최장 길이 우선 검색에 기초한 프리픽스 길이에 따른 이진 IP 검색 구조

준회원 추 하 늘*, 정회원 임 혜 숙**

Longest First Binary Search on Prefix Length for IP Address Lookup

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

요 약

라우터는 입력된 패킷의 목적지 주소에 따라 IP 주소검색을 통해 패킷의 최종 목적지로 갈 수 있는 다음 홉으로 패킷을 전달하는 역할을 한다. 인터넷에 접속된 단일 호스트 네트워크 수의 증가로 인해 라우팅 테이블의 크기가 급격히 증가하고 있으며, 통신 링크의 속도 또한 기하급수적으로 빠르게 증가하고 있다. 라우터에 입력된 패킷은 선속도(wire-speed)로 처리되어야 하므로, 링크 속도의 증가는 라우터에서의 패킷 처리시간이 감소됨을 의미한다. 그러므로 차세대 라우터는 더 효율적이고 빠른 IP 주소검색 기술을 필요로 한다. 기존에 연구되어온 대부분의 검색 구조들에서는 짧은 길이의 프리픽스로부터 긴 길이의 프리픽스로 검색 영역을 확장하였다. 이 때문에 일치하는 가장 긴 프리픽스를 찾을 때까지 현재까지 일치된 가장 긴 프리픽스를 기억하면서 검색을 진행하였다. 본 논문에서는 긴 프리픽스를 먼저 검색하는 프리픽스 길이에 따른 이진 IP 주소 검색 구조를 제안한다. 제안하는 구조는 트라이의 리프에 존재하는 프리픽스들만으로 이루어진 독립적인 여러 개의 트라이를 구성하고, 길이에 따르는 이진 검색을 통해 긴 길이의 프리픽스와의 일치 여부를 먼저 확인함으로써 보다 빠른 검색속도를 제공한다. 또한, 이 구조는 기존의 프리픽스 길이에 따른 이진검색 구조가 선처리(pre-processing)가 많아 프리픽스의 부가적 추가가 힘들었던 것과는 다르게 선처리가 없이 프리픽스의 부가적 추가가 가능한 장점을 갖는다. 본 논문에서는 제안하는 구조의 성능을 실험한 후, 기존에 연구되어온 다른 IP 주소 검색 구조와 성능을 비교하였다.

Key Words : IP address lookup, Longest first search, Prefix grouping, Binary trie, Binary search on length

ABSTRACT

Based on the destination IP address of incoming packets, the Internet routers determine next hops and forward packets toward final destinations through IP address lookup. The bandwidth of communication links increases exponentially fast as well as the routing table size grows significant as the number of single host networks attached to the Internet increases. Since packets should be processed at wire-speed, the increased link speed reduces the processing time of a packet in routers, and hence more efficient and fast IP address lookup algorithms and architectures are required in the next generation routers. Most of the previous IP lookup schemes compare routing prefixes of shorter length first with a given input IP address. Since IP address lookup needs to

※ 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 (thousandsky@ewhain.net)

** 이화여자대학교 정보통신학과 SoC설계연구실 (hlim@ewha.ac.kr)

논문번호 : KICS2006-06-285, 접수일자 : 2006년 6월 27일, 최종논문접수일자 : 2006년 8월 11일

find the most specific route of the given input, search continues until the longest matched prefix is found while it keeps remembering the current best matching prefix. In this paper, based on binary search on prefix length, we proposed a new IP address lookup algorithm which compares longer prefixes first. The proposed scheme is consisted of multiple tries with prefixes on leaves only. The trie composed of the longest prefixes is primarily searched whether there is a match with the given input. This processing is repeated for the trie of the next longer prefixes until there finds a match. Hence the proposed algorithm provides the fast search speed. The proposed algorithm also provides the incremental update of prefixes while the previous binary search on length scheme does not provide the incremental update because of pre-processing requirement. In this paper, we performed extensive simulations and showed the performance comparisons with related works.

Ⅰ. 서 론

라우터는 패킷을 최종 목적지까지 보내기 위해 들어오는 모든 패킷에 대해 실시간으로 IP 주소검색을 수행하여야 한다. 현재 사용되고 있는 인터넷 주소 체계는 주소들을 클래스로 구분하지 않는 CIDR(classless inter-domain routing)로서, 이 체계에서의 프리픽스는 임의의 길이를 가질 수 있으므로 IP 주소 공간을 훨씬 효율적으로 사용할 수 있다. 가변 길이를 갖는 CIDR 주소 체계에서는 라우터에 들어오는 IP 주소와 가장 길게 일치하는 프리픽스(longest matched prefix, LMP)를 찾아야 하며, 이는 기존의 확정 일치(exact match) 알고리듬보다 훨씬 복잡한 최장 길이 프리픽스 일치(longest prefix matching) 알고리듬이 필요함을 의미한다. 게다가, 최근의 인터넷 기술의 급속한 성장은 링크 속도의 증가로 인하여, 비디오 스트리밍(video streaming)이나 온라인 게임 등의 실시간 응용 프로그램의 발전을 동반하였으며, 급격한 트래픽의 증가로 인해 라우터에서의 IP 주소 검색은 인터넷의 새로운 병목점이 되고 있다.

IP 주소검색과 관련해서 라우터의 성능을 평가하는데 있어 고려되는 주요한 요소들이 있다. 가장 중요하게 생각되는 것은 검색속도로서 검색속도는 메모리 접근 횟수에 가장 큰 영향을 받는데, 이는 메모리 접근 속도는 링크 속도가 증가하는 경향만큼 빠르게 향상되고 있지 않기 때문이다. 다음으로 고려되는 요소는 필요 메모리 사이즈이다. 무선 네트워크 기기를 이용한 호스트 라우트(host route)등의 증가로 인하여 라우팅 테이블을 위한 메모리 사이즈가 급격히 증가하고 있다. 따라서, 적은 메모리를 사용하여 라우팅 테이블을 구성하는 것이 고려되어야 한다. 또한 와이브로등의 새로운 네트워크 기술의 발전으로 일시적인 네트워크 접속 시도가 많아짐에 따라 프리픽스의 부가적인 추가의 복잡성도 고려되어야 하는 중요한 사항이다.

