균형 다중 트리를 이용한 고속 IP 어드레스 검색 기법

High-speed IP Address Lookup using Balanced Multi-way Trees

김 원 정 † 이 보 미 †† 임 혜 숙 †††
(Wonjung Kim) (Bomi Lee) (Hyesook Lim)

요 약 링크기술이 발달하면서 인터넷 라우터로 들어오는 패킷의 수가 급격히 증가하고 있으며, 이로 인하여 입력 패킷을 실시간으로 처리하는 일은 점점 더 어려운 작업이 되어가고 있다. IP 어드레스 검색은 라우터의 패킷 처리 기능 중 가장 필수적인 기능 중의 하나로서, 효율적인 IP 어드레스 검색을 위한 알고리즘과 구조에 대한 연구가 진행되고 있다. 본 논문에서는 작은 크기의 메모리를 사용하면서도 검색 속도를 빠르게 향상시킨 IP 어드레스 검색 구조를 제안한다. 제안하는 방법은 한번의 메모리 접근으로 여러 프리픽스를 동시에 비교하는 멀티웨이 트리에 기반을 두고 있는 구조로서, 약 4만개의 프리픽스를 저장하기 위해 280KByte 크기의 SRAM을 사용하고, 평균 5.9번의 메모리 접근으로 IP 어드레스를 검색하는 구조이다.

키워드 : IP 어드레스 검색, 균형 트리, 다중 트리, 라우터, 메모리 접근, 이진 프리픽스 트리

Abstract Packet arrival rates in internet routers have been dramatically increased due to the advance of link technologies, and hence wire-speed packet processing in Internet routers becomes more challenging. As IP address lookup is one of the most essential functions for packet processing, algorithm and architectures for efficient IP address lookup have been widely studied. In this paper, we propose an efficient IP address lookup architecture which shows very good performance in search speed while requires a single small-size memory. The proposed architecture is based on multi-way tree structure which performs comparisons of multiple prefixes by one memory access. Performance evaluation results show that the proposed architecture requires a 280KByte SRAM to store about 40000 prefix samples and an address lookup is achieved by 5.9 memory accesses in average.

Key words : IP address lookup, balanced tree, multi-way tree, router, memory access, binary prefix tree

1. 서 론

어드레스 검색은 입력 패킷의 목적지 IP 어드레스의 네트워크 부분 (프리픽스)을 보고 출력포트를 결정하는 작업이다. 이전에 사용된 class-ful addressing 구조에서는 프리픽스의 길이가 8비트, 16비트, 24비트인 class A, class B, class C로 구성되어 있어, 해쉬과 바이너리 검색 등 exact matching이 가능하였다. 그러나 호스트 수가 증가하고 네트워크가 점점 커지면서, 낭비되는 어드레스의 수가 증가하였고, 라우팅 테이블의 크기가 증가하는 문제점이 생기게 되었다.

이러한 문제점들을 해결하기 위해 classless inter-domain routing(CIDR) 구조가 도입되었다. CIDR 구조에서는 프리픽스 길이가 고정되어 있지 않고 임의의 길이를 가질 수 있어, 낭비되는 주소 공간이 줄어들어 어드레스를 효율적으로 사용할 수 있게 되었고, 여러 개의 어드레스를 하나의 어드레스로 합치는 address aggregation이 가능하여 라우팅 테이블의 크기가 기하급수적으로 커지는 것을 방지할 수 있게 되었다. 반면에, 입력 패킷의 프리픽스 길이를 알 수 없기 때문에, 라우팅 테이블에 있는 모든 프리픽스에 대해 어드레스 검색을 수행하여, 해당하는 여러 엔트리들 중 가장 길게

† 학생회원 : 이화여자대학교 정보통신학과
aksmfwhd@ewhain.net
†† 비 회 원 : 이화여자대학교 정보통신학과
shoutoi@ewhain.net
††† 비 회 원 : 이화여자대학교 정보통신학과 교수
hlim@ewain.ac.kr
논문접수 : 2004년 12월 1일
심사완료 : 2005년 3월 7일

매치하는 프리픽스를 찾는 longest prefix matching 작업을 수행하여야 한다. 더욱이 인터넷 대역이 커지고 라우터가 처리해야 할 양이 많아지면서 어드레스 검색은 더욱 더 어려운 작업이 되었고, 라우터에서의 병목점으로 작용하게 되었다. 이로서 검색속도, 메모리 사이즈, IPv6 어드레스로의 확장성, pre-processing 여부, 업데이트 측면에서 보다 효율적인 IP 어드레스 검색 구조가 요구된다 할 것이다[1].

