고속의 최장 IP 주소 프리픽스 검색을 위한 비트-맵 트라이

(A Bit-Map Trie for the High-Speed Longest Prefix Search of IP Addresses)

오 승 현 † 안 종 석 ††
(Seung-Hyun Oh) (Jong-Suk Ahn)

요 약 본 논문은 IPv4와 IPv6을 지원하는 라우터에서 기가비트의 속도로 포워딩 검색을 수행하는 효율적인 포워딩 테이블 구조를 제안한다. 포워딩 검색은 최장 프리픽스 일치검색, LPM(Longest Prefix Matching)의 복잡도가 포워딩 테이블 및 주소크기에 따라 증가하여 라우터 성능의 병목지점으로 알려져 있다. 포워딩 검색의 고속화를 위해 본 논문에서는 빈번한 메모리 접근을 최소화할 수 있는 BMT(Bit-Map Trie) 자료구조를 소개한다. BMT 포워딩 검색은 필요한 모든 검색연산이 캐쉬에 저장된 소형 인덱스 테이블에서만 발생한다. 포워딩 테이블의 트라이로부터 소형 인덱스 테이블을 구축하기 위해서 BMT는 차일드(child) 노드 포인터와 포워딩 테이블 엔트리에 대한 포인터를 각각 한 비트로 표현하는 비트-맵을 구성한다. 또한 IPv6와 같이 주소길이가 증가하면 트라이의 깊이가 깊어져서 전통적인 트라이 검색속도가 느려지는 문제점을 해결하기 위해서 BMT에서는 검색을 시작할 적절한 트라이의 레벨을 결정하는 이진검색 알고리즘을 사용한다. 실험 결과 BMT는 IPv4 백본 라우팅 테이블을 펜티엄-II 프로세서의 L2 캐쉬 크기인 512KB 보다 작게 압축하였으며, 최대 250ns/패킷의 검색속도를 제공하여 기존의 알려진 가장 빠른 최장 검색 알고리즘의 성능과 같은 속도를 실현하였다.

키워드 : 라우터, 기가비트, 포워딩 검색, 트라이, LPM

Abstract This paper proposes an efficient data structure for forwarding IPv4 and IPv6 packets at the gigabit speed in backbone routers. The LPM(Longest Prefix Matching) search becomes a bottleneck of routers' performance since the LPM complexity grows in proportion to the forwarding table size and the address length. To speed up the forwarding process, this paper introduces a data structure named BMT(Bit-Map Trie) to minimize the frequent main memory accesses. All the necessary search computations in BMT are done over a small index table stored at cache. To build the small index table from the trie representation of the forwarding table, BMT represents a link pointer to the child node and a node pointer to the corresponding entry in the forwarding table with one bit respectively. To improve the poor performance of the conventional tries when their height becomes higher due to the increase of the address length, BMT adopts a binary search algorithm for determining the appropriate level of tries to start. The simulation experiments show that BMT compacts the IPv4 backbone routers' forwarding table into a small one less than 512-kbyte and achieves the average speed of 250ns/packet on Pentium II processors, which is almost the same performance as the fastest conventional lookup algorithms.

Key words : router, gigabit, forwarding lookup, trie. LPM

· 본 연구는 한국과학재단 목적기초연구[R01-2000-000-00280-0(2002)] 지원으로 수행되었음
† 정 회 원 : 동국대학교 컴퓨터학과 교수
shoh@dongguk.ac.kr
†† 종신회원 : 동국대학교 컴퓨터공학과 교수
jahn@dgu.edu
논문접수 : 2002년 5월 17일
심사완료 : 2003년 1월 21일

1. 서 론

최근 들어 인터넷 백본이 기가비트급 링크로 대치되며 백본 트래픽이 폭발적으로 증가함에 따라 기가비트급 고속 라우터에 대한 연구[1]가 활발히 진행되고 있다. 인터넷 백본의 고속화로 인해, 초당 기가비트 또는

테라비트 속도로 패킷을 처리하는 고속 라우터 개발이 활발히 추진되고 있다. 이러한 고속 라우터 연구는 스위칭 조직(switching fabric)연구와 더불어 라우터 성능의 병목지점으로 알려진 포워딩 검색을 고속화하는 연구가 활발히 진행되고있다. 포워딩 검색은 완전 일치 검색과는 달리 O(1)의 성능을 이루기가 어려운데, 이는 포워딩 검색은 목적지 IP 주소와 가장 긴 IP주소가 일치하는 포워딩 테이블 엔트리(entry)를 찾아내는 최장 프리픽스 일치 검색(LPM: Longest Prefix Matching)이기 때문이다. LPM검색의 성능은 테이블의 크기나 IP 주소의 길이에 비례하는 것으로 알려졌다[2].

포워딩 검색 성능을 향상하기 위한 기존의 연구로는 빠른 접근속도를 갖는 SRAM, CAM과 같은 별도의 하드웨어를 사용하는 *하드웨어 기반* 연구들[1,3~7]과 새로운 프로토콜을 사용하는 *프로토콜 기반* 연구[8~11] 및 포워딩 테이블의 자료구조를 개선하는 *소프트웨어 기반*의 연구[2,12~18]가 있다. 특히 소프트웨어 기반의 연구는 다른 두 가지 종류의 연구와는 달리 개별적으로 각 라우터에 새 포워딩 테이블 구조와 알고리즘을 적용하기 때문에 쉽게 구현되며 비용이 저렴한 장점이 있다. 최근에 기가비트급 속도를 지원하기 위해 한 패킷을 수백~수십 나노초(nanoseconds)에 처리하는 자료구조들이 제안되었다. 이 부류의 연구들은 대개 포워딩 테이블을 압축하거나, LPM 검색을 완전 일치검색으로 변환하여 접근속도가 느린 메모리 사용을 제한한다.

또한 포워딩 검색은 새로운 IP 프로토콜인 IPv6을 사용할 때에는 쉽게 고속화할 수 있을 것으로 기대되었다. 이는 체계적인 IP주소 분배와 CIDR(Classless InterDomain Routing) 방식으로 기존 포워딩 테이블의 크기를 획기적으로 줄일 수 있기 때문이다[19]. 그러나 IPv6을 사용할 경우에도 사용자가 다수의 서비스 제공자에 가입하는 경우에 멀티-홈(multi-home) 문제가 발생하여 포워딩 테이블의 크기가 여전히 증가할 것으로 예상된다. 즉 사용자가 서로 다른 서비스 제공자에게서 IP주소를 할당받는 경우에는, 이들 사용자의 개별 주소가 서비스 제공자의 라우터에서 대표자의 주소로 통합되지 않기 때문이다. 또한 포워딩 검색의 복잡도는 대체로 주소 길이의 개수에 비례하기 때문에, IPv6 주소의 길이 증가는 포워딩 검색 속도를 저하시킨다. 일례로 [16]은 프리픽스의 길이별로 해시 테이블을 구성하고, 프리픽스의 길이를 키로 해시 테이블을 이진 검색한다. IPv6에서 포워딩 테이블의 프리픽스 길이도 IPv4에 비해 다양하게 구성되어 해시 테이블이 늘어남에 따라 검색속도도 저하된다.