기존에 연구되어온 알고리듬들은 대부분 프리픽스의 길이가 짧은 것에서부터 긴 것으로 확장해나가면서 가장 긴 프리픽스를 검색하는 구조를 갖고 있다. 따라서, 하나의 프리픽스와 일치하였다 하더라도, 그 프리픽스보다 더 긴 프리픽스와 일치하는 BMP(best matching prefix)가 존재할 가능성이 있으므로 모든 프리픽스가 검색될 때까지 검색을 계속 진행하여야 한다. 본 논문에서는 이와는 반대로, 가장 긴 길이의 프리픽스를 먼저 검색하고 짧은 길이의 프리픽스로 검색을 확장시켜나가는 알고리즘을 제안한다. 또한 제안하는 구조는 기존의 길이에 따른 이진 검색 구조에서 선계산(pre-computation)이 요구되었던 단점을 디스조인트한 프리픽스들만으로 트라이를 만듦으로써 해결하였으며, 이로 인해 부가적인 업데이트가 용이한 장점을 갖는다.

본 논문의 구성은 다음과 같다. 먼저 Ⅱ장에서는 기존에 나와 있는 인터넷 주소 검색 구조들을 소개한다. Ⅲ장에서는 제안하는 구조에 대해 설명한다. Ⅳ장에서는 제안하는 구조의 성능 실험 결과를 보이고, 제안하는 구조와 기존의 다른 구조들과의 성능을 비교한다. 마지막으로, Ⅴ장에서는 간단한 결론을 맺는다.

Ⅱ. 기존의 구조

2.1 이진 트라이(Binary Trie[1])

이진 트라이는 트라이의 경로에 따라 해당하는 프리픽스의 라우팅 정보를 저장하는 방법으로써 검색 경로를 쫓아감을 통해서 프리픽스를 추론할 수 있으므로, 프리픽스 자체를 저장하지 않아도 되는 장점이 있다. 하지만, 프리픽스가 저장되는 노드까지 거쳐오는 모든 경로에 빈 내부 노드가 존재하게 되고, 이것은 이진 트라이를 위한 필요 메모리 사이즈를 크게 할 뿐만 아니라 검색속도를 느리게 하는 단점이 있다. 특정 프리픽스는 다른 프리픽스의 프리픽스가

표 1. 라우팅 데이터의 예

Prefix	Next hop	Prefix	Nest Hop
00000*	P0	1111*	P7
00*	P1	01110*	P8
001*	P2	11100*	P9
111*	P3	1*	P10
00100*	P4	0011*	P11
00101*	P5	110*	P12
011*	P6		

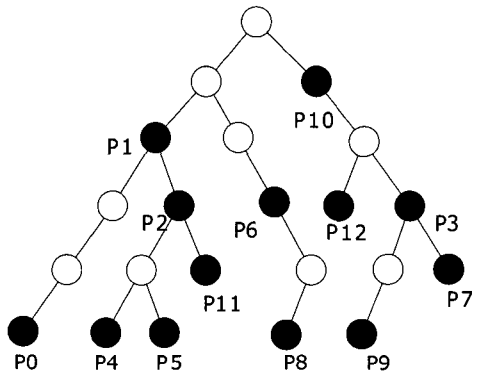

그림 1. 표 1의 라우팅 데이터를 이용해 구성된 이진 트라이의 예

될 수 있으며 이를 프리픽스 네스팅(prefix nesting) 관계라고 한다. 프리픽스의 네스팅 관계 때문에, 이진 트라이 구조에서는 상위레벨의 노드에 존재하는 프리픽스와 일치하였다고 하더라도, 현재 노드까지의 BMP를 기억하면서 트라이의 리프까지 입력 주소의 한 비트(bit)씩 살피면서 검색을 진행하여야 한다. 따라서 최악의 경우 메모리 접근 횟수가 IPv4의 경우 32, IPv6의 경우 128이 될 수 있다. 그림 1은 표 1에서 주어진 라우팅 데이터를 이용해서 구성된 이진 트라이의 예이다. 그림 1에서 검은 색으로 표시된 것은 해당하는 노드가 프리픽스임을 나타내며, 흰 노드는 비어있는 내부 노드에 해당한다.

2.2 멀티비트 트라이(Multi-bit Trie[2])

이 구조는 앞서 소개된 이진 트라이를 확장하는 알고리듬이라고 할 수 있다. 이진 트라이가 한 번에 한 비트만을 살피면서 검색을 진행하는 것과는 다르게, 한 번에 여러 비트를 보면서 검색을 수행한다. 한 번에 몇 개의 비트를 살필 것인지 스트라이드(stride)를 정하고, 이에 따라 트라이를 만든다. 이 때 스트라이드의 배수에 해당하는 길이를 갖지 않는 프리픽스에 대해서는 스트라이드의 배수에 해당하는 여러 개의 프리픽스들로 확장하는 작업이 필요하다. 예를 들어, 스트라이드가 2 비트 인데 프리픽스가 101*이라면, 프리픽스는 101*로 시작하는 두 개의 1010*과 1011*의 프리픽스로 확장되어야 하며, 이 두 프리픽스가 갖는 라우팅 정보는 101*의 정보를 상속받게 된다. 프리픽스가 여러 개로 확장되면서 라우팅 정보의 복사가 이루어지므로 필요한 메모리의 사용량이 커지게 된다. 하지만, 한 번에 여러 비트를 살피므로, 최대 프리픽스의 길이를 W라 할 때, 트라이의 깊이가 스트라이드 K에 따라서 W/K로 줄게 된다. 또한, 트라이에서의 검색 시간은 트라이의 깊이에 의존하므로 검색 시간을 줄일 수 있는 장점이 있다.

2.3 프리픽스 길이에 따른 이진 검색(Binary Search on Length[3])

이 구조는 프리픽스 길이에 대해서는 이진 검색을, 프리픽스 값에 대해서는 해싱을 사용하는 구조이다. 이 구조에서는 이진 트라이를 프리픽스 길이에 따른 각각의 길이 레벨로 분리하고, 각 길이 레벨에 존재하는 모든 노드들을 하나의 해쉬 테이블에 저장하는 라우팅 테이블을 만든다. 입력된 주소에 대해 각각의 길이 레벨에 따르는 해싱 값을 구해서 해당 해싱 테이블을 접근 했을 때, 프리픽스와 일치하는 경우에는 더 긴 프리픽스가 존재할 것을 가정하여 더 긴 길이 레벨로 검색을 진행하고, 일치하지 않는 경우에는 현재의 길이 레벨보다 긴 프리픽스가 존재하지 않을 것을 가정하여 짧은 길이 레벨로 검색을 진행함으로써, 프리픽스 길이에 따른 이진 검색을 수행하는 방법이다. 그러나 프리픽스의 네스팅 관계 때문에 해당 해쉬 테이블의 프리픽스와 일치하였다고 해도, 더 긴 길이 레벨의 프리픽스가 존재하지 않을 수도 있고, 해당 해쉬 테이블의 프리픽스와 일치하지 않았다고 해도, 더 긴 길이 레벨의 프리픽스와 일치할 수 있기 때문에, 길이에 따른 이진 검색을 위한 이 구조의 기본 가정에 모순이 존재한다.

