# 패킷 분류를 위한 이차원 이진 프리픽스 트리
## (A Two-Dimensional Binary Prefix Tree for Packet Classification)

정 여 진 † 김 혜 란 †† 임 혜 숙 †††
(Yeojin Jung) (Hyeran Kim) (Hyesook Lim)

**요 약** 인터넷은 그 급속한 성장과 더불어 점차 더 나은 서비스를 제공할 것을 요구받게 되었다. 이에 따라 차세대 인터넷 라우터들에서의 지능적인 패킷 분류 기능은 필수 불가결한 것으로 여겨지고 있다. 패킷 분류란 미리 정의된 classifier에 의거하여 입력된 패킷에 매치하는 가장 순위가 높은 룰을 찾는 과정이다. 기존에 나와있는 많은 패킷 분류 검색 구조들이 출발지, 목적지 프리픽스 필드에 기반하여 룰을 추려내는 접근 방법을 사용하고 있다. 그러나 대부분의 검색 구조들은 출발지, 목적지 프리픽스 검색을 위하여 트라이 구조에 바탕을 둔 순차적인 일차원 검색을 따르고 있으며, 매우 큰 메모리를 요구한다는 단점을 가지고 있다. 본 논문에서는 메모리를 매우 효율적으로 사용하면서도 출발지-목적지 프리픽스 쌍에 기반한 이차원 패킷 분류 구조를 제안하고자 한다. 코드워드로 구성된 이진 프리픽스 트리를 구성함으로써, 출발지 프리픽스 검색과 목적지 프리픽스 검색이 하나의 이진 트리를 통해 동시에 가능하도록 하였다. 또한 본 논문에서 제안하는 구조인 이차원 이진 프리픽스 트리는 트리 구조 내부에 비어있는 노드를 포함하고 있지 않으므로 트라이 구조가 가지고 있는 메모리의 비효율성 문제를 완전히 제거하였다.

**키워드** : 패킷 분류, 이진 프리픽스 트리, 이차원 검색, 코드워드, 메모리 효율

***Abstract*** Demand for better services in the Internet has been increasing due to the rapid growth of the Internet, and hence next generation routers are required to perform intelligent packet classification. For a given classifier defining packet attributes or contents, packet classification is the process of identifying the highest priority rule to which a packet conforms. A notable characteristic of real classifiers is that a packet matches only a small number of distinct source-destination prefix pairs. Therefore, a lot of schemes have been proposed to filter rules based on source and destination prefix pairs. However, most of the schemes are based on sequential one-dimensional searches using trie which requires huge memory. In this paper, we proposea memory-efficient two-dimensional search scheme using source and destination prefix pairs. By constructing binary prefix tree, source prefix search and destination prefix search are simultaneously performed in a binary tree. Moreover, the proposed two-dimensional binary prefix tree does not include any empty internal nodes, and hence memory waste of previous trie-based structures is completely eliminated.

**Key words** : packet classification, binary prefix trie, two-dimensional search, memory efficiency

## 1. 서 론

군사적 목적의 작은 네트워크에서 시작된 인터넷은, 성장을 거듭한 끝에 이제는 전 세계를 하나로 묶은 거대한 네트워크로 발전하게 되었으며, 이에 따라 제한된 서비스 만을 제공하는 초기의 소극적이고 한정적인 역할에서 벗어나 사용자의 요구에 따라 맞춤 서비스를 제공할 것을 요구받게 되었다. 이와 같은 한차원 높은 서비스를 지원하기 위해서는 인터넷 상의 플로우들을 식별하고 이에 따라 다른 처리가 행해져야 하는데, 이것은 라우터의 주된 기능 중에 하나인 패킷 분류(packet classification)를 통해서 가능하여진다.

패킷 분류란 말 그대로 라우터로 들어오는 패킷들을 미리 정해진 룰들에 따라 그 클래스를 분류하는 과정을 말하며, 패킷 분류를 통해 분류된 패킷들은 각 플로우에 따라 적절한 서비스를 제공받게 된다. 패킷들의 클래스

† 비 회 원 : 삼성전자 정보통신총괄
surya@ewha.ac.kr
†† 학생회원 : 이화여자대학교 정보통신학과
hranhi@hanmail.net
††† 비 회 원 : 이화여자대학교 정보통신학과 교수
hlim@ewha.ac.kr
논문접수 : 2004년 10월 14일
심사완료 : 2005년 4월 14일

를 결정하기 위해 사용되는 룰들을 이루는 필드에는 주로 목적지 IP 주소, 출발지 IP 주소, 목적지 포트 넘버, 출발지 포트 넘버, 프로토콜 등 패킷의 헤더 정보들이 사용되게 된다. 표 1은 classifier의 예를 보이고 있다. 결국 패킷 분류는 여러 개의 필드들을 보고 패킷을 분류하는 다차원 검색으로, 이것은 패킷을 적절한 아웃풋 포트로 내보내기 위하여 목적지 IP 주소를 사용하는 일차원 검색인 IP 주소 검색을 다차원 검색으로 확장한 것으로 이해될 수 있다.

패킷 분류는 매 패킷마다 룰을 구성하는 모든 필드들에 대하여 조건을 검사하고, 매치하는 여러 룰 중에서 가장 순위가 높은 룰을 선택하는 방식으로 수행된다. 이 때 필드의 표현 방식에 따라 각기 다른 연산이 수행되어야 하며 각 패킷들이 여러 룰에 매치할 수 있으므로 매치하는 모든 룰들을 찾아 가장 순위가 높은 룰을 선택해야 하는 복잡성이 있음에도 불구하고 검색은 매 패킷에 대하여 실시간으로 이루어져야 한다는 데에 패킷 분류의 어려움이 있다고 할 수 있다[1].