본 논문에서는 크기가 작은 메모리를 사용하면서도 검색 성능이 좋고 일반적인 프로세서로 처리 가능한 소프트웨어에 기반한 IP 어드레스 검색 구조를 제안한다. 라우팅 테이블을 여러 개의 밸런스 트리들로 구성하여 하나의 SRAM에 저장하고 멀티웨이 검색을 수행하는 방식이다. 제안하는 방식은 3개의 키를 두고 4개의 범위에서 비교하는 4-way 검색 방식을 적용함으로써 N이 전체 프리픽스의 개수라고 할 때 검색 횟수가 $O(\log_4 N)$로 줄어들어 검색 속도가 개선된다. 그리고 한 엔트리에 여러 개의 프리픽스를 저장해서 캐쉬 라인 사이즈를 활용할 수 있고, 여기에 범위 테이블을 추가하여 검색 범위를 더욱 좁힘으로써 검색 횟수를 더욱 줄일 수 있다. 또한 IPv4 어드레스와 IPv6 어드레스를 하나의 메모리에 저장해 IPv4 어드레스에 대해서는 4-way 검색을, IPv6 어드레스에 대해서는 바이너리 검색을 적용함으로써 IPv6 어드레스로의 확장에도 매우 용이한 구조이다. 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 살펴보고 3장에서는 제안하는 구조에 대해 살펴본다. 4장에서는 시뮬레이션 결과를 비롯하여 기존의 소프트웨어에 기반한 방식과의 성능을 비교한 후 5장에서 결론을 맺는다.

2. 관련 연구

IP 어드레스 검색을 위한 가장 기본적인 구조로서, 그림 1에 보인 것과 같은 이진 트라이(trie) 구조가 있다. 이 구조는 간단하고 업데이트를 쉽게 할 수 있는 장점이 있으나 중간에 비어 있는 노드가 생겨 메모리가 낭비되고, 트리의 높이가 프리픽스의 최대 길이와 같게 되어 검색 속도가 느린 단점이 있다[2].

다음은 이진 트라이의 높이를 줄인 path-compressed trie 구조를 들수 있다. 하나의 차일드를 갖는 비어있는 노드들을 제거함으로써 트라이의 높이를 줄여 요구되는 메모리를 줄이고 검색속도를 향상시킨 방법이다. 그러나 이 구조는 트라이가 조밀한 경우는 이진 트라이와 비슷한 성능을 갖게 된다. 한 번에 한 비트씩이 아니라 여러 비트를 보는 multi-bit trie도 있는데 트라이의 높이가 줄어들어 검색 속도가 향상되나 여러 비트를 동시에 보기 위해 프리픽스들을 확장해야 되므로 메모리 사이즈가 늘어나게 된다. 또 다른 구조로 level-compressed trie가 있는데, path-compressed trie와 multibit trie 개념을 병합한 구조로서, 이진 트라이를 path compression 한 후 multi-bit trie로 변환시킨 구조이다. 이 구조는 검색 성능은 좋으나 요구되는 메모리 사이즈가 크고 업데이트가 쉽지 않은 단점이 있다.

다음은 큰 전달 테이블을 데이타 압축을 통하여 작은 구조로 표현한 Lulea 구조[3]가 있는데, 이 구조는 요구되는 메모리 사이즈가 작아 L1/L2 캐쉬에 저장 가능한 소프트웨어에 기반한 방식이다. 그러나 메모리 접근 횟수가 많고, pre-processing이 복잡해서 업데이트가 쉽지 않으며, 큰 라우팅 테이블로의 확장성이 좋지 않은 단점이 있다.

