.

이렇게 크기에 따라 정렬된 프리픽스 리스트에 대해 이진 검색을 수행하면 입력된 주소와 최장 길이 일치하는 프리픽스(longest prefix match)의 검색에 실패하게 된다. 예를 들어, 어떤 프리픽스 A를 BMP(best matching prefix)로 갖는 주소가 입력될 때, A를 프리픽스로 갖는 다른 프리픽스가 존재하고, 이 프리픽스가 A보다 먼저 비교되는 경우에는 A를 포함하고 있지 않은 다른 쪽 리스트로 검색이 진행되어 주어진 입력과 최장 길이 일치 프리픽스를 찾을 수 없게 된다. 이러한 문제를 해결하기 위해서 이진 트리 구조에서는 프리픽스들을 디스조인트(disjoint)와 인클로저(enclosure), 인클로즈드(enclosed) 프리픽스로 분류한다. 인클로저 프리픽스는 자신을 프리픽스로 하는 또 다른 프리픽스가 존재하는 프리픽스이고, 인클로즈드 프리픽스는 인클로저 프리픽스를 프리픽스로 갖는 프리픽스이다. 디스조인트 프리픽스는 인클로저도, 인클로즈드도 아닌 프리픽스이다. 주소가 입력되면, 디스조인트 프리픽스와 인클로저 프리픽스만으로 이루어진 리스트에 대해 이진 검색을 먼저 수행한다. 이 때, 입력된 주소가 인클로저 프리픽스와 비교되는 경우에는, 이 인클로저를 프리픽스로 갖는 인클로즈드 프리픽스가 리스트에 추가되는 방식으로 이진 프리픽스 트리가 구성된다.

이 구조는 빈 노드가 존재하지 않는 트리를 구성

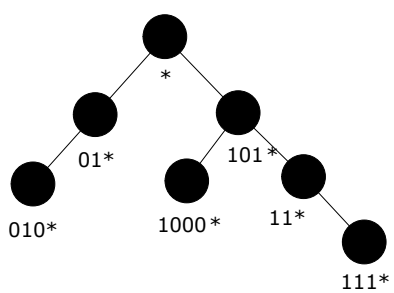


그림 4. 표 1 분류 테이블의 근원지 주소를 이용하여 구성한 일차원 이진 트리

함으로써 프리픽스들만으로 이루어진 이진 트리를 구성할 수 있는 장점을 갖는다. 하지만, 인클로저 프리픽스가 인클로즈드 프리픽스보다 항상 상위 레벨에 존재하여야 하므로 구성된 프리픽스 트리의 깊이가 프리픽스 네스팅 관계에 따라서 매우 깊어질 수 있는 단점이 있다. 그림 4는 표 1의 분류테이블에서 근원지 주소만을 이용하여 구성한 일차원 이진 트리의 예를 보여준다.

3.2 이차원 계층 이진 트리 (Two-dimensional Hierarchical Binary Search Tree, HBST)

3.2.1 이진 트리 테이블 빌드

우선 프리픽스 형태로 표현된 근원지 주소를 이용해서, 첫 번째 트리를 구성한다. 이 때, 첫 번째 필드에 대한 이진 트리가 구성되므로 모든 노드는 프리픽스 노드로서 유효하다. 이 유효한 프리픽스 노드에 룰의 두 번째 필드에 해당하는 목적지 주소 프리픽스를 이용하여 계층적으로 두 번째 필드 이진 트리를 구성한다. 프리픽스로 표현된 두 개의 필드에 대해서 같은 프리픽스 길이와 프리픽스 값을 갖고 나머지 세 필드에 대해 다른 값을 갖는 룰이 존재할 수 있으므로 이러한 룰 중에서 가장 높은 순위를 갖는 룰에 대한 정보를 노드에 저장해둔다. 그림 5에 패킷 분류를 위해 제안하는 계층 이진 트리 구조를 보였다.

