IP 주소 검색을 위한 트리 레벨을 사용한 이진 검색 구조

준회원 문 주 형*, 정회원 임 혜 숙*

Binary Search on Tree Levels for IP Address Lookup

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

요 약

인터넷 트래픽의 급속한 성장으로 인하여, 인터넷 라우터에서는 보다 빠른 주소 검색을 지원하면서도 매우 큰 라우팅 데이터에 대하여도 잘 동작하는 인터넷 주소 검색 구조를 요구하고 있다. 본 논문에서는 이진 검색에 기초한 인터넷 주소 검색 구조를 심도있게 연구하였다. 기존에 연구되어온 대부분의 이진 검색 구조들은 프리픽스의 값에 따르는 이진 검색을 수행하는 구조로서 프리픽스 개수의 로그 함수에 비례하는 검색 속도를 보인다. 한편 프리픽스 길이에 따르는 이진 검색 구조가 연구되어 검색 성능에 있어서는 매우 우수한 성질을 보이나, 순수한 의미의 이진 검색이 불가능하여, 이진 검색시 접근되는 노드에 특수 목적의 마커를 저장하고, 또한 가장 잘 일치하는 프리픽스를 미리 계산하여 저장하는 방식을 통하여 해결하였다. 이러한 복잡한 선계산은 라우팅 테이블의 구성을 매우 어렵게 할 뿐 아니라, 프리픽스의 부가적 추가를 불가능하게 만드는 단점이 있다. 본 논문에서는 이러한 복잡한 선계산 없이 리프-푸싱만을 통하여 프리픽스 길이에 대하여 이진 검색을 수행하는 매우 효율적인 구조를 제안하고, 제안하는 구조의 성능을 실험한 후, 기존에 연구되어온 다른 이진 검색 구조와 성능을 비교하였다.

Key Words : Address lookup, Internet Protocol, Binary Trie, Binary Search on Length

ABSTRACT

Address lookup is an essential function in the Internet routers, and it determines overall router performance. In this paper, we have thoroughly investigated the binary-search-based address lookup algorithms and proposed a new algorithm based on binary search on prefix lengths. Most of the existing binary search schemes perform binary search on prefix values, and hence the lookup speed is proportional to the length of prefixes or the log function of the number of prefixes. The previous algorithm based on binary search on prefix lengths has superior lookup performance than others. However, the algorithm requires very complicated pre-computation of markers and best matching prefixes in internal nodes since naïve binary search is not possible in their scheme. This complicated pre-computation makes the composition of the routing table and incremental update very difficult. By using leaf-pushing, the proposed algorithm in this paper removes the complicated pre-computation of the previous work in performing the binary search on prefix lengths. The performance evaluation results show that the proposed scheme has very good performance in lookup speed compared with previous works.

※ This research was partially supported by the MIC(Ministry of Information and Communication), Korea, under the Chung-Ang University 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 : KICS2005-09-365, 접수일자 : 2005년 9월 6일

Ⅰ. 서 론

최근 급속한 성장을 이룬 인터넷 기술은, Ethernet 기술을 비롯하여, video streaming이나 interactive game과 같은 응용 프로그램의 발전을 동반하였으며, 인터넷 트래픽의 급속한 증가로 인하여, 인터넷 주소 검색은 효율적인 데이터 교환에 장애가 되는 새로운 병목점으로 등장하고 있다. 현재 사용되고 있는 인터넷 주소 체계는 클래스를 갖지 않는 주소 체계(classless inter-domain routing, CIDR)로서, CIDR 체계에서의 인터넷 주소 검색은 라우터로 입력된 인터넷 주소와 가장 길게 일치하는 프리픽스(longest matched prefix, LMP)의 라우팅 정보를 찾아야 한다. 이는 기존의 확정 일치(exact match) 알고리듬 보다 훨씬 복잡한 알고리듬이 수행되어야 함을 의미하며, 더욱이 라우터가 저장하는 라우팅 테이블의 크기가 급속히 증가하는 추세에 있어, 큰 라우팅 테이블에서도 좋은 성능을 나타내는 검색 구조가 요구된다고 할 것이다[1].

본 논문에서는 인터넷 주소 검색을 위하여 기존에 연구되어온 이진 검색 구조들을 심도있게 연구한 후, 프리픽스 길이에 대해 이진 검색을 적용하는 구조를 제안한다. 기존에 연구되어진 프리픽스 길이에 대해 이진 검색을 수행하는 구조의 경우 비어있는 내부 노드에 마커 및 현재까지 가장 잘 일치하는 프리픽스(best matching prefix, BMP)를 미리 계산하여 저장하여야 하는 제약점이 있었으나, 본 논문에서는 리프-푸싱 기술을 이용하여 마커 및 BMP를 미리 계산하지 않고도 프리픽스 길이에 대하여 이진 검색을 수행하는 구조를 제안한다. 또한 기존 구조의 경우는 프리픽스 값에 대하여 완전 해싱 (perfect hashing)를 구할 수 있음을 가정하여 실용적이지 못한 단점이 있었으나, 본 논문에서는 복수 해싱 (multiple hashing) 을 사용하여 실제 구현 가능한 구조를 제안한다. 제안하는 구조는 최악의 경우 6번의 메모리 접근을 통하여 인터넷 주소 검색을 수행하는 구조이며 메모리 접근 횟수에 있어서 큰 라우팅 테이블에 대해서도 작은 라우팅 테이블과 비슷한 성능을 나타내는 장점을 갖는다.

