셋-프루닝 이진 검색 트리를 이용한 계층적 패킷 분류 알고리즘

(A Hierarchical Packet Classification Algorithm Using Set-Pruning Binary Search Tree)

이 수 현 † 임 혜 숙 ††
(Soohyun Lee) (Hyesook Lim)

요 약 인터넷 라우터에서의 패킷 분류는 들어오는 모든 패킷에 대하여 패킷이 입력되는 속도와 같은 속도로 수행되어야 하는데, 여러 헤더 필드에 대해 다차원 검색을 수행하여야 하므로, 라우터 설계에 있어 가장 어려운 문제중의 하나이다. 계층적 패킷 분류 구조는 하나의 필드 검색이 끝날 때마다 검색 영역이 현저하게 줄어듦으로 매우 효율적이다. 그러나 계층적 구조들은 빈 노드와 역추적이라는 두가지 문제를 내재하고 있다. 본 논문에서는 두가지 문제를 동시에 해결하는 새로운 계층적 패킷분류 구조를 제안한다. 역추적 문제는 셋-프루닝 기법을 이용하여 해결하였으며, 이진 검색트리를 적용하여 빈노드를 제거하였다. 시뮬레이션 결과 제안된 알고리즘은 메모리 요구량의 증가 없이 검색 성능을 현저히 향상시킴을 확인하였다. 또한 셋-프루닝에 있어 제한된 룰의 복사를 적용하는 최적화 기법을 제안한다.

키워드 : 패킷 분류, 역추적, 셋-프루닝, 이진 검색 트리

Abstract Packet classification in the Internet routers requires multi-dimensional search for multiple header fields for every incoming packet in wire-speed, hence packet classification is one of the most important challenges in router design. Hierarchical packet classification is one of the most effective solutions since search space is remarkably reduced every time a field search is completed. However, hierarchical structures have two intrinsic issues; back-tracking and empty internal nodes. In this paper, we propose a new hierarchical packet classification algorithm which solves both problems. The back-tracking is avoided by using the set-pruning and the empty internal nodes are avoided by applying the binary search tree. Simulation result shows that the proposed algorithm provides significant improvement in search speed without increasing the amount of memory requirement. We also propose an optimization technique applying controlled rule copy in set-pruning.

Key words : Packet classification, Back-tracking, Set-pruning, Binary search tree

1. 서 론

기존의 인터넷 라우터는 서비스의 대역폭이나 전송 시간에 있어 각각의 입력 패킷을 동일하게 처리하여 품질보장이 전혀 없는 베스트 에포트(best-effort) 서비스를 제공하여 왔다. 이는 파일 전송, 메일링, 웹 브라우징과 같이 지연에 민감하지 않으면서 정확한 정보의 전송을 중요시 하는 데이타 트래픽에 적합하였다. 그러나 최근 들어 멀티미디어 스트리밍과 같은 실시간 트래픽이 기하급수적으로 증가함에 따라 다양한 플로우에 따른 품질 보증(Quality of Service, QoS)을 제공하는 정책 기반 서비스가 중요시 되고 있다. 인터넷 라우터가 각각의 플로우에 따라 다른 정책을 제공하기 위해서는 패킷 분류(packet classification)가 필수적으로 수행되어야

· 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)

† 학생회원 : 이화여자대학교 전자정보통신학과
hiroko77@ewhain.net
†† 비 회 원 : 이화여자대학교 전자정보통신학과 교수
hlim@ewha.ac.kr

논문접수 : 2008년 2월 28일
심사완료 : 2008년 8월 27일

한다. 패킷 분류란 라우터에 입력된 패킷을 미리 정의된 룰 셋에 따라 입력된 패킷이 속하는 클래스로 분류하는 것으로서, 그 클래스에 속하는 정책에 따라 서비스를 제공할 수 있도록 하여 품질 보증을 가능하게 한다[1].

패킷 분류의 어려움은 크게 세 가지 요인에 기인한다. 첫째로, 패킷이 입력되면 여러 개의 헤더 필드에 대한 다차원 검색을 동시에 수행하여야한다는 점이다. 패킷 분류를 위한 클래스는 패킷의 여러 헤더 필드에 속하는 룰들에 의하여 정의된다. 따라서 입력된 패킷이 속하는 클래스를 결정하기 위해서는 여러 헤더 필드에 대한 각각의 일치여부를 판단하여야 한다. 각 헤더 필드에 따라 완전 일치(exact match), 영역 일치(range match), 프리픽스 일치(prefix match)라는 각각 다른 방법에 의하여 일치여부를 판단하여야 하므로 검색이 복잡하다. 둘째로, 패킷 분류는 일치 가능한 모든 룰 가운데서 가장 높은 우선순위를 갖는 룰을 검색하여 그에 따른 정책으로 패킷을 처리해야 한다는 점이다. 패킷 분류는 초당 수 천만 개로 들어오는 모든 입력 패킷에 대하여 선속도(wire-speed) 프로세싱을 수행하여야 하는 어려움이 있다. 그러므로, 라우터 설계에 있어 선속도의 패킷 분류를 제공하기 위한 효율적인 데이타 구조 및 알고리즘[2]에 대한 연구가 활발히 진행되고 있다.

다양한 응용 프로그램으로 인한 데이타 플로우를 처리하기 위하여 라우터의 패킷 분류 속도가 관건이 되는데, 현재 기가 비트(gigabit) 또는 테라 비트(terabit) 라우터 시장에 초점이 맞춰져 있다. 그러므로 라우터상에서 링크 속도의 발전에 상응하는 빠른 패킷 분류 속도를 갖는 것이 중요한데 이는 입력 패킷이 속하는 클래스를 검색하는 과정에서 요구되는 평균 메모리 접근 횟수로서 평가된다[3].

빠른 검색 속도, 즉 적은 메모리 접근 횟수를 갖기 위해서는 헤더 필드 중 근원지 IP주소와 목적지 IP 주소 2개의 필드를 이용하여 계층적으로 룰을 검색하는 계층적 구조가 효율적이며 이러한 구조를 이용한 알고리즘이 많이 제안되어 왔다[4]. 계층적 구조의 효율성은 근원지 IP 주소와 목적지 IP 주소의 2개 필드의 조합에 의한 플로우의 다양성이 나머지 필드의 조합에 의한 다양성보다 훨씬 크다는 점에 근거한다[4]. 그러나 기존의 계층적 트라이는 빈 노드의 존재가 불가피하고 역추적(back-tracking)이 요구되기 때문에 메모리 접근 횟수가 증가하여 개선의 여지가 있다. 따라서 본 논문에서는 셋-프루닝(set-pruning) 기법과 이진 검색을 동시에 적용하여 빈 노드를 포함하지 않으면서도 역추적을 제거함으로써 검색 성능을 높이는 새로운 계층적 패킷 분류 알고리즘을 제안하고자 한다. 또한 셋-프루닝에 있어 와일드카드를 제외한 근원지 프리픽스에 대해 제한된 룰의 복사(controlled rule copy)를 적용하여 메모리 요구량 및 검색 성능을 향상시키는 최적화 기법을 제안하고자 한다.

2. 기존의 패킷 분류 구조

2.1 패킷 분류 문제의 정의(Problem Statement)

패킷 분류를 위해서 라우터에서는 검색의 기준으로써 $R=\{R_i \mid i=0,1,...,N\}$로 표현되는 N개의 룰을 갖는 분류 테이블을 갖는다. 분류 테이블 내의 모든 룰은 높은 우선순위를 갖는 룰부터 낮은 우선순위를 갖는 룰까지 순서대로 정렬되어 있다. 표 1은 실제 인터넷 라우터의 룰 특성을 반영하는 클래스벤치(class-bench)[5,6]로부터 15개의 룰들을 선택하여 구성한 분류기(classifier)이며 룰 번호가 작을수록 더 높은 우선순위(priority)를 갖는다.

표 1 분류기 예시

Src IP	Dst IP	Src port	Dst Port	Proto	Rule
010*	10*	0,65535	25,25	6	R0
00*	01*	53,53	443,443	4	R1
*	10*	53,53	1024,65535	17	R2
*	01*	53,53	443,443	4	R3
1*	1*	53,53	25,25	4	R4
1101*	001*	0,65535	2788,2788	17	R5
110100*	11*	53,53	5632,5632	6	R6
*	11*	53,53	25,25	6	R7
*	001*	0,65535	2788,2788	17	R8
111*	01*	67,67	5632,5632	17	R9
010*	10*	67,67	443,443	4	R10
111*	1*	67,67	25,25	4	R11
1101*	001*	53,53	2788,2788	4	R12
111*	*	1024,65535	5632,5632	4	R13
1*	10*	53,53	1024,65535	17	R14

각각의 룰 R_i는 근원지 IP 주소, 목적지 IP 주소, 근원지 포트 번호, 목적지 포트 번호, 프로토콜 타입의 5개의 필드들의 특정 값들로 이루어져 있으며 $R_i = \{F_i^j \mid j = 0,1,...,5\}$로 표현된다. 라우터에 들어오는 입력 패킷은 여러 개의 다양한 필드로 구성되어 있는데 그 중에서 관계있는 5개의 필드를 추출한 것, 즉 $H = (H^1, H^2, ..., H^5)$이 검색의 주체가 된다. 입력 헤더인 $H = (H^1, H^2, ..., H^5)$가 특정 룰 $R_i = \{F_i^j \mid j = 0,1,...,5\}$가 갖는 5개의 필드 값에 대해 $H^j \in R_i^j$의 관계를 갖게 되면 입력 패킷 H와 룰 R_i가 일치하였다고 정의한다.