제안하는 구조는 두 개의 테이블에 의하여 구성되는데 제안하는 패킷 분류 테이블의 구조를 그림 6에 보였다. 먼저 필드 1 테이블은 첫 번째 근원지 주소로 구성되며, 필드 2 테이블은 두 번째 필드인 목적지 주소로 구성되나, 기본적으로 분류테이블의 모든 필드를 포함하게 된다. 각각의 테이블의 엔트리 구조는 그림 7과 같다. 필드 1 테이블은 먼저 해당 엔트리가 유효한지를 알려주는 유효 엔트리 비트, 프리픽스 값, 프리픽스 길이, 왼쪽 노드 포인터, 오른쪽 노드 포인터, 그리고 다음 트리 포인터를 갖는다. 여기서 다음 트리 포인터는 첫 번째 이진 트리의 유효한 노드에서 두 번째 이진 트리의 루트 노드로의 검색 확장을 위해 사용되며, 이 값은 필드 2 테이블의 해당 엔트리 주소가 된다.

필드 2 테이블은 필드 1에서의 유효 노드에 계층적으로 연결된 두 번째 목적지 주소, 근원지 포트, 목적지 포트, 프로토콜의 네 필드로 이루어진 룰의 정보들을 갖는다. 구체적으로 살펴보면 유효 엔트리 비트, 프리픽스 값, 프리픽스 길이, 근원지 포트 시작 번호, 근원지 포트 끝 번호, 목적지 포트 시작 번

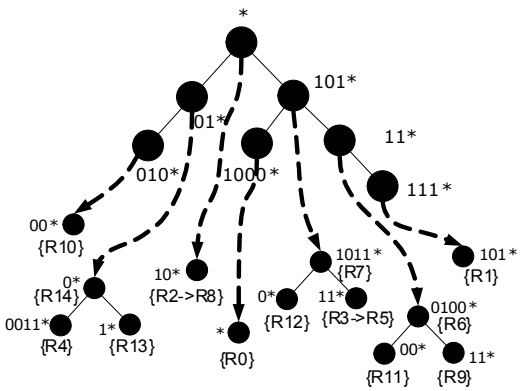


그림 5. 제안하는 이차원 계층 이진 트리

호, 목적지 포트 끝 번호, 프로토콜 와일드, 프로토콜 타입, 왼쪽 노드 포인터, 오른쪽 노드 포인터, 그리고 룰 번호, 연결 리스트 포인터를 갖는다. 룰 번호는 테이블의 엔트리 정보에 해당하는 룰 번호를 갖고 있는 것이다. 연결 리스트 포인터는 목적지 주소까지는 같고 포트와 프로토콜이 다른 경우에 해당하는 룰에 대해서도 검색을 할 수 있게 한 것으로, 필드 2 테이블의 엔트리 주소를 나타낸다. 인터넷 상에서의 많은 트래픽은 근원지와 목적지의 주소뿐만 아니라 포트와 프로토콜의 다양함에 의해 생성될 수 있는 여러 개의 플로우가 존재하므로, 들어온 패킷은 이 모든 경우에 대해 일치할 가능성이 있다. 따라서 일치 가능한 모든 플로우를 검색하여 최우선순위 룰을 검색하여야 하기 때문에, 우선순위가 높은 룰부터 낮은 룰로 연결된 연결 리스트를 이용해 검색을 할 수 있게 하였다.

3.2.2 검색

패킷 분류를 위한 검색은 다음의 과정을 통해 이루어진다. 주어진 패킷의 근원지 주소를 필드 1 트리의 루트에 해당하는 필드 1 테이블 엔트리에 저장된 프리픽스와 비교하고, 그것보다 크면 오른쪽 포인터를, 작으면 왼쪽 포인터를 따라 가면서 필드 1 테이블을 이진 검색한다. 접근한 엔트리의 프리픽스와 일치하는 경우에는, 필드 1 트리에서 유효한 노드에 일치한 것과 동일하므로 필드 2 트리로 검색 영역을 확장한다. 이 때, 필드 1 테이블의 다음 트리 포인터 정보를 이용하여 필드 2 트리의 루트 노드에 해당하는 테이블 엔트리에 접근한다. 필드 2 트리에서는 목적지 주소뿐만 아니라, 근원지 포트, 목적지 포트, 프로토콜의 정보까지 함께 갖고 있어 동시에 비교 가능하게 하였다. 또한, 앞서 언급한 대로, 인터넷의