이러한 점을 해결하기 위하여, 이 구조에서는 선계산(pre-computation)을 사용하였다. 비어있는 내부 노드에 마커를 미리 계산 저장하여, 현재의 해쉬 값이 프리픽스와 일치하지 않은 경우에도 더 긴 프리픽스가 존재함을 표시한다. 그러나 마커만으로 모든 종류의 프리픽스를 커버할 수 없다. 어느 길이 레벨에 존재하는 노드에 접근하였을 때, 마커가 있기 때문에 더 긴 길이 레벨에 대해서 이진 검색을 진행하였는데, 더 길게 일치하는 프리픽스가 없는 경우에는, 해당하는 마커가 존재하는 길이 레벨보다 더 짧은

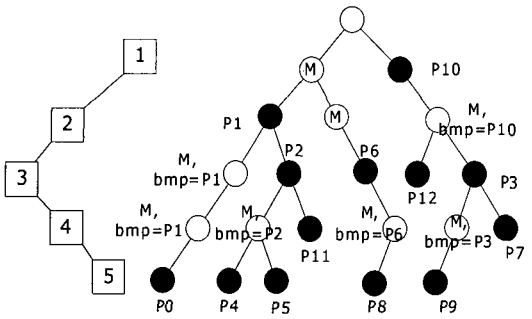

그림 2. 표 1의 라우팅 데이터를 이용해 구성된 프리픽스 길이에 따른 이진 검색의 예

길이 레벨에 대해서 다시 검색을 해야 하는 백-트랙킹(back-tracking) 문제가 발생한다. 이 경우 검색시간이 매우 길어지는 문제점이 있어 모든 마커마다 현재까지의 BMP를 미리 계산하여 저장해 두는 것을 통해 해결하였다. 마커보다 더 긴 길이 레벨을 검색했으나 일치하는 프리픽스를 찾지 못한 경우, 현재까지의 BMP를 저장하고 있으므로, 그것이 입력된 주소와 가장 길게 일치하는 프리픽스가 되는 것이다. 마커와 BMP는 모든 비어있는 내부 노드마다 저장될 필요는 없으며, 이진 검색을 수행할 때 먼저 접근되는 길이 레벨의 내부 노드에만 미리 계산해서 저장해두면 된다. 이 구조의 경우, 프리픽스 길이에 따른 이진 검색을 수행하므로 최대 프리픽스의 길이를 W라 할 때, 하나의 주소 검색을 위한 최대 메모리 접근 횟수가 $\log_2 W$가 되는 장점을 갖는다. 하지만, 내부 노드에 마커와 BMP를 선계산하여야 하고, 이런 선계산으로 인해 프리픽스의 부가적인 추가가 힘들다는 단점이 있다. 그림 2는 표 1에서 주어진 라우팅 데이터를 이용해 구성된 길이에 따른 이진 검색 트라이의 예를 보여준다. 그림 2에서 왼쪽에 있는 네모 안의 숫자들은 길이 레벨을 나타내는데, 레벨 3이 가장 먼저 접근되는 레벨이며 주어진 입력과의 일치여부에 따라 다음으로는 레벨 2와 레벨 4가, 마지막으로 레벨 1과 레벨 5가 접근됨을 보여준다. 그림 2에서 검은 색의 노드는 해당하는 노드가 프리픽스임을 나타내고, 흰 노드는 비어있는 내부 노드로써 각각의 내부 노드까지의 BMP와 마커를 선계산하여 저장하고 있다.

2.4 이진 트리(Binary Prefix Tree[4], BPT)

이 구조는, 이진 트라이에서 필연적으로 존재하는 비어있는 내부 노드를 없애고, 프리픽스에 대한 이진 검색을 수행하는 구조이다. 프리픽스에 대한 이진 검색이 어려웠던 것은 프리픽스가 임의의 길이를 가지기 때문에 서로 다른 길이를 갖는 두 프리픽스에 대한 크기를 정의할 수 없어 비교가 불가능하기 때문이다. 따라서, 이 구조에서는 길이가 다른 프리픽스들 간의 크고 작음을 비교할 수 있는 크기 비교 정의가 제안되었다. 두 개의 서로 다른 길이를 갖는 프리픽스에 대하여, 짧은 길이를 갖는 프리픽스 길이까지 값을 비교하여 크기가 더 큰 쪽이 큰 프리픽스로 정의된다. 짧은 길이까지의 값이 같다면, 긴 프리픽스의 다음 bit가 0이면 길이가 짧은 프리픽스가 큰 프리픽스로, 1이면 길이가 긴 프리픽스가 큰 프리픽스로 정의된다. 예를 들어, 01*과 100*을 비교해 보면, 짧은 쪽 프리픽스 길이인 처음 두 비트를 비교하여 100*이 01*보다 큰 프리픽스로 정의된다. 01*과 0111*을 비교해보면, 짧은 쪽 프리픽스 길이까지가 같으므로, 긴 프리픽스의 다음 비트를 보고, 그 값이 1이므로 0111*이 01*보다 큰 프리픽스로 정의된다.

그러나, 이러한 방법으로 크기 순서대로 정렬된 프리픽스 리스트에 대해서 이진 검색을 수행한다면, 들어온 주소와 가장 길게 일치하는 프리픽스 검색에 실패하게 된다. 예를 들어 프리픽스 A를 BMP로 갖는 주소가 들어왔을 때, A를 부-스트링(sub-string)으로 갖는 다른 프리픽스가 존재하고, 이 프리픽스가 A보다 먼저 비교되는 경우에는 A가 존재하지 않는 다른 쪽 리스트로 검색이 진행될 수 있기 때문이다. 이러한 문제를 해결하기 위하여 이진트리 구조에서는 프리픽스들을 디스조인트(disjoint), 인클로져(enclosure), 인클로즈드(enclosed) 프리픽스로 분류한다. 인클로져는 자신을 프리픽스로 하는 또 다른 프리픽스가 존재하는 프리픽스이고, 인클로즈드 프리픽스는 인클로져를 프리픽스로 갖는 프리픽스이며, 디스조인트 프리픽스는 인클로져도, 인클로즈드 프리픽스도 아닌 프리픽스이다. 디스조인트 프리픽스와 인클로져 프리픽스만으로 이루어진 리스트에서 먼저 이진 검색을 수행하는데, 이 때 인클로져 프리픽스가 입력된 주소와 비교될 때, 비교되는 인클로져를 부-스트링으로 갖는 인클로즈드 프리픽스가 리스트에 추가되는 방식으로 이진 프리픽스 트리가 구성된다.