그간 패킷 분류를 위하여 많은 연구가 이루어졌는데, 주된 연구 방향은 어떻게 하면 효율적으로 다차원 검색을 수행할 것인가로, 룰을 이루는 여러 필드 중에서도 출발지 IP 주소와 목적지 IP 주소에 그 초점이 맞추어져 왔다. 이것은 출발지 IP 주소와 목적지 IP 주소에 따른 플로우의 다양성이 다른 필드에 따르는 플로우의 다양성보다 월등히 크다는 분석에 기초하고 있다[2]. 효율적인 이차원 패킷 분류 방법에의 수많은 연구에도 불구하고 대다수의 기존의 검색 구조들은 진정한 의미의 이차원 검색이 아닌 일차원 검색의 순차적 연결에 그 논의를 국한하고 있고, 이차원 검색 방법의 경우에도 개선의 여지가 매우 크다고 여겨진다. 따라서 본 논문에서는 IP 주소 검색을 위한 일차원 검색 구조인 이진 프리픽스 트리를 이차원으로 확장한 이차원 이진 프리픽스 트리(binary prefix tree) 구조를 제안하고자 한다.

본 논문의 구성은 다음과 같다. 우선, 2장에서 기존의 패킷 분류 검색 구조들에 대하여 살펴본 후, 3장에서 제안하는 구조에 대하여 설명한다. 4장에서는 제안하는 구조의 시뮬레이션 결과 및 기존의 다른 검색 구조와의 비교를 보이고, 끝으로 5장에서 결론을 맺는다.

## 2. 기존의 알고리즘

다차원 검색을 위한 가장 간단한 구조는 룰들에 대하여 순차적으로 검색을 수행하는 것이다. 이 방법은 가장 간단한 방법이면서 작은 메모리를 사용하지만 룰의 수에 따라 검색 시간이 선형적으로 증가하기 때문에 룰의 수가 많은 경우에는 적합하지 못한 방법이다. 따라서 검색 시간을 줄이기 위하여 여러가지 패킷 분류 검색 구조들이 제안되었는데 본 장에서는 그 중에서 몇 가지 검색 구조들을 그 접근 방법에 따라 일차원 검색의 순차적 연결에 기반을 둔 방법과 이차원 검색에 기반을 둔 방법으로 나누어 살펴보고자 한다.

본격적인 논의에 앞서, 앞으로 본 논문의 전개에 있어서의 다차원 검색이란 출발지 IP 주소와 목적지 IP 주소로 이루어진 이차원 검색에 기반한 검색을 의미함을 미리 밝혀둔다.

우선 일차원 검색의 순차적 연결에 기반을 둔 방법으로 가장 많이 연구되는 것이 트라이 구조를 이용한 검색 구조이다[3]. 이 구조의 가장 간단한 형태인 계층 트라이(hierarchical trie)는 룰을 구성하는 필드들 중, 프리픽스로 표현된 각각의 필드들에 대하여 별도의 트라이를 구성하고 그 트라이들을 계층적으로 연결한 것이다. 첫번째 필드에 해당되는 트라이에서 매치되는 모든 엔트리에 대하여, 그 엔트리에 계층적으로 연결된 하위 필드 트라이로 검색을 계속 진행하는 방식으로 각 필드에 대한 검색을 순차적으로 수행하게 된다. 패킷 분류는 매치할 수 있는 모든 룰을 찾고 이 중에서 가장 순위가 높은 룰에 따라 클래스를 결정하게 되므로, 이러한 계층

표 1 classifier의 예

| | Src. Prefix | Dst. Prefix | Src. Port Num. | Dst. Port Num. | Protocol |
|---|---|---|---|---|---|
| 1 | 10111* | * | 23 | 23 | TCP |
| 2 | 1001* | 101* | 80 | 30 | TCP |
| 3 | * | 10* | >1024 | >1024 | UDP |
| 4 | 10* | 11* | * | * | * |
| 5 | 01* | 0011* | 80 | * | TCP |
| 6 | 100* | 1011011* | 21 | 21 | TCP |
| 7 | 1001* | 0111* | <1023 | * | * |
| 8 | 101110* | 1001* | 4767 | * | UDP |
| 9 | 10* | 1011* | * | * | UDP |
| 10 | 100100* | 1010011* | * | 21 | * |
| 11 | * | 0011* | * | * | * |

트라이를 이용한 검색 방식의 경우 매치하는 모든 룰을 찾기 위한 back-tracking이 요구된다[4]. [4]와 [5]에서는 룰의 복사나 스위치 포인터를 이용하여 이러한 back-tracking을 방지, 검색 시간의 복잡도를 $O(W^d)$에서 O(dW)로 단축시킨 방법으로 셋 푸루닝 트라이(set-pruning trie)와 그리드 트라이(grid-of-trie)를 제안하고 있다. 이와 같은 구조들은 계층 트라이 구조보다 검색 시간을 줄인 대신에 선처리 연산 시간이 많이 소요된다.