(a) 필드 1 테이블

Entry	valid	prefix	length	Left	Right	Next
0	1	*	0	1	3	0
1	1	01*	2	5	-	1
2	1	1000*	4	-	-	4
3	1	101*	3	2	4	6
4	1	11*	2	-	6	9
5	1	010*	3	-	-	11
6	1	111*	3	-	-	12

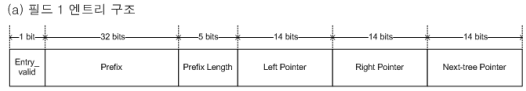
(b) 필드 2 테이블

Entry	valid	Prefix	length	srcPort Start	srcPort End	dstPort Start	dstPort End	Protocol Wild	Prototype	Left Pointer	Right Pointer	Rule Number	srcPort End
0	1	10*	2	53	53	25	25	1	17	-	-	2	14
1	1	0*	1	0	65535	5632	5632	1	6	3	2	14	-
2	1	1*	1	53	53	443	443	1	17	-	-	13	-
3	1	0011*	4	1024	65535	1024	65535	1	6	-	-	4	-
4	1	*	0	53	53	443	443	1	17	-	-	0	-
5	1	0*	1	0	65535	5632	5632	1	6	-	-	12	-
6	1	1011*	4	0	65535	5632	5632	1	6	5	7	7	-
7	1	11*	2	67	67	5632	5632	1	6	-	-	3	13
8	1	00*	2	53	53	25	25	1	6	-	-	11	-
9	1	0100*	4	0	65535	5632	5632	1	6	8	10	6	-
10	1	11*	2	0	15576	2788	2783	1	4	-	-	9	-
11	1	00*	2	53	53	443	443	1	17	-	-	10	-
12	1	101*	3	53	53	25	25	1	6	-	-	1	-
13	1	11*	2	53	53	25	25	1	4	-	-	5	-
14	1	10*	2	53	53	25	25	1	6	-	-	8	-

그림 6. 표 1의 분류테이블의 예시를 이용하여 구성된 계층 이진 트리 테이블

많은 트래픽에는 근원지 주소와 목적지 주소뿐 아니라, 포트와 프로토콜의 다양함에 의해 생겨날 수 있는 많은 트래픽의 클래스가 존재한다. 따라서 주소는 같더라도 포트와 프로토콜의 값이 다름에 의해 일치 가능한 많은 룰이 검색될 수 있으며 이 중에서도 가장 높은 우선순위를 갖는 룰을 찾아야 하므로 일치 가능한 모든 룰에 대해 검색이 가능해야 한다. 이를 위해 필드 2 테이블에는 룰을 구성하는 4가지의 필드 값과 왼쪽, 오른쪽 포인터와 함께 일치 가능한 모든 룰의 검색을 위한 연결 리스트 포인터를 이용한다.

(a) 필드 1 엔트리 구조



(b) 필드 2 엔트리 구조

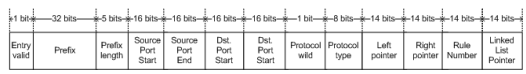


그림 7. 제안하는 테이블의 엔트리의 구조

필드 2 테이블의 검색은 필드 1에서의 검색 진행과 유사하게 이루어진다. 해당 엔트리의 목적지 주소, 근원지/목적지 포트 번호, 프로토콜의 값을 비교한다. 모두 일치하는 경우에는 연결 리스트가 존재한다고 하더라도, 해당 엔트리의 룰 번호가 가장 우선순위가 높은 룰에 해당하므로 그 값을 BMR로 저장하고 인풋의 다음 비트를 보고 엔트리의 프리픽스 값과의 크기 비교를 한 후에 이진 검색을 수행한다. 목적지 주소는 일치하는데 다른 필드 값이 일치하지

않고, 연결 리스트가 존재하는 경우는 연결 리스트로 이동하여 검색을 진행한다. 목적지 주소는 일치, 다른 필드 값에 대해서는 일치하지 않고 연결 리스트도 존재하지 않는 경우에는 인풋의 다음 비트를 보고 엔트리의 프리픽스와 크기 비교를 통해 이진 검색을 수행하며 검색을 진행한다.