본 논문의 구조는 다음과 같다. 먼저 2장에서는 기존에 나와 있는 이진 검색에 기초한 인터넷 주소 검색 구조들을 소개한다. 3장에서는 제안하는 구조에 대해 설명한다. 4장에서는 제안하는 구조의 성능 실험 결과를 보이고, 제안하는 구조와 기존의 다른 이진 검색 구조들과의 성능을 비교한다. 5장에서 간단한 결론을 맺는다.

Ⅱ. 기존의 구조

2.1 이진 트라이(Binary Trie[1])

이진 트라이는 트라이의 경로에 따라 해당하는 노드에 프리픽스의 라우팅 정보를 저장하는 방법으로, 거쳐온 경로로부터 프리픽스를 추론할 수 있으므로, 따로 프리픽스를 저장해 줄 필요가 없다는 장점이 있다. 그러나 프리픽스가 거치는 모든 경로마다 프리픽스가 저장되지 않은 내부 노드를 두어야 하고, 이러한 내부 노드때문에 요구되는 메모리의 크기가 커진다는 단점이 있고, 최악의 경우 트라이의 깊이가 IPv4의 최대 프리픽스 길이인 32까지 커질 수 있어, 32번의 메모리 접근를 필요로 한다.

2.2 이진 트리(Binary prefix tree, BPT[2])

이 구조는 이진 트라이가 갖는 비어있는 내부 노드를 없앤 후, 프리픽스 값에 대하여 이진 검색을 수행하기 위한 방식이다. 이진 검색을 수행하기 위해서는 프리픽스들이 값의 크기 순서대로 정렬되어져야 하는데, 프리픽스의 길이가 같은 경우에는 직접적인 크기 비교가 가능하지만, 길이가 다른 경우에는 직접적인 크기비교가 불가능하다. 따라서 이 구조에서는 길이가 다른 프리픽스들 간의 크기 비교를 정의하였다. 두 개의 서로 다른 길이를 갖는 프리픽스가 있을 때, 짧은 쪽 프리픽스 길이까지의 값을 비교하여, 크기가 더 큰 쪽이 큰 프리픽스로 정의된다. 짧은 쪽 프리픽스 길이까지의 값이 같다면, 긴 프리픽스의 다음 비트가 **0**이면 길이가 짧은 프리픽스가 큰 프리픽스로 정의되고, **1**이면 길이가 긴 프리픽스가 큰 프리픽스로 정의된다. 예를 들어, **10***과 **0010***을 비교해 보면, 짧은 쪽 프리픽스 길이인 처음 두 비트를 비교하여 **10***이 **0010***보다 큰 프리픽스로 정의된다. **10***과 **1011***을 비교해보면, 짧은 쪽 프리픽스 길이까지가 같으므로, 긴 프리픽스의 다음 비트를 보고, 그 값이 **1**이므로, **1011***이 큰 프리픽스로 정의된다.

이러한 방법으로 정렬된 프리픽스 리스트에 대해 이진 검색을 수행한다면, 입력된 어드레스와 가장 길게 매치하는 프리픽스 검색에 실패하게 된다. 예를 들어 프리픽스 A를 BMP로 갖는 입력 어드레스가 들어왔음을 가정할 때, A를 부-스트링으로(sub-string) 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 이진 트리 구조에서는 프리픽스들을 디스조인트(disjoint), 인클로져(enclosure), 인클로즈드(enclosed) 프리픽스로 분류한다. 인크로져는 자신을 부-스트링으로 하는 또 다른 프리픽스가 존재하는 프리픽스이고, 인클로즈드 프리픽스는 인클로져를 부-스트링으로 갖는 프리픽스이고, 디스조인트 프리픽스는 인클로져도 인클로즈드 프리픽스도 아닌 프리픽스이다. 디스조인트 프리픽스와 인클로져 프리픽스만으로 이루어진 리스트에서 먼저 이진 검색을 수행하는데, 인클로져 프리픽스가 입력과 비교되어지기 위하여 선택되어진 경우, 선택된 인클로져를 부-스트링으로 갖는 인클로즈드 프리픽스가 리스트에 추가되는 방식으로 이진 검색 트리가 구성된다. 이 방법은 비어있는 내부 노드를 갖지 않고 프리픽스들만으로 이루어진 이진 검색 트리를 구성한다는 장점이 있으나, 인클로져는 항상 인클로즈드 프리픽스보다 상위에 위치하여야 한다는 제약점을 갖기 때문에, 트리의 깊이가 매우 깊어진다는 단점을 갖는다.

이러한 단점을 개선하기 위하여 인클로져가 갖는 인클로즈드 프리픽스들의 개수까지를 미리 고려하여 트리를 구성한다면 이진 트리보다 훨씬 균형이 잡힌 트리를 구성할 수 있다는 점을 이용하여 이진 트리의 성능을 개선한 가중 이진 트리 구조[3]가 제안되었다. 또한 프리픽스들을 성질에 따라 여러 그룹으로 나누고, 각 그룹에는 서로 디스조인트한 프리픽스만이 존재하게 함으로서 여러 개의 완전 이진 트리를 구성하고, 각각의 완전 트리에 대하여 이진 검색을 수행하는 구조가 제안되었다[4].