$H \in F$는 각각의 필드 마다 판단 방법이 다르다. 근원지 IP 주소, 목적지 IP 주소 필드에 대해서는 입력된 패킷의 해당 필드의 주어진 프리픽스 길이까지만을 비교하는 프리픽스 일치가 수행되어야 하고, 근원지 및 목적지 포트 번호 필드에 대해서는 입력 패킷의 해당 필드 값이 특정 룰의 해당 필드에 주어진 영역에 포함되는 지를 확인하는 영역 일치가 수행되어야 하며, 프로토콜 타입 필드에 대해서는 입력 패킷의 해당 필드 값이 특정 룰의 해당 필드와 정확히 일치하는지를 판단하는 완전일치가 수행되어야 한다. 입력 패킷이 하나 이상의 룰과 일치하는 경우 일치하는 룰 중 가장 우선순위가 높은 룰을 패킷 분류의 최종 결과로 결정하고 해당 룰의 정책(policy)에 따라 패킷을 처리한다.

예를 들어 입력 패킷 H (111010, 101000, 53, 1024, 17)는 R_2의 근원지 및 목적지 IP 주소 필드 값인 *, 10*과 프리픽스 일치하며 근원지 및 목적지 포트는 영역 [53,53]과 [1024,65535]에 포함되며 근원지 포트번호인 17과 완전일치 하므로 R_2와 일치한다. 또한 R_2보다 더 낮은 인덱스 값을 가지는 일치하는 룰이 존재하지 않으므로 R_2가 패킷 분류의 최종 결과가 된다.

2.2 계층적 트라이(Hierarchical Trie)

계층적 트라이[1]는 룰을 구성하는 필드들 중 근원지와 목적지의 IP 주소 프리픽스로 표현된 필드들에 대하여 별도의 트라이를 구성하고 근원지 주소 트라이에서 목적지 주소 트라이로 계층적으로 연결한 것으로서 트라이에 기초한 기본적 구조이다. 각 트라이 구조는 프리픽스의 가장 왼쪽에 있는 비트인 MSB(Most Significant Bit)부터 한 비트씩 보아 0이면 루트(root) 노드의 왼쪽 노드를, 1이면 오른쪽 노드를 따라 한 레벨씩 진행하며 LSB(Least Significant Bit)의 해당 위치에 프리픽스 노드를 저장한다. 근원지 IP 주소 트라이인 상위 트라이의 프리픽스 노드는 목적지 IP 주소 트라이인 다음 트라이 포인터 값을 저장하고 있으며 하위 트라이의 프리픽스 노드는 나머지 3개의 필드 값을 저장하고 있다. 동일 위치에 여러 개의 룰이 존재하는 경우에는 두 프리픽스 쌍과는 일치하고 다른 필드 조합을 갖는 룰의 존재 여부를 나타내는 링크드 리스트(linked list)를 통해 우선순위 순서대로 연결된다. 그림 1에서는 표 1의 분류테이블을 바탕으로 근원지 IP주소와 목적지 IP 주소 필드를 이용한 계층적 트라이를 보였다. 그림에서 프리픽스 노드는 검은 색, 빈 노드는 흰 색으로 표현하였다. 계층적 트라이의 검색 과정은 첫 번째 필드에 해당되는 트라이에서 매치되는 모든 노드에 대하여, 그 노드에 계층적으로 연결된 하위 필드 트라이로 검색을 확장하며 각 필드에 대한 검색을 순차적으로 수행하는 방식으로 이루어진다. 각 계층의 트라이에서의 검색은 트라이의 루트에서부터 잎노드(leaf node)까지 입력 패킷의 MSB부터 한 비트씩 비교하며 0이면 왼쪽 포인터를, 1이면 오른쪽 포인터를 따라가며 이루어진다. 먼저 입력 패킷의 근원지 IP 주소 값에 따라 상위 트라이의 검색 경로를 따라가다가, 일치 가능한 노드에 도달하면 다음 트라이 포인터를 이용해 목적지 IP주소 값을 따라 하위 트라이의 검색을 진행한다.

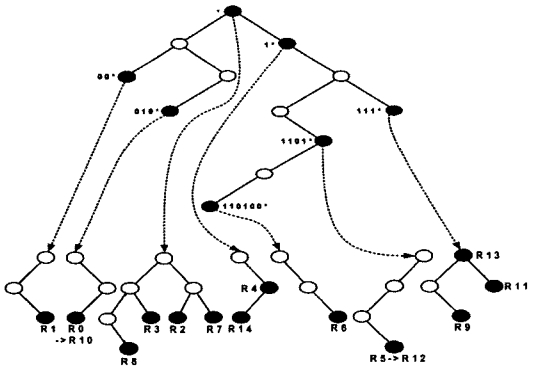


그림 1 계층적 트라이

위 두 필드 값에 대해 일치하면 해당 노드의 룰 정보를 살피고 나머지 3개의 필드의 값을 비교하여 일치하면 해당 룰을 현재까지의 BMR(Best Matching Rule)로 저장하고 상위 트라이로 검색 영역을 돌려주는 역추적을 한다. 나머지 3개의 필드 값과 일치하지 않으면 링크드 리스트 포인터의 존재 유무를 살핀다. 링크드 리스트 포인터가 있는 경우 이에 따라 나머지 값을 비교하여 일치여부를 판단하고, 링크드 리스트가 존재하지 않는 경우 상위 트라이로 역추적을 하여야 한다. 즉 계층적 트라이에서는 하위 트라이로부터 상위 트라이로의 역추적이 요구되는데, 이는 일치 가능한 룰 중에서도 가장 높은 우선순위를 갖는 룰을 검색하여야 하기 때문에 상위트라이의 잎노드를 만날 때까지 진행되어야 한다.

이 구조의 메모리 요구량은 트라이 상에서의 노드수, 즉 노드를 구성하는 엔트리 수로서 평가된다. 따라서 분류기의 룰의 개수를 N, 프리픽스의 최대 길이를 W 라

할 때, 메모리 요구량의 복잡도(complexity)는 O(2NW)를 가지며 검색 시간의 복잡도는 검색 과정에서의 메모리 접근 횟수와 연관되기 때문에 O(W2)를 갖는다.

이 구조에서는 빈 노드의 존재가 불가피하기 때문에 불필요한 메모리 접근으로 검색 성능이 떨어지고 가장 높은 우선순위를 갖는 룰을 검색하기 위해 하위 트라이에서 상위 트라이로의 역추적이 요구된다. 역추적은 메모리 접근 횟수를 늘려 검색 속도 성능을 저하시킨다.

2.3 계층적 셋-프루닝 트라이(Set-pruning Trie)

셋-프루닝[3] 트라이는 계층적 트라이에서 역추적에 의한 검색의 비효율성을 제거함으로써 검색 성능을 높이기 위해 제안된 구조이다. 계층적 트라이의 상위 트라이에서 상위 레벨 프리픽스 노드에 속하는 룰 정보를 하위 레벨의 프리픽스 노드에 연결된 하위 트라이에 복사하여 저장하는 구조이다. 그림 2에 그림 1의 계층적 이진 트라이에 셋-프루닝을 적용하여 구성한 계층적 셋-프루닝 트라이를 보였다. 원래 상위 레벨 프리픽스에 속하는 룰을 저장하는 프리픽스 노드는 검은색으로, 이로부터 복사된 룰이 저장된 프리픽스 노드는 회색으로 나타내었다. 계층적 셋-프루닝 트라이에서의 검색 과정은 상위 트라이에서 최장길이 검색을 통해 BMP(Best Matching Prefix)를 검색하여 이에 연결된 하위 트라이에서 우선순위가 가장 높은 룰을 찾을 수 있으므로, 상위 트라이에서 일치 가능한 모든 노드마다 하위 트라이로 내려가 검색이 진행될 필요가 없어 역추적이 제거된다. 따라서 검색 성능은 $O(W^2)$에서 $O(2W)$로 향상되었으나 구조상 여전히 빈 노드를 포함하고 있어 불필요한 메모리 접근으로 검색 성능이 저하된다. 또한 1개의 룰이 여러 노드로 복사되기 때문에 많은 메모리가 추가적으로 사용되어 메모리 요구량이 급격히 증가한다.

2.4 영역 분할 사분 트리 (AQT: Area-Based Quad Tree)

영역 분할 사분 트리(AQT)[7]는 효율적 패킷 분류를 위하여 트라이에 기초하여 구성된 구조이나, 계층적 트라이와는 다른 접근 방식을 통해 빠른 검색 속도를 제공하는 알고리즘이다. 영역 분할 사분 트리는 프리픽스로 표현되는 근원지 IP주소와 목적지 IP주소의 두 필드의 정보를 동시에 이용하여 하나의 새로운 프리픽스인 코드워드(codeword)를 생성하고 이를 이용하여 사분 트라이 구조를 만들어 패킷 분류를 수행한다.

각 룰이 갖는 프리픽스 형태로 표현된 근원지 IP 주소와 목적지 IP주소의 이차원 프리픽스는 하나의 평면상의 직사각형 영역으로 정의된다. W를 필드에 속하는 프리픽스의 최대 길이라고 할 때, 전체 검색영역은 2^W*2^W의 크기를 가지는 정사각형 평면으로 나타낼 수 있으며, 근원지 프리픽스의 길이가 m, 목적지 프리픽스의 길이가 n인 룰은 전체 정사각형 평면에서 가로 $2^{(W-m)}$, 세로 $2^{(W-n)}$인 직사각형 영역으로 표현된다.