프리픽스들 사이에는 네스팅 관계가 존재하며, 이러한 네스팅 관계는 이진 트리의 구성에 있어서 인클로저 프리픽스가 인클로즈드 프리픽스보다 상위에 존재하게 하므로, 일치 가능한 인클로저 프리픽스에서 우선적으로 비교된다. 따라서 검색이 진행되는 동안 일치 가능한 매 노드까지의 BMR(best matching rule)을 기억하면서 이진 트리의 리프까지 검색이 진행되어야 하므로 계층 이진 트라이에서와 마찬가지로 백-트랙킹(back-tracking)이 필요하다. 또한, 패킷 분류는 기존 라우터에서의 IP 주소 검색과는 달리 일치 가능한 모든 룰 가운데서 가장 높은 우선순위를 갖는 룰을 최종 결과로 출력하므로, 이러한 백-트랙킹은 필수적이라 할 수 있다. 하지만 백-트랙킹으로 인해 필드 1, 필드 2 테이블의 메모리 접근 횟수가 증가하게 되며 특히, 필드 2 테이블에서는 연결 리스트의 검색도 추가적으로 이루어지게 되므로 필드 2의 메모리 접근 횟수가 필드 1의 메모리 접근 횟수보다 크게 된다.

그림 8은 두 프리픽스 필드가 (10111, 10110)이고, 근원지 포트 번호가 53, 목적지 포트 번호가 25, 프로토콜 타입이 6인 패킷이 입력으로 들어왔을 경우에 제안하는 구조의 검색 과정에 대해 설명한 것이다. 근원지 주소가 10111이므로 처음으로 접근된 필드 1 테이블의 엔트리의 프리픽스인 와일드카드 *와 비교한다. 와일드카드와 비교되어 일치하였으므로 다음 트리 포인터 정보를 이용해 나머지 필드의 정보를 비교한다. 필드 2 테이블의 0번째 인덱스에 접근되었으므로 프리픽스 10*과 비교하고, 동시에 근원지 포트, 목적지 포트, 프로토콜에 대해서도 비교한다. 모두 일치하지 않고 연결 리스트가 존재하므로 연결리스트 포인터를 따라 이동한다. 14번째 엔트리와 비교하여 4개의 필드 값이 모두 일치하므로 해당하는 룰 번호 8을 BMR로 기억하고 검색을 진행한다. 현재 의 엔트리에는 왼쪽 포인터와 오른쪽 포인터 정보가 없는데, 이는 트리 구조에서의 리프에 해당하므로 더 이상 검색을 진행할 수 없음을 뜻한다. 따라서 필드 2 테이블에서의 검색을 마치고 BMR을 8로 기억하고 필드 1 테이블로 돌아간다. 필드 1테이블에서 0 번째 엔트리의 값보다 입력된 근원지 주

(a) 필드 1 테이블

	Entry	valid	prefix	length	Left	Right	Next
1	0	1	*	0	1	3	0
	1	1	01*	2	5	–	1
	2	1	1000*	4	–	–	4
4	3	1	101*	3	2	4	6
7	4	1	11*	2	–	6	9
	5	1	010*	3	–	–	11
	6	1	111*	3	–	–	12