가장 최근에 발표된 구조로서 이진 프리픽스 트리[4]가 있다. 그림 2에 보인 바와 같이 비어있는 노드를 포함하지 않는 트리가 구성되어 메모리 낭비를 막은 매우 우수한 구조라고 할 수 있다. 길이가 다른 프리픽스 간에 크기를 비교하는 새로운 정의를 사용하여 바이너리 검색을 가능하게 하였는데, 크기 비교를 위해 사용된 정의는 다음과 같다.

정의 1. 두 프리픽스의 길이가 같으면 값을 비교한다. 길이가 다른 경우 짧은 프리픽스의 길이까지의 값을 비교한다. 짧은 길이까지의 값이 같은 경우에는 긴 길이 프리픽스의 그 다음 비트가 1이면 긴 길이 프리픽스가 크고, 0이면 긴 길이 프리픽스가 작다.

정의 2. 두 프리픽스가 동일하거나 두 프리픽스 중 짧은 프리픽스의 길이까지 비교해서 값이 같으면 매치한다.

정의 3. 두 프리픽스가 매치하지 않으면 디스조인트하다.

정의 4. 두 프리픽스가 매치하는 경우, 짧은 길이의 프리픽스는 긴 길이의 프리픽스의 인클로져이다.

그러나 이 구조는 인클로저들이 자신을 프리픽스로 갖는 다른 프리픽스보다 트리의 상위 노드에 위치해야 한다는 제약점을 갖으므로, 언밸러스한 트리가 형성되며 트리의 높이가 가장 긴 프리픽스 길이에 비례하여 커지게 되는 단점이 있다.

이러한 단점을 보완하기 위해 Lim[5]에서는 여러 개의 밸런스 트리를 하나의 메모리에 저장하여 각 트리마다 순차적인 바이너리 검색을 적용하는 방식을 제안하고 있는데, 그림 3은 이 구조를 나타내고 있다. 언밸러스한 하나의 트리로부터 인클로저를 가지는 프리픽스들을 분리하여 인클로저의 서브트리 형태로 여러 개의 밸런스 트리를 구성함으로써 트리의 높이를 줄여 검색 성능을 향상 시켰다. 트리의 각 노드들은 차일드로 가는 포인터를 저장함 없이 메모리의 인덱스를 사용하여 바

이너리 검색을 수행하므로, 캐쉬에 저장 가능한 작은 메모리를 사용하고, 메모리의 사용 효율이 매우 우수한 구조이다. 또한 라우팅 테이블의 크기가 프리픽스 길이가 아닌 프리픽스 개수에 따라 결정되므로 IPv6로의 전환이 용이하다[5]. 그러나 바이너리 검색을 하기 때문에 메모리 접근 횟수가 여전히 많고, 캐쉬 라인 하나에 프리픽스 한 개와 서브트리 포인터 (저장된 프리픽스가 인클로저인 경우)만 저장하기 때문에 캐쉬 라인 사이즈가 낭비된다는 단점이 있다[6].

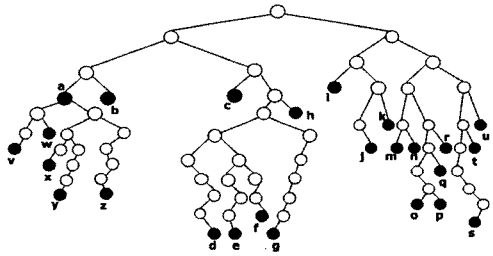

그림 1 이진 트리(trie)

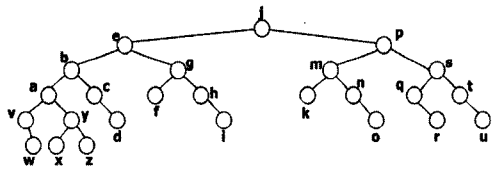

그림 2 이진 프리픽스 트리(tree)