본 논문은 백본 라우팅 테이블을 펜티엄 프로세서[20]의 L2 캐쉬에 저장할 수 있는 작은 크기의 포워딩 테이블로 압축하여 저속 메모리 접근을 최대한 억제함으로써 기가비트 포워딩 검색을 수행할 수 있는 새로운 자료구조와 검색 알고리즘을 제시한다. 본 논문에서 제안한 BMT[21] 자료구조는 기존 논문 [13]의 비트열 아이디어를 기반으로 하고 있으나, 본 논문은 다음 두 가지 점에서 기존의 방식과 다르다. 첫째는 트라이와 해쉬를 병행해서 사용하는 것과는 달리 포워딩 테이블 전체를 하나의 트라이로 표현하였으며 모든 트라이의 포인터를 비트열의 균일 구조로 전환하였다. 둘째, IPv4 주소를 검색하는 BMT를 확장하여 128비트 IPv6 주소를 고속으로 검색하는 이중 이진검색 알고리즘을 제시한다. .

본 논문의 구조는 다음과 같다. 2장에서는 세 가지 부류의 포워딩 검색방식을 소개하며, 3장에서는 IPv4를 위한 BMT와 검색 알고리즘에 대해 기술하며, 4장에서는 IPv4 BMT의 포워딩 검색 실험 결과를 제시한다. 5장에서는 IPv6을 위해 확장된 BMT 기반 이중 이진검색 알고리즘을 기술하고, 마지막으로 6장에서는 결론과 향후 연구과제를 제시한다.

2. 관련연구

2.1 하드웨어 기반 연구

SRAM을 이용한 고속 포워딩 방법[1]은 10ns 정도의 접근시간을 가진 SRAM에 포워딩 테이블을 저장하는 방법이다. SRAM은 가격이 비싸 중소규모 라우터에 주로 사용한다. [4]에서는 포워딩 테이블의 프리픽스들을 길이별로 각각의 CAM에 저장하여 병렬로 주어진 IP 주소를 검색한다. 이 방법에서는 IPv4는 32개의 CAM, IPv6은 128개의 CAM이 필요하므로 구현비용이 비싸다는 단점이 있다. 비용을 낮추기 위해서는 하나의 CAM에 모든 길이의 프리픽스를 저장하여야 하는데, 이 경우에는 저장된 엔트리를 서로 다른 비트 길이로 검색할 수 있어야한다. 그러나 이 방식은 CAM 셀에 무관심(don't care) 비트 등을 추가로 구현해야 하므로 더 많은 트랜지스터를 요구하게 되어 CAM의 데이타 용량을 축소시키게 된다.

캐쉬를 이용하는 방법은 포워딩 엔트리를 캐쉬에 미리 저장하여 성능을 향상시키는 것인데, 낮은 캐쉬 적중률(hit ratio) 때문에 백본 라우터에는 적용하기 어렵다. [3]에서는 소규모 캐쉬를 이용하여 포워딩 검색속도를 최소 65% 향상시킨다고 보고하였다. 그러나 인터넷상에서의 트래픽의 멀티플렉싱 정도와 포워딩 테이블의 크기를 고려하며, 높은 적중률을 유지하기 위해서는 수천개 이상의 캐쉬 라인이 필요하다. [7]은 트라이를 기반

으로 한 링크-리스트(link-list)를 이용하여 포워딩 테이블을 구성하고, 구성된 테이블을 고속의 SRAM에 저장하여 빠른 검색속도를 얻을 수 있음을 보여준다. 마지막으로 [5,6]에서는 파이프라인과 병렬처리 구조를 이용하여 검색 성능을 증가시킬 수 있음을 보여주었다.

2.2 프로토콜 기반 연구

프로토콜을 이용한 연구에는 ATM과 MPLS[9] 같이 LPM문제를 완전 일치 방식으로 변환하는 기법과 IP 패킷헤더를 변형하는 기법[8]이 있다. ATM에서는 출발지-목적지 노드 사이의 경로에 설정되어 있는 VCI/VPI 라벨(label)을 이용하여 포워딩 엔트리를 검색한다. 이러한 라벨을 스위칭 테이블의 인덱스 또는 해시함수의 키로 사용하여 검색을 고속화한다. MPLS, 태그(tag) 스위칭[10], 그리고 플로우(flow) 스위칭[11]은 ATM과 마찬가지로 IP 주소를 라벨 또는 플로우 식별자로 변환하여 검색을 고속화한다. 일례로, MPLS의 경우는 진입(ingress) 라우터와 진출(egress) 라우터 사이에 형성된 경로에 대하여 ATM과는 달리 호-설정 없이 동적으로 라벨을 배정한다. 그리고 중간 라우터들은 이 라벨을 이용하여 포워딩을 수행한다. 진출 라우터에서는 최종적으로 패킷에 부착된 라벨은 제거하고 이후 포워딩은 종래의 검색방법을 사용한다.

IP 패킷헤더를 변형하는 기법[8]은 클루(clue)정보를 IP 헤더에 포함하여 라우터가 검색해야 할 포워딩 테이블의 범위를 줄인다. 큐는 전 라우터에서 경험한 검색과정과 포워딩 검색범위 정보를 의미하는 것으로, 다음 라우터에서 전체 포워딩 테이블을 검색하지 않고 이 클루 정보를 이용하여 검색범위를 축소한다. 이러한 방식은 포워딩 검색에 필요한 부담을 패킷 이동경로상의 전체 라우터에 분산시켜 포워딩 검색속도를 향상시키게 된다. 그러나 이러한 프로토콜 기반 연구들은 순수 IP 네트워크에서 타 라우터와의 연동 및 확장성에 문제가 있다.

2.3 소프트웨어 기반 연구

소프트웨어 기반 연구는 먼저 프리픽스 트라이를 이용하는 패트리샤 트라이(patricia trie)[12] 방식이 있다. 이 방법은 NetBSD 1.2에서 사용한 기법[2]으로, 한번에 한 비트씩 트라이 노드와 목적지 IP 주소를 비교한다. [2]에서 보고한 주소 당 검색시간은 $1.44 \log N$이다. 이때 N은 포워딩 엔트리의 개수인데, 백본 포워딩 테이블의 크기인 50,000를 가정하면 최악의 경우에는 22회 이상의 메모리 접근이 필요하다. 50ns이상의 느린 메모리 속도를 감안할 때, 이러한 메모리 접근 횟수로는 기가급 속도를 성취할 수 없다. 이러한 단점을 보안하기 위해, [2]에서는 반복되는 0이나 1의 비트열을 압축해서 표현하는 스킵 카운트(skip count)를 사용하는 개선된 방법을 제시하였다. 그러나 여전히 검색이 실패할 경우 백-트래킹(back-tracking)이 발생하여 검색 속도가 저하된다.

[14]는 16비트 길이의 해시 테이블을 먼저 적용하고 16비트 이상의 더 긴 프리픽스에 대해서는 이진검색 기법을 적용하였다. [15]는 프리픽스 길이의 개수를 최소화함으로써 프리픽스 길이를 키로 한 이진검색을 빠르게 한다. 프리픽스 길이 개수의 최소화는 제한된 프리픽스 확장을 통해 형성되며, 결과적으로 프리픽스 길이 별로 만들어진 이진검색 테이블의 개수가 최소화됨으로써 검색속도를 향상시킨다. [16]은 프리픽스 길이별로 해시 테이블을 만든 후 프리픽스 길이를 키로 이진검색을 한다. [17]의 LC-trie 기법은 트라이 레벨을 압축하여 검색시간을 단축하는 방법이다. 이진 트라이의 링크에서 연속적으로 나타나는 동일한 비트를 제거하여 레벨을 압축하고, 다시 차일드 노드가 두 개 이상인 다중(multiway) 트라이로 변환하여 레벨을 이중으로 압축한다. [18]은 포워딩 테이블을 프리픽스 및 포스트픽스(postfix) 트라이로 분리하여 트라이 저장에 필요한 메모리 크기를 절감한다. 포스트픽스 트라이는 프리픽스의 뒷 부분 즉, 프리픽스의 하위 비트 값들을 공유하는 트라이로써 포워딩 테이블의 메모리 크기를 줄임으로써 속도를 개선한다. [13]은 프리픽스 트라이의 차일드 링크정보와 포인터 정보를 하나의 비트로 표현하여 포워딩 테이블의 크기를 캐쉬 크기로 압축한다. 트라이 깊이 16, 24 및 32에서 구성된 비트열은 트라이 저장에 필요한 메모리 크기를 줄여 캐쉬에 저장함으로써 포워딩 검색을 빠른 캐쉬검색으로 전환한다.