(b) 필드 2 테이블

	Entry	valid	prefix	length	SrcPort	SrcPort	DstPort	DstPort	Protield	Prototype	Left	Right	Rule	Linked
2	0	1	10*	2	53	53	25	25	1	17	–	–	2	14
	1	1	0*	1	0	65535	5632	5632	1	6	3	2	14	–
	2	1	1*	1	53	53	443	443	1	17	–	–	13	–
	3	1	0011*	4	1024	65535	1024	65535	1	6	–	–	4	–
	4	1	*	0	53	53	443	443	1	17	–	–	0	–
6	5	1	0*	1	0	65535	5632	5632	1	6	–	–	12	–
5	6	1	1011*	4	0	65535	5632	5632	1	6	5	7	7	–
	7	1	11*	2	67	67	5632	5632	1	6	–	–	3	13
	8	1	00*	2	53	53	25	25	1	6	–	–	11	–
	9	1	0100*	4	0	65535	5632	5632	1	6	8	10	6	–
	10	1	11*	2	0	15576	2788	2783	1	4	–	–	9	–
	11	1	00*	2	53	53	443	443	1	17	–	–	10	–
	12	1	101*	3	53	53	25	25	1	6	–	–	1	–
	13	1	11*	2	53	53	25	25	1	4	–	–	5	–
3	14	1	10*	2	67	67	5632	5632	1	17	–	–	8	–

그림 8. 검색의 예

소 값이 크므로 오른쪽 포인터로 이동하여 일치 여부를 검사한다. 필드 1 테이블의 3번째 엔트리의 101*과 일치하므로 다음 테이블 포인터의 정보 6을 이용하여 필드 2 테이블로 검색을 진행한다. 필드 2 테이블의 6번째 엔트리의 프리픽스는 주어진 패킷의 목적지 주소와 일치하지만, 목적지 포트의 정보가 일치하지 않으므로 일치 가능성이 없다. 따라서 연결 리스트의 존재를 확인한다. 해당 엔트리는 연결 리스트가 존재하지 않고 주어진 패킷의 목적지 주소 값은 엔트리의 프리픽스 값보다 작으므로 왼쪽 포인터 5를 따라서 검색이 이루어진다. 5번째 엔트리는 목적지 주소와 목적지 포트가 일치하지 않으므로 일치 가능성이 없다. 따라서 엔트리의 연결 리스트가 존재하는지 살피고 존재하지 않으므로 주어진 인풋이 엔트리의 프리픽스 값보다 큰지 작은지를 확인한다. 이 경우에는 주어진 인풋이 크지만 오른쪽 포인터가 존재하지 않으므로 BMR의 값을 업데이트 하지 않고 필드 1테이블로 돌아간다. 주어진 근원지 주소의 값이 101*보다 크므로 오른쪽 포인터 4를 따라 검색을 진행하는데, 근원지 주소가 4번째 엔트리의 11*보다는 작으므로 왼쪽 포인터를 쫓아서 검색을 진행해야 한다. 하지만 해당 엔트리의 왼쪽 포인터 값이 존재하지 않으므로 검색이 종료되고, BMR 8은 최종 패킷 분류의 결과가 된다.

Ⅳ. 제안하는 구조의 성능 평가

본 논문에서는 패킷 분류를 위해서 프리픽스로 표현되는 두 필드만을 이용해 일치 가능한 룰 정보를 갖는 이진 트리를 계층적으로 연결하여 구성한 이차

표 2. acl 타입의 룰 셋에 대한 성능 실험 평가

	N	*Prefix Nesting*	T_{avg}		T_{max}		M(KByte)	
			Proposed	H-trie	Proposed	H-trie	Proposed	H-trie
acl50	49	4	10.99	83.59	18	108	1.34	11.33
acl100	100	4	12.75	87.13	20	132	2.53	18.95
acl500	450	4	20.64	91.94	55	154	9.46	40.94
acl1k	958	4	17.46	77.17	41	124	20.20	82.91
acl5k	4660	4	21.16	84.01	69	177	101.94	410.5

표 3. ipc 타입의 룰 셋에 대한 성능 실험 평가

	N	*Prefix Nesting*	T_{avg}		T_{max}		M(KByte)	
			Proposed	H-trie	Proposed	H-trie	Proposed	H-trie
ipc50	50	4	10.12	63.31	13	103	1.46	13.83
ipc100	100	4	10.90	66.63	19	104	2.83	24.53
ipc500	497	4	17.44	72.49	30	129	12.63	92.33
ipc1k	988	4	21.11	71.91	29	128	22.28	121.57
ipc5k	4468	4	26.61	85.64	52	138	92.88	224.70