영역분할 사분 트리의 구성은 다음과 같다. 전체 평면은 사분 트리의 루트(root)에 대응되며 4개로 분할된 영역은 루트 노드의 4개의 자식 노드(child node)로 대응된다. 이 같은 과정이 재귀적으로 반복되어 2차원 트라이를 구성한다. 그림 3에 표 1의 룰 셋에 대해 각 룰이 차지하는 영역과 이에 대응하는 영역분할 사분트리 구조를 보였다.

검색과정은 다음과 같다. 검색의 각 단계가 진행될 때

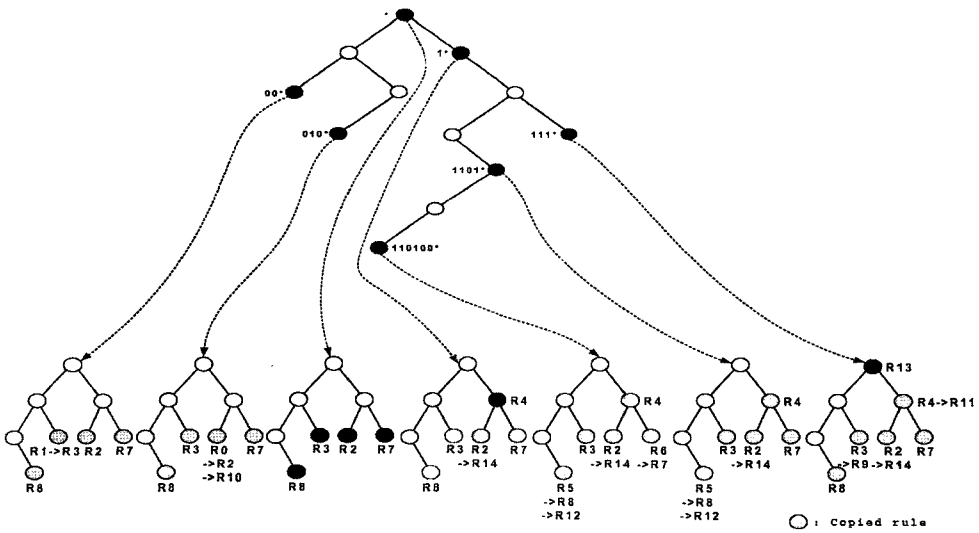


그림 2 계층적 셋-프루닝 트라이

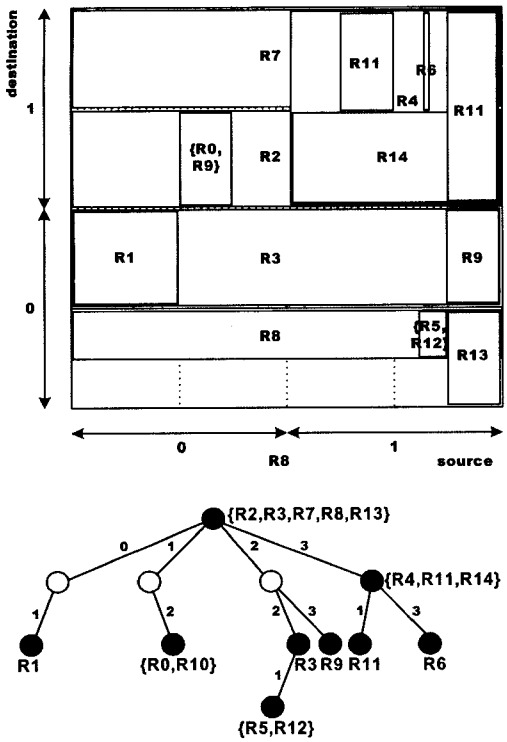


그림 3 룰 영역과 영역분할 사분 트라이

마다 근원지 주소와 목적지 주소의 각 비트를 확인하여, 두 비트의 조합으로 표현되는 같은 크기를 갖는 00, 01,10,11의 4개 정사각형 영역 중 일치하는 영역으로 검색영역이 제한되는 과정을 반복한다. 그러나 하나의 룰이 갖는 근원지와 목적지 프리픽스의 길이가 다를 수 있기 때문에 이를 적용하기 위하여 CFS(Crossing Filter Set)의 개념이 제안되었다. 이는 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 CFS가 AQT의 각 노드에 저장된다. 영역분할 사분 트라이 구조는 이진 검색에 기초한 트라이 구조로서 검색이 효율적이나 여전히 빈 노드의 존재가 불가피하며 따라서 프리픽스의 길이가 길어짐에 따라 메모리 효율성이 떨어지며 불필요한 메모리 접근을 필요로 한다.

이와 같이 검색 성능 향상을 위한 많은 알고리즘들이 제안되었으나 이 구조들은 빈 노드를 포함하거나 역추적이 요구되므로 검색 성능 향상이 제한된다. 따라서 본 논문에서는 셋-프루닝 기법과 이진 검색을 동시에 적용하여 역추적과 빈 노드를 동시에 제거하여 검색 성능을 높이는 새로운 계층적 패킷 분류 알고리즘을 제안한다.

3. 제안하는 패킷 분류 구조

3.1 제안하는 셋-프루닝 이진검색을 이용한 계층적 패킷분류 알고리즘

기존의 계층적 패킷 분류 알고리즘의 경우, 각 필드로 구성된 트라이 내부에 빈 노드가 존재하게 되므로 메모리 요구량이 증가할 뿐 아니라 불필요한 메모리 접근으로 검색 성능이 저하된다. 제안하는 셋-프루닝 이진검색을 이용한 계층적 패킷 분류 트리 알고리즘은 근원지 및 목적지 IP주소 필드의 이진 트라이에서 내부의 빈 노드를 완전히 제거한 이진 트리로 구성하여 메모리 비효율성을 해결하였다. 제안하는 구조에 앞서 일차원 이진검색 트리에 대해 설명한다.

3.1.1 일차원 이진검색 트리(One-dimensional Binary Search Tree)

이진 검색 트리는 1차원 IP 주소 검색을 위하여 이진 검색을 적용하는 구조로서, Yazdani[8]에 의해 제안되었다. 프리픽스간의 이진 검색을 하기 위해서는 프리픽스의 크기에 따른 정렬이 필요한데, 프리픽스의 길이가 고정되어 있지 않기 때문에 서로 다른 길이의 프리픽스간의 크기 비교에 대해 정의함으로써 프리픽스의 이진 검색을 가능하게 하였다.

같은 길이를 갖는 두 프리픽스에 대해서는 프리픽스의 수치 값의 비교로 크기를 결정한다. 그러나 길이가 다른 프리픽스를 비교하기 위해서는 짧은 쪽 프리픽스 길이까지의 값을 비교하여, 프리픽스 값이 더 큰 쪽을 큰 프리픽스로 정의한다. 또한 짧은 쪽 프리픽스 길이까지의 두 값이 같으면, 긴 프리픽스의 다음 비트의 값이 0인 경우 짧은 프리픽스를 더 큰 프리픽스로, 1인 경우에는 긴 프리픽스를 큰 프리픽스로 정의한다.

그러나 이러한 정의에 의하여 정렬된 리스트에 대해 가장 잘 일치하는 프리픽스(BMP: Best Matching Prefix)를 찾기 위하여 바로 이진검색이 적용될 수 없다. 예를 들어 정렬된 리스트인 00*, 010*, 0101*, *, 1*, 110101*, 1101*, 110110*, 111*, 11111*에 대하여 입력 주소가 11010010인 경우 1101*을 서브 스트링(sub-stiring)으로 갖는 110101*이 1101*보다 먼저 비교되기 때문에 이진 검색이 110101* 보다 작은 방향으로 진행되어 주어진 입력의 BMP인 1101*을 찾을 수 없게 된다.

따라서 이를 피하기 위해 프리픽스의 네스팅 관계에 관한 분류가 필요하다. 인클로저(enclosure)는 자신을 프리픽스로 하는, 즉 서브 스트링(sub-string)으로 갖는 다른 프리픽스가 같은 데이타 집합 내에 최소한 하나는 존재하는 프리픽스이고, 인클로즈드(enclosed)는 인클로저 프리픽스를 서브 스트링으로 갖는 프리픽스이다. 디스조인트 프리픽스는 인클로저도 인클로즈드도 아닌 프리픽스이다. 예를 들어 1101*와 110100*의 프리픽스를

고려할 때, 110100*이 1101*을 프리픽스로 하므로 1101*이 인클로저, 110100*이 인클로즈드이다.

이런 분류를 바탕으로 디스조인트(disjoint) 프리픽스와 첫 번째 레벨의 인클로저 프리픽스들만으로 이루어진 리스트에 대해 이진검색 트리를 구성하되, 인클로저가 선택될 때에, 이 인클로저를 프리픽스로 갖는 인클로즈드 프리픽스를 리스트에 추가하는 방식을 반복함에 의하여 트리를 구성한다. 이러한 이진검색을 적용하면 트라이에 존재하던 빈 노드를 제거한 프리픽스들만으로 이루어진 트리를 구성할 수 있고 프리픽스의 이진 검색을 통해 한번 검색이 이루어질 때마다 검색 영역이 최대 1/2로 감소한다는 장점이 있다. 그림 4에서는 표 1에서 보인 근원지 프리픽스 셋에 대한 네스팅 관계와 이를 통해 구성된 이진 트리 구조를 보였다.