3. BMT(Bit-Map Trie)

3.1 BMT 자료구조

트라이[22]는 저장공간을 절감하기 위해 프리픽스 부분을 상호 공유하는 문자열들을 표현하는데 적합한 트리 형태이다. <프리픽스/길이> 형태인 라우팅 테이블의 프리픽스 엔트리가 비트열로 표현될 수 있고 많은 프리픽스 부분의 비트가 서로 중복되어 있기 때문에, 트라이는 라우팅 테이블을 압축하여 작고 효율적인 포워딩 테이블을 만들기에 좋은 자료구조이다. 일례로 그림 1은 라우팅 테이블의 전형적인 예와 그에 상응하는 트라이 표현을 보여주고 있다. 트라이의 링크는 위치에 따라 좌측은 0으로, 우측은 1로 표현하여 IP 주소의 비트열을 표시한다. 노드는 루트로부터 해당노드까지의 경로 상에 있는 링크가 나타내는 비트열의 총합과 같은 프리픽스

엔트리를 표시한다. 이때 본 논문에서 "프리픽스 노드"라고 명명된 검은색 노드는 포워딩 테이블에 해당 프리픽스 엔트리가 존재하는 노드로써 해당 엔트리의 위치정보를 갖는다. 그림 1에서 프리픽스 "8*/8"에 해당하는 검은색 프리픽스 노드는 루트노드로부터 비트열 "00001000" 프리픽스를 나타낸다. 또한 이 프리픽스는 포워딩 테이블에 존재하므로 이 노드는 "8*/8" 엔트리에 대한 위치 정보를 보유하고 있다

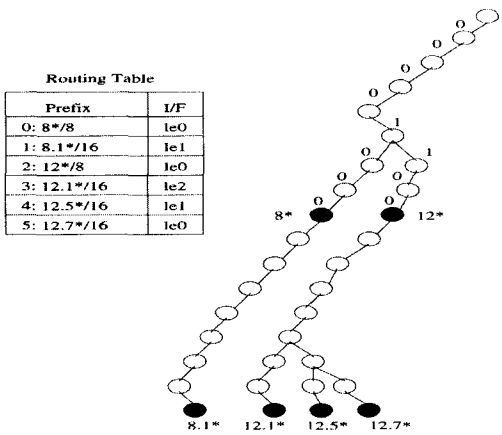


그림 1 전형적인 라우팅 테이블과 트라이

32비트 IPv4주소에서 표현 가능한 모든 주소를 그림 1과 같은 트라이로 표현하면 깊이는 32, 리프노드의 최대 개수는 2^{32}개가 된다. 이때 트라이의 모든 리프노드를 좌측에서 우측으로 펼쳐놓으면 0부터 2^{32}-1까지 가능한 모든 IP 주소 값을 일렬로 늘어놓은 것과 같다. 이때 포워딩 테이블에 모든 IP 주소에 해당되는 2^{32}개의 엔트리를 만들어 각 IP주소에 해당하는 프리픽스와 출력포트 정보를 저장하면, IP주소 값을 이 포워딩 테이블 배열의 인덱스로 사용하여 O(1) 복잡도로 출력포트를 검색할 수 있다. 이러한 자료구조는 방대한 메모리 크기 때문에 구현이 불가능하므로 다음과 같이 메모리 양을 줄여갈 수 있다.

첫째, 일렬로 펼쳐진 포워딩 테이블에서 같은 프리픽스 값을 갖는 인접한 엔트리들은 하나의 엔트리로 통합한다. 이 경우 IP주소 자체를 배열의 인덱스 값으로 사용할 수 없게된다. 또한 어느 엔트리가 인접 엔트리와 통합되었는지를 알아야 하는데, 이러한 정보는 트라이의 리프노드에 0과 1을 할당하여 표현될 수 있다. 즉 0은 인접한 엔트리와 동일한 프리픽스 값을 갖는 엔트리이므로 인접한 엔트리에 통합되었음을 의미하며, 1은 해당 엔트리가 포워딩 테이블에 존재한다는 것을 의미한다. 1로 표시되는 리프노드는 위에서 정의한 프리픽스 노드에 해당된다. 이때 포워딩 테이블에서 임의의 IP에 대한 엔트리의 인덱스는 IP 주소 0.0.0.0에 해당되는 리프노드부터 주어진 IP 주소에 해당되는 리프노드 사이에 표시된 1의 개수이다. 이렇게 각 리프노드를 0과 1로 표시한 비트열을 프리픽스 비트-맵(PBM: Prefix Bit-Map)이라 명한다.

둘째, 이러한 PBM을 하나의 배열로만 표현하게 되면 메모리 사용량이 여전히 방대해지므로 트리 구조의 계층적 PBM으로 변환한다. 참고로 32비트 주소에 해당되는 PBM 배열은 2^{32}비트, 즉 512Mbyte가 필요하다. 트리 구조 PBM을 만들기 위해서는 그림 2와 같이 임의의 계층에서 리프노드에 대한 PBM을 만들고 이들 PBM을 트리 구조와 같이 연결한다. 즉 이진 트리에서 각 8계층마다 PBM을 만들게 되면 256비트의 PBM들이 만들어지며, 이 PBM의 256개 비트들 각각은 다시 256비트의 차일드 PBM을 갖는다. 이러한 트리 PBM은 일차원 PBM과는 달리 모든 IP 주소에 해당되는 리프노드를 갖고 있지는 않으며, 임의 노드의 하단 트리에 프리픽스 노드가 존재하지 않는 경우에는 이 노드의 PBM에서는 더 이상의 차일드 PBM을 만들지 않는다. 이러한 트리 PBM에서 인덱스를 계산하기 위해서는 주어진 IP 주소의 프리픽스를 나타내는 차일드 PBM을 루트 PBM에서 시작하여 차례로 검색한다.

임의의 계층에서 PBM을 만들기 위해서는 먼저 그림 1에서와 같이 이진 트리에서 프리픽스 노드를 결정하는 데, 프리픽스 노드는 루트로부터 이 노드까지의 링크들에 배정된 0과 1의 조합이 포워딩 테이블내의 프리픽스 값과 같은 노드이다. 트리 PBM 구조에서 포워딩 테이블 인덱스 값은 트리 PBM에서 해당되는 프리픽스 노드를 검색했을 때, 이 노드를 트리 PBM에서 DFS(Depth-First-Search) 방식으로 검색할 때 이 노드에 도달하기 전까지 방문했던 모든 프리픽스 노드의 개수이다. 일차원 PBM과 마찬가지로 프리픽스 노드에 1을 배정하므로 모든 프리픽스 노드들은 서로 다른 1의 개수를 갖게 되고 따라서 서로 다른 포워딩 테이블 인덱스를 갖는다.