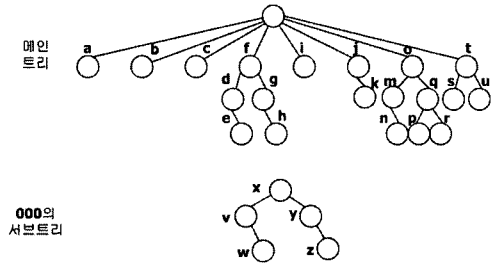

그림 3 Lim 이진 트리(tree)

3. 제안하는 구조

3.1 프리픽스 분류

라우팅 테이블의 모든 프리픽스는 이진 프리픽스 트리 방식의 정의를 사용하여 인클로저와 디스조인트 프리픽스로 분류된다. 전체 프리픽스 중 디스조인트와 인클로저 프리픽스들을 첫번째 레벨로 분류하고 인클로저의 서브트리에 들어가는 프리픽스들은 두번째 레벨로 분류된다. 두번째 레벨에서 인클로저가 또 존재하는 경우, 인클로저의 서브트리에 포함시키고 이 프리픽스들을 세번째 레벨로 분류된다. 즉, 어떤 프리픽스가 어느 인클로저의 서브트리에 속한다면 그 인클로저가 속하는 레벨보다 하나 증가한 레벨이 된다. 그림 4는 표 1의 프리픽스들을 가지고 레벨이 분류된 예를 보여준다. 예를 들어 000, 001, 0110001101은 첫번째 레벨 프리픽스이고, 000의 서브트리에 속하는 000000, 00001 등의 5개 프리픽스는 두번째 레벨임을 나타내고 있다.

표 1 프리픽스의 예

a	000	o	11010010
b	001	p	11010011
c	010	q	110101
d	0110001101	r	11011
e	0110010101	s	111000110
f	011001101	t	11101
g	0110100000	u	1111
h	0111	v	000000
i	100	w	00001
j	10101	x	0001000
k	1011	y	000101000
m	11000	z	000110101
n	11001		

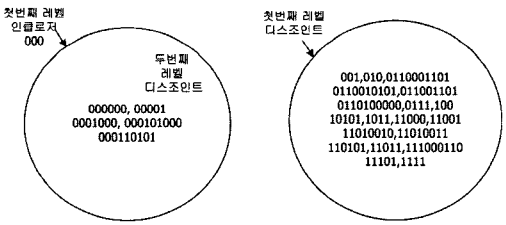

그림 4 프리픽스 분류

3.2 멀티웨이 트리

멀티웨이 검색은 한 번에 여러 개의 범위를 비교해서 검색하는 방식으로, 본 논문에서는 3개의 key를 두고 4개의 범위를 비교하는 4-way 검색 방식을 적용하였다. 이를 위해한 노드에 여러 개의 프리픽스가 저장되는데, 촘촘하게 트리가 구성되도록 프리픽스들을 선택한다. 표 1의 프리픽스들을 예로 들면, 첫 번째 레벨에 속하는 인클로저와 디스조인트인 프리픽스들 중에서 100(i), 11001(n), 11011(r)을 선택하면, 100보다 작은 범위, 100에서 11001사이의 범위, 11001에서 11011 사이의 범위, 11011 보다 큰 범위인 총 4개의 범위로 나누어진다. 여기서 가장 왼쪽에 위치하는 노드에는 100보다 작은 범위에 속하는 8개의 프리픽스 중 4번째인 0110001101(d), 7번째인 0110100000(g), 8번째인 0111(h)이 선택되어 저장되고 또 4개의 범위로 나누어지게 된다. 이와 같은 방법으로 멀티웨이 트리를 구성하게 되면 그림 5와 같다. 그림 1, 그림 2, 그림 3, 그림 5를 비교해보면 이진

트리(trie)의 높이는 가장 긴 프리픽스의 길이인 W로 4개의 트리 중에서 제일 크고 이진 프리픽스 트리[4], Lim[5] 이진 트리, 멀티웨이 트리 순으로 트리의 높이가 줄어들고 있음을 알 수 있다. 그 중 제안하는 멀티웨이 트리가 높이가 가장 낮아 검색횟수가 가장 줄어들 것임을 알 수 있다.