3.1.2 제안하는 계층적 셋-프루닝 이진 검색 트리 빌드

제안하는 계층적 셋-프루닝 이진 검색 트리를 구성하기 위해서 먼저 역추적을 제거하기 위한 셋-프루닝의 적용이 필요하다. 룰의 복사 여부를 판단하기 위하여 근원지 IP주소 필드의 프리픽스들을 프리픽스의 네스팅 관계 정의에 따라 분류하여야 한다. 즉 근원지 IP 주소 필드의 프리픽스 간에 네스팅 관계가 있는 경우 인클로저에 연결된 룰들을 인클로즈드에 연결된 검색 영역에 복사, 저장을 한다. 복사는 인클로즈드의 근원지 주소와 인클로저의 룰 번호, 목적지 주소, 근원지 포트, 목적지 포트, 프로토콜 정보를 갖는 새로운 룰 엔트리를 만들어 냄으로써 이루어진다. 예를 들어, R5의 근원지 프리픽스 1101*와 R6의 110100*은 네스팅 관계에 있으므로 인클로저인 1101*의 룰 R5의 정보가 인클로즈드인 110100*에 연결된 목적지 검색 영역에 저장 되어야한다. 즉 근원지 및 목적지 IP 주소 필드가 (110100*, 001*), 근원지 및 목적지 포트 번호의 영역이 각각 [0,65535], [2788,2788], 프로토콜 타입이 17, 룰 인덱스가 5인 엔트리를 추가하여야 한다.

셋-프루닝의 적용 후에는 근원지 IP 주소 필드의 프리픽스들을 이용하여 상위 트리를 구성한다. 이 때, 모든 룰의 근원지 IP 주소 필드에 대한 이진 트리가 구성되므로 상위 트리의 노드는 빈 노드 없이 프리픽스 노드만으로 구성된다. 각 프리픽스 노드에 근원지 IP 주소 필드값이 프리픽스 노드의 프리픽스 값과 일치하는 모든 룰의 목적지 IP 주소 프리픽스를 이용한 하위 트리를 구성한 후 계층적으로 연결한다. 두 프리픽스 값이 같지만 나머지 세 필드에 대해 다른 값을 갖는 룰이 존재할 수 있으므로 이러한 룰 중에서 가장 높은 우선순위를 갖는 룰에 대한 정보를 노드에 저장한다. 나머지 룰은 두 프리픽스 값이 같지만 나머지 세 필드에 대해 다른 값을 갖는 룰이 존재함을 나타내는 링크드 리스트로 우선순위의 순서에 따라 연결하였다. 그림 5에서는 표 1의 분류테이블을 예로 구성된 제안하는 계층적 셋-프루닝 이진 검색트리를 보였다. 그림 2의 계층적 셋-프루닝 트라이와 그림 5의 제안하는 계층적 셋-프루닝 이진검색트리 구조를 비교하여 보면 그림 2의 계층적 셋-프루닝 트라이에서는 상위 트리 및 모든 하위 트리에 빈 노드가 존재하나 그림 5의 제안하는 구조에서는 빈 노드가 모두 제거되어 룰을 저장하는 노드만으로 트리가 구성된 것을 볼 수 있다.

HSP-BST의 구조는 상위 트라이와 하위 트라이의 두 개의 테이블에 의하여 구성되는데 그 구조는 표 2와 표 3과 같다. 먼저 필드 1의 테이블은 근원지 주소 필드에 의하여 구성되며 엔트리의 너비는 10바이트(10 bytes)이다. 필드 1 테이블은 해당 엔트리의 유효함을 나타내는 유효 엔트리 비트, 근원지 주소의 프리픽스 값, 프리픽스의 길이와 트리의 구조를 나타내는 왼쪽 및 오른쪽 노드의 포인터, 다음 트리로의 포인터를 갖는다. 다음 트리로의 포인터는 상위 트리와 하위 트리를 계층적으로 연결하기 위한 것으로 해당 값은 필드 2 테이블의 엔트리 주소가 된다.

필드 2 테이블은 원래의 룰과 복사된 룰의 개수의 합과 같은 수의 엔트리를 갖으며 유효 엔트리 비트와 목적지 프리픽스 값, 프리픽스의 길이, 근원지 포트 시작 번호, 근원지 포트 끝 번호, 목적지 포트 시작 번호, 목적지 포트 끝 번호, 프로토콜 와일드, 프로토콜 타입, 왼쪽 및 오른쪽 노드 포인터, 룰 번호, 링크드 리스트 포

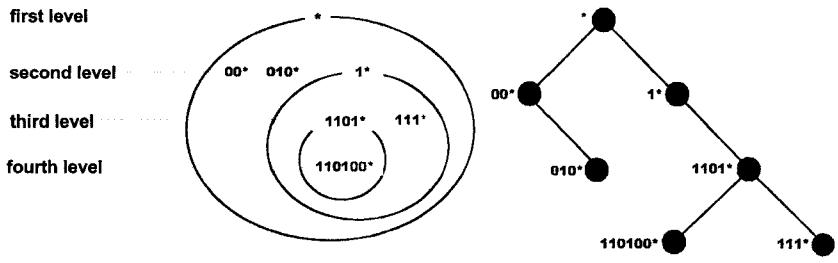


그림 4 프리픽스 네스팅 관계와 이진검색 트리

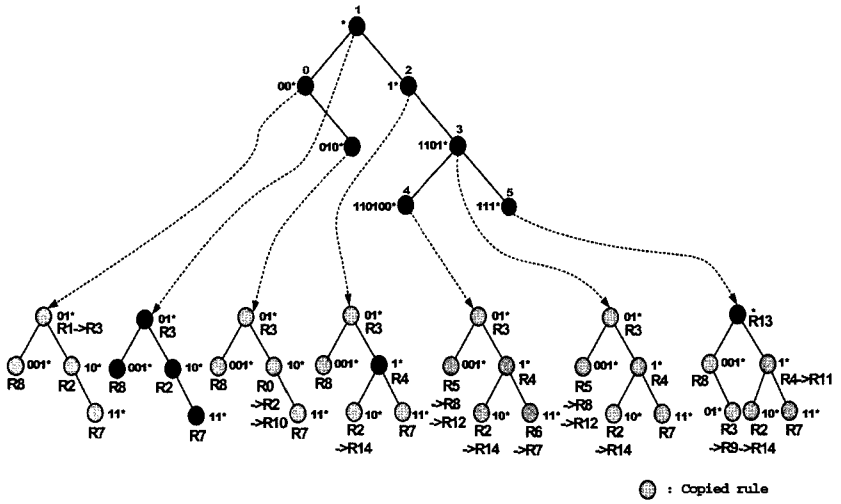


그림 5 제안하는 HSP-BST

표 2 Field 1 테이블

1 bit	32 bits	5 bits	14 bits	14 bits	14 bits
Entry valid	prefix	Prefix Length	Left Pointer	Right Pointer	Next-tree Pointer

Entry	Valid	Prefix	Length	Left	Right	Next
0	1	*	0	1	3	1
1	1	00*	2	-1	2	5
2	1	010*	3	-1	-1	9
3	1	1*	1	-1	4	13
4	1	1101*	4	6	5	18
5	1	111*	3	-1	-1	22
6	1	110100*	6	-1	-1	29

표 3 Field 2 테이블

1 bit	32 bits	5 bits	16 bits	16 bits	16 bits	16 bits	1 bit	8 bits	14 bits	14 bits	14 bits	14 bits
Entry valid	Prefix	Prefix length	Source Port Start	Source Port End	Dst. Port Start	Dst. Port End	Protocol wild	Protocol type	Left pointer	Right pointer	Rule Number	Linked List Pointer

E	V	Prefix	Len	Src_S	Src_E	Dst_S	Dst_E	W	PT	L	R	#	Lk
0	1	01*	2	53	53	443	443	1	4	-1	-1	1	-1
1	1	10*	2	0	65535	25	25	1	6	-1	-1	0	16
2	1	1*	1	53	53	25	25	1	4	3	-1	2	-1
3	1	10*	2	53	53	1024	65535	1	17	-1	-1	10	-1
4	1	001*	3	0	65535	2788	2788	1	17	-1	5	11	18
5	1	1*	1	53	53	25	25	1	4	6	7	9	-1
6	1	10*	2	53	53	1024	65535	1	17	-1	-1	5	-1

42	1	001*	3	53	53	2788	2788	1	4	-1	-1	12	-1
43	1	10*	2	53	53	1024	65535	1	17	-1	-1	14	-1
44	1	10*	2	53	53	1024	65535	1	17	-1	-1	14	-1
45	1	10*	2	53	53	1024	65535	1	17	-1	-1	14	-1
46	1	10*	2	53	53	1024	65535	1	17	-1	-1	14	-1

인터 필드로 구성되어 한 엔트리 당 21바이트(21 bytes)의 너비를 갖는다. 여기서 검색에 쓰이는 필드는 목적지 주소와, 근원지 포트, 목적지 포트, 프로토콜의 필드이다. 링크드 리스트 포인터는 근원지 주소와 목적지 주소는 같고 포트와 프로토콜이 다른 경우에 계층적 트리의 구조로는 같은 위치이기 때문에 식별할 수 없으므로 이를 검색할 수 있게 한 것으로써 필드 2의 엔트리 주소를 나타낸다. 모든 일치 가능한 플로우를 검색하여 우선

순위가 가장 높은 룰을 검색하여야 하기 때문에, 우선순위가 높은 룰부터 낮은 룰로 연결된 연결 리스트를 이용해 검색을 할 수 있게 하였다.