그림 2의 프리픽스 노드들은 검은색으로 칠해져 있는데, 일차원 PBM과는 달리 이때의 PBM은 상위노드가 프리픽스 노드인 경우에는 가장 왼편에 있는 리프노드에 1을 배정한다. 일례로 0, 4, 7, 8, 14 및 15번 노드에는 1을 나머지는 0을 할당하게 된다. 이는 노드 0, 1, 2, 3에 해당되는 IP와 최장 일치되는 프리픽스는 상위의 부모 프리픽스 노드이므로 이들 네 개의 노드들은 같은 1의

개수를 공유해야 하고 따라서 노드 0에만 1을 배정한다.

또한 각 리프노드들은 차일드 노드를 가질 수 있으므로 트리 PBM 구조는 차일드 비트열(CBM: Child Bit-Map)을 갖는다. 즉 이 비트열은 차일드 노드의 존재유무를 1과 0으로 표현하게 되는데, 차일드 노드가 있을 때는 1을 없을 때는 0을 배정한다. 트리 PBM에서는 이 CBM을 보고 검색 계속 여부를 결정한다. 차일드 노드가 없을 때는 이 계층에서 계산한 인덱스 값이 LPM을 만족하는 엔트리의 인덱스이며, 차일드 노드가 있을 때는 다시 차일드 노드의 CBM에서의 1의 개수를 계산해야 한다. 차일드 노드의 CBM의 위치는 PBM에서 인덱스를 계산하는 방식과 마찬가지로 현재의 차일드 노드까지의 1의 개수의 총합이 CBM 테이블의 인덱스가 된다. 즉 PBM과 CBM이 일정 길이를 갖게되고 일정 방식에 의해 정돈되어 배열에 저장되어 있을 때는 1의 개수에 의해 각각의 차일드 PBM과 CBM을 찾을 수 있다.

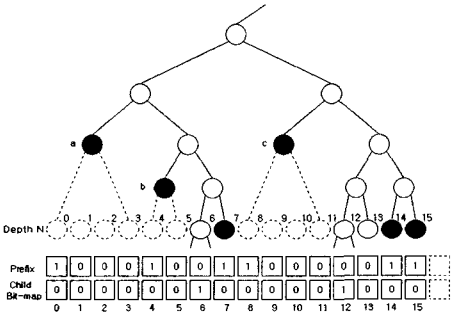


그림 2 비트 맵 구성

그림 3은 그림 1의 라우팅 테이블이 완전이진 트라이[21]로 확장된 모양과 CBM을 보여준다. 본 연구에서 트리 BM은 트라이의 8의 배수 깊이마다 구성되므로 각 CBM 엔트리의 길이는 256이다. 그림 3에서 점선으로 표시된 삼각형, 부 트라이는 표시의 간략화를 위해 깊이 16의 트라이를 4의 배수 깊이에서 분할하여 CBM을 구성한 예를 보여준다. CBM은 프리픽스 노드가 가리키는 포워딩 테이블 인덱스의 순서를 지키기 위해 부 트라이의 WFS(Width-First-Search) 순서로 수집된다. 참고로 CBM을 구성하는 트라이의 깊이가 깊어지면 비트열의 길이가 길어져서 비 프리픽스 노드를 표시하는 0이 많아지고, 깊이가 너무 얕을 때에는 CBM 엔트리의 개수가 과다해진다. CBM에서 0은 트라이의 구조를 표현하지만 포워딩 테이블 인덱스를 계산하는데는 의미가 없다. 따라서 과다한 0의 중복은 메모리의 낭비를 초래할 뿐이다. CBM을 구성하기 적절한 트라이 깊이를 정하는 문제는 다음 연구로 미루어진 상태이다.

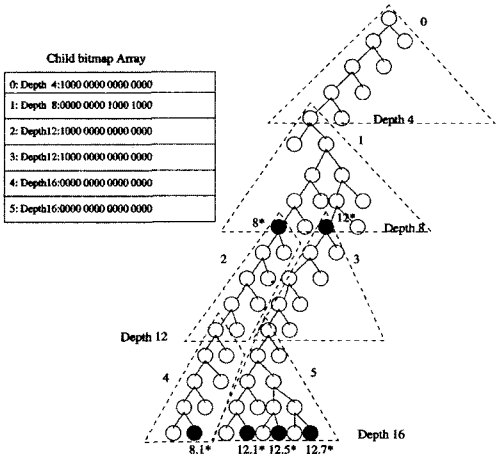


그림 3 트라이의 CBM 노드 표현 예제

그림 3의 CBM에서 차일드 노드 탐색을 위해 1의 개수를 계산할 때 발생하는 반복적인 비트 시프트(shift) 연산은 오버헤드가 된다. CBM의 비트열을 일정한 길이의 구간으로 구분하여 각 구간별로 1의 개수를 미리 카운트하여 함께 저장함으로써 CBM 탐색시간을 줄일 수 있다. 이때 필요한 공간은 구간의 길이 64비트 당 한 바이트로 약 5만개의 프리픽스에 대해 35K~40K 바이트가 소요되는데, 이 크기는 포워딩 테이블을 캐쉬에 저장할 때 부담이 되지 않는다. 그림 3의 트라이에서 PBM의 구성은 CBM을 만드는 방법과 동일하며, 앞에서 기술한 바와 같이 노드에 값을 할당하는 방법만이 다르다.

그림 4는 본 논문에서 제안한 비트-맵 트라이(BMT)의 구성 예제로, 그림 4 (a)는 비트-맵 트라이를, 그림 4 (b) 비트-맵 테이블은 별도로 구성되었던 PBM과 CBM이 통합되어 하나의 비트-맵 테이블을 구성함을 보여준다.

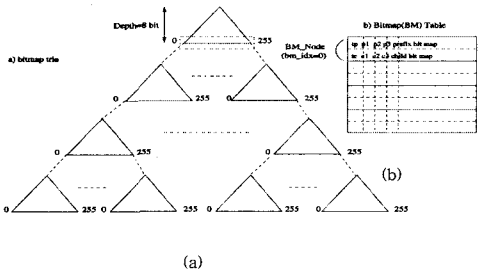


그림 4 비트-맵 트라이와 비트-맵 테이블

3.2 BMT 검색 알고리즘

BMT 기반의 포워딩 검색에는 두 가지 방법이 있다. 첫 번째 검색 알고리즘은 주어진 IP 주소를 네 개의 8비트 조각으로 분리, 검색키로 사용하여 BMT를 루트노드로부터 탑-다운(top-down) 방식으로 검색하므로 최대 4회의 검색이 발생한다. 두 번째 검색 알고리즘은 확장된 검색 알고리즘으로 중간 레벨의 노드에서 검색을 시작하여 BMT 검색횟수를 절감할 수 있다.

3.2.1 검색 알고리즘

BMT 검색은 1로 표시된 비트 하나가 라우팅 테이블의 프리픽스 엔트리 하나를 표시하며, BMT의 첫 번째 비트부터 특정 비트 사이의 1의 총계가 라우팅 테이블의 엔트리의 오프셋(offset)이라는 점을 이용한다. 하지만 BMT 검색은 1의 개수를 세기 위해 한 비트씩 비교하여 탐색하지는 않는다. BMT의 엔트리 즉, 노드는 깊이가 8인 부 트라이의 리프노드를 표시하고 있다. 따라서 8비트 키 검색은 키 값을 인덱스로 직접 비트 위치를 결정할 수 있다. BMT 검색은 두 가지 단계로 구성된다. 먼저 주어진 IP 주소를 네 개의 8비트 조각으로 분리하고 각 조각을 검색키로 CBM에서 탐색 가능한 마지막 노드 즉, 검색 대상 IP 주소에 가장 많은 비트가 일치하는 노드를 결정한다. 두 번째 단계는 PBM의 첫 번째 비트부터 첫 번째 단계에서 결정된 노드에 해당하는 비트 사이에 존재하는 1의 총계를 계산한다. 참고로 CBM과 PBM은 동일한 위치에서 구성되므로 CBM에서 찾은 비트의 위치는 PBM에서의 비트위치와 같다.