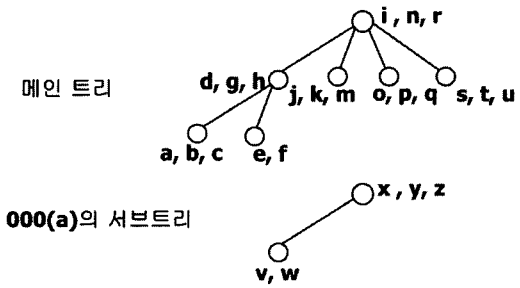

그림 5. 제안하는 멀티웨이 트리

3.3 테이블 구성

IPv4 어드레스는 3.1에서 제시한 방법으로 프리픽스를 분류한 후 4-way search 방식을 적용하기 위해 하나의 엔트리에 3개의 프리픽스를 저장한다. 각각의 해당하는 범위에 프리픽스가 있으면 child valid를 1로, 그렇지 않으면 0으로 저장하고 인클로저 프리픽스이면 서브트리 포인터를 갖는다. IPv6 어드레스는 Lim에서 저장한 방식을 사용하여 하나의 엔트리에 한 개의 프리픽스가 저장되고 인클로저이면 서브트리를 가리키는 포인터와 개수를 갖는다.

메인 테이블의 처음 부분에 첫번째 레벨의 디스조인트와 인클로저 프리픽스로 구성되어 있는 메인 트리가 저장된 후, 첫번째의 서브트리에 해당하는 두번째 레벨의 디스조인트와 인클로저 프리픽스들이 저장된다. 그 다음으로 세번째 레벨의 프리픽스들이 저장되는 방식이 반복되어 메인 테이블이 만들어진다.

여기에 범위 테이블을 사용해 검색 범위를 줄여 검색 성능을 더욱 향상시켰다. IPv4 어드레스와 IPv6 어드레스에 대해 범위 테이블이 각각 있고, 프리픽스의 처음 8비트가 범위 테이블의 인덱스로 사용된다. IPv4 어드레스의 경우에는 해당하는 프리픽스들이 저장된 처음 위치만 지정되어 있고, IPv6 어드레스의 범위 테이블에는 처음 위치와 개수가 저장된다.

메인 테이블에는 처음 8비트를 저장할 필요없이 8비트를 제외한 나머지 부분만 저장하면 되는데, 그림 6에서는 처음 3비트를 가지고 범위 테이블을 구성한 경우의 메인 테이블을 보여주고 있다. 메인 테이블에 저장되지 않는 처음 3비트는 괄호로 표현하였고 업데이트에 대비해 비워둔 엔트리는 생략하였다. 그림 7은 범위 테이블을 사용한 구조를 나타내는 멀티웨이 트리이다.