그림 1은 표 1의 classifier를 사용하여 계층 트라이를 구성한 예이다. 그림 1의 계층 트라이의 경우, 출발지 프리픽스 트라이(F1 트라이)에서도 빈 노드로 인하여 메모리의 효율성이 떨어지며, 목적지 프리픽스 트라이(F2 트라이)의 경우에는 F1의 유효한 노드마다 별도의 F2 트라이를 가지게 되므로 메모리의 효율성은 더욱 현저하게 떨어지게 된다.

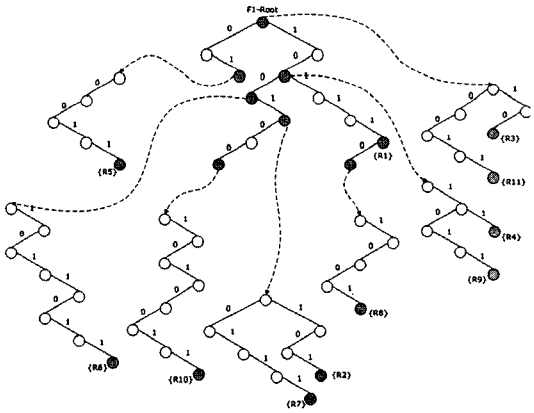


그림 1 계층 트라이

그 다음으로 이차원 검색 구조로서 area-based quadtree(AQT)가 있다[6]. AQT는 매 단계에서 프리픽스로 표현된 두 필드를 동시에 보면서 전체 검색 범위를 1/4로 줄여가며 패킷이 해당하는 영역(range) 내의 룰들을 얻는 방법이다. 즉 매 단계에서 출발지 프리픽스와 목적지 프리픽스 각각의 영역을 이등분하게 된다. 이와 같은 AQT의 검색 구조는 출발지 IP 주소와 목적지 IP 주소를 동시에 포함하고 있는 4-way 트라이 구조로 생각될 수 있다. 이 방법의 경우 프리픽스로 표현된 두 필드가 매치하는 룰의 리스트를 얻기 위하여 최대 프리픽스 길이만큼의 검색 시간이 요구된다.

그림 2는 표 1의 classifier를 사용하여 AQT 트라이를 구성한 예이다. 그림 2의 AQT의 경우에도 트라이 구조에 기반을 두고 있기 때문에 빈 노드는 불가피하며, 룰을 이루고 있는 프리픽스의 길이가 길어질수록 메모리의 비효율성은 더욱 심화된다.

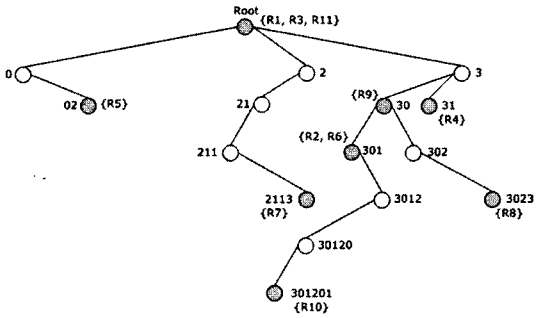


그림 2 AQT

이와 같은 검색 방법들은 트라이 구조의 최대 단점인 비어있는 노드를 저장하는데서 오는 메모리 비효율성의 한계를 극복하지 못하고 있다. 본 논문에서는 이차원 검색 구조를 표현함에 있어 중간에 빈 노드를 포함하는 트라이 구조 대신 빈 노드를 포함하지 않는 트리 구조를 채택함으로써 이차원 검색을 위해 메모리 효율을 극대화하면서 이진 검색이 가능한 구조를 제안한다.

## 3. 제안하는 알고리즘

### 3.1 1차원 이진 프리픽스 트리

일차원 검색인 IP 주소의 검색은 프리픽스 길이의 가변성 때문에 매치하는 프리픽스들 중 가장 길게 매치하는 프리픽스를 선택하는 longest prefix matching (LPM) 방식으로 이루어진다. 프리픽스 길이의 가변성은 프리픽스간에 크기 비교를 어렵게 하고, 또한 프리픽스간의 포함관계를 형성하여 IP 주소 검색에 있어 바이너리 검색을 불가능하게 하였다. 따라서 메모리의 비효율성에도 불구하고 기존의 일차원 IP 주소 검색 구조들은 주로 트라이 구조에 기반을 두고 있다. 그러나 그림3에서 보이는 바와 같이 Yazdani 등은 프리픽스들간의 크기 비교를 정의하고, 포함관계를 가지는 프리픽스간의 관계를 정의하여 새로운 바이너리 검색 구조를 제안하였다[7]. 여기에 Lim 등은 그 논의를 더욱 발전시켜 서로 디스조인트한 프리픽스들은 균형 트리를 형성할 수 있다는 점에 기초, Yazdani의 이진 프리픽스 트리(binary prefix tree)를 여러 개의 균형 트리로 재구성하였다[8]. 이와 같은 이진 프리픽스 트리는 메모리의 효율을 극대화시켜 트라이 구조에 기반한 많은 검색 구조들이 가진 메모리 비효율성 문제를 해결함과 동시에 검색 시간에 있어서도 우수한 성능을 보였다. 본 논문에서는 이러한 일차원 IP 주소 검색을 위한 이진 프리픽

스 트리 개념을 패킷 분류를 위한 이차원 트리로 확장하였다.