표 1 BMT를 이용한 포워딩 검색 알고리즘

```
1:  Let Bit-Map table BM and an index bm_idx:=0, and
2:  routing entry pointer ptr:=default routing entry
3:  for (addr=IP_Address[m*8:m*8+7], m=0,1,2,3) {
4:     get BM_Node:=BM(bm_idx);
5:     get byte_idx:=addr / 8;
6:     get bit_idx:=addr mod 8;
7:     if (BM_node.childBit(byte_idx)<<bit_idx & 0x80) {
8:           set bm_idx:=number of all child bit 1; and
9:   }
10:    else
11:          set ptr:=number of all prefix bit 1;
12: }
13: return pointers[ptr];
```

표 1에 BMT 검색 알고리즘을 간략하게 표시하였다. 앞에서 기술한 두 단계의 검색에서 첫 단계 CBM 검색은 라인 3~10 부분이다. 라인 7은 각 8비트 조각을 키로 차일드 노드를 검색한다. 두 번째 단계의 검색은 PBM의 첫 비트와 CBM에서 결정된 비트와 같은 위치의 PBM 비트 사이에 있는 1의 개수를 라인 11에서 계산한다. 어떤 IP 주소에 대해 CBM에서 검색된 비트가 n 엔트리의 120번째 비트라면 1의 개수 계산은 0~(n-1) 엔트리에 속한 1의 개수와 n 엔트리의 0~120번 비트 사이의 1의 개수를 합한 값이 된다.

3.2.2 확장 검색 알고리즘

확장된 BMT 포워딩 검색 알고리즘은 표 2에서 볼 수 있다. 확장 검색의 의미는 트라이를 탑-다운 검색할 때 루트노드에서 검색을 시작하지 않고 중간 노드에서부터 검색을 시작하는 것이다. 3.1절의 BMT 자료구조에서 기술한 바와 같이 IP 주소는 그림 4의 작은 삼각형으로 표시된 4단계의 8비트 부 트라이로 구성되므로 최대 4회의 부 트라이 탐색이 발생한다. 이에 비해 확장 검색기법은 두 번째 부 트라이 레벨에서 검색을 시작하여 최대 3회로 탐색 회수를 줄이는 것이다. 다만 표 2의 라인 6에서 사용된 자료구조와 같이 레벨 16에서 수집된 별도의 비트열 자료구조가 추가된다. 참고로 확장 검색 알고리즘은 라우팅 프리픽스의 길이가 16비트 근처에서 가장 많이 형성되는 점을 반영한 것이다.

표 2 확장된 BMT 검색 알고리즘

```
1:  Let Bit-Map table BM and an index bm_idx:=0, and
2:  strt_pos:=0 and
3:  routing entry pointer ptr:=default routing entry
4:  get b_byte_idx:=IP_Address[0:15] / 8;
5:  get b_bit_idx:=IP_Address[0:15] mod 8;
6:  if (pnt16_child[b_byte_idx]<<b_bit_idx & 0x80) {
7:    bm_idx:=number of all child bit 1; and
8:    set strt_pos:=2; and
9:  }
10: else {
11:   set ptr:=number of all prefix bit 1;
12:   return pointers[ptr];
13: }
14: for (addr=IP_Address[m*8:m*8+7], m=strt_pos~3) {
15:   same to #4 ~ #11 lines of Algorithm 1
16: }
17: return pointers[ptr];
```

4. BMT 실험

BMT 포워딩 테이블에 대한 성능측정은 표 3의 IPMA [23] 라우팅 테이블과 본 연구기관에서 사용중인 BGP (Border Gateway Protocol)[24] 라우터의 테이블을 이용하였다. 실험에 사용된 운영체제와 CPU는 리눅스 커널

2.2.x에서 400㎒ 펜티엄Ⅱ 프로세서(L1 cache 32KByte, L2 cache 512KByte)를 이용하였다. 실험에서 사용된 라우팅 테이블은 표 2에서 보는바와 같이 4만8천여 개 이상의 엔트리를 가진 백본급 라우팅 테이블부터 수천 개의 엔트리를 가진 중소규모까지 다양하게 구성되어 있다. 본 실험에서 BMT 포워딩 테이블은 약 48,000여 개의 엔트리에 대해 284KB로 L2 캐쉬에 충분히 저장될 수 있는 작은 크기로 만들어짐이 확인되었다.

BMT 포워딩 테이블의 성능은 두 가지 종류의 라우팅 테이블과 두 가지 종류의 IP 주소를 혼합한 네 가지 시나리오에 대해 측정되었다. 실험에 사용된 IP 주소는 랜덤으로 발생시킨 IP 주소와 네트워크에서 수집한 실제 IP 주소이며, 포워딩 테이블을 캐쉬에 저장한 것과 비슷한 효과를 얻기 위해 동일한 IP 주소를 2회 연속 검색하여 평균값을 구했다. 검색시간 측정은 *getrusage()*함수를 사용하였으며, 전체 검색시간을 IP 주소 개수로 나누어 IP 주소별 검색시간을 결정하였다. 표 4~7은 서로 다른 네 가지 시나리오에 의한 성능측정 결과를 보여주고 있다.

표 3 IPMA 라우팅 테이블 별 BMT 크기와 생성시간

Forwarding Tables	# of Routing Entries	# of Prefix Nodes	Bit-Map Table Size (KB)	Built Time(ns)
Mae-East	48,290	52,858	284	290
Mae-West	29,588	32,462	247	210
PacBell	25,275	26,897	226	190
AADS	16,864	18,093	190	140
Paix	9,618	10,434	141	80

표 4는 각 IPMA 라우팅 테이블이 <# of packets> 행의 개수만큼 IP 주소를 포워딩 검색할 때 BMT의 두 가지 검색방법인 <Seek time>과 확장검색 <Extended seek time>에 대한 평균 검색속도를 정리하고 있다. 일반검색과 확장검색의 성능은 각각 215~260ns와 215~250ns의 범위이며, <Resolved ratio>는 33% 이하로 매우 낮은 값을 보인다. 이것은 오직 33% 정도의 패킷만이 라우팅 엔트리를 찾았으며 나머지 패킷은 목적지 엔트리 검색에 실패하여 처리되지 않았음을 의미한다. 즉, 대다수의 검색이 루트노드에 대한 첫 번째 검색 즉, 첫 번째 8비트 조각을 키로 한 검색에서 실패하였음을 의미한다. 이러한 결과의 이유는 랜덤으로 만들어진 IP 주소 값이 실제 사용되지 않는 IP 주소 범위에서 많이 만들어지고 있기 때문이다. 표 5는 이와는 달리 실제 수집된 트래픽의 주소를 사용하여 <Resolved ratio>가 100%에 가깝게 되고, 표 4에 비해 거의 2배의 속도저하를 보여준다. 이것은 트라이 검색이 첫 번째 단계보다 더 하위 단계까지 진행되고 있음을 의미한다.