3.2 검색

3.2.1 필드 1 테이블에 대한 검색

가장 우선순위가 높은 룰을 검색하기 위한 과정은 다음과 같다. 필드 1 테이블에 대한 검색이 먼저 수행된다. 입력된 패킷의 근원지 주소 필드의 주소 값과 저장된 필드 1테이블의 근원지 주소 프리픽스를 비교한다. 그 두 값이 같은 경우 현재 노드의 다음 트리로의 포인터를 현재까지의 베스트 매치 값으로 저장하고, 입력된 패킷의 주소 값이 저장된 프리픽스 값보다 작은 경우 테이블의 왼쪽 포인터를 따라, 큰 경우 테이블의 오른쪽 포인터를 따라서 이진 검색을 진행한다. 더 이상 진행할 곳이 없는 경우, 필드 1테이블에서의 검색이 끝나게 되며 현재까지 저장된 다음 트리로의 포인터 값에 따라 하위 트리인 필드 2 테이블의 엔트리로 이동하게 된다. 이 구조에서는 인클로즈드의 하위 트리에 인클로저에 해당하는 룰들을 모두 복사하였기 때문에 계층적 트라이나 2차원 이진 트리와는 달리 상위트리에서 최장 길이 일치(longest prefix match) 검색을 수행한다.

3.2.2 필드 2 테이블에 대한 검색

필드 2 테이블의 검색은 입력된 패킷의 목적지 주소와 엔트리에 저장된 프리픽스의 크기 비교를 통하여 이진검색의 방향을 결정한다. 입력된 패킷의 목적지 주소와 저장된 프리픽스가 일치하는 경우에만 해당 엔트리의 목적지와 근원지 포트 번호, 프로토콜의 값을 비교한다. 모두 일치하는 경우 링크드 리스트가 존재하여도, 해당 엔트리의 룰 번호가 가장 우선순위가 높으므로 링크드 리스트로 이동하지 않은 채 그 값을 가장 우선순위가 높은 룰 BMR로 저장한 후, 크기 비교에 따라 이진검색을 진행한다. 목적지 주소는 일치하는데 다른 필드 값이 일치하지 않고, 연결 리스트가 존재하는 경우 연결 리스트로 이동하여 검색을 한다. 더 이상의 연결 리스트가 존재하지 않는 경우, 입력 패킷의 목적지 주소와 엔트리 프리픽스의 크기 비교를 통해 다음 노드로 진행하며 이진 검색을 수행한다.

예를 들어, 근원지 IP주소, 목적지 IP 주소, 근원지 포트 번호, 목적지 포트 번호, 프로토콜 타입이 (111010, 101000, 53, 1024, 17)인 입력 패킷에 대한 검색 과정은 그림 6과 같다. 필드 1 테이블에서 루트 노드인 0번째 엔트리에 접근하여 해당 프리픽스 값인 와일드카드(*)와 입력된 패킷의 근원지 IP주소인 111010를 비교하면 매치하는 경우에 해당한다. 따라서 다음 트리로의 포인터인 1을 저장한 후 다음 비트가 1이므로 오른쪽 포인터에 따라 3 번째 엔트리로 검색을 이동한다. 그 후 3번째 엔트리에 접근하면 해당 프리픽스 값이 1*로 매치하므로 다음 트리로의 포인터값을 13으로 업데이트 한 후 오른쪽 포인터를 따라 4번째 엔트리로 이동하여 검색을 진행한다. 4번째 엔트리에 접근하여 해당 프리픽스 값인 1101*과 입력 근원지 IP주소인 111010를 비교하면 입력된 패킷의 주소 값이 저장된 프리픽스 값보다 큰 경우에 해당하므로 해당 엔트리의 오른쪽 포인터에 따라 5번째 엔트리로 이동하여 검색을 진행한다. 5번째 엔트리의 프리픽스 값은 111*로서 입력 주소값과 일치하므로 다음 트리로의 포인터인 22를 베스트 매치값으로 저장한다. 이 때 입력 주소의 다음 비트값은 0이지만, 5번째 엔트리는 트라이에서의 잎노드에 해당하여 더 이상 검색이 진행되지 않고 검색을 종료한다. 따라서 필드 1테이블에서의 최종 출력은 다음 트리로의 포인터로서 베스트 매치값인 22를 가지며 필드 2 테이블로 검색 영역이 확장된다.

	Entry	Valid	Prefix	Length	Left	Right	Next
1→	0	1	*	0	1	3	1
	1	1	00*	2	-1	2	5
	2	1	010*	3	-1	-1	9
2→	3	1	1*	1	-1	4	13
3→	4	1	1101*	4	6	5	18
4→	5	1	111*	3	-1	-1	22
	6	1	110100*	6	-1	-1	29

	E	V	Prefix	Len	Src_S	Src_E	Dst_S	Dst_E	W	PT	L	R	#	Lk
	0	1	001*	3	0	65535	2788	2788	1	17	-1	-1	8	-1
	1	1	01*	2	53	53	443	443	1	4	0	2	3	-1
	2	1	10*	2	53	53	1024	65535	1	17	-1	3	2	-1
	3	1	11*	2	53	53	25	25	1	6	-1	-1	7	-1
	4	1	001*	3	0	65535	2788	2788	1	17	-1	-1	8	-1
5→	22	1	*	0	1024	65535	5632	5632	1	4	23	25	13	-1
6→	25	1	1*	1	53	53	25	25	1	4	26	27	4	40
8→	26	1	10*	2	53	53	1024	65535	1	17	-1	-1	2	44
7→	40	1	1*	1	67	67	25	25	1	4	26	27	11	-1

그림 6 입력 (111010,101000,53,1024,17)에 대한 검색의 예

필드 2 테이블에서의 검색은 필드 1 테이블의 검색 결과인 22에 의해 22번째 엔트리로부터 시작된다. 입력된 패킷의 목적지 주소인 101000과 저장된 프리픽스인 와일드카드(*)는 매치하지만 다른 필드 값이 매치하지 않으므로 다음 비트인 1에 따라 오른쪽 포인터 값인 25번째 엔트리로 검색이 이동한다. 25번째 엔트리에서는 1*과 입력 주소값이 매치하고 다른 필드 값이 일치하지 않으며 링크드 리스트가 존재하므로 링크드 리스트 포인터를 따라 이동한다. 40번째 엔트리로 이동하여 필드 값을 비교하면 프리픽스 1*와 일치하나 다른 필드 값이 일치하지 않으므로 왼쪽 포인터를 따라 26번째 엔트리로 이동한다. 26번째 엔트리에서 4개의 필드 값이 모두 일치하므로 룰 번호인 2를 BMR로 저장한 후 트라이의 잎노드에 해당하므로 검색을 종료한다. 이 때 링크드 리스트가 존재하지만 링크드 리스트는 룰의 우선순위에 따라 정렬되어 있으므로 링크드 리스트로 검색을 이동할 필요가 없다. 따라서 BMR 2는 최종 패킷 분류의 결과이다.

4. 성능 개선을 위한 최적화 기법

4.1 제한된 룰의 복사(Controlled Rule Copy)

본 논문에서 제안하는 구조는 역추적을 제거하기 위해 셋-프루닝을 적용하여 인클로저의 하위 트라이에 속한 룰들을 인클로즈드의 하위 트라이로 복사한 후 이진 검색구조를 적용하는 방식이다. 먼저 룰 셋의 특성을 알아보기 위하여 현재 인터넷 라우터에서 사용되는 분류기와 비슷한 성질을 갖는 classBench[5,6]를 사용하여, 100개에서 5000개의 룰을 갖는 ACL(access control list), FW(firewall), IPC(IP chain)의 셋을 생성하였다.

그림 7에서는 각 룰 셋의 근원지 프리픽스 길이에 따른 룰 개수의 분포를 보였다. 근원지 프리픽스들의 길이별 분포도를 살펴본 결과, IPC와 FW룰셋의 경우, ACL과 매우 다른 특성을 갖음을 확인할 수 있었다. ACL룰셋은 프리픽스 길이의 분포가 주로 32비트 근처에 분포하는데 반해, IPC 및 FW룰 셋의 경우 와일드카드 등 다수의 짧은 길이 프리픽스를 포함함을 발견할 수 있었다. 근원지 프리픽스에 와일드카드가 많은 경우 다른 모든 프리픽스에 대하여 인클로저이기 때문에 근원지 프리픽스가 와일드카드인 모든 룰들이 다른 모든 노드의 하위 트라이들로 복사되어야 하므로, 하위 트라이의 노드 수를 증가시켜 메모리 요구량과 메모리 접근 횟수를 크게 증가시킨다.

표 4, 표 5에서는 근원지 주소가 와일드 카드인 룰들과 와일드 카드가 아닌 룰들의 우선순위에 따른 개수

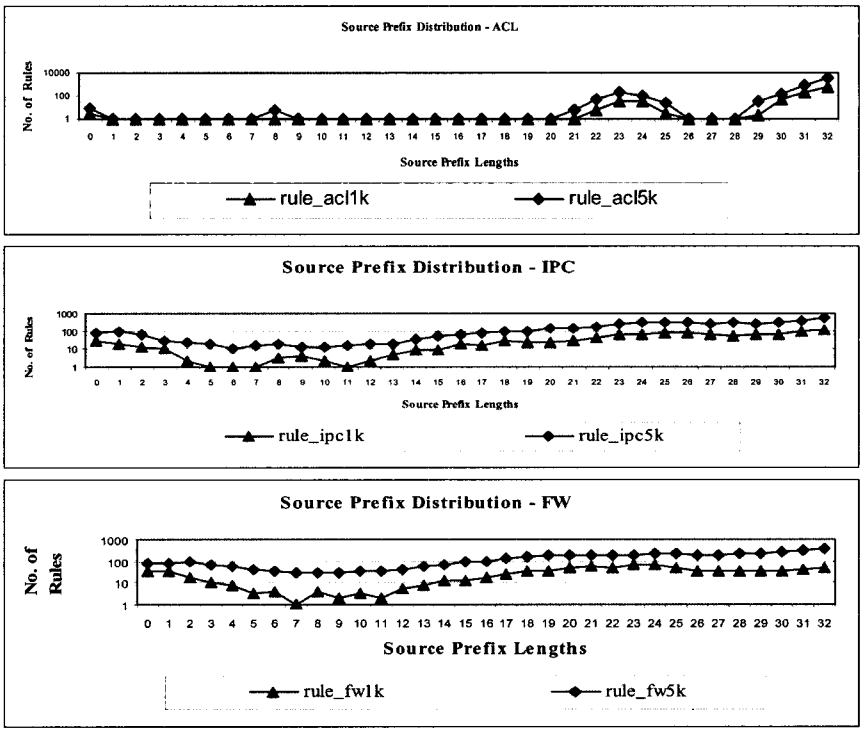


그림 7 각 룰 셋의 근원지 프리픽스 길이의 분포도

표 4 각 1k 룰 셋의 우선순위 분포도

Rule set \ Priority		Top 20%	Top 40%	Top 60%	Top 80%	Top 100%
ACL	main	191	383	574	766	955
	wildcard	0	0	0	0	3
IPC	main	197	395	584	772	961
	wildcard	0	0	8	18	27
FW	main	174	348	522	688	839
	wildcard	0	0	0	8	32