Definition 1 Compare
Assume there are two prefixes $A=a_1a_2a_3...a_n$ and $B=b_1b_2b_3...b_m$ where $a_i$ and $b_j$ are 0 or 1 for all is and js.
a. If n=m, two strings have the same length, the numerical values of A and B are compared
Ex) For A=1001 and B=1100 , B>A
b. If n!=m (assume n<m), the two substrings $A=a_1a_2a_3...a_n$ and $B=b_1b_2b_3...b_n$ are compared.
The prefix with bigger (smaller) numeric value is considered bigger (smaller).
If $a_1a_2a_3...a_n$ and $B=b_1b_2b_3...b_n$ are equal, then the $(n+1)^{th}$ bit of string B is checked.
It is considered B>A if $b_{n+1}=1$ and B<=A otherwise
Ex) For A=1001 and B=110000 , B>A
Ex) For A=1001 and B=100110 , B>A
Ex) For A=1001 and B=100100 , A>B
Definition 2 Match
A Prefix is a substring of other prefix.
Ex) For A=1001 and B=100100, A and B are matched.
Definition 3 Disjoint
Two prefixes are not matched
Ex) For A=1001 and B=111, A and B are disjoint.
Definition 4 Enclosure
A prefix A is called an enclosure if there exist at least one element such that A is a prefix of that data element
Ex) For A=1001 and B=100100, A is an enclosure of B.

그림 3 Yazdani의 정의

### 3.2 이차원 이진 프리픽스 트리로의 확장

제안하는 구조인 이차원 이진 프리픽스 트리 (2-dimensional binary prefix tree: 2D-BPT)는 출발지 IP 주소와 목적지 IP 주소로부터 코드워드들을 생성하여 이 코드워드들을 가지고 이진 프리픽스 트리를 구성하여 하나의 이진 검색을 통해 이차원 검색을 수행할 수 있는 구조이다. 앞서 보인 표 1의 classifier 의 예를 사용하여 제안하는 구조에 대하여 설명하고자 한다.

• 코드워드

이차원 BPT를 구성하기 위한 첫 단계는 출발지 프리픽스와 목적지 프리픽스로부터 코드워드를 생성하는 것이다. 출발지 프리픽스와 목적지 프리픽스를 같은 위치의 한 비트씩 조합하여 두 프리픽스 중 짧은 길이에 해당하는 코드워드를 만든다. (출발지 프리픽스 비트, 목적지 프리픽스 비트)이 (0, 0)이면 0이고 (0, 1)이면 1, (1, 0)이면 2, (1, 1)이면 3으로 대치하여 코드워드를 만들면 (출발지 프리픽스, 목적지 프리픽스)의 이차원 검색 영역은 두프리픽스를 모두 포함하는 일차원 코드워드 검색으로대치되게 된다. 그림 4는 표 1의 룰들로부터 코드워드를 생성하는 과정을 보이고 있다. 예를 들어 룰 2의 경우를 보면 같은 위치의 (출발지 프리픽스 비트, 목적지 프리픽스 비트) 쌍을 짧은 프리픽스 길이까지조합하면 (1, 1), (0, 0), (0, 1)로 조합이 가능하고 이는 코드워드 '301'로 표현된다. 이와 같은 방식으로 두 프리픽스가 모두 와일드카드가 아닌 룰들에 대하여 코드워드를 생성하고, 한쪽이 와일드카드인 경우에는 유효한 프리픽스를 그대로 코드워드로 사용하게 된다.

• 이차원 이진 프리픽스 트리 구성

그림 4와 같이 각 룰의 (출발지 프리픽스, 목적지 프리픽스) 쌍이 일차원 코드워드로 대치되면 이 코드워드를 가지고 이진 프리픽스 트리 (BPT)를 구성할 수있다.

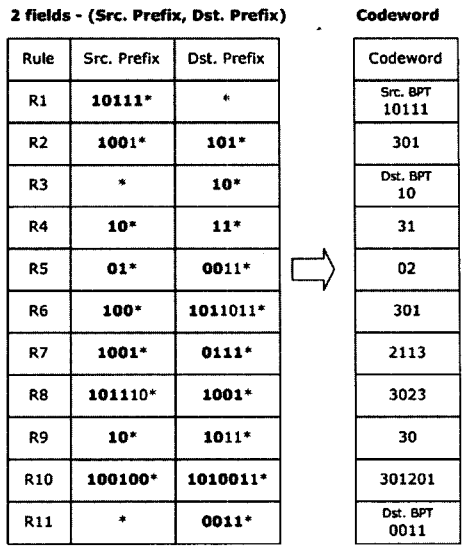


그림 4 코드워드 생성

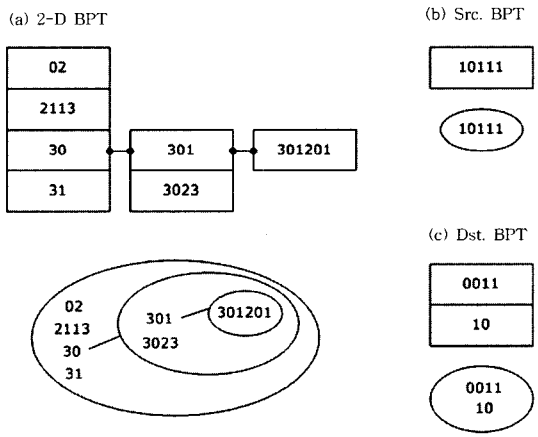


그림 5 이차원 이진 프리픽스 트리를 위한 분류 과정

이때 2D-BPT, Src-BPT, Dst-BPT, 이렇게 세가지 종류의 BPT가 구성되는데, 2D-BPT는 (출발지 프리픽스, 목적지 프리픽스) 쌍이 모두 와일드카드가 아닌 룰들로부터 생성된 코드워드들로 구성되고, 목적지 (출발지) 프리픽스가 와일드카드인 룰들은 출발지(목적지) 프리픽스를 그대로 코드워드로 사용하여 Src(Dst)-BPT를 구성하게 된다. BPT를 구성하는 방법은 앞서 일차원 BPT와 동일하며 프리픽스 대신에 코드워드로 대치된 것만이 다르다. 이때 Yazdani의 인클로져와 디스조인트의 정의는 그대로 사용되나 본 논문에서는 Lim[8]등이 제안한 균형 트리 방식을 사용하였다.