2.3 영역에 따르는 이진 검색(Binary Search on Range[5])

인터넷 주소 검색이 어려운 이유는 앞서 서론에서 잠시 언급한 바와 같이 입력된 어드레스의 프리픽스 길이 정보를 알지 못하는 상태에서 라우팅 테이블에 저장된 프리픽스중 가장 길게 일치하는 프리픽스를 찾아야 하기 때문이다. 영역에 따르는 이진 검색 구조에서는 모든 프리픽스를 같은 길이의 시작점과 끝 점을 갖는 영역으로 표시하는 방법으로 프리픽스의 길이정보를 제거하고, 각 구간에 따르는 BMP를 미리 계산하여, 인터넷 주소의 이진 검색을 가능하게 하였다. 그러므로 구간의 개수는 최악의 경우 프리픽스 개수의 두배와 같으며, BMP의 선계산으로 인하여, 프리픽스의 부가적 추가(incremental upate)가 가능하지 않은 단점을 갖는다.

2.4 리프-푸쉬(Leaf-Pushing[6])

리프-푸쉬이란, 이진 트라이의 내부 노드에 위치한 모든 프리픽스를 리프로 내려, 프리픽스가 리프에만 존재하도록 만든 구조이다. 그림 1을 예로 설명하면, **0***, **000***, **0010*** 의 3개의 프리픽스가 존재함을 가정할 때, 리프-푸쉬을 수행하면 프리픽스가 **01***, **000***, **0010***, **0011***로 변환되며, 프리픽스 **01***와 **0011***는 프리픽스 **0***로부터 라우팅 정보를 물려받게 된다. 리프-푸쉬을 이용하면, 모든 프리픽스들이 서로 디스조인트한 성질을 갖게 되므로, 디스조인트한 프리픽스들에 대하여 이진 검색을 수행하는 알고리듬[6]이 적용 가능하여 진다. 하지만 프리픽스의 수가 증가한다는 단점을 갖는다.

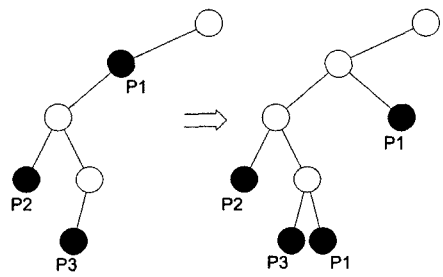


그림 1. 리프-푸쉬의 예

2.5 프리픽스 길이에 대한 이진 검색(Binary Search on Length[8])

이 구조는 앞서 서론에서 언급한 바대로 프리픽스 길이에 대하여는 이진 검색을, 프리픽스 값에 대하여는 해쉬을 사용하는 구조이다. 이 구조에서는 이진 트라이를 레벨별로 나누고 각 레벨에 존재하는 모든 노드들을 하나의 해쉬 테이블에 저장하는 라우팅 테이블을 만든다. 입력된 어드레스에 대해 각각의 레벨에 따르는 해쉬값을 구하여 해당 해쉬 테이블을 접근하였을 때, 프리픽스와 일치하는 경우에는 더 긴 프리픽스가 존재할 것을 가정하여, 더 긴 레벨로 진행하고, 프리픽스와 일치하지 않는 경우에는 현재 레벨보다 긴 프리픽스가 존재하지 않는 것을 가정하여, 짧은 레벨로 진행함으로서, 프리픽스 길이에 대하여 이진 검색을 수행하는 구조이다. 그러나 해당 해쉬 테이블의 프리픽스와 일치하였다고 하여도, 더 긴 레벨의 프리픽스가 존재하지 않을 수 있고, 해당 해쉬 테이블의 프리픽스와 일치하지 않았다고 하여도, 더 긴 레벨의 프리픽스와 일치할 수 있기 때문에, 길이에 대한 이진 검색을 위

한 이 구조의 기본적인 가정에 모순이 존재한다.

이러한 모순을 해결하기 위하여 이 구조에서는 선계산(pre-compution) 기법을 이용하였다. 비어있는 내부 노드에 마커를 미리 계산, 저장하여, 현재의 해쉬 값이 프리픽스와 일치하지 않은 경우에도 더 긴 프리픽스가 존재함을 표시한다. 그러나 마커가 모든 종류의 프리픽스를 커버할 수는 없다는 문제점이 있다. 마커가 있었기 때문에, 아래 레벨에 대해서 이진 검색을 진행하였으나, 더 길게 일치하는 프리픽스가 존재하지 않은 경우, 해당 마커의 위 쪽 레벨에 대하여 다시 검색을 해야하는 백-트랙킹(back-tracking) 문제가 발생한다. 이경우 검색 시간을 증가시키기 때문에 바람직하지 않은데, 이는 모든 마커마다 현재까지의 BMP를 미리 계산하여 저장함으로써 해결한다. 마커보다 더 긴 레벨을 검색하였으나 실패한 경우 현재까지의 BMP를 기억하고 있으므로, 그것이 입력된 인터넷 주소와 가장 길게 일치하는 프리픽스가 되는 것이다. 마커와 BMP는 모든 비어있는 내부 노드에 저장될 필요가 없으며, 이진 검색시 먼저 접근이 되는 레벨의 내부 노드에만 미리 계산하여 저장하면 된다. 이 구조의 경우 프리픽스 길이에 대하여 이진 검색을 수행하므로, 하나의 어드레스 검색을 위한 최대 메모리 접근 횟수가 매우 작다는 장점을 갖는다.