이 구조는 트리의 구성에 있어서 비어있는 내부 노드가 전혀 존재하지 않고, 프리픽스들만으로 이루어진 이진 트리를 구성할 수 있는 장점이 있으나 인클로져 프리픽스는 인클로즈드 프리픽스 보다 항상 상위에 존재하여야 하므로 프리픽스 네스팅 관계에 따라서 프리픽스 트리의 깊이가 매우 깊어질 수 있다는 단점이 있다.

2.5 영역에 따르는 이진 검색(Binary Search on Range[5], BSR)

영역에 따르는 이진 검색 구조는 모든 프리픽스를 같은 길이의 시작점과 끝점을 갖는 하나의 영역으로 표현함으로써, 프리픽스의 길이 정보를 제거하고, 각 구간에 따르는 BMP를 미리 계산하여, 인터넷 주소의 이진 검색을 수행하는 구조이다. 이 구조의 평균 검색속도는 테이블 엔트리 개수의 로그 스케일에 비례하므로 매우 우수한 성능을 보이나, 라우팅 테이블 엔트리의 개수는 최악의 경우 프리픽스 개수의 두 배가 될 수 있어 메모리 사용량이 많아지고, BMP를 위한 선계산으로 인하여, 프리픽스의 부가적인 추가가 가능하지 않은 단점을 갖는다.

Ⅲ. 제안하는 구조

IP 주소 검색 문제는 결국 최장 길이 프리픽스 일치 문제로 요약되므로, 가장 긴 길이의 프리픽스들부터 검색한다면 빠른 검색 속도를 제공할 수 있을 것이다. 가장 긴 프리픽스란 이진 트라이의 리프 노드에 위치하는 프리픽스들로서 정의 될 수 있다. 제안하는 구조에서는 가장 긴 길이의 레벨부터 검색하기 위하여 이진 트라이의 리프 노드들만으로 이루어진 여러 개의 독립적인 트라이를 구성한다. 각 트라이에 대해서는 메모리 접근 횟수 성능이 가장 좋은 길이에 따른 이진 검색(BSL)을 적용하여 검색을 수행하는 구조를 제안한다. 이 때, 각각의 트라이는 모두 디스조인트한 프리픽스들로만 이루어지게 되므로, 마커와 BMP를 선계산할 필요가 없는 장점을 갖게 된다.

3.1 프리픽스 그룹핑

이진 트라이의 검색 경로 상에서 가장 긴 길이의 프리픽스들은 그림 1에서 보인 바와 같이 트라이의 리프에 존재한다. 이러한 점을 이용해 프리픽스를 길이가 긴 순서부터 리프 레벨을 정의하여 그룹핑할 수 있으며, 그 과정은 다음의 두 단계로 이루어진다.

먼저, 이진 트라이의 검색 경로 상에서 만나는 모든 리프노드들에 대하여 리프 레벨을 부여하고 그에 따른 그룹 리스트에 추가하는 것을 모든 리프 프리픽스에 대해 반복적으로 수행한다. 여기서, 리프 레벨이란 프리픽스에 해당하는 길이 레벨에 트라이의 가장 밑인 리프에서부터 트라이의 루트로 향하는 경로에서 만나는 프리픽스들에게 순차적인 번호를 부여함으로써 얻어지는 새로운 레벨을 의미한다. 예를

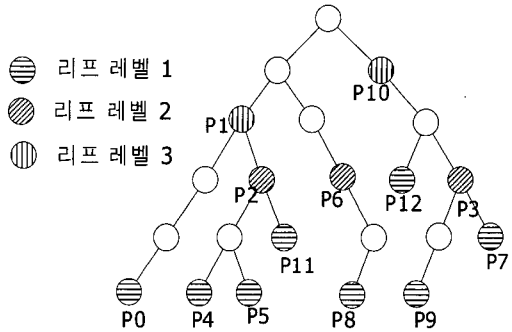

그림 3. 제안하는 프리픽스 그룹핑의 예

들어, 그림 1의 예에서 P0는 가장 첫 번째 리프에 해당하므로 리프 레벨을 1로 하고, 해당 프리픽스를 트라이에서 제거한 후 1번째 그룹 리스트에 포함시킨다. 이렇게 반복적으로 모든 리프에 대해서 그룹 리스트를 작성한다.

다음으로, 리프 노드를 떼어냄으로써 생겨난 빈 노드들을 제거한다. 그림 1에서 보면, P0를 떼어내고, 1번째 그룹에 포함시킨 후에는 000*과 0000*이 빈 노드로 존재하게 된다. 이러한 빈 노드를 모두 제거함으로써, 두 번째로 긴 프리픽스 노드들이 모두 리프에 존재하게 된다. 첫 단계를 반복함에 의하여 이 프리픽스들을 2번째 그룹 리스트에 포함시킨다. 이러한 일련의 두 과정을 리프 레벨을 1씩 증가시켜 가면서, 라우팅 데이터에서 주어진 프리픽스 개수와 그룹 리스트에 추가된 총 프리픽스 개수가 같아질 때까지 반복적으로 수행해나간다. 이 때, 특이할만한 점은 그림 3의 예에서, P1의 리프 레벨이 3이 된다는 점이다. P1은 P0까지의 경로상에서는 리프 레벨이 2가 되지만, P4, P5, P11의 경로에서는 P1의 리프 레벨이 3이 된다. 이는 리프 레벨이 1인 프리픽스와 빈 노드들을 제거한 후에도, P1이 새로운 리프 자리에 존재하는 P2의 인클로져로서 계속 남아있기 때문이다.

그림 3은 표 1의 라우팅 데이터를 이용하여 제안하는 리프 레벨에 따른 프리픽스 그룹핑을 수행한 후의 트라이를 나타낸다.