그림 5는 BPT들의 구성 단계를 보이고 있다. 그림 5의 (a)를 보면, 우선 Yazdani의 정의에 따라 코드워드

간 인클로져와 인클로져와 매치하는 엔트리간의 포함관계가 결정된 후, 인클로져를 포함하는 엔트리들을 제외한, 서로 디스조인트한 관계에 있는 코드워드만을 사용하여 균형 바이너리 트리를 구성하게 된다. 이때 인클로져를 포함하는 엔트리들은 자신의 인클로져의 하위 트리로 연결되어 다시 밸런스한 바이너리 트리를 구성하게 된다. 2D 코드워드에 해당하는 02, 2113, 30, 31, 301, 3023, 301201 중 301, 3023, 301201는 30의 하위 트리에 속하므로 이들을 제외한 02, 2113, 30, 31이 바이너리 트리를 구성하게 되며, 메모리에는 크기 순서로 저장되어 검색시 바이너리 검색이 수행되게 된다. 301, 3023, 301201는 다시 인클로져와 하위 트리에 속하는 엔트리, 그리고 디스조인트한 엔트리로 구별되어 같은 과정을 반복하게 된다. 이렇게 정렬된 코드워드들은 그림 5의 관계로 정렬되어 메모리에 순차적으로 저장되며 그림 6과 같은 트리 구조를 형성하게 된다. 그림 6에서 보이는 바와 같이 본 논문에서 제안하는구조는 그림 1의 계층 트라이나 그림 2의 AQT와는 달리 빈 노드없이 유효한 노드만으로 구성되므로 메모리의 낭비가 없고 여러개의 룰이 하나의 코드워드에 매핑되는 경우에도 하나의 코드워드만이 저장되므로 검색 시간에 대한 패널티없이 메모리 효율의 극대화를 가져오게 된다.

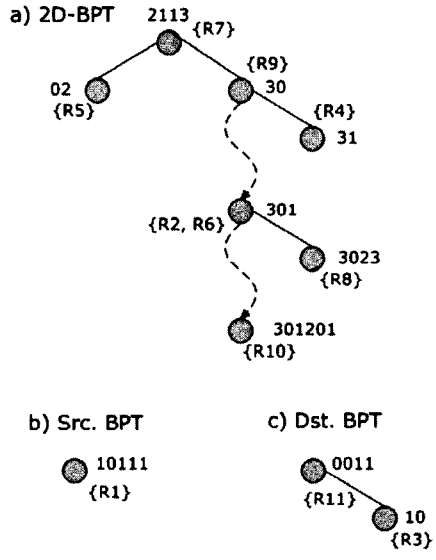


그림 6 제안하는 이차원 이진 프리픽스 트리

• 메모리 구조

본 논문에서 제안하는 패킷 classification 검색 구조는 그림 7과 같은 세 가지 종류의 테이블로 구성된다. 우선 2D-BPT, Src-BPT, Dst-BPT의 시작 주소와 엔트리 수를 저장하는 어드레스 테이블을 갖으며, 이러한 BPT 데이타 구조를 저장하는 BPT 테이블, (출발지 프리픽스, 목적지 프리픽스) 쌍중 BPT에서 완전히 커버하지 못한, 둘 중 더 긴 프리픽스와 (출발지 프리픽스, 목적지 프리픽스)를 제외한 나머지 필드들이 저장되어 있는 룰 테이블을 갖는다. 그림 7은 표 1의 룰들을 이용하여 본 논문에서 제안하는 구조에 따라 테이블들을 구성한 것이다. 그림 7의 (a)의 어드레스 테이블은 위에서 언급한 것처럼 BPT 테이블에서의 각 BPT의 시작 주소와 엔트리 수를 알려줌으로써 BPT 테이블의 전체 영역을 작은 BPT들의 검색 영역으로 분할, 순차적 검색을 가능하게 하고, (b)의 BPT 테이블은 BPT를 저장하고 있으며 이 테이블에 대해서는 바이너리 검색이 수행된다. BPT 테이블은 각 BPT를 이루는 코드워드뿐만이 아니라 각 엔트리들의 인클로져와 child의 관계에 따라 하위 트리 관계를 형성, 하위 트리의 위치 정보와 각 엔트리에 해당하는 룰 메모리의 주소 정보를 포함하고 있다. 룰 테이블(그림 8의 (c))은 검색의 마지막 단계에서 사용되는데, 동일한 코드워드를 가지는 룰들이 우선 순위에 따라 링크드리스트로 연결되어 있다. 이 단계에서

(a) Address Table

| 2-D BPT Address | No. of Entries | Src. BPT Address | No. of Entries | Dst. BPT Address | No. of Entries | Wildcard Linked Address |
|---|---|---|---|---|---|---|
| 0 | 4 | 7 | 1 | 8 | 2 | - |