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

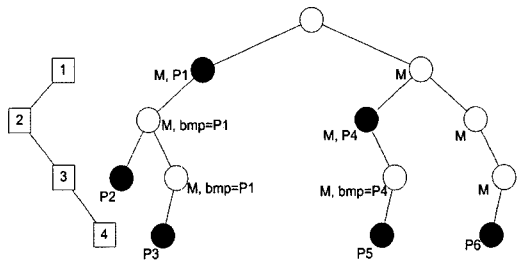


그림 2. 프리픽스에 대한 이진 검색의 예

라우팅 정보가 사용된다. 다른 내부 노드들에 대해서도 마커와 BMP를 선계산 해준 결과가 그림2에 표시되어 있다. 이 구조는 IPv4의 경우 최대 5번의 메모리 접근으로 하나의 어드레스 룩업이 가능한 장점을 갖으나, 마커와 해당 마커에 대한 BMP를 미리 계산해주어야 하며, 이러한 선계산 때문에 프리픽스의 부가적 추가(incremental update)가 가능하지 않다는 단점을 지닌다.

Ⅲ. 제안하는 구조

앞서 2.5절에서 설명한 Waldvogel[8]의 프리픽스 길이에 따르는 이진 검색 구조는 길이에 대해서 이진 검색을 수행하기 때문에 검색에 필요한 메모리 접근 횟수에 있어 매우 우수한 성능을 보이지만, 마커와 각각의 마커에 대한 BMP를 미리 계산해 주어야 한다는 단점이 있었다. 마커 및 BMP를 없앨 수 있다면, 복잡한 선계산을 없앨 수 있는 장점이 있다. 본 논문에서는 마커 및 BMP의 선계산을 없앨 수 있는 방법을 연구하였다.

먼저 마커와 마커의 BMP가 왜 필요한 것인지 정리하여 보면, 마커는 특정 레벨에서 일치하는 프리픽스가 없더라도, 그보다 더 긴 레벨에서 일치하는 프리픽스가 존재할 수 있기 때문에 필요하다. 그러나 마커가 있다고 해도 더 긴 레벨에 프리픽스가 존재하지 않을 수도 있기 때문에 짧은 레벨로 다시 검색이 거슬러 돌아가야하는 문제점을 해결하기 위하여 마커에 해당하는 BMP를 미리 계산하여 저장하는 것이다.

만약 마커보다 더 긴 레벨에 입력과 일치하는 프리픽스가 반드시 존재한다면, 마커와 BMP를 모두 계산해 줄 필요가 없게 된다. 이것은 특정 트리의 리프가 모두 프리픽스로 차있다는 것과 동일한 의미를 갖는다. 2.4절에서 설명한 리프-푸싱[6]을 수행하면, 트리의 리프를 모두 프리픽스로 채울 수 있게 된다. 따라서 리프-푸싱된 데이터를 이용하면, 마커

와 BMP를 모두 저장할 필요가 없어진다. 즉 본 논문에서는 라우팅 데이터에 대하여 먼저 이진 트라이를 구성하고, 이진 트라이의 내부 노드에 존재하는 모든 프리픽스들은 리프로 푸쉬한 후 프리픽스 길이에 대하여 이진 검색을 수행하는 구조를 제안한다.

길이에 대한 이진 검색은 다음의 세가지 경우로 나누어 생각될 수 있다. 먼저 해쉬에 의하여 검색된 엔트리가 프리픽스라면 모든 프리픽스는 리프에 존재하기 때문에 가장 긴 프리픽스를 찾았으므로 검색을 종료한다. 다음은 검색된 엔트리가 내부 노드인 경우로서, 더 긴 프리픽스가 반드시 존재하므로 더 긴 레벨로 검색을 진행한다. 마지막으로, 검색된 엔트리에 노드가 존재하지 않는 경우, 짧은 레벨로 검색을 진행한다. 따라서 리프-푸쉬을 이용하면 마커나 BMP를 미리 계산함 없이, 프리픽스 길이에 대한 진정한 의미의 이진 검색이 가능하여 진다.

그림 3은 그림 2의 트라이를 리프-푸쉬한 트리이다. 앞의 그림 2에서의 예를 사용하여 설명하면, 입력된 어드레스가 **001100**인 경우, 먼저 레벨 2의 노드 **00***를 검색하게 된다. 노드 **00***는 내부 노드이므로, 더 긴 레벨로 검색을 진행하고, 레벨 3의 **001*** 역시 내부 노드이므로 더 긴 레벨로 검색을 진행하고, 레벨 4의 **0011***가 검색된다. **0011***는 프리픽스이므로, 일치한 프리픽스인 P1을 출력한 후 검색이 종료된다. 입력된 어드레스가 **110000**인 경우, 레벨 2인 노드 **11***가 검색되고 내부 노드이므로 더 긴 레벨로 검색을 진행하여 레벨 3의 **110***가 검색되나 검색된 엔트리에는 노드가 존재하지 않으므로 짧은 레벨로 진행하여야 한다. 그러나 레벨 2와 레벨 3 사이에는 다른 레벨이 존재하지 않으므로, 일치하는 프리픽스 없이 검색이 종료된다.