표 5 각 5k 룰 셋의 우선순위 분포도

Rule set \ Priority		Top 20%	Top 40%	Top 60%	Top 80%	Top 100%
ACL	main	932	1864	2796	3728	4652
	wildcard	0	0	0	0	8
IPC	main	893	1787	2677	3532	4392
	wildcard	0	0	3	42	76
FW	main	613	1226	1840	2415	2953
	wildcard	0	0	0	38	114

표 6 각 룰 셋에 대한 임계값

	Number of rules	Threshold value
ACL1K	958	955
IPC1K	988	544
FW1K	871	525
ACL 5K	4660	4645
IPC 5K	4468	2633
FW 5K	3067	1902

분포도를 보였다. 이를 통해 근원지 주소가 와일드 카드가 아닌 룰들은 모든 우선순위 구간에 대하여 고르게 분포하는 반면, 와일드카드인 룰들은 대부분 낮은 우선순위인 80%~100% 구간에 분포함을 알 수 있다. 즉 실제 라우터 상에서 근원지 주소가 와일드카드인 룰들은 다른 구체적 프리픽스를 갖는 룰들에 비해 우선순위가 낮다는 특성을 가지고 있음을 확인하였다. 이는 근원지 프리픽스가 와일드카드가 아닌 룰들을 먼저 검색한다면 우선순위가 낮은 와일드카드의 룰에 대한 검색을 피할 수 있기 때문에 보다 빨리 BMR을 결정할 수 있음을 의미한다.

따라서 이같은 성질을 바탕으로 근원지 프리픽스에 와일드카드를 가지는 룰들에 대하여 별도의 트리를 구성하여 메인 트리에서는 와일드카드의 룰들에 대한 복사가 이루어 지지 않도록 하는 제한된 룰의 복사를 적용함으로써 메모리 요구량을 감소시키고, 와일드 카드 트리에서 가장 우선순위가 높은 값을 임계값(threshold value)으로 정하여 메인 트리에서의 지역 BMR(Local BMR)이 임계값보다 우선순위가 낮은 경우에만 와일드 트리로 검색을 이동함으로써 검색 속도 성능을 향상시키는 최적화 기법을 제안한다.

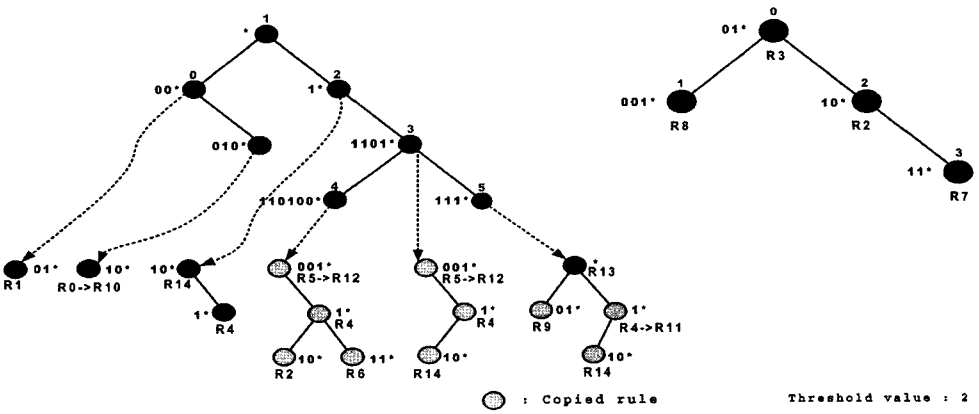
표 6에서 각 룰 셋에 대한 임계값을 나타내었는데 임계값이 대부분 낮은 우선순위 구간에 존재하므로 와일드 카드 트리로 검색을 이동하는 경우가 적다. 예를 들어 ACL1K 룰 셋의 경우, 총 958개의 룰들 중에서 룰 번호가 955,956,957인 3개의 룰과 일치하는 입력패킷을 제외하고는 모든 입력에 대해 메인 트리의 검색만으로 검색이 종료되어 메모리 접근 횟수를 줄일 수 있다.

4.2 최적화 구조 빌드와 검색

첫째, 주어진 룰 셋으로부터 근원지 프리픽스가 와일드카드인 룰들을 분리하여야 한다. 둘째로, 와일드카드가 아닌 룰들에 대하여 앞서 제안한 구조인 셋-프루닝 이진검색 트리인 메인 트리를 구성한다. 마지막으로 근원지 프리픽스가 와일드카드인 룰들에 대하여 목적지 주소 프리픽스로 구성된 일차원 이진검색 트리(1D-BST)를 구성한 후 이 트리에서의 우선순위가 가장 높은 룰 번호를 임계값으로 저장한다. 이 때의 임계값은 메인 트리의 지역 BMR 검색 후에 근원지 프리픽스가 와일드카드인 룰들로 구성된 트리로의 검색 필요성 여부를 판단하는 기준으로 이용된다. 그림 8에는 표 1의 룰 셋에 대하여 만들어진 최적화 구조를 보였다. 그림 8(a)에 보여진 메인트리구조는 제안하는 HSP-BST와 같은 구조를 가지나 와일드카드에 있던 룰들이 다른 노드의 하위 트리로 복사되어 지지 않았다는 점만이 다르다. 그림 8(b)는 근원지 프리픽스에 와일드카드를 갖는 룰들로 이루어진 별도의 와일드카드 이진검색트리를 보여준다. 그림 8의 최적화된 HSP-BST와 그림 5의 구조를 비교하여 보면, 루트노드에 존재하여 모든 하위트리로 복사되었던 룰들이 와일드카드 트리로 분리되어 저장됨으로 매우 효율적인 계층적 셋 프루닝 이진트리가 형성되었음을 알 수 있다.

이 구조에서의 검색과정은 다음과 같다. 메인트리에서 제안하는 구조의 검색 방법과 같은 방법으로 지역 BMR을 결정한다. 그 다음 지역 BMR 값과 와일드카드 트리의 임계값을 비교한다. 지역 BMR 값이 임계값보다 작은 경우, 즉 지역 BMR이 임계값보다 더 높은 우선순위를 갖는 경우에는 와일드카드 트리의 검색을 하여도 그 출력 BMR의 우선순위가 지역 BMR보다 낮은 우선순위를 가지므로 와일드카드 트리로의 검색은 불필요하며 지역 BMR이 최종 BMR로 저장된다.

그러나 지역 BMR 값이 임계값보다 큰 경우에는 와일드카드 트리로 검색 영역이 확장되어야 한다. 입력된 패킷과 와일드카드 트리 노드에 저장된 엔트리의 비교는 저장된 엔트리의 룰의 우선순위가 지역 BMR 보다 높은 우선순위를 갖는 경우에만 수행된다. 입력된 패킷이 와



(a) non-wild card : HSP-BST (b) wild card : 1D-BST

그림 8 최적화된 HSP-BST

일드카드 트리의 노드에 저장된 룰과 일치하며, 메인 트리의 지역 BMR보다 높은 우선순위를 갖는 경우에만 해당 엔트리의 룰 번호를 현재까지의 BMR로 저장한 후 이진 검색을 진행한다. 이 때 링크드 리스트가 존재하여도 링크드 리스트로 이동하지 않는데 이는 링크드 리스트가 우선순위에 따라 연결되어 있으므로 현재 노드에서 매치한다면 그 중 가장 우선순위가 높기 때문이다.

와일드카드 트리의 노드에 저장된 룰과 입력된 패킷이 일치하지 않고 링크드 리스트가 존재하는 경우 현재 엔트리의 룰 번호와 지역 BMR의 룰 번호를 비교한다. 현재 엔트리의 룰이 높은 우선순위를 갖는 경우에는 링크드 리스트의 엔트리로 검색이 이동하지만 그렇지 않은 경우에는 링크드 리스트의 엔트리로 이동하지 않은 채 다음 노드로의 이진 검색을 진행한다. 이는 링크드 리스트는 룰의 우선순위에 따라 연결되어 있기 때문에 현재 노드의 우선순위가 지역 BMR보다 낮은 경우 더 이상 링크드 리스트에 따라 검색을 진행할 필요가 없기 때문이다. 그림 9는 최적화 구조의 검색 과정을 개략적으로 나타낸 코드이다. 와일드카드 프리픽스를 갖는 룰들은 우선순위가 낮은 경우가 대부분이기 때문에 다수의 입력 패킷은 와일드카드 트리로 검색이 확장될 필요가 없어 메인 트리의 검색만으로 최종 BMR을 얻을 수 있어 메모리 접근 횟수가 감소하여 검색 성능이 향상된다.