(b) BPTs Table

| | Codeword | Codeword Length | Rule Table Address | Sub-tree Address | No. of Sub-tree Entries |
|---|---|---|---|---|---|
| 0 | 02 | 2 | 4 | - | - |
| 1 | 2113 | 4 | 6 | - | - |
| 2 | 30 | 2 | 8 | 4 | 2 |
| 3 | 31 | 2 | 3 | - | - |
| 4 | 301 | 3 | 1 | 6 | 1 |
| 5 | 3023 | 4 | 7 | - | - |
| 6 | 301201 | 6 | 9 | - | - |
| 7 | 10111 | 5 | 0 | - | - |
| 8 | 0011 | 4 | 10 | - | - |
| 9 | 10 | 2 | 2 | - | - |

(c) Rule Table

| | Rule Number | CFS Flag | Prefix | Prefix Length | Src. Port Number | Dst. Port Number | Protocol | Linked-list Pointer | Stack Pointer |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | * | 0 | 23 | 23 | TCP | - | - |
| 1 | 2 | 2 | 1001* | 4 | 80 | 80 | TCP | 5 | 8 |
| 2 | 3 | 0 | * | 0 | >1024 | >1024 | UDP | - | - |
| 3 | 4 | 0 | * | 0 | * | * | * | - | - |
| 4 | 5 | 1 | 0011* | 4 | 80 | * | TCP | - | - |
| 5 | 6 | 1 | 1011011* | 7 | 21 | 21 | TCP | - | 8 |
| 6 | 7 | 0 | * | 0 | <1023 | * | * | - | - |
| 7 | 8 | 2 | 101110* | 6 | 4767 | * | UDP | - | 8 |
| 8 | 9 | 1 | 1011* | 4 | * | * | UDP | - | - |
| 9 | 10 | 1 | 1010011* | 7 | * | 21 | * | - | 1 |
| 10 | 11 | 0 | * | 0 | * | * | * | - | - |

그림 7 제안하는 테이블 구조

는 코드워드로 다 표현되지 못한 프리픽스와 룰의 나머지 필드들이 비교된다. CFS 플래그는 테이블에 저장된 프리픽스가 유효한 경우와 그렇지 않은 경우를 구별해 주며, 유효한 프리픽스인 경우에는 출발지 프리픽스인지 목적지 프리픽스인지에 따라 다른 값을 가지게 된다. 스택 포인터는 링크드리스트를 따라 검색을 수행하다가 매칭 룰을 발견하거나 링크드리스트의 끝에 도달하였을 때 추가적으로 검색을 진행해야 하는 가장 가까운 인클로져의 링크드리스트의 시작 주소를 의미한다.

• 검색 과정

제안하는 패킷 분류 구조에서의 검색은 다음의 세단계로 이루어진다. 첫 단계는 어드레스 테이블에서 검색해야 할 BPT의 주소와 엔트리 수를 얻어내는 과정이다. 이 때 얻은 정보를 이용하여 각 BPT에 대해서는 바이너리 검색을 적용하되, 2D-BPT, Src-BPT, Dst-BPT는 순차적으로 검색되게 된다. 여기서 BPT의 엔트리 수는 각 BPT를 이루는 전체 엔트리 수가 아니라 각 BPT의 첫번째 트리의 엔트리 수, 즉 첫 단계에서 바이너리 검색이 되어야 할 엔트리 수를 의미하게 된다. 다음 단계는 BPT 테이블에서의 바이너리 검색이다. 첫번째 단계에서 얻은 BPT의 시작 주소와 엔트리 수를 이용하여 바이너리 검색에 의해 매칭엔트리를 찾고 매칭 엔트리가 존재할 경우, 하위 트리의 존재 여부를 보고 하위 트리 존재시, 하위 트리에서 다시 매칭 엔트리를 찾는 과정을 반복하여 가장 길게 매치하는 코드워드를 선택하게 된다. 즉, BPT의 구성 및 검색은 일차원 BPT에서와 동일하게 이루어지며 차이점은 BPT가 프리픽스가 아닌 코드워드로 구성되는 점이다. 따라서 하나의 BPT를 검색하지만 이 과정에서 (출발지 프리픽스, 목적지 프리픽스) 쌍의 이차원에 대한 검색이 이루어지게 되는 것이다.

검색의 마지막 단계는 룰 테이블에서의 순차적 검색이다. 두번재 단계에서 매칭 엔트리를 발견하게 되면 그 엔트리에 해당하는 룰들의 링크드리스트의 시작 주소를 얻게 되는데, 이 주소를 이용하여 룰 테이블에서 해당 룰들을 순차적으로 검색하게 되는 것이다. 이 룰들을 우선 순위순서로 연결되어 있기 때문에 검색을 진행하다가 매칭 룰을 찾게 되는 경우 링크드리스트의 하위 룰들에 대해서는 검색을 진행할 필요가 없다. 이것은 패킷 분류가 룰의 우선 순위에 기반을 두고 결정되기 때문이다.