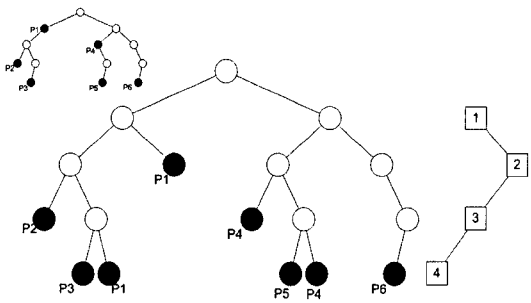


그림 3. 제안하는 트리 레벨을 사용한 이진 검색 구조의 예

3.1 라우팅 테이블 빌드

완전 해쉬 함수를 가정하였을 때 본 논문에서 제안하는 구조의 라우팅 테이블을 만드는 방법은 다음과 같다. 먼저 이진 트라이를 만든 후, 리프-푸쉬을 수행하여, 리프-푸쉬된 이진 트라이를 만든다. 트라이의 레벨별로 존재하는 모든 노드들의 해쉬값을 구하고, 해쉬값을 메모리의 어드레스로 하는 엔트리에 해당 노드 값 및 노드의 종류(내부 노드 혹은 프리픽스)와 같은 노드의 정보를 저장한다.

3.2 프리픽스 추가

새로운 프리픽스를 추가하려 하는 경우, 먼저 추가하려는 프리픽스를 검색한다. 검색결과 다음의 세가지 경우를 나누어 생각할 수 있다.

먼저 추가하려는 프리픽스가 리프에 추가되는 경우이다. 이 경우에는 이미 존재하는 리프 프리픽스와 일치할 수도 있고, 그림 4의 **1111***와 같이 비어있는 엔트리에 추가되어질 수도 있는데, 이 경우에는 기존의 프리픽스를 새로운 프리픽스로 대치하거나, 새로운 리프를 생성하면 된다.

다음은 추가하려는 프리픽스가 기존의 내부 노드와 일치하는 경우이다. 이 경우 리프-푸쉬을 수행하여, 리프에 존재하는 프리픽스와 비교하여 추가하려는 프리픽스가 기존에 존재하는 프리픽스보다 긴 경우에만 기존의 프리픽스를 대치한다. 그림 3을 예로 설명하면 새로운 프리픽스 **P7=11***를 추가하고 싶은 경우, 레벨 2의 내부 노드와 일치하였으므로, 리프-푸쉬을 수행하고, **110***은 리프이므로 프리픽스를 추가하고, **111***에는 내부 노드가 존재하므로, 한번 더 리프-푸쉬을 수행한다. **1110***에는 보다 긴 프리픽스가 존재하므로, 새로운 프리픽스로 대치하지 않고, **1111***에는 리프가 존재하지 않으므로 새로운 프리픽스를 추가한다. 이 과정을 그림 4에 보였다.

다음은 추가하려는 프리픽스에 해당하는 내부 노드도 리프도 존재하지 않는 경우로서, 이 경우에는 해당 엔트리에 리프 프리픽스를 추가한 후, 위로 검

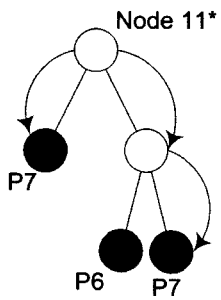


그림 4. 제안하는 트리 레벨을 사용한 이진 검색 구조의 프리픽스 추가의 예

색을 진행하여 이미 존재하는 내부 노드를 만날 때까지 내부 노드를 추가한다. 만약 위로 검색하는 중에 프리픽스가 있었다면, 그 프리픽스를 두 번째 경우와 같이 리프-푸싱을 해준다. 예를 들어 그림 3에 **0100***를 추가하고 싶다면, 먼저 **0100***를 해당 리프에 저장한 후, 한 레벨 위의 **010***를 검색한다. 내부 노드가 존재하지 않으므로 해당 엔트리에 내부 노드 **010***를 추가한다. 다시 한 레벨 위의 **01***를 검색한다. 프리픽스가 존재하므로, 이 노드를 내부 노드로 변환하고, 이 프리픽스는 두번째 경우와 같이 리프-푸싱을 수행하여, 리프 **011***와 **0101***에 저장한다.

3.3 복수 해싱(Multiple hashing[9])

4장에서 보이려는 바와 같이 완전 해싱 함수를 가정하였을 때 제안하는 알고리듬이 효과적으로 동작하는 것을 확인하였다. 하지만 실제 하드웨로 구현하는 데 있어 완전 해싱 함수를 기대하는 것은 불가능하기 때문에, 본 논문에서는 완전 해싱 함수를 사용한 실험에 더하여 구현가능한 해싱 함수를 사용한 경우를 실험하였다. 해싱에서 발생할 수 있는 충돌 현상을 줄이고, 라우팅 테이블에 데이터를 골고루 분포시키기 위하여 본 논문에서는 복수의 해싱 함수를 사용하였다.