5. 시뮬레이션 및 성능 평가

본 논문에서 제안하는 구조의 성능을 평가하기 위해 현재 인터넷 라우터에서 사용되는 룰 특성을 반영하는 classBench[5,6]를 사용하여 1000개와 5000개의 룰을 갖는 ACL, FW, IPC의 셋과 인풋을 구성하여 실험하였다. 인터넷 라우터는 초당 수백 개의 데이타 플로우에 대하여 선속도의 비율로 처리하여야 하며 패킷 분류 또

```
Void Search(){
    if (( Local BMR ! = -1) && ( Threshold > Local BMR) {
          // Done : BMR = Local BMR ;
     }

    if (( Local BMR == -1) || ( Threshold < Local BMR) {
          // Wildcard Prefix Tree Search
         longestpfx_nextptr = F1Search(root);
         BMR=F2Search();
    }
}

int F1Search(){
   if (node==-1)
         return longestpfx_nextptr;
   if(src field match){
         longestpfx_nextptr = currentnode.nextptr;
         if(next bit is '1'){
              longestpfx_nextptr = F1Search(right_ptr);
         }
         if(next bit is '0'){
              longestpfx_nextptr = F1Search(left_ptr);
         }
         else return longestpfx_nextptr;
   }
   else if( input is bigger than stored prefix)
         longestpfx_nextptr = F1Search(right_ptr);
   else  longestpfx_nextptr = F1Search(left_ptr);
}

Int F2Search(){
    if (node==-1)
         return BMR;
    if(3 fields match){
         if(Local BMR > Rule #){
              BMR = Rule #;
         }
         else BMR = Local BMR;
         if(next bit is '1'){
              BMR = F2Search(right_ptr);
         }
         if(next bit is '0'){
              BMR = F2Search(left_ptr);
         }
         else return BMR;
    }
    if(only dst field match && linked list && Local BMR > Rule #){
              BMR = F2Search(linked list);
    }
   if(only dst field match && linked list doesn't exist){
         if(next bit is '1'){
              BMR = F2Search(right_ptr);
         }
         if(next bit is '0'){
              BMR = F2Search(left_ptr);
         }
         else return BMR;
   }
   if( input is bigger than stored prefix)
         BMR = F2Search(right_ptr);
  else if (input is smaller than stored prefix)
         BMR = F2Search(left_ptr);
   else
         return BMR;
}
```

그림 9 최적화 구조의 검색 슈도 코드

한 실시간으로 그와 같은 비율로 처리되어야 한다. 빠른 패킷 분류를 위해서는 빠른 검색 속도가 반드시 요구되며 검색 속도는 입력 패킷이 속하는 클래스를 검색하는 과정에서 요구되는 메모리 접근 횟수의 영향이 가장 중요하게 평가된다[3]. 따라서 검색 성능은 BMR을 찾기 위해 필드를 검색하는데 소요되는 평균 메모리 접근 횟수(Tavg)와 최대 메모리 접근 횟수(Tmax)로 평가될 수 있다. 또한 메모리 요구량은 테이블을 저장하는데 소요되는 총 메모리의 크기(M)로서 평가되며 메모리 사이즈가 클수록 패킷 분류를 위한 검색 시간이 많이 소모될 수 있으므로 가능한 최소의 메모리로 구성할 수 있는 알고리즘일 좋은 알고리즘이라 할 수 있다.

표 7에서는 각 룰 셋의 룰의 개수 및 근원지 프리픽스의 네스팅 수, HSP-BST 구조, 최적화 된 HSP-BST 구조의 복사된 룰의 개수를 보였다. HSP-BST 구조와 최적화 된 HSP-BST 구조를 비교하여 보면 최적화된 구조에서 제한된 룰의 복사를 적용함으로써 복사된 룰의 개수가 최대 7674개 줄었음을 알 수 있다. 이는 그만큼의 노드수가 감소함을 의미하므로 메모리 요구량의 감소와 검색 성능의 향상의 근거가 된다.

그림 10, 그림 11, 그림 12와 그림 13에서는 1000개의 룰을 갖는 셋과 5000개의 룰을 갖는 셋에 대하여 HSP-BST 구조, 최적화 된 HSP-BST 구조, 계층적 트라이, 계층적 셋-프루닝 트라이 구조의 평균 메모리 접근 횟수와 최대 메모리 접근 횟수를 비교하였다. 먼저 HSP-BST 구조와 최적화된 HSP-BST 구조를 비교하여 보면 평균 메모리 접근 횟수 면에서 ACL 5K룰 셋의 경우 17.76에서 16.14으로, IPC5K룰 셋의 경우 17.88에서 15.46으로 검색 속도가 향상 되었다. 이는 메인트리의 하위 트리에서 제한된 룰의 복사를 통해 노드 수가 감소하였고, 와일드카드 트리로 검색 영역이 확대되지 않는 경우가 다수이기 때문에 메모리 접근 횟수가 감소한다는 점을 뒷받침한다. FW 5K룰 셋의 경우에는 24.65에서 24.66으로 거의 변화가 없었는데 룰 셋의 특성상 이는 임계값이 높은 우선순위를 갖는 경우로서, 와일드카드 트리로의 검색이 필요한 경우가 많기 때문이다.

표 7 룰 셋

	No. of rules	No. of prefix nesting	No.of rules after rule copy	No.of rules after controlled rule copy
ACL1K	958	4	1334	1163
IPC1K	988	4	8898	3313
FW1K	871	3	7911	3329
ACL 5K	4660	4	13591	8367
IPC 5K	4468	4	21306	11573
FW 5K	3067	3	14294	6620

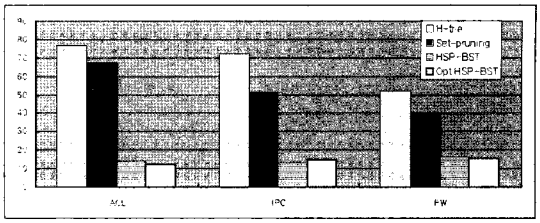


그림 10 1k 룰 셋에 대한 평균 메모리 접근 횟수 비교

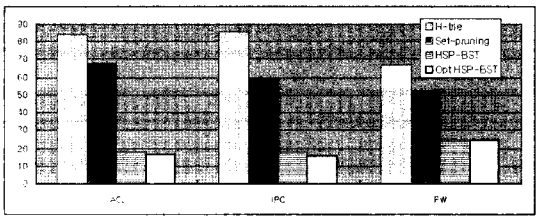


그림 11 5k 룰 셋에 대한 평균 메모리 접근 횟수 비교

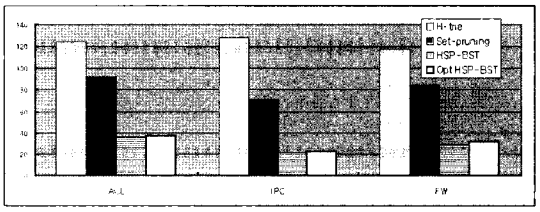


그림 12 1k 룰 셋에 대한 최대 메모리 접근 횟수 비교

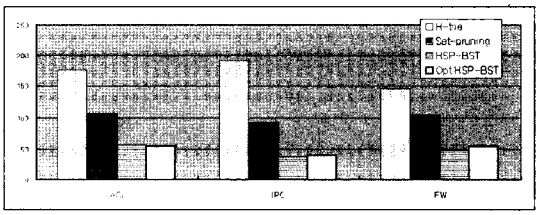


그림 13 5k 룰 셋에 대한 최대 메모리 접근 횟수 비교

최대 메모리 접근 횟수는 ACL 5K룰 셋의 경우 57에서 54로 줄었으나 IPC 5K룰 셋의 경우 37에서 41로, FW 5K 룰 셋의 경우 46에서 54로 늘어나 룰 셋에 따라 다른 특성을 보였다. 이는 전체적으로 평균 메모리 접근 횟수가 감소하는 경우라도 특정 입력값에 대한 최적화된 HSP BST의 메인 트리에서의 검색 결과가 임계값보다 우선순위가 낮은 경우 와일드 트리로 검색 영역이 확장되어 최대 메모리 접근 횟수가 증가하기 때문이다.

제안된 최적화된 HSP-BST 구조의 평균 메모리 접근 횟수는 12.44에서 24.66의 분포를 이루고, HSP-BST 구조의 경우에는 13.0에서 24.65의 분포를 이루고 있는 반면, 계층적 트라이는 52.14에서 85.64, 계층적 셋

-프루닝 트라이는 36.36에서 67.64의 분포를 이루고 있어 제안하는 두 구조의 평균 메모리 검색 성능이 크게 향상되었다는 점을 확인할 수 있다. 이는 룰의 복사를 통하여 역추적이 제거되어 메모리 접근 횟수가 감소하였기 때문이다. 평균 메모리 접근 횟수는 룰 셋의 크기가 1k에서 5k로 증가함에도 크게 증가하지 않는 것을 보아 큰 룰 셋에 대해서도 잘 동작하는 확장성(scalability)이 우수한 구조임을 보여준다.

그림 14와 그림 15에서는 1000개의 룰을 갖는 셋과 5000개의 룰을 갖는 셋에 대하여 HSP-BST 구조, 최적화 된 HSP-BST 구조, 계층적 트라이, 계층적 셋-프루닝 트라이 구조의 메모리 요구량을 비교하였다. 메모리 요구량 측면에서는 최적화를 통해 모든 룰 셋에 대하여 메모리 요구량이 40~60% 감소한다는 점을 확인할 수 있었다. 이는 제한된 룰 복사로 하위 트리로 복사된 노드 수가 감소하였기 때문이다.