표 4. fw 타입의 룰 셋에 대한 성능 실험 평가

	N	*Prefix Nesting*	T_{avg}		T_{max}		M(KByte)	
			Proposed	H-trie	Proposed	H-trie	Proposed	H-trie
fw50	50	4	10.39	53.75	19	139	1.12	5.02
fw100	98	4	15.58	50.57	27	134	2.34	9.93
fw500	425	4	34.20	83.43	75	201	9.37	23.37
fw1k	871	3	19.80	52.14	35	117	19.26	39.44
fw5k	4351	3	36.75	69.20	72	162	89.88	119.10

원 계층 이진 트리 구조를 제안하였다. 본 논문에서 제안된 구조의 성능을 평가하기 위해 실제 라우터에서의 룰 특성을 반영하는 클래스벤치(classbench[4],[5])의 룰 셋(rule set)과 인풋을 이용하여 C언어를 통해 실험하였다. 실험에 사용된 룰 셋 들은 약 50개에서 5000여 개에 이르는 다양한 프리픽스 개수를 가지며, acl(access control flow), fw(firewall), ipc(IP chain)의 세 가지 특성을 갖는 셋 들로 구성되었다.

표 2, 표 3, 표 4는 위에서 언급한 클래스벤치의 세 가지 특성에 따라 다양한 사이즈를 갖는 룰 셋을 사용하여 이차원 계층 이진 트리의 성능을 계층 이진 트라이와 평가, 비교한 결과를 보여준다. 이는 다양한 룰 개수(N)에 따라, 근원지 주소들 사이의 최대 프리픽스 네스팅의 개수(*Prefix Nesting*)와 입력된 패킷 헤더 필드를 검색하는 데 소요되는 평균 메모리 접근 횟수(T_{avg}), 검색하는데 소요되는 최대 메모리 접근 횟수(T_{max})와 필드 1, 필드 2 테이블을 저장하는데 소요되는 총 메모리의 크기(M)를 나타낸 것이다.

표 2에서 보는 바와 같이, 제안하는 구조에서는 평균 11.0에서 21.2번의 메모리 접근이 필요함을 알 수 있다. 평균 메모리 접근 횟수는 5k개의 룰 셋에 대해서도 크게 증가하지 않음을 볼 수 있으며, 이는 본 논문에서 제안하는 구조가 큰 룰 셋에 대해서도 잘 동작하는 확장성이 매우 우수한 구조임을 보여준다. 그러나 최대 메모리 접근 횟수는 룰 셋이 커짐에 따라 매우 커지는 것을 알 수 있는데, 이는 상대적으로 짧은 길이의 프리픽스를 갖는 두 필드 값에 대해 각각의 트리의 리프까지 검색이 이루어지게 되며, 이 때 프리픽스의 네스팅 관계에 따라 접근해야 하는 노드가 많아지기 때문이다. 또한, 룰 셋의 크기가 크면 클수록, 필드 1, 필드 2 테이블의 엔트리 수가 커지므로 요구되는 메모리 사이즈가 증가하는 것을 알 수 있다. 클래스벤치의 세 가지 룰 특성을 분석한 결과, fw 타입의 룰 특성이 다른 타입들에 비해 짧은 길이의 프리픽스 필드를 갖고 이로 인해 네스팅 관계의 복잡도가 월등히 높은 것으로 나타났다. 따라서 다른 타입의 룰 셋보다 fw 타입의 룰 셋의 경우에 메모리 접근 횟수가 조금 높게 나타나는 것을 알 수 있었다. 표 3, 표 4의 결과를 통해, 클래스벤치의 다른 두 타입에 대해서도 표 2와 비슷한 경향을 보임을 알 수 있다.