여기서 한가지 간과하지 말아야 할 점은 패킷분류는 IP 주소 검색에서의 검색과 달리 longest 프리픽스 match(LPM)에 기반하지 않고 매칭되는 모든 룰들은 검색하여 가장 우선 순위가 높은 룰들을 결정해야 한다는 것이다. 따라서 본 구조의 BPT에서 매칭되는 모든 엔트리의 링크드리스트를 검색하여 매칭하는룰들을 찾아야 한다. 그럼에도 불구하고 BPT에서 LPM과 같은 방식의 검색의 적용은 룰 테이블의 스택 포인터로 가능해지게 된다. 제안하는 구조의 BPT에서 LPM 방식으로 엔트리가 찾아지게 되는 경우, 그 엔트리의 인클로져에 해당하는 엔트리들도 매칭 엔트리가 되며 이 인클로져들의 링크드리스트의 룰들 또한 검색의 대상이 되어야 한다. 그러나 본 논문에서 제안하는 구조에서는 BPT의 특정 엔트리에 매치되는 룰들이 가장 가까운 인클로져의 링크드리스트의 시작 주소를 스택 포인터로 연결하는 방식을 취하였기 때문에 BPT에서 LPM 방식으로 검색이 가능하다.

링크드리스트 포인터와 스택 포인터는 그림 7에 잘 나타나 있다. 그림 7을 보면, 룰 2와 룰 6은 모두 동일한 코드워드인 '301'에 해당하고 따라서 두 둘은 우선 순위에 따라 링크드리스트에 의해 연결되어 있다. 여기서 코드워드 '30'은 '301'의 인클로져가 되고 '301'은 '301201'의 인클로져가 된다. BPT의 검색에서 '301201'이 매칭 엔트리로 결정되면 우선 이 엔트리의 링크드리스트인 룰 10의 매칭 여부를 확인하면, '301201' 엔트리의 링크드리스트에서의 검색은 끝이 나지만, 룰 10은 '301201'의 인클로져인 '301'의 링크드리스트의 시작 주소인 1을 스택 포인터로 가지고 있어 상위 인클로져에 매칭하는 룰들의 검색이 연이어지도록 한다. 마찬가지로 '301'의 매칭 룰들은 '301'의 인클로져인'30'의 링크드리스트의 주소인 '8'을 스택 포인터로 가지고 있어 '301'의 링크드리스트의 검색 후, '30'의 링크드리스트의 검색이 가능하도록 한다. 인클로져와 child 관계에 있는 링크드리스트들을 하나의 리스트로 연결하는 방법도 가능하지만, 이 경우에는 각 리스트들은 우선 순위에 따라 연결되어도 전체적인 리스트는 우선 순위에 따라 연결되어 있지 않으므로 이들 리스트에서 우선 순위가 가장 높은 룰을 찾기 위해서는 리스트의 모든 룰들을 검사해야 한다. 그러나 스택 포인터를 이용하게 되면 각 리스트들이 연결되어 있어도 각 링크드리스트의 구별이 가능하므로 하나의 리스트 내에서 순위가 낮은 룰들은 검사하지 않고 건너뛰는 것이 가능해져 링크드리스트의 검색 시간을 줄일 수 있게 된다.

본 논문에서 제안하는 구조에서는 하나의 코드워드에 매칭하는 룰들을 링크드리스트로 연결하여 순차적으로 검색하는 가장 단순한 방법을 채택하였지만 이들 리스트들의 검색에 있어 BPT와 동일한 방법, 또는 다른 영역 (range)검색 등의 방법을 통해 검색 시간을 줄이는 방법도 가능하다.

## 4. 실험 결과

본 논문에서 제안하는 구조의 알고리즘을 실제 라우

표 2 실험 결과 (One memory access is one word)

| Rule | | DB1 | DB2 | DB3 |
|---|---|---|---|---|
| No. of rules | | 2250 | 2375 | 4710 |
| Address Table | | 1 | 1 | 1 |
| BPT Table Search | Max. | 25 | 37 | 35 |
| | Min. | 8 | 9 | 7 |
| | Avg. | 18 | 23 | 23 |
| Rule Table Search | Max. | 16 | 45 | 31 |
| | Min. | 1 | 1 | 1 |
| | Avg. | 10 | 18 | 29 |
| Total Search Time | Max. | 39 | 77 | 67 |
| | Min. | 13 | 16 | 18 |
| | Avg. | 29 | 42 | 53 |
| Required Memory Size(byte) | | 56k | 60k | 141k |

터에서 사용되는 룰들을 이용하여 시뮬레이션하였다. 라우터에서 사용 중인 3개의 룰 셋에 대하여 각 룰들이 동일한 확률로 들어온다는 가정 하에 각 룰에 매치하는 헤더 정보들을 입력으로 사용하고 룰의 와일드카드에 해당하는 부분에 대해서는 임의의 값을 생성하여 사용하였다. 표 2에서 본 논문에서 제안하는 구조의 검색 시간과 소요되는 메모리 크기에 관한 시뮬레이션 결과를 보여주고 있다. 표 2의 메모리 횟수는 각 테이블을 액세스하는 횟수로 BPT 테이블의 액세스 횟수는 2D-BPT, Src-BPT, Dst-BPT를 순차적으로 검색하는데 소요되는 검색 횟수를 모두 더한 것이고, 룰 테이블의 액세스 횟수 역시 각 BPT의 매칭 에트리의 링크드리스트를 순차적으로 검색하는 데 소요되는 횟수이다.