3.2 멀티플 트라이의 구성

리프 레벨별로 그룹 리스트를 완성한 후에는, 각 그룹에 속한 프리픽스들만으로 이진 트라이를 구성한다. 이 때 생성되는 트라이의 개수는 주어진 라우팅 데이터의 프리픽스 네스팅 관계가 몇 개의 리프 레벨을 갖느냐에 따라 달라진다. 그림 3의 예로 멀티플 트라이를 구성한 것을 그림 4에 보였다. 그림 3

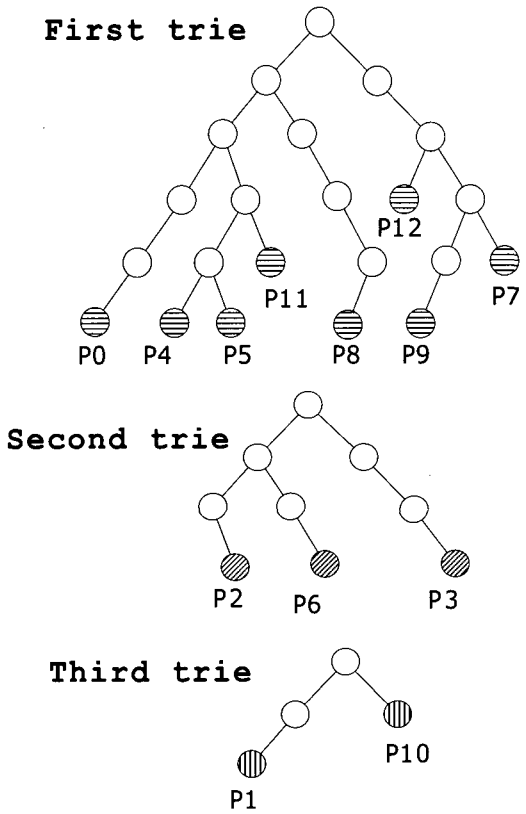

그림 4. 제안하는 구조의 멀티플 트라이

에서 보이는 바와 같이, 최대 리프 레벨이 3이므로 3개의 트라이가 구성된 것을 볼 수 있다. 특징적인 것은, 각 그룹에 속한 프리픽스들이 모두 리프에 위치하였던 프리픽스들이므로, 이들을 이용해 만든 트라이들 역시 리프가 프리픽스로 가득 차 있으며, 이것은 모두 디스조인트한 프리픽스로 이루어졌다는 것을 의미한다.

3.3 라우팅 테이블 빌드

완전 해쉬 함수를 가정했을 때 본 논문에서 제안하는 구조의 라우팅 테이블을 만드는 방법은 다음과 같다. 먼저 이진 트라이를 만든 후, 리프 레벨로 프리픽스를 그룹핑한 후에 해당하는 리프 레벨별로 독립적인 이진 트라이를 구성한다. 이 때 생성되는 트라이의 개수는 최대 리프 레벨 개수와 같다. 각각의 트라이에 존재하는 모든 노드의 값을 저장하는 것이 아니라, 각 트라이에서 프리픽스가 존재하는 길이 레벨에 속한 내부 노드와 프리픽스 노드들의 해쉬 값을 구하고, 그 값을 메모리의 주소로 하는 엔트리에 해당 노드 값 및 노드의 종류(내부 노드 혹은 프리픽스 노드)등의 노드 정보를 저장한다.

3.4 각 트라이에 길이에 따른 이진 검색 적용

Waldvogel[3]의 프리픽스 길이에 따른 이진 검색 구조는 프리픽스의 길이에 대해서 이진 검색을 수행하기 때문에 검색에 필요한 메모리 접근 횟수에 있어 매우 우수한 성능을 보인다. 하지만 임의의 길이를 갖는 프리픽스들 사이에는 한 프리픽스가 다른 프리픽스의 프리픽스(prefix of prefixes)가 되는 프리픽스 네스팅 관계가 존재하게 되고, 이 때문에 길이에 따른 이진 검색을 위해 마커와 BMP를 미리 계산해주어야 하는 어려움이 있었다. 마커와 BMP가 필요한 이유를 정리해 보면, 마커는 특정 길이에 해당하는 레벨에서 일치하는 프리픽스가 없더라도, 그보다 긴 길이의 레벨에 일치하는 프리픽스가 존재할 수 있기 때문에 존재한다. 그러나 마커가 존재하더라도 그보다 더 긴 길이의 레벨에 프리픽스가 존재하지 않을 수 있으므로 검색이 짧은 길이의 레벨로 거슬러 올라가야 하는 문제점을 해결하기 위해 BMP를 미리 계산할 필요가 있다.

만약 마커보다 더 긴 길이의 레벨에 반드시 프리픽스가 존재함을 보장한다면, 마커와 BMP를 선계산할 필요가 없게 된다. 어느 특정 길이에 해당하는 레벨의 검색 시 프리픽스를 만난 경우에는 같은 트라이상에서 이보다 더 긴 프리픽스가 없음을 보장하고, 또한 어느 특정 길이에 해당하는 레벨의 검색 시 내부 노드를 만난 경우에는 같은 트라이상에서 이보다 더 짧은 프리픽스는 없다는 것을 보장한다면 마커와 BMP의 선계산이 전혀 필요 없게 된다. 이것이 제안하는 구조에서 디스조인트한 프리픽스들로만 이루어진 여러 개의 트라이를 구성하는 이유이다.

제안하는 구조에서 검색은 멀티플 트라이에 대해 순차적으로 진행된다. 리프 레벨이 작을수록 긴 프리픽스이므로, 리프 레벨이 작은 트라이에서 큰 트라이로 검색을 진행하다가, 입력된 주소와 일치하는 프리픽스를 찾으면 검색을 종료한다. 각 트라이에서의 길이에 따른 이진 검색은 프리픽스가 존재하는 길이 레벨의 노드들만을 해쉬 테이블에 저장하므로 프리픽스가 존재하는 길이 레벨에 대해서만 이루어지며, 이진 검색은 다음의 세 경우로 나누어 생각할 수 있다. 먼저, 해쉬에 의해서 접근된 엔트리가 프리픽스라면, 모든 프리픽스는 리프에 존재하고 이것이 가장 긴 프리픽스에 해당하므로 검색을 종료한다. 다음으로는 접근된 엔트리가 내부 노드인 경우이다. 이 경우에는 현재 노드보다 더 긴 길이의 프리픽스가 반드시 하나 이상 존재하므로 프리픽스가 존재하는 더 긴 길이에 해당하는 길이 레벨로 검색을 진행한다.