다음으로는 기존의 패킷 분류 구조와 제안하는 구조의 성능을 비교하는 실험을 수행하였다. 표 5, 표 6, 표 7에 약 5k의 엔트리를 갖는 클래스벤치의 세 타입의 룰 셋에 대하여 최대 메모리 접근 횟수(T_{max}),

평균 메모리 접근 횟수(T_{avg})와 소요되는 총 메모리의 크기(M)를 비교하였다. 본 논문에서 제안하는 계층 이진 트리는 계층 이진 트라이에 비해서 메모리 접근 횟수가 1/2에서 약 1/4 정도로 검색 성능이 향상되었으며, 같은 노드에서의 선형 검색이 많이 요구되는 AQT에 비해서는 최대 1/15의 메모리 접근 횟수를 보인다. 이는 앞서 설명한 대로 기존의 트라이에 기초한 구조들이 빈 내부 노드를 필연적으로 갖게 되고 이에 따른 메모리 접근 횟수가 증가하며, 프리픽스의 길이에 대해서는 선형적인 검색이 이루어짐으로 인해 평균 메모리 접근 횟수가 큰 값을 갖는 단점을 제안하는 구조에서 해결하였음을 뜻한다. 또한, 계층 이진 트라이, 비트 벡터, 영역분할 사분 트리의 세 경우에는 두 필드 트라이의 내부에 빈 노드가 존재함으로 인해 트라이 구성에 필요한 메모리 요구량이 상당히 많은 것을 확인할 수 있다. 게다가, 비트 벡터 구조의 경우에는 각 노드에 룰의 개수만큼의 엔트리를 갖는 비트 벡터를 가지고 있어야 하므로, 이를 위한 메모리 요구량은 다른 패킷 분류 구조에 비해 월등히 많은 것을 알 수 있다. 반면에, 제안하는 구조는 각 두 필드에 대해 구성된 트리 내부에 빈 노드가 존재하지 않게 되므로, 각각의 필드 트리를 구성하는데 필요한 메모리 요구량이 현저히 적은 것을 확인할 수 있다. 또한, 빈 노드에 대한 접근이 없으므로, 평균 메모리 접근 횟수도 다른 세 구조에 비해 최대 약 4배 줄어든 것을 확인할 수 있다. 표 6은 ipc 5k 룰 셋에 대한 성능 비교를 보여주며, 이 경우에도 역시 제안하는 구조가 다른 구조에 비해 빠른 검색 성능과 적은 량의 메모리를 요구함을 알 수 있다. 표 7의 fw타입의 룰 셋에 대해서는, 평균 메모리 접근 횟수가 다른 룰 타입에 대해 큰 것을 알 수 있다. 이것은, fw타입의 경우 같은 개수의 룰 셋이라도 짧은 길이에 해당하는 프리픽스의 차지 비율이 월등히 높고, 이에 따라 네스팅 관계가 다른 룰 셋의 경우보다 복잡하기 때문이다. 따라서 패킷의 두 필드에 대해 일치 가능성이 높은 프리픽스의 존재율이 높으므로 검색 시에 접근해야 하는 엔트리의 개수가 증가하게 되고 이것은 평균 메모리 접근 횟수와 최대 메모리 접근 횟수가 큰 값을 갖게 되는 요인이 된다. 하지만 그럼에도 기존의 구조에 비해 현저하게 향상된 검색 속도를 나타내고 있음을 알 수 있다.

현재의 네트워크 프로세서들은 초당 수백 기가 비트의 속도로 제공되는 패킷을 선속도로 처리해야 하고, 이를 위해 패킷 분류 또한 매우 빠르게 이루어

표 6. ipc5K 룰 셋에 대한 성능비교

	T_{max}	T_{avg}	M(KByte)
Linear Search	4468	1957.17	82.90
Hierarchical Trie[1]	192	85.64	224.70
Bit Vector[2]	230	151.86	2531.48
AQT[4]	415	344.79	234.26
Proposed HBST	*52*	*26.61*	*92.88*

표 5. acl5K 룰 셋에 대한 성능비교

	T_{max}	T_{avg}	M(KByte)
Linear Search	4660	2399.02	86.47
Hierarchical Trie[1]	177	84.01	410.48
Bit Vector[2]	76	64.10	2793.23
AQT[4]	94	50.11	200.22
Proposed HBST	*69*	*21.16*	*101.94*