표 4 IPMA DB에 대한 랜덤 IP 주소 성능측정

	Mae-East	Mae-West	Pac-Bell	AADS	Paix
# of packets(K)	2000	2000	2000	2000	2000
Resolved ratio(%)	32.9	32.9	26.4	26.1	21.6
Seek time(ns)	260	250	240	240	215
Extended seek time(ns)	250	245	230	230	215

표 5 IPMA DB에 대한 실제 IP 주소 성능측정

	Mae-East	Mae-West	Pac-Bell	AADS	Paix
# of packets(K)	6199	6199	6199	6199	6199
Resolved ratio(%)	100	99.9	99.2	99.9	99.9
Seek time(ns)	492	501	492	503	492
Extended seek time(ns)	417	417	416	417	416

표 6, 7은 본 연구기관의 라우팅 테이블을 이용하여 두 가지 검색기법의 성능을 정리하고 있다. 두 개의 표에서 R11~R23은 두 개의 라우터에서 사흘 간 수집된 라우팅 테이블을 의미하며, 이들 표를 통해 우리는 실제 IP 주소에 대한 성능이 거의 두 배로 악화되었지만 모두 기가비트급 포워딩 성능을 제공할 수 있음을 확인하였다. 이러한 결과를 바탕으로 볼 때 기존의 연구성과 [19,22]에 대해서도 실제 IP 주소를 사용하여 성능을 다시 평가해볼 필요가 있다고 생각된다. 마지막으로 네 가지 시나리오의 결과를 비교해볼 때 테이블의 크기는 입력 트래픽에 비해 검색성능에 적은 영향을 미친다고 평가할 수 있다.

표 6 본 연구기관의 DB에 대한 랜덤 IP 주소 성능측정

	R11	R12	R13	R21	R22	R23
# of packets(K)	2000	2000	2000	2000	2000	2000
Resolved ratio(%)	30.5	31.3	31.3	30.5	31.3	31.3
Seek time(ns)	190	195	195	190	195	195
Extended seek time(ns)	210	210	215	210	215	210

그림 5는 왜 트래픽의 특성에 따라 성능이 큰 차이를 보이는지를 트라이 레벨별 검색 성공률의 분포를 통해 보여주고 있다. 실험 대상 테이블은 IPMA의 Mae-East와 본 기관의 R12이다. 랜덤주소는 대부분 레벨 1과 2에서 검색이 종료되며, 실제주소를 투입하였을 때는 본

표 7 본 연구기관의 DB에 대한 실제 IP 주소 성능측정

	R11	R12	R13	R21	R22	R23
# of packets(K)	2262	6199	6184	12446	3088	2553
Resolved ratio(%)	100	100	100	100	100	100
Seek time(ns)	344	312	312	314	314	321
Extended seek time(ns)	260	243	244	245	242	242

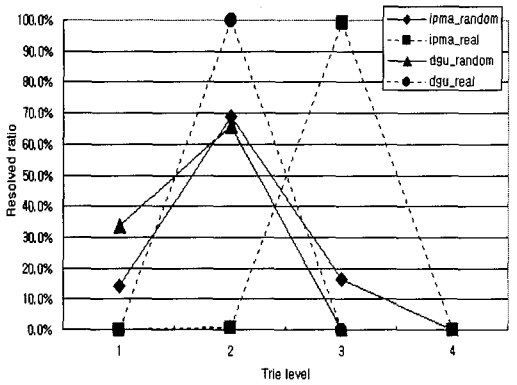


그림 6 포워딩 검색성공 레벨 분포

기관 테이블에서는 레벨 2에서, IPMA 테이블에서는 레벨 3에서 검색이 종료되는 비율이 절대 다수이다. 이러한 비교는 랜덤주소의 검색보다 실제주소의 검색에 더 많은 검색회수가 필요함을 보여준다.

표 8은 기존의 소프트웨어 기반의 연구들과 본 논문의 BMT 자료구조의 성능을 비교 표시한 것이다. 소프트웨어로 구현된 기존 연구성과들을 비교 평가하는 것은 의미가 있지만 이때 각 연구들의 지향점을 주의 깊게 살펴볼 필요가 있다. 즉, [14]와 BMT 기법은 포워딩 테이블을 L2 캐쉬 크기이하로 압축함으로써 고속검색 성능을 얻고자하는 반면에 타 연구들은 포워딩 테이블을 주 메모리에 저장하면서 검색성능을 향상시키고자 한다. 이러한 두 가지 종류의 연구를 표 8에서 비교하면 [15]는 3.2MB의 포워딩 테이블을 해시 테이블로 구성하여 비교적 빠른 속도를 얻고 있다. 하지만 [15]는 완전 해시를 위한 해시함수 구성에 필요한 시간이 13분 정도로 매우 길다는 약점이 있다. 표 8에서 본 연구의 결과를 제외한 나머지 연구들의 성능 값들은 직접 구현하여 측정한 값이 아니고 [15]에서 인용한 값과 직접적으로 성능이 발표되지 않은 것들은 펜티엄 II 프로세서로 속도를 환산하여 기록한 값이다.

표 8 기존 알고리즘과의 성능비교

	Size(KB/# of prefix)	Lookup speed per packet
[13]	160/32,732	652ns / PII 400MHz
[14]	950/38,816	430ns / PII 300MHz
[15]	3,200/38,816	226ns / PII 300MHz
[16]	1,200/33,000	650ns / PII 300MHz
[17]	800/41,578	1000ns / PII 300MHz
BMT	300/48,290	250ns / PII 400MHz

[13]은 BMT와 비트열을 이용한 연구로 BMT의 비트-맵과 비슷한 개념에서 출발한 연구이다. [13]에서 포워딩 테이블의 크기는 160KB로 매우 작지만 검색속도는 652ns로 BMT보다 2배 이상 느리다. [13]은 3단계의 비트열과 베이스 어드레스 테이블, 맵 테이블 및 컨버전 테이블로 구성되며, 각 단계별로 4회의 메모리 접근이 필요하므로 검색시간이 길어진다. 이에 비해 BMT는 비트-맵 테이블과 컨버전 테이블로 구성되는 단순한 구조이다. BMT는 각 단계별 검색에서 더 긴 프리픽스 검색을 위해 비트-맵의 비트 1의 개수를 카운트하는 번거로운 과정이 있지만 [13]은 비트 카운트 값을 베이스 어드레스 테이블과 맵 테이블에 대한 인덱스 값을 IP주소로부터 직접 계산을 통해 산출하는 장점이 있다.

5. 이중 이진검색 알고리즘

IPv6 주소의 길이는 128비트로 기존의 IPv4의 32비트 주소에 비해 4배의 길이로 확장되었다. 이러한 주소 확장으로 포워딩 검색시간이 주소 길이에 비례하여 증가하는 경우에는 IPv6용 기가비트급 고속 라우터를 소프트웨어로 구현하는 것은 어렵다. 일례로 [16]은 프리픽스 길이별로 해시 테이블을 만들고, 프리픽스 길이를 키로 해시 테이블을 이진 검색하므로 $O(\log_2(\text{주소 길이}(W)))$의 해시 검색을 한다. 그러나 해시 테이블 검색에 필요한 시간은 $O(W/M)$, (M=워드 크기)이므로 [16]의 실제 검색속도는 $O(W/M \cdot \log_2(\text{주소 길이}))$이며, 주소길이가 128로 확장된 IPv6의 검색시간은 느려진다.