표 8에는 제안된 알고리즘의 성능을 평가하기 위하여 기존의 패킷 분류 알고리즘들과 시뮬레이션 결과를 비교하였다. 비교를 위하여 시뮬레이션한 기존의 패킷 분류 알고리즘들로는 위에서 언급한 H-trie[1], Set-pruning trie[3], AQT[7]외에 Priority-Based Quad-tree (PQT)[9], Bit-Vector(BV)[10], Hicut[11], Hypercut[12]를 추가하였다. 간략하게 이 알고리즘들을 소개하면 PQT는 룰의 각 필드를 영역으로 표현하고 영역을 줄여가면서 패킷이 속하는 가장 구체적인 영역을 찾는 방법이다. AQT는 프리픽스로 표현되는 두 필드를 동시에 보면서 전체 검색 범위를 1/4로 줄여가며 패킷이 해당하는 영역 내의 룰들을 얻는데 PQT는 AQT를 개선하기 위하여 룰의 우선 순위를 고려하여 구성한 트리 구조이다. BV는 프리픽스 형태로 표현되는 두 필드를 이용하여 독립적인 이진 트라이를 구성하고, 각각의 필드 트라이의 프리픽스 노드마다 주어진 룰 셋의 사이즈에 해당하는 비트 벡터를 갖는 구조이다. HiCut과 HyperCuts은 커팅을 이용한 패킷 분류 알고리즘으로 룰 셋의 각 필드(field)를 영역(range)으로 표현하고 영역을 줄여가면서 패킷이 속하는 가장 구체적인 영역을 찾는 구조이다. Hicut은 분할 된 영역을 바탕으로 디시전 트리를 구성하며 리프에 저장된 룰의 수를 일정한 수(빈스) 이하로 하여 검색 시간을 조절할 수 있다. HyperCuts은 한번에 여러 필드의 영역을 동시에 검사하여 높이를 2로 제한하는 디시전 트리를 구성하여 검색 성능을 높인 구조이다. 표 8로부터 본 논문에서 제안된 구조는 계층적 트라이, 계층적 셋-프루닝 트라이 구조 뿐 아니라 기존에 연구되어 온 다른 구조에 비해 우수한 검색 성능을 보임을 알 수 있다.

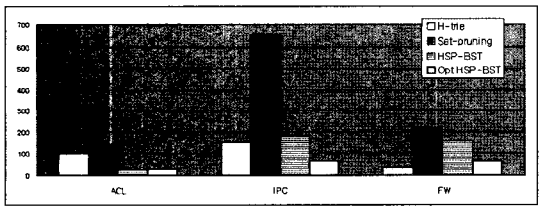


그림 14 1k 룰 셋에 대한 메모리 요구량 비교

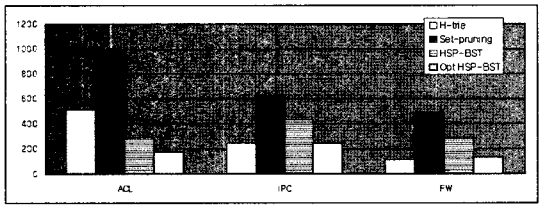


그림 15 5k 룰 셋에 대한 메모리 요구량 비교

표 9에는 각 구조에서 요구되는 메모리 사용량의 비교를 보였다. 메모리 요구량을 살펴보면 제안하는 구조는 약 5000개의 룰을 갖는 커다란 룰 셋에 대하여 500Kbyte 미만의 메모리를 요구하고 있어 패킷분류를 위해 구현된 칩 안에 내장 가능한 메모리를 요구함을 볼 수 있다. 이를 통해 본 논문에서 제안하는 구조는 칩 안에 내장 가능한(on-chip) 적정한 정도의 메모리를 요구하나, 타 구조에 비하여 좋은 검색 성능을 보임을 알 수 있다.

6. 결 론

빠른 패킷 분류 속도를 갖기 위해서는 헤더 필드 중 근원지 IP 주소와 목적지 IP 주소 2개의 필드를 이용하여 계층적으로 룰을 검색하는 계층적 구조가 효율적이며 이러한 구조를 이용한 알고리즘이 많이 제안되었다. 계층적 구조의 효율성은 근원지 IP 주소와 목적지 IP 주소 2개 필드의 조합에 의한 플로우의 다양성이 나머지 필드의 조합에 의한 다양성보다 훨씬 크며, 하나의 필드에 대한 검색이 완료되었을 때 검색해야 할 영역이 크게 감소한다는 점에 근거한다. 그러나 기존의 계층적 트라이는 트라이 내부에 빈 노드가 불가피하였고 최종적으로 가장 우선순위가 높은 룰을 찾기 위한 역추적이 요구되어 검색 성능이 떨어지는 한계가 있었다. 계층적 셋-프루닝 트리는 역추적을 제거하였으나 여전히 빈 노드가 존재하여 메모리 접근 횟수가 증가하므로 개선의 여지가 있었다. 따라서 본 논문에서는 셋-프루닝 기법과 이진 검색을 동시에 적용하여, 비어있는 내부 노드를 포함하지 않으면서도 역추적이 없는 새로운 계층적 패킷 분류 알고리즘을 제안하였다. 제안된 구조는 칩 안에 내장 가능한(on-chip) 적정한 정도의 메모리를 사용하여 구현될 수 있으며, 패킷 분류를 위한 검색성능이 타 구

표 8 룰 셋에 대한 평균 메모리 접근 횟수 비교

Rule set \ Algorithm		HSP-BST	Opt HSP-BST	H-trie [1]	Set-pruning trie[3]	AQT [7]	PQT [9]	BV [10]	HiCuts [11]	HyperCuts [12]
ACL	1K	13.75	12.44	77.2	67.63	38.6	35.6	66.0	68.2	65.2
	5K	17.76	16.14	84.0	67.64	50.13	59.6	64.1	231.5	228.5
IPC	1K	13.32	14.94	71.9	50.86	94.5	73.6	63.6	45.7	42.7
	5K	17.88	15.46	85.6	59.79	344.8	202.1	151.9	132.8	129.8
FW	1K	16.21	15.45	52.1	39.36	369.3	197.9	196.6	106.2	103.2
	5K	24.65	24.66	67.5	52.17	660.5	571.1	738.8	366.5	363.5

표 9 룰 셋에 대한 메모리 요구량 비교

Rule set \ Algorithm		HSP-BST	Opt HSP-BST	H-trie [1]	Set-pruning trie[3]	AQT [7]	PQT [9]	BV [10]	HiCuts [11]	HyperCuts [12]
ACL	1K	27.9	24.5	102.5	138.6	56.4	29.9	153.3	147.2	157.9
	5K	286.1	178.1	509.0	990.9	200.2	145.6	2793	734.2	770.6
IPC	1K	184.5	70.5	157.2	655.7	71.2	30.9	154.3	203.3	212.2
	5K	438.2	240.1	246.8	620.4	234.3	139.9	2351	883.1	925.6
FW	1K	163.6	70.6	41.9	219.0	35.2	27.2	111.9	342.3	359.2
	5K	293.8	138.8	109.9	491.8	479.8	136.0	2340	966.4	1013.0

조에 비하여 향상됨을 시뮬레이션을 통하여 확인하였다.

근원지 프리픽스가 와일드카드인 룰의 경우, 다른 모든 프리픽스에 대하여 인클로저이기 때문에 모든 하위 트리로의 복사가 일어나며 실제 라우터 상에서 대부분 긴 프리픽스를 갖는 구체적인 룰들에 비해 우선순위가 낮다는 특성을 가지고 있다. 따라서 이를 바탕으로 근원지 프리픽스가 와일드카드가 아닌 룰들의 트리를 구성하여 검색한 후 와일드카드인 룰로 이루어진 별도의 트리에서 검색하도록 함으로써, 전자에 해당하는 트리에서는 와일드카드에 의한 복사가 이루어 지지 않도록 하는 제한된 룰의 복사를 적용하여 메모리 요구량을 감소시키고 검색 속도 성능을 향상시키는 최적화 기법을 또한 제안하였다. 시뮬레이션 결과를 통해 이러한 최적화 구조는 제안된 구조의 메모리 요구량을 절반 정도로 감소시켜 타 구조들에 비해 적은 메모리를 사용하여 구현되고 검색 속도 성능이 더욱 향상되었음을 확인하였다.

참 고 문 헌

[1] H.Jonathan Chao, "Next Generation Routers," Predeedings of the IEEE JPROC.2002.802001, Volume 90, Issue 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] G.Vaghese, "Network Algorithmics," Morgan Kaufmann, 2005.

[4] 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.

[5] D. E.Taylor, J. S. Turner, J.S, "ClassBench: a packet classification benchmark," Proc. IEEE INFOCOM 2005, pp. 2068-2079, March 2005.

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

[7] 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.

[8] N.Yazdani and P. S. Min, "Fast and Scalable Schemes for the IP Address Lookup Problem," Proc, IEEE HPSR2000, pp. 83-92, 2001.

[9] 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.

[10] T. V. Lakshnam and D. Stiliadas, "High-speed policy based packet forwarding using efficient multi-dimensional range matching," in Proc. ACM SIGCOMM, pp. 203-214, 1998.

[11] P. Gupta and N. Mckeown, "Classification using hierarchical intelligent cuttings," IEEE Micro, Vol.20, No.1, pp. 34-41, Jan./Feb., 2000.

[12] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in Proc. SiGCOMM, 2003.

이 수 현

2007년 8월 이화여자대학교 정보통신학과, 학사. 2007년 9월~현재 이화여자대학교 전자정보통신공학과, 석사과정. 관심분야는 Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

임 혜 숙

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