3.3.1 해싱 함수

각각의 라우팅 데이터에 대하여 최적화되어 있는 해싱 함수를 찾는 것은 매우 어렵고, 시간이 오래 걸리는 작업이므로, 본 논문에서는 구현이 간단한 해싱 함수인 CRC(cyclic redundancy check) 해싱 함수를 사용하였다. CRC 해싱 함수는 구현이 간단할 뿐만 아니라, 여러 길이에 해당하는 해싱 값을 하나의 해싱 함수를 사용하여 찾아낼 수 있다는 장점이 있다. CRC-32는 32개의 레지스터들이 직렬로 연결되어 있으며, generator polynomial의 형태에 따라 레지스터의 중간 중간에 EXOR가 수행된다. 본 논문에서는 그림 5에 나타난 것처럼 CRC-32를 해싱 함수로 사용하였다. 예를 들어 첫 레벨 검색이 프리픽스 길이 16에 대하여 수행된다면, 입력된 어드레스의 상위 16 비트가 한 비트씩 차례로 CRC-32로 입력된 후, 상위 16개의 레지스터 중 미리 지정된 몇 개의 레지스터로부터 값을 읽어 해쉬값을 결정한다. 다음은 레벨 24에 대하여 검색이 수행된다면, 다시 입력 어드레스의 상위 24 비트를 한 비트씩 차례로 CRC-32에 입력한 후, 상위 24개의 레

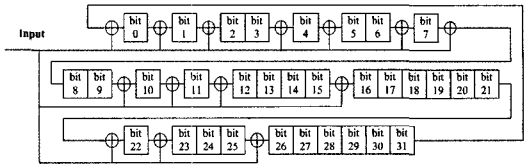


그림 5. CRC-32의 구조

지스터중 미리 지정된 레지스터들의 값을 읽어 해싱값을 결정한다. 이런 방법을 사용함으로, 프리픽스 길이에 대해 이진 검색을 수행하는 구조를 구현함에 있어, 하나의 해싱 함수를 사용하여 구현이 가능하다.

3.3.2 병렬 검색

오버플로우 엔트리는 TCAM 에 저장됨을 가정하였다. 리프가 오버플로우 되어 TCAM 에 저장되었다면, 해쉬 테이블의 검색을 끝내고 TCAM 을 검색하여 가장 길게 매치하는 프리픽스를 찾는데 아무런 아무런 문제가 없다. 그러나 내부 노드가 오버플로우되어 TCAM 에 저장되는 경우, 그 내부 노드 보다 긴 레벨에 존재하는 프리픽스는 해쉬 테이블에 올바로 저장되어 있다 하더라도 제대로 검색이 되지 않는다. 그러므로 본 논문에서 제안하는 구조에서는 그림 6에 나타낸 바와 같이 각 레벨 별로 해싱 테이블을 접근할 때, 동시에 TCAM 검색을 수행하여 이진 검색을 진행한다.

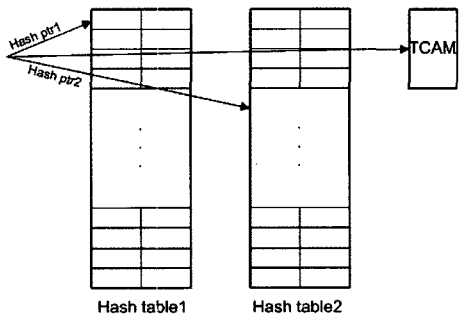


그림 6. 복수 해싱을 사용하였을 때의 병렬 검색

Ⅳ. 제안하는 구조의 성능 평가

본 논문에서는 리프-푸싱 기법을 이용하여 모든 프리픽스들을 트라이의 리프에 위치하게 함으로써, 마커와 BMP를 미리 계산하지 않고도 프리픽스 길이에 따르는 이진 검색을 수행하는 구조를 제안하였다. 이를 완전 해싱 함수를 가정하여 실험해 본 결

표 1. 여러 라우팅 데이타에 대한 성능 실험 결과

	N	N'	N^t	N_a	$M(byte)$
MAE-West1	14,553	19,968	82,669	4.06	402.4
MAE-West2	14,937	20,995	87,162	4.23	424.4
Aads	20,204	25,751	105,243	4.61	512.6
Pb	20,519	25,337	104,183	4.62	507.5
MAE-West3	29,584	42,908	127,019	3.81	619.5
MAE-East1	37,993	58,976	194,084	4.58	946.4
MAE-East2	39,464	59,377	193,739	4.49	944.9
Funet	40,905	59,249	144,924	3.82	707.0

과 제대로 동작함을 확인할 수 있었다. 표 1은 15000여개에서 40000여개를 갖는 라우팅 데이터(N)[10] 에 대하여 리프-푸쉬 결과 확장된 프리픽스의 개수(N'), 내부 노드를 포함하는 전체 노드의 개수(N^t), 입력된 인터넷 주소를 검색하는 데 소요되는 평균 메모리 접근 회수(N_a) 및 라우팅 테이블을 저장하는 데 소요되는 메모리의 크기(M)를 보여주고 있다.