표 7. fw5K 룰 셋에 대한 성능비교

	T_{max}	T_{avg}	M(KByte)
Linear Search	4351	2292.30	80.73
Hierarchical Trie[1]	162	69.20	119.10
Bit Vector[2]	1044	738.75	2394.65
AQT[4]	1193	660.12	479.77
Proposed HBST	*72*	*36.75*	*89.88*

져야 한다. 또한 무선 네트워크 응용 프로그램들의 발전으로 많은 양의 네트워크 장비들이 네트워크에 접속되고, 이것은 패킷 분류를 위한 룰 셋의 급격한 증가로 이어진다. 따라서 패킷 분류를 위해서는 가능한 한 최소의 메모리로 빠르게 패킷을 처리하는 것이 중요한 성능 척도라 할 수 있겠다. 본 논문에서 제안하는 이차원 계층 이진 트리는 실시간 패킷 분류를 위한 평균 검색 속도와 메모리 요구량의 면에서 기존의 패킷 분류 구조에 비해 월등한 성능을 보이는 우수한 구조라고 할 수 있다.

Ⅴ. 결 론

본 논문에서는 패킷 분류를 위해서 이진 트리를 계층적으로 구성한 이차원 계층 이진 트리를 제안하였다. 기존의 계층 이진 트라이는 트라이에 기초한 구조로써 각 필드 트라이 내부에 빈 노드의 존재가 불가피하였고, 이로 인해 메모리 비효율성과 검색 시의 메모리 접근 횟수가 증가되는 한계점이 있었다. 또한, 이차원 영역분할 사분 트리 역시 트라이에 기초한 구조로 빈 노드로 인한 메모리 비효율성을 극복하지 못하였다. 따라서 본 논문에서는 각 필드 트

라이 내부의 빈 노드를 제거함으로써 메모리 효율성을 높이는 이진 트리를 계층적으로 구성함으로써 메모리의 효율성을 현저히 향상시키는 새로운 패킷 분류 구조를 제안하였다. 또한 제안하는 구조는 빈 노드에 대한 메모리 접근을 없앰으로써 검색의 효율성 또한 향상됨을 확인하였다.

참 고 문 헌

〔1〕 H. Jonathan Chao, "Next Generation Routers," *Proceedings of the IEEE JPROC.2002.802001, Volume 90, Issue 9, pp.1518 - 1558, Sept. 2002*

〔2〕 G. Varghese, "Network Algorithmics," *Morgan Kaufmann*, 2005.

〔3〕 V. Srinivasan, G. Varghese, S. Suri, and M. Waldvagel, "Fast and scalable layer four switching," *in Proc. ACM SIGCOMM'98, pp. 191-202, Aug. 1998*

〔4〕 D. E. Taylor, J. S. Turner, J.S, "ClassBench: a packet classification benchmark," *Proc. IEEE INFOCOM 2005, pp.2068 - 2079, March 2005*

〔5〕 D. E. Taylor, J. S. Turner. The source code of Packet Classification Bench, http://www.arl.wustl.edu/~det3/ClassBench/index.htm

〔6〕 T. V Lakshman and D. Stiliadis, "High-speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," *Proc. of ACM SIGCOMM, pp. 203-214, October 1998.*

〔7〕 F. Baboescu and G. Varghese, "Scalable Packet Classification," *Proc. of ACM SIGCOMM, pp. 199-210, August 2001.*

〔8〕 Tong Liu, Huawei Li, Xiaowei Li, Yinhe Han, "Fast Packet Classification using Group Bit Vector," *in Proc. IEEE Globecom 2006*

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

〔10〕 N. Yazdani and P. S. Min, "Fast and Scalable Schemes for the IP Address Lookup Problem," *Proc. IEEE HPSR2000, pp.83-92, 2000*

추 하 늘 (Ha Neul Chu) 준회원

2005년 8월 : 이화여자대학교 정보통신학과, 학사

2005년 9월~현재 : 이화여자대학교 정보통신학과, 석사과정

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