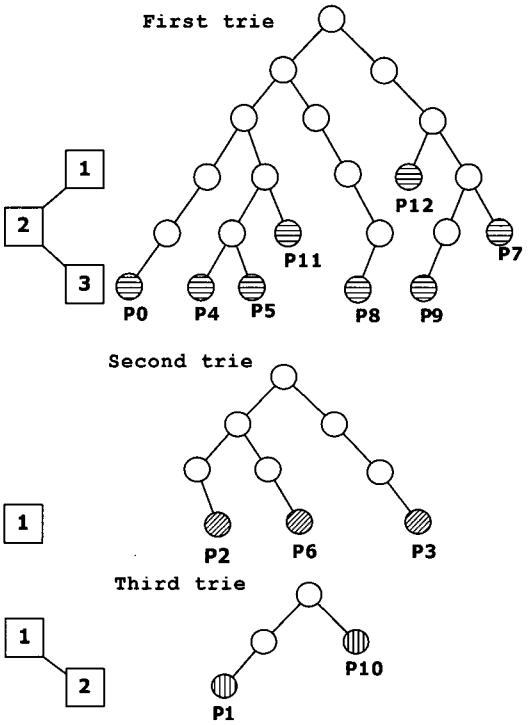

그림 5. 제안하는 구조에서의 이진 검색의 예

마지막으로, 접근된 엔트리에 노드가 존재하지 않는 경우로서, 그보다 긴 길이 레벨에도 역시 노드가 존재하지 않을 것이므로 프리픽스가 존재하는 짧은 길이의 길이 레벨로 검색을 진행한다. 이렇게 디스조인트한 프리픽스들로만 이루어진 트라이를 통해 길이에 따른 이진 검색이 가능하게 된다.

그림 5는 그림 4의 멀티플 트라이를 이용한 검색의 예를 보여준다. 첫 번째 트라이에 프리픽스가 존재하는 길이 레벨은 프리픽스 길이 3, 4, 5에 해당하는 레벨이므로 이들에 대해서만 이진 검색이 이루어진다. 예를 들어, 입력된 주소가 011101인 경우 먼저 첫 번째 트라이에서 레벨 2인 0111*을 검색한다. 노드 0111*은 내부 노드이므로, 더 긴 레벨로 검색을 진행하고, 레벨 3의 01110*이 검색된다. 01110*은 프리픽스이고, 긴 프리픽스를 먼저 검색하였으므로 일치한 프리픽스인 P8은 가장 긴 프리픽스가 된다. 이를 출력하고 검색을 종료한다. 또 다른 예로, 입력된 어드레스가 100000인 경우, 첫 번째 트라이부터 검색을 시작하는데, 레벨 2의 엔트리 1000에 노드가 존재하지 않으므로, 더 짧은 길이인 레벨 1로 진행하고, 레벨 1의 엔트리 100에도 노드가 존재하지 않으므로, 더 짧은 길이로 검색이 진행하여야 하나, 첫 번째 트라이에는 더 짧은 길이가 존재하지 않으므로, 다음 트라이로 검색을 진행한다. 두 번째 트라이에서는 프리픽스가 존재하는 레벨이 길이가 3인 레벨 1뿐이므로, 이 레벨에 대해서만 검색을 한다. 이때, 레벨 1의 엔트리에 해당하는 노드가 여전히 존재하지 않으므로, 세 번째 트라이로 이동하고, 이 트라이의 레벨 1에 1* 노드를 검색한다. 노드 1*은 프리픽스이므로, 일치한 프리픽스 P10을 출력하고 검색을 종료한다.

3.5 프리픽스 삭제 및 추가

구성된 라우팅 테이블에서 프리픽스를 제거하기를 원하는 경우, 해당하는 프리픽스가 존재하는지 각각의 트라이에서 검색을 한 후에 삭제한다.

새로운 프리픽스를 추가하는 경우, 먼저 추가하려는 프리픽스를 가장 긴 프리픽스들로 이루어진 첫 번째 이진 트라이에서 검색한다. 검색결과 다음의 세 가지 경우에 대해 생각해 볼 수 있다.

먼저, 추가하려는 프리픽스에 해당하는 노드가 존재하지 않는 경우이다. 이 경우에는, 모든 프리픽스와 디스조인트한 경우이므로 첫 번째 트라이의 리프에 새로운 노드로 만들어 주면 된다.

다음으로는 추가하려는 프리픽스를 검색하는 경로상에서 기존의 프리픽스를 만나는 경우이다. 이 경우 추가하려는 프리픽스는 리프 자리에 새로운 노드로 만들어주고, 경로 상에서 만났던 프리픽스를 현재의 리프 레벨 트라이에서 제거한 후 다음 트라이에 추가한다.

마지막으로 추가하려는 프리픽스가 기존 트라이의 내부 노드에 해당하는 경우이다. 이 경우는 추가하려는 프리픽스가 가장 긴 프리픽스보다 짧으므로, 새로운 노드를 추가할 수 있는 트라이에 갈 때까지 다음 트라이로 프리픽스 추가를 위한 검색을 진행한다. 이때에, 프리픽스는 리프에 추가되고 경로 상에 프리픽스가 존재한다면 그 프리픽스를 현재 트라이에서 제거하고 다음 트라이에 추가한다.

상위 트라이에서 제거된 프리픽스를 다음 트라이에 추가하고자 할 때, 트라이가 존재하지 않는 경우에는 새로운 트라이를 만들고, 해당하는 프리픽스 노드를 리프 위치에 생성한다.

그림 6-1의 예를 통해 설명하면, 예를 들어 P13인 010*을 추가하려는 경우, 첫 번째 트라이에 일치하는 프리픽스가 없으므로, 그 자신에 해당하는 프리픽스 노드를 새로이 추가한다. 그림 6-2는 P14인 0000*를 추가하려는 경우를 설명한다. 첫 번째 트라이의 내부 노드에 해당하므로 두 번째 트라이에서