표 1의 결과를 분석하여 보면 입력된 어드레스와 가장 길게 일치하는 프리픽스를 찾는 데 평균 3.8에서 4.6 번의 메모리 접근이 필요한 것을 볼 수 있는데, 이는 최악의 경우인 5번의 메모리 접근에 매우 근접하는 결과로서 이런 결과가 나타난 이유를 연구하여 보았다. 그림 7은 라우팅 데이터의 길이별 분포를 나타낸 것이다. 그림에서 보는 바와 같이 프리픽스는 길이 24와 길이 16에 매우 많이 존재함을 알 수 있다. 그러나 표 2의 실험 결과는 그림 8의 왼쪽에 있는 길이별 검색 트리, 즉 대칭 검색 방법에 의한 것으로서, 모든 프리픽스가 균일하게 존재할 때의 길이별 이진 검색 트리이다. 이 경우 프리픽스가 많이 분포하는 길이 24나 길이 16이 초반에 검색되지 아니하고 후반에 검색되기 때문에 평균 메모리 접근 횟수가 최악의 경우에 근접하는 것으로 생각될 수 있었다.

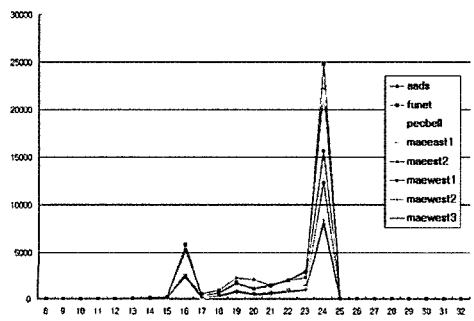
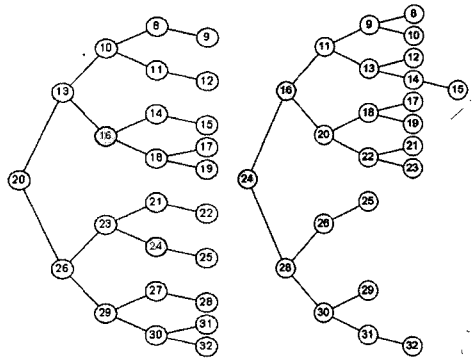


그림 7. 프리픽스의 분포 특성



(a) 대칭 이진 검색 트리 (b) 비대칭 이진 검색 트리

그림 8. 프리픽스 길이별 이진 검색 트리

프리픽스 분포 특성을 고려하여 평균 메모리 접근 회수를 줄이기 위한 방법은 두 가지를 고려하여 볼 수 있을 것이다. 먼저 자주 입력되는 어드레스의 프리픽스를 먼저 검색하는 방법이다. 이는 매우 효과적인 방법일 것으로 생각되나 시간에 따르는 편차가 심하여 트래픽 발생 분포가 변할 때마다 길이별 이진 검색 트리를 재구성하지 않는다면 의미가 없을 것으로 생각한다. 또 다른 방법은 각각의 네트워크들이 비슷한 양의 트래픽을 발생시킴을 가정하여, 가장 많은 프리픽스들이 존재하는 레벨을 먼저 검색하는 방법으로서 합리적인 방법으로 생각된다. 그림 8의 오른쪽 트리는 많은 프리픽스를 갖는 길이 24와 길이 16을 먼저 검색하기 위하여 재구성된 트리이다. 왼쪽의 대칭 검색 트리와 비교하여 볼 때 최악의 경우 6번까지 메모리 접근이 필요하다는 단점이 있으나 실험 결과 평균 메모리 접근 횟수에 있어서는 매우 우수한 것으로 나타났다. 표 2에 대칭 이진 검색과 비대칭 이진 검색의 실험 결과를 비교하였다. 비대칭 이진 검색 결과 평균 약 2.4번의 메모리 접근으로 어드레스 검색이 가능함을 알 수 있었다.

표 2. 대칭 이진 검색과 비대칭 이진 검색 비교

	N	대칭 이진 검색		비대칭 이진 검색	
		N_a	Min,Max	N_a	Min,Max
MAE-West1	14,553	4.06	1, 5	2.12	1, 5
MAE-West2	14,937	4.23	1, 5	2.25	1, 5
Aads	20,204	4.61	1, 5	2.07	1, 5
Pb	20,519	4.63	1, 5	2.04	1, 5
MAE-West3	29,584	3.81	1, 5	2.41	1, 6
MAE-East1	37,993	4.58	1, 5	2.40	1, 5
MAE-East2	39,464	4.49	1, 5	2.10	1, 5
Funet	40,905	3.82	1, 5	2.02	1, 6

표 3. 성능비교(38k 엔트리 라우팅 데이터)

	n	N_a	M(KB)	N'
Binary Trie[1]	30	23.2	864.2	-
Multi-bit Trie[1]	15	12.2	948.7	106,460
BPT[2]	25	15.8	351.4	-
WBPT[3]	17	15.3	351.4	-
DPT[7]	16	14.9	324.4	58,976
Binary search on Range[5]	17	15.9	404.0	78,867
Waldvogel[8]	6	2.2	0.98MB	171,647
Proposed	5	2.1	946.4	194,084

다음은 기존의 구조와 제안하는 구조의 성능을 비교하는 실험을 수행하였다. 표 3에 약 38000여개의 엔트리를 갖는 MAE-East1 라우팅 데이터에 대하여 평균 메모리 접근 횟수와 소요되는 메모리의 크기를 비교하였다. 표 3에서 보여주는 바와 같이 제안하는 구조는 타 구조에 비하여 2-3배의 메모리를 요구하나, 트리의 깊이(n, 최악의 경우 메모리 접근횟수)나 평균 메모리 접근 횟수(N_a)에 있어 타 구조에 비하여 월등히 좋은 결과를 보여줌을 알 수 있었다.