BMT를 IPv6 주소에 적용하면 트라이의 최대 깊이는 128, 최대 부 트라이 레벨은 16이므로 검색시간이 증가할 수 있다. 그러므로 IPv4 주소 검색방법과는 달리 확장된 주소길이를 반영한 검색 방법이 필요하다. 본 연구에서는 이중 이진검색(dual binary search) 방법을 이용하여 IPv6 주소를 BMT에서 빠르게 검색할 수 있음을 보이고자 한다. 이중 이진검색은 외부와 내부 두 단계의 중첩된 이진검색으로 포워딩 검색을 한다. 첫 번째

단계의 외부 이진검색에서는 포워딩 검색을 할 프리픽스의 길이를 이진검색 방법으로 결정하고 두 번째 단계는 결정된 길이의 프리픽스에 대해 이진검색을 한다. 첫 번째 과정의 프리픽스 길이에 대한 이진검색은 최장 일치 검색을 만족시키기 위해서 가장 짧은 길이에서부터 가장 긴 길이의 프리픽스를 검색하는 대신 우선적으로 검색을 시작할 길이 L을 선택하는 것이다. 참고로 BMT에서 포워딩 검색을 할 때 8비트 단위 즉, 부 트라이 단위로 검색을 하므로 L은 부 트라이의 레벨이다. 두 번째 과정은 내부 이진검색으로 첫 단계에서 결정된 레벨 L에 대해 이진검색을 함으로써 고속의 일치검색을 할 수 있도록 한다. 이것은 모든 프리픽스 중에서 길이가 L*8이상인 프리픽스만을 대상으로 1~L*8 비트만을 검색함으로써 서로 다른 길이의 프리픽스에 대해서 이진검색을 할 수 있도록 한다. 이 방법의 특징은 프리픽스를 확장하거나 축소하지 않고 BMT에서 이진검색을 할 수 있다는 것이다.

5.1 이중 이진검색 기법을 위한 자료구조

128비트 주소에 대한 BMT 구성은 IPv4의 경우와 동일하다. 다만 프리픽스 이진검색에 필요한 두 가지 자료구조가 추가된다. 첫째, BMT의 각 엔트리가 속하는 부 트라이 레벨 L 정보가 필요하다. 이 정보는 부 트라이의 각 레벨별 이진검색을 수행할 때 레벨 L에 속하는 BMT 노드의 범위를 제공한다. 둘째, BMT의 각 노드에 대한 프리픽스 주소이다. 이 주소는 이진검색에서 직접 목적지 IP 주소와 비교하기 위한 주소이다.

BMT 엔트리에 대응하는 부 트라이 레벨의 범위정보는 각각의 엔트리에 모두 지정될 필요는 없고 단지 레벨별로 처음과 마지막 부 트라이에 대응하는 엔트리의 인덱스 정보만 구성하면 된다. 레벨 범위정보는 BMT를 구성하는 단계에서 간단한 조작을 통해 얻을 수 있으며, 이 정보를 저장하기 위한 메모리 크기는 부 트라이 레벨 개수(16) × 2바이트 = 32바이트이다.

두 번째 정보인 BMT 엔트리별 프리픽스가 필요한 이유는 BMT가 트라이 구조와 포워딩 테이블의 엔트리에 대한 인덱스 정보만을 비트로 압축하여 표현하므로 이진검색에 사용할 수 있는 프리픽스 주소정보가 BMT에 없기 때문이다. BMT 엔트리별 프리픽스 주소는 각 부 트라이의 루트에 대응하는 프리픽스 주소이다. 부 트라이의 각 리프노드에 대응하는 프리픽스 검색은 IPv4 BMT 검색기법에 의해 손쉽게 검색할 수 있으므로 프리픽스를 따로 저장할 필요가 없다. BMT 엔트리별 프리픽스의 길이는 해당 부 트라이의 루트의 깊이와 같으므로 부 트라이 레벨 × 8(레벨=0~N, 루트 부 트라이 레벨=0)이다. 참고로 BMT는 이진 트리와 같은 방법으로 구성되고, 각 부 트라이는 WFS 순서로 BMT의 엔트리가 되므로 BMT 엔트리별 프리픽스 주소는 자연스럽게 오름차순으로 정렬된 상태가 된다.

5.2 검색 알고리즘

이중 이진검색 기법에서 두개의 이진검색은 각각 내부와 외부로 중첩된 구조를 형성하고 있는데, 먼저 외부 이진검색은 IP 주소검색의 대상이 될 BMT 엔트리의 레벨 L을 선택한다. 두 번째 내부 이진검색은 첫 번째 이진검색에서 선택된 레벨 L에서 프리픽스 이진검색을 한다.

외부 이진검색은 최종적으로 선택될 프리픽스의 길이를 알 수 없는 상태에서 중간 길이의 프리픽스부터 검색을 시작하여 이진검색으로 가장 빠른 시간에 목적지

표 9 이중 이진검색 알고리즘

```
Function Dual_b_search(DA) /* search for destination address DA */
1: Initialize a low and high limit of outer binary search;
2: Initialize final routing index BMP to null;
3: While low_o <= high_o do          /* outer binary search */
4:        mid_o := (low_o+high_o)/2;
5:        Calculate a inner binary search key from DA [0:mid*8];
6:        Initialize a low and high limit of inner binary search from the base_address[] table;
7:        While low_i <= high_i do
          /* inner binary search */
8:           mid_i := (low_i+high_i)/2;
9:           To do binary search on sub-trie's prefix address;
10:       Endwhile
11:       If not found a matched prefix node then
12:                Modify the high_o to search upper half of trie for shorter prefix;
13:       Else if found a matched prefix and there is no child then
14:                BMP := number of all prefix bit;
15:                return BMP;
          /* end of search */
16:       Else if found a matched prefix and there is child link then
17:                Modify the low_o search the lower half of trie for longer prefix;
18:       Endif
19: Endwhile
20: return0;
                    /* fail of dual binary search */
```

IP 주소와 일치하는 최장 프리픽스의 길이를 결정한다. 내부 이진검색은 외부 이진검색에서 선택된 레벨 L에서 L*8 길이의 프리픽스에 대해 이진검색을 수행한다. 검색결과에서 일치하는 프리픽스가 검색되면 리프노드에 대한 검색을 실시한 후 다음의 세 가지 경우에 대해 추가검색 여부를 판단하게 된다. 첫째 프리픽스 검색이 성공하고 차일드 노드가 존재하지 않는 경우에는 찾은 프리픽스가 최장 프리픽스이므로 검색은 종료된다. 둘째 프리픽스 검색이 성공하고 차일드 노드가 존재할 경우에는 더 긴 프리픽스가 있다는 의미이므로 외부 이진검색에서 트라이의 상위 레벨 즉, L*8보다 더 긴 프리픽스를 대상으로 검색범위를 조정한다. 마지막으로 세 번째 경우는 프리픽스 검색에 실패한 경우로 현재 검색한 L*8 길이에서 일치하는 프리픽스는 없으므로 외부 이진검색에서 BMT의 하위 레벨 즉, L*8보다 더 짧은 길이의 프리픽스를 검색하도록 검색범위를 조정한다.

이중 이진검색 과정에 대한 알고리즘은 표 8에 표시되어있다. 표 8은 라인 3~19와 라인 7~10에서 두 개의 이진검색이 중첩되고 있음을 보여준다. 라인 3~19의 외부 이진검색은 BMT 레벨에 대한 검색이므로 최대 메모리 접근 회수는 $\log_2 L$이다. 라인 7~10의 내부 이진검색은 외부 이진검색에서 선택된 레벨 L에서 프리픽스 주소에 대한 이진검색이므로 트라이의 최하위 레벨이 선택되었을 경우 최대 검색범위가 $1 \sim 2^{120}$이 된다. 이 값은 128비트의 IPv6 주소에서 모든 IP 주소가 포워딩 테이블에 포함되었을 때 발생할 수 있는 최악의 경우이다.