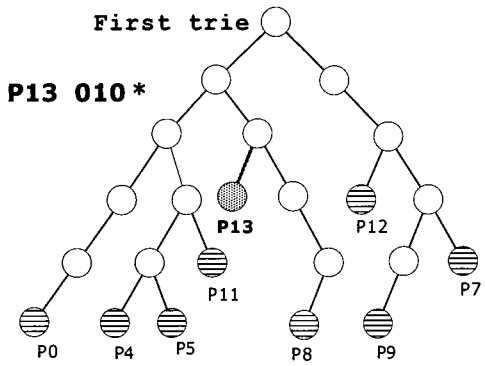

그림 6-1. 프리픽스 010*의 추가의 예

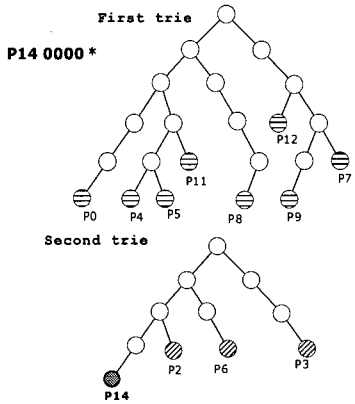

그림 6-2. 프리픽스 0000*의 추가의 예

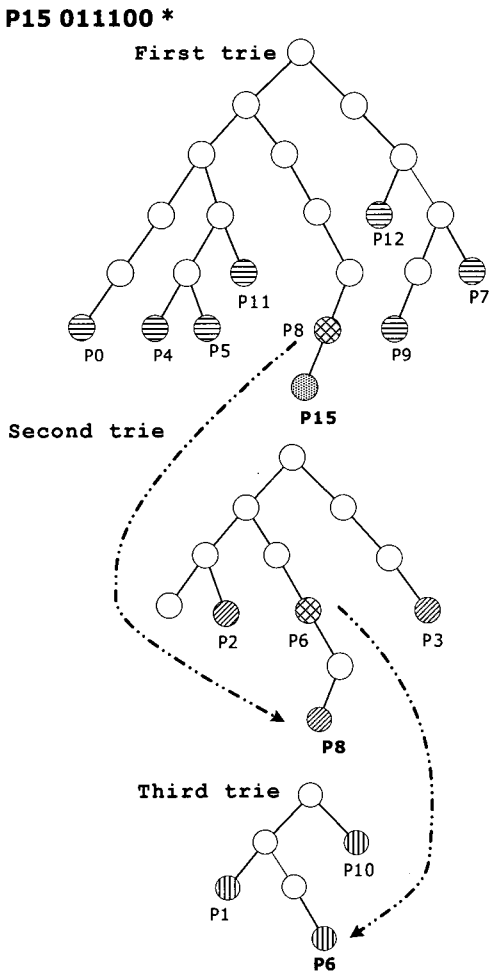

그림 6-3. 프리픽스 011100*의 추가의 예

검색을 진행하고, 이 트라이에서는 노드가 없으므로 해당하는 자리에 새로운 노드로 추가한다. 마지막으로, P15인 011100*을 추가하는 경우를 그림 6-3에 보였다. 프리픽스 P8과 일치하고, 프리픽스 P15가 프리픽스 P8보다 길므로, 리프인 011100*에 새로운 프리픽스 노드 P15를 추가하고, P8을 해당 트라이에서 제거한 후에 다음 트라이에 추가한다. 두 번째 트라이에서 P8을 추가하고자 할 때, 또 한 번 프리픽스 P6(011*)을 만나므로 P8을 해당하는 리프로 추가한 후에 P6을 두 번째 트라이에서 제거하고, 세 번째 트라이로 추가하게 한다. 세 번째 트라이에는 P6에 해당하는 노드가 존재하지 않으므로, 새로운 노드를 만들어 준다. 이와 같이 제안하는 구조에서의 프리픽스 추가는 프리픽스의 네스팅 관계에 따라 최대 트라이의 개수만큼의 프리픽스들만이 영향을 받으므로, 부가적인 추가가 가능하다.

Ⅳ. 제안하는 구조의 성능 평가

본 논문에서는 리프들만으로 이루어진 멀티플 트라이를 구성하고, 각 트라이에 대해서는 길이에 따른 이진 검색을 통해 최장 길이의 프리픽스를 먼저 검색할 수 있게 한 구조를 제안하였다. 본 논문에서 제안된 구조의 성능을 평가하기 위해 C언어를 사용하여 실제 사용된 백본 라우터(backbone routers)의 라우팅 데이터[6]에 대하여 실험하였다. 이 데이터들은 다양한 프리픽스 개수를 갖는 MAE-West, Aads, PORT80, Grouptlcom, Telstra 백본 라우터 라우팅 데이터[6]이다. 또한, 완전 해쉬 함수에 의해 각 노드에 해당하는 정보를 해쉬 테이블에 저장함을 가정하였으므로, 완전 해쉬 함수를 가정하여 성능 평가를 진행한 결과 제안하는 구조가 제대로 동작함을 확인할 수 있었다. 표 2는 위에서 언급한 라우팅 데이터 (N)[6]들에 대하여 그룹 리스트 혹은 트리의 개수 (N_{group}), 입력된 인터넷 주소를 검색하는 데 소요되는

표 2. 여러 라우팅 데이터에 대한 성능 실험 결과

	N	N_{group}	T_{avg}	T_{min}, T_{max}	M(KB)
MAE-West1	14,553	3	4.53	1,10	460
Aads	20,204	3	3.76	1,9	589
MAE-West2	29,584	5	4.11	1,18	732
PORT80	112,310	25	5.51	1,62	2.22M
Grouptlcom	170,601	18	4.88	1,52	3.14M
Telstra	227,223	18	5.04	1,54	3.99M

평균 메모리 접근 횟수(T_{avg}), 검색하는데 소요되는 최소 및 최대 메모리 접근 회수(T_{min}, T_{max}) 와 라우팅 테이블을 저장하는데 소요되는 메모리의 크기(M)를 보여주고 있다.

표 2의 결과를 살펴보면, 기존에 알려진 프리픽스 네스팅 관계는 최대 7레벨을 갖는다[7]는 것과는 달리 프리픽스의 네스팅 관계가 최대 25 레벨까지도 존재함을 알 수 있었다. 표 2에서 보는 바와 같이, 제안하는 구조에서는 입력된 어드레스와 가장 길게 일치하는 프리픽스를 찾는데 평균 3.8에서 5.6번의 메모리 접근이 필요한 것을 알 수 있다. 평균 메모리 접근 횟수는 라우팅 테이블의 사이즈가 매우 큰 PORT80, Grouptlcom, Telstra의 경우에도 그다지 증가하지 않는 것을 볼 수 있어 본 논문에서 제안하는 구조가 확장성이 매우 우수함을 알 수 있다. 그러나 최대 메모리 접근 횟수는 라우팅 테이블이 커짐에 따라 매우 커지는 것을 볼 수 있는데, 이는 상대적으로 짧은 길이를 프리픽스로 갖는 패킷에 대해서는 긴 프리픽스들로 이루어진 첫 번째 트라이부터 짧은 길이의 프리픽스들로 이루어진 트라이들로 검색이 진행되어, 프리픽스 네스팅 관계에 따라 접근해야 하는 트라이의 개수가 많아지기 때문이다. 즉, 최대 메모리 접근 횟수가 주어진 라우팅 데이터의 프리픽스 네스팅 관계의 정도에 비례하여 나타나는 것을 확인할 수 있다.