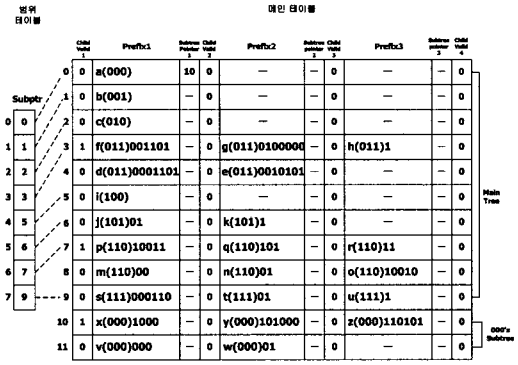

그림 6 제안하는 구조의 프리픽스 저장 방식

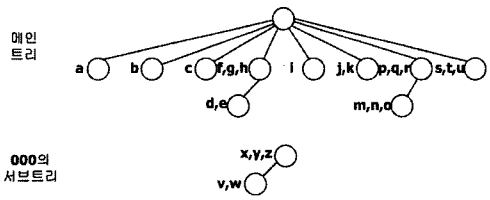

그림 7 제안하는 구조의 멀티웨이 트리

3.4 검색 방법

제안하는 구조의 검색은 범위 테이블에서 시작되는데, 먼저 IPv4 어드레스에 대해 살펴보면 다음과 같다. 어드레스의 처음 8비트를 보고 범위 테이블에서 위치 정보를 얻게 되면, 메인 테이블로 가서 3개의 프리픽스와 매치하는지 비교한다. 매치하지 않는 경우에는 4개의 범위 중 해당하는 범위를 찾아 다시 검색을 수행한다. 매치하는 경우는 디스조인트 프리픽스와 매치할 때와 인클로저 프리픽스와 매치할 때의 두가지 경우로 나눌 수 있는데, 매치한 프리픽스가 디스조인트이면 검색이 끝나고 해당하는 엔트리의 아웃 포트 포인터를 출력한다. 매치한 프리픽스가 인클로저이면 아웃 포트 포인터를 저장한 후 서브트리 포인터가 가리키는 곳으로 이동하여 다시 검색을 수행한다. 서브트리에서 새로운 프리픽스와 매치하게 되면 아웃 포트 포인터를 업데이트한다. 그림 6을 예로 들면 들어오는 주소가 00010000인경우 처음 3비트가 000이므로 메인 테이블의 주소 0으로 가서 검색을 수행한다. 프리픽스 000과 매치하므로 해당하는 아웃 포트 포인터를 저장한 후 서브포인터가 가리키는 주소로 가게 된다. 3개의 프리픽스와 매치하는지 비교하는데, 프리픽스 x와 새로이 매치하고 프리픽스 x의 서브 포인터가 0이므로 아웃 포트 포인터를 업데이트하여 출

력하고 검색을 종료한다.

들어오는 패킷의 어드레스가 IPv6일 경우도 위의 IPv4 어드레스를 검색하는 방식과 마찬가지로 범위 테이블에서 위치 정보와 개수를 얻어 바이너리 검색을 수행한다. 매치한 프리픽스가 디스조인트이면 검색을 마치고, 인클로저이면 서브트리 포인터가 가리키는 위치와 개수를 얻어 다시 바이너리 검색을 수행한다.

3.5 업데이트

제안하는 구조는 incremental 업데이트가 가능하다. 새로운 프리픽스를 추가하거나 제거할 때 먼저 범위 테이블에서 위치 정보를 얻어 메인 테이블로 간다. IPv4 어드레스인 경우는 검색 과정과 마찬가지로 세 개의 프리픽스와 매치하는지 비교해, 매치하지 않는 경우에는 해당하는 범위에 있는 프리픽스들과 비교해서 크기 순으로 정렬해 다시 저장하고 해당하는 범위에 프리픽스가 없는 경우에는 비워있는 엔트리에 저장된다. 매치할 때는 두가지 경우가 있는데, 엔트리에 있는 프리픽스가 인클로저인 경우에는 서브포인터가 가리키는 곳으로 가서 매치하는지 비교하여 저장한다. 추가하는 프리픽스가 인클로저인 경우는 그 자리에 저장되고 원래 저장되어 있던 프리픽스는 서브트리가 가리키는 곳으로 가서 저장된다. IPv6 어드레스를 추가하거나 제거하고자 할 때도 앞의 업데이트 방법을 동일하게 적용할 수 있다.

3.6 엔트리 구조

IPv4 어드레스를 저장할 때의 엔트리 구조는 그림 8과 같다. 메인 테이블에는 4개의 child valid 비트와 3개의 엔트리를 갖는다. 각각의 엔트리는 프리픽스, 프리픽스의 길이, 서브트리가 존재하는지의 여부를 표시하게 위한 1 비트와, 서브트리 포인터, 그리고 아웃 포트 포인터를 저장하는 필드를 갖는다. 범위 테이블에는 메인 테이블의 인덱스가 저장된다.