그림 6은 단지 편리성을 이유로 32비트 주소를 사용하여 간략하게 이중 이진검색 알고리즘의 실제 동작과정을 보여주고 있다. 또한 아래의 표 9는 그림 6의 BMT를 대상으로 검색할 주소로 "4.17.1.1"이 주어진 경우 검색과정을 정리한 것이다.

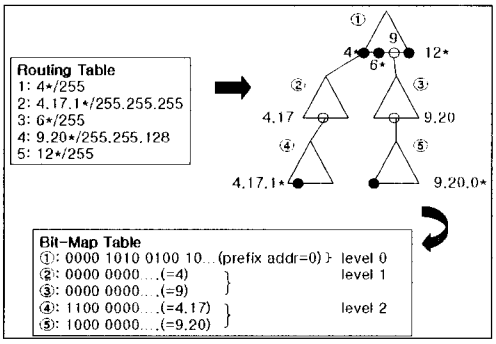


그림 7 이중 이진검색을 위한 32 비트 주소의 BMT 구성

표 10 이중 이진검색의 예제

```
Step 1: 외부 이진검색 레벨 선택
 - (그림 13) line 3 computes target level with low and high bound :=
   (low_o+high_o)/2 = 1
 - (그림 14) Bit-Map table entry of level 1 = ②, ③
Step 2: 내부 이진검색 범위 계산
 - (그림 13) line 8 computes middle node index := (low_i+high+i)/2 = 2
 - key of binary search := "4.17.1.1"&0x(level*8) = '4'
 - (그림 13) line 9 found a matched node ②
 - node ② has a child link that means existing a longer prefix
 - so, modify the lower bound of outer binary search := 1
 - if found a matched node and there is no child link, return
   an index as a routing index and exit searching
Step 3: goto Step 1 and to do next level searching
```

6. 결론 및 향후과제

본 논문은 기가비트 포워딩을 지원할 수 있는 새로운 자료구조인 BMT와 IPv6의 128비트 주소를 빠르게 검색할 수 있는 이중 이진검색 알고리즘을 제안하였다. 실험을 통해 BMT가 48,000 엔트리 이상의 라우팅 테이블을 300-KByte 이하로 압축할 수 있음을 보였다. 또한 랜덤 주소에 대해 190~250ns 속도로 포워딩 검색을 수행할 수 있었으며, 이 속도는 약 4백만 lps(lookup per second)의 성능을 나타낸다. 실제주소에 대한 검색 속도도 240~410ns로 약 240만 lps를 지원할 수 있다.

본 연구의 향후과제는 BMT와 다른 소프트웨어 기반의 연구를 동일한 플랫폼에 구현하여 네트워크에서 수집한 트래픽의 주소를 이용하여 실질적인 비교를 하는 것이다. 또한 랜덤으로 생성된 IPv6 주소와 가능한 한 실제 IP 주소를 수집하여 이중 이진검색 알고리즘을 실험하여 그 성능을 측정하는 것이다.

참 고 문 헌

[1] Peter Newman, Greg Minshall, and Larry Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, January 1997.

[2] Keith Sklower, "A Tree-Based Routing Table for Berkeley Unix," Technical report, University of California, Berkeley.

[3] David C. Feldmeier, "Improving gateway performance with a routing-table cache," *In proceedings of IEEE Infocom'98*, New Orleans, Louisiana, March 1998.

[4] Anthony J. Bloomfeld NJ McAuley, Paul F. Lake Hopatcong NJ Tsuchiya, and Daniel V. Rockaway Township Morris Country NJ Wilson, "Fast Mul-

tilevel hierarchial routing table using content-addressable memory," U.S. Patent serial number 034444.

[5] P. Gupta, et al., "Routing Lookups in Hardware at Memory Access Speeds," *In Proceedings of IEEE Infocom'98*, San Francisco, April 1998.

[6] A. Moestedt, et al., "IP Address Lookup in Hardware for High-Speed Routing," *Hot Interconnects*, August 1998.

[7] Pinar A. Yilmaz, Andrey Belenkiy, Necdet Uzun, and Nitin Gogate, A Trie-based Algorithm for IP Lookup Problem , *In Proceedings of Globecom'00*, 2000.

[8] A. Bremler-Barr, Y. Afek, and S. Har-Peled, "Routing with Clue," *In Proceedings of ACM SIGCOMM 99*, Cambridge, September 1999.

[9] E. Rosen, et al., "Multiprotocol Label Switching Architecture," ftp://ds.internic.net/internet-drafts/draft-ietf-mpls-arch-07.txt, July 2000.

[10] Yakov Rekhter et al, "Tag switching architecture overview,"ftp://ds.internic.net/internet-drafts/draft-rfcedinfo-rekhter-00.txt, 1996.

[11] Peter Newman, Tom Lyon, and Greg Minshall, "Flow labeled IP: a connectionless approach to ATM," *In Proceedings of IEEE Infocom'96*, San Francisco, California, March 1996.

[12] Donald R. Morrison, "PATRICIA-Practical Algorithm to Retrieve Information Coded In Alfanumeric," *journal of the ACM*, 15(4):514-534, October 1968.

[13] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen pink, "Small Forwarding Tables for Fast Routing Lookups," *In Proceedings of ACM SIGCOMM'97*, October 1997.

[14] B. Lampson, V. Srinivasan and G. Varghese, "IP Lookups using Multiway and Multicolumn Search," *In Proceeding of INFOCOM'98*, March 1998.

[15] S. Venkatachary and G. Varghese, "Faster IP Lookups using Controlled Prefix Expansion," *In Proceedings of ACM Sigmetrics'98*, June 1998.

[16] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner, "Scalable High Speed IP Routing Lookups," *In Proceedings of ACM SIGCOMM'97*, October 1997.

[17] S. Nilsson and G. karlsson, "Fast Address Look-Up for Internet Routers," *In Proceedings of IEEE Broadband Communications'98*, April 1998.

[18] T. Kijkanjanarat and H.J.Chao, "Fast IP Lookups using a Two-Trie Data Structure," *In Proceedings of Globecom'99*, 1999.

[19] S. Deering, and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *IETF RFC 2460*, December 1998.

[20] www.intel.com/design/PentiumII/

[21] 오승현, 안종석, "고속 라우터의 기가비트 포워딩 검색을 위한 비트-맵 트라이 구조", 정보과학회 논문지:정보통신, 2001.

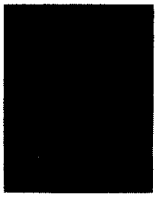
[22] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, "Data Structure and Algorithms," *Addison-Wesley*, 1983.

[23] Michigan University and merit Network, Internet Performance Management and Analysis (IPMA) project, http://nic.merit.edu/~ipma

[24] Y. Rekhter and T. Li. "A Border Gateway Protocol 4 (BGP-4)," *IETF RFC 1771*, March 1995.

오 승 현

1988년 동국대학교 전자계산학과(학사)
1998년 동국대학교 컴퓨터공학과(석사)
2001년 동국대학교 컴퓨터공학과(박사)
1987년~1996년 (주)대우엔지니어링
2002년~현재 동국대학교 컴퓨터학과 전임강사. 관심분야는 실시간 프로토콜, 네트워크 시뮬레이션, IPv6, 무선통신



안 종 석

1983년 서울대학교 전자공학과(학사)
1985년 한국과학기술원 전기 및 전자공학과(석사). 1995년 University of Southern California 컴퓨터공학과(박사)
1983년~1995년 삼성전자 선임연구원
1996년~현재 동국대학교 컴퓨터공학과 부교수. 관심분야는 실시간 프로토콜, 네트워크 시뮬레이션 멀티캐스트, 무선통신