라우팅 데이터의 프리픽스 네스팅 관계 정도에 비례하여 리프 레벨의 수가 증가하게 되고, 이에 따라 구성되는 멀티플 트라이의 개수가 많아지게 되므로 표 2에서 보이듯이, 프리픽스 네스팅 관계가 적은 MAE-West1이나 MAE-West2, Aads에 대해서는 필요 메모리 사이즈가 작은 편이지만, 프리픽스 네스팅 관계가 많아지면 많아질수록, 트라이를 많이 구성하게 되므로 필요 메모리 사이즈가 커지는 것을 알 수 있다.

다음은 기존의 구조와 제안하는 구조의 성능을 비교하는 실험을 수행하였다. 표 3에 약 38000여개의

표 3. 성능비교(38k 엔트리 라우팅 데이터)

	T_{avg}	M(KB)	*Incremental updates*
Binary Trie[1]	23.2	864.2	Yes
Multi-bit Trie[2]	12.2	948.7	Yes
BPT[4]	15.8	351.4	No
WBPT[8]	15.3	351.4	No
DPT[9]	14.9	324.4	No
Binary search on Range[5]	15.9	404.0	No
Waldvogel[3]	2.2	0.98MB	No
Leaf-pushed BSL[10]	2.1	946.4	No
Proposed	4.57	1.04MB	Yes

엔트리를 갖는 MAE-East1 라우팅 데이터에 대하여 평균 메모리 접근 횟수와 소요되는 메모리의 크기를 비교하였다. 표 3에서 보여주는 바와 같이, 제안하는 구조는 기존의 이진 검색의 구조에 비해 2~3배의 메모리를 요구하나, 평균 메모리 접근 횟수(T_{avg})에 있어서 월등히 좋은 결과를 보여줌을 알 수 있다. 제안하는 구조는 독립적인 트라이를 여러 개 구성하게 되므로, 상대적으로 짧은 길이의 프리픽스와 일치하는 입력에 대해서는 여러 개의 트라이에 대해 검색을 진행하게 된다. 이로 인해 메모리 접근 횟수가 늘어나게 되므로, 표 3에서 보이는 바와 같이 제안하는 구조의 평균 메모리 접근 횟수가 하나의 트라이에 대해서 길이에 따른 이진 검색을 수행하는 Waldvogel[3]의 구조보다 많은 것을 알 수 있다. 하지만, 무선 네트워크나 인터넷에서 사용되는 어플리케이션들의 증가로 프리픽스 길이가 32인 호스트 라우트(host route)들이 급격히 증가하고 있고 라우팅 데이터에서 프리픽스 길이가 긴 데이터들이 많아지는 추세이므로 짧은 길이의 프리픽스에 의한 성능 저하는 큰 문제가 되지 않을 것으로 생각된다. 또한 프리픽스의 네스팅 관계에 따라 멀티플 트라이를 구성하게 되므로 메모리 요구량에 있어 Waldvogel[3]의 구조보다 조금 큰 것을 볼 수 있다. 하지만 링크 테크놀로지의 발전으로 인해 라우터에 입력되는 많은 양의 패킷을 입력되는 선속도(wire-speed), 즉 실시간으로 처리하는 것이 보다 중요한 인터넷 주소 검색의 이슈가 되고 있고, 많은 무선 네트워크 테크놀로지의 발전으로 인해 네트워크에 일시적으로 참여하는 엔드 시스템(end system)의 수가 증가되고 있어, 라우팅 테이블의 빠른 업데이트가 중요한 이슈로 떠오르고 있다. 기존의 Waldvogel[3]의 구조는 마커와 BMP의 선계산 때문에 프리픽스의 부가적 추가가 불가능하나 본 논문에서 제안하는 구조에서는 프

리픽스의 부가적인 추가가 가능한 장점이 있다.

본 논문에서 제안하는 구조가 서론에서 언급한 세 가지의 중요 성능 평가의 척도를 모든 면에서 만족시키지는 못하나, 실시간 패킷 포워딩을 위한 평균 검색속도에 있어 매우 우수한 구조이며, 라우팅 테이블의 부가적 추가 기능을 제공하는 구조로서 제안된 기법이 종합적인 면에서 기존의 여러 이진 검색 알고리듬에 비해 우수한 구조라고 할 수 있다.

V. 결 론

본 논문에서는 라우팅 테이블에 들어가는 프리픽스를 네스팅 관계에 따라 그룹핑하여 여러 개의 독립적인 트라이를 구성하고, 각 트라이에 대해서는 긴 길이의 프리픽스 트라이로부터 시작하여 길이에 따른 이진 검색을 수행하는 구조를 제안하였다.

완전 해싱 함수를 가정한 경우 본 논문에서 제안하는 구조는 40K 엔트리 이하의 라우팅 데이터에 대해서는 평균 4.6번의 메모리 접근으로 주소 검색이 가능함을 보였다. 기존의 길이에 따른 이진 검색이 선계산이 필요하고, 이에 따른 프리픽스의 부가적인 추가가 전혀 불가능 하였던 것에 반하여, 본 논문에서 제안하는 구조는 프리픽스의 부가적 추가가 가능한 구조이다. 또한 라우팅 데이터의 크기가 커져도 성능에 영향을 받지 않는, 확장이 매우 용이한 구조이다.

참 고 문 헌

[1] M.A. Ruiz-Sanchex, E.W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, pp.8-23, March/April 2001.

[2] H. Jonathan Chao, "Next Generation Routers," *Proceeding of the IEEE*, vol.90, no.9, September 2002.

[3] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM Conf.*, Cannes, France, pp.25-35, 1997.

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

[5] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search," *IEEE/ACM Transaction on Networking*, Vol.7, No.3, pp.324-334, Jun. 1999.

[6] http://www.potaroo.net

[7] G. Varghese, "Network Algorithmics," *Morgan Kaufmann*, 2005.

[8] Changhoon Yim, Bomi Lee, and Hyesook Lim, "Efficient Binary Search for IP Address Lookup," *IEEE Communications Letters*, Vol.9, No.7, pp.652-654, Jul. 2005.

[9] Hyesook Lim, Wonjung Kim, and Bomi Lee, "Binary Search in a Balanced Tree for IP Address Lookup," *Proc. IEEE HPSR2005*, May 2005.

[10] Ju Hyoung Mun, Hyesook Lim, and Changhoon Yim, "Binary Search on Prefix Lengths for IP Address Lookup," *IEEE Communications Letters*, Vol.10, No.6, pp.492-494, June. 2006.

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