표 3은 기존의 다른 패킷 분류 구조들과 본 논문에서 제안하는 구조의 성능을 비교해 놓은 것이다. 기존의 구조를 위해서는 참고 논문 [2]의 2799개의 룰을 갖는 데이타베이스의 결과를 이용하였고, 제안하는 구조를 위해서는 4710개의 룰을 갖는 DB3의 결과를 이용하였다. 본 논문에서 사용된 룰들과 기존의 검색 구조의 성능 평가에 사용된 룰들은 동일하지 않은 것이기 때문에 직접적인 결과 비교는 어렵지만, 본 시뮬레이션에서 사용된 룰 역시 실제로 사용되는 룰이고 여러 크기의 룰 셋에 대하여 시뮬레이션 한 결과 일관성있는 결과를 보이고 있으므로, 본 논문에서 제안하는 구조의 성능을 기존의 검색 구조의 성능에 견주어 평가함에 있어 부족함이 없다고 여겨진다. 표 3에서 쓰인 제안하는 구조의 메모리 액세스 횟수는 기존의 검색 구조와의 비교를 위하여 [2]에서 쓰인 방법으로 메모리 검색 횟수를 재계산한 것이다. 메모리 액세스 한번에 한 워드의 데이타를 읽어온다는 가정하에 각 테이블의 폭을 워드로 나눈 값에 각 테이블 액세스 횟수를 곱하여 계산하였다. 표 3을 보면 소요되는 메모리 크기면에서는 괄목할만한 성능을 보이

표 3 성능비교

| Algorithms | Required Memory Size (Word) | Number of Memory Accesses (Worst case) |
|---|---|---|
| RFC | 747,271 | 12 |
| HiCuts - 4 | 117,801 | 82 |
| HiCuts - 1 | 25,543 | 172 |
| BV | 276,604 | 846 |
| ABV | 285,099 | 196 |
| EGT | 75,376 | 154 |
| EGT - PC | 30,753 | 87 |
| Proposed 2D- BPT | 35,095 | 264 |

고 있으며, 메모리 액세스 면에서도 룰 테이블에서의 검색을 순차적으로 수행하였음에도 불구하고 좋은 성능을 보임을 알 수 있다. 룰 테이블에서의 검색도 순차적 검색 대신 BPT 테이블에서와 같은 바이너리 검색이나 range 검색 방식을 적용한다면 검색 시간에 있어서 성능 개선을 가져올 것으로 사료된다.

## 5. 결 론

본 논문에서는 메모리를 효율적으로 사용하면서도 이차원 검색이 가능한 2D-BPT를 제안하였다. (출발지 프리픽스, 목적지 프리픽스) 쌍에 대한 룰의 다양성의 의존도가 매우 크다는 점에 착안하여, 그 동안 (출발지 프리픽스, 목적지 프리픽스) 쌍에 기반한 이차원 검색을 위한 많은 패킷 분류 구조들이 연구되어 왔음에도 불구하고, 일차원 검색의 순차적 연결내지는 메모리 비효율성의 한계를 극복하지 못한 이차원 검색에서 벗어나지 못하였다. 본 논문에서는 빈 노드없이 바이너리 트리를 구성하여 메모리를 효율적으로 사용하면서도 검색 시간에 있어서 좋은 성능을 보이는 일차원 BPT를 이차원으로 확장하여 패킷 분류에 적용하였다. (출발지 프리픽스, 목적지 프리픽스) 쌍으로부터 코드워드를 생성, 코드워드 BPT를 구성함으로써 하나의 BPT를 이용하여 (출발지 프리픽스, 목적지 프리픽스) 쌍을 위한 이차원 검색이 가능하도록 하였다. 본 논문에서 제안하는 구조인 2D-BPT는 기존의 검색 시간과 소요되는 메모리 크기의 절충 관계에서 벗어나 검색 시간에 대한 패널티 없이 메모리를 효율적으로 활용할 수 있는 실용적인 구조라 할 수 있다.

## 참 고 문 헌

[1] P.Gupta and N.McKeown, "Algorithms for Packet Classification," IEEE Network, vol.15, pp24-32, 2001.

[2] Florin Baboescu, Sumeet Singh, and George

Varghese, "Packet Classification for Core Router: Is there an alternative CAMs?," IEEE Infocom 2003, vol.1, pp. 5363, Apr., 2003.
[3] H.Jonathan Chao, "Next Generation Routers," Proceedings of the IEEE, Vol.90, No.9, pp. 1518-1558, Sep, 2002.
[4] P. Tsuchiya, "A search algorithm for table entries with non-contiguous wildcarding," Bellcore, internet document.
[5] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *in Proc. ACM SIGCOMM'98*, Aug. 1998, pp. 191-202.
[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*, Aug. 1999, pp. 25-41.
[7] N.Yazdani and P.S.Min, "Fast and Scalable Schemes for the IP Address Lookup Problem," Proc. IEEE HPSR2000, pp. 83-92, 2000.
[8] Hyesook Lim and Bomi Lee, "A New Pipelined Binary Search Architecture for IP Address Lookup," Proc. IEEE HPSR2004, pp. 86-90, Apr. 2004.

정 여 진
2003년 2월 이화여자대학교 정보통신학과 학사. 2005년 2월 이화여자대학교 정보통신학과 석사. 2005년 3월~현재 삼성전자 정보통신총괄. 관심분야는 Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

김 혜 란
2004년 2월 이화여자대학교 정보통신학과 학사. 2004년 3월~현재 이화여자대학교 정보통신학과 석사과정. 관심분야는 Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

임 혜 숙
정보과학회논문지 : 정보통신
제 32 권 제 3 호 참조