다음은 10만개의 엔트리를 갖는 매우 큰 라우팅 데이터에 대하여 기존의 구조와의 성능을 비교하였다. 표 4에서 보는 바와 같이 제안하는 구조는 라우팅 테이블의 크기에 따라 크게 성능 차이를 보이지 않음을 알 수 있었다. 이는 본 논문에서 제안하는 구조가 라우팅 데이타가 커지면 평균 메모리 접근 횟수가 증가하는 BPT, WBPT 그리고 DPT보다 커다란 라우팅 데이타에 적합함을 나타낸다.

다음은 완전 해쉬 함수를 가정하지 않고, 본 논문에서 제안하는 구조를 복수의 해쉬 함수를 사용하여 구현한 경우의 성능 실험이다. 해쉬 함수로는

표 4. 성능비교(100k 엔트리 라우팅 데이터)

	n	N_a	(MB)	N'
Binary Trie[1]	26	23.6	1.26	-
Multi-bit Trie[1]	13	12.0	1.85	201,681
BPT[2]	32	18.2	1.26	-
WBPT[3]	20	16.6	1.26	-
DPT[7]	18	16.1	722.2KB	137,553
Binary search on Range[5]	18	16.9	1.17	178,984
Waldvogel[8]	6	3.4	1.48	258,095
Proposed	6	3.3	1.48	310,717

표 5. 동일한 크기의 해쉬 테이블(2.6 Mbyte) 메모리를 사용한 경우의 오버플로우

	N	메모리 사용효율	오버플로우 확률(개수)
MAE-West1	14,553	15.21 %	0 %
MAE-West2	14,937	16.03 %	0 %
Aads	20,204	19.37 %	0 %
Pb	20,519	19.17 %	0 %
MAE-West3	29,584	23.41 %	0.004 %(5)
MAE-East1	37,993	35.75 %	0.046%(87)
MAE-East2	39,464	35.68 %	0.047%(91)
Funet	40,905	26.70 %	0.053%(77)

앞서 설명한 바대로 CRC-32를 사용하였다. 표 5는 버켓당 두개의 엔트리를 갖는 해쉬 테이블 두개를 사용하여 실험한 결과 여러 가지 사이즈의 라우팅 데이터에 대하여 오버플로우가 발생한 횟수를 보여주고 있다. 실험 결과에서 나타내주는 바와 같이 동일한 크기의 해쉬 테이블을 사용한 경우, 메모리 사용 효율(memory efficiency)이 올라감에 따라 오버플로우의 수가 증가함을 알 수 있다. 본 논문에서 제안하는 구조는 메모리 사용 효율을 30%내외로 유지하는 경우 매우 적은 수의 엔트리를 갖는 TCAM을 사용하여 구현 가능함을 알 수 있다.

V. 결 론

본 논문에서는 프리픽스 길이에 따르는 이진 검색을 위해 기존에 제안되었던 구조의 단점이었던 복잡한 선계산을 없애고, 리프-푸쉬만으로 길이에 따르는 이진 연산을 가능하게 한 구조를 제안하였다. 완전 해쉬 함수를 가정하고 비대칭 검색 방법을 사용한 경우 본 논문에서 제안하는 구조는 40K 엔트리 이하의 라우팅 데이터에 대하여는 최대 6번 평균 2.5번의 메모리 접근으로 어드레스 검색이 가능함을 보였으며, 100K 엔트리 라우팅 데이터에 대하여는 최대 6번 평균 3.3번의 메모리 접근 성능을 보였다. 실제적인 완전 해쉬 함수의 구현은 불가능하다고 판단하여 복수의 해쉬 함수를 사용하여 제안하는 구조를 구현하여 제안하는 구조가 잘 동작함을 확인하였다. 기존의 프리픽스 길이에 따르는 이진 검색 구조가 부가적 추가가 전혀 불가능하였던 것에 반하여 본 논문에서 제안하는 구조는 부분적으로 프리픽스의 부가적 추가가 가능한 구조이다. 또한 라우팅 데이타의 크기가 커져도 성능에 영향을 받지 않는, 확장성이 매우 우수한 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] N. Yazdani and P. S. Min, "Fast and Scalable Schemes for the IP Address Lookup Problem", Proc. *IEEE HPSR2000*, pp. 83-92, 2000.

[3] 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.

[4] Hyesook Lim, Bomi Lee, and Won-Jung Kim, "Binary Searches on Multiple Small Trees for IP Address Lookup", *IEEE Communications Letters*, vol.9, no. 1, pp.75-77, Jan. 2005.

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

[6] V. Srinuvasan and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion", *ACM Transactions on Computer Systems*, Vol. 17, No. 1, pp.1-40, Feb. 1999.

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

[8] 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.

[9] H. Lim and Y. Jung, "A Parallel Multiple Hashing Architecture for IP Address Lookup", Proc. *IEEE HPSR2004*, pp.91-98, Apr. 2004.

[10] Merit Networks, Inc. http://www. merit. edu

문 주 형(Ju Hyoung Mun) 준회원

2005년 2월 이화여자대학교 정보통신학과, 학사

2005년 3월~현재 이화여자대학교 정보통신학과, 석사과정

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

임 혜 숙(Hyesook Lim) 정회원

1986년 2월 서울대학교 제어계측공학과, 학사

1986년 8월~1989년 2월 삼성휴렛 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