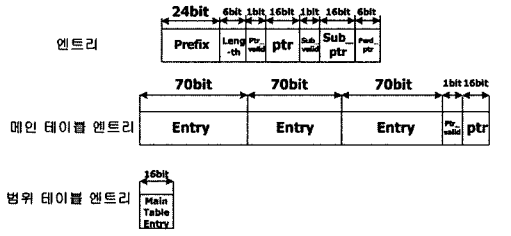

그림 8 엔트리 구조

4. 시뮬레이션 결과와 성능 비교

본 논문에서는 실제 사용된 라우팅 데이타를 가지고 C언어를 사용하여 성능을 실험하였다. 이 데이타들은 MAE-WEST, MAE-EAST, ADDS, PAC BELL, FUNET 라우터에서 수집된 결과이다. 표 2를 보면 라우팅 엔트리 수에 따르는 이진 프리픽스 트리, Lim의 구조와 제안하는 구조의 메모리 사이즈를 비교하고 있는데, 제안하는 구조가 가장 작은 메모리를 요구함을 알 수 있다.

표 2 메모리 사이즈 비교

	라우팅 엔트리 수	이진 프리픽스 트리	Lim 이진 트리	제안하는 구조
MAE-WEST(1)	14553	130.9K	105.5K	99.6k
MAE-WEST(2)	14937	134.4K	108.29K	102.7k
AADS	20204	181.8K	146.47K	137.5k
PAC BELL	20519	184.7K	148.76K	139.4k
MAE-WEST(3)	29584	273.7K	214.48K	211.5k
MAE-EAST(1)	37993	351.4K	275.4K	261.3k
MAE-EAST(2)	39464	365.0K	286.1K	270.1k
FUNET	40905	378.4K	296.5K	282.8k

그림 9와 그림 10은 평균 메모리 검색횟수와 최대 메모리 검색 횟수를 비교한 그래프로, 멀티웨이 검색 방식을 적용함으로써 평균 메모리 접근 횟수와 최대 메모리 접근 횟수가 약 1/3로 줄어드는 것을 볼 수 있고 이로 인해 검색 성능이 향상됨을 알 수 있다. 표 3은 약 4만개의 엔트리에 대해 기존의 소프트웨어에 기반한 방식과의 성능을 비교한 표이다. 본 논문에서 제안하는 구조는 메모리 사이즈가 다른 방식에 비해 비교적 적게 요구되고 가장 적은 수의 메모리 접근으로 어드레스 검색이 가능해 검색성능이 가장 좋은 것을 알 수 있다.

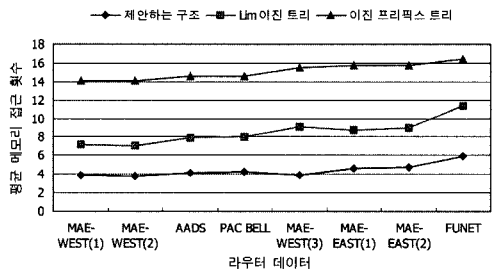

그림 9 평균 메모리 접근 횟수 비교

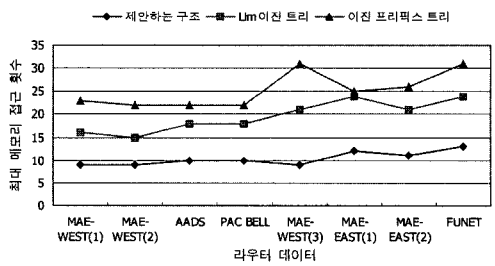

그림 10 최대 메모리 접근 횟수 비교

표 3 제안하는 구조와 기존 소프트웨어에 기반한 방식의 비교

	메모리 접근횟수			메모리 크기
	평균	최소	최대	
Lulea	Not Available	5	40	160KB
이진 프리픽스 트리	16.4	14	31	342.5KB
Lim 이진 트리	11.4	1	24	296.5KB
제안하는 구조	6.1	1	13	282.8kB

5. 결 론

본 논문에서는 IP 어드레스 검색을 위한 멀티웨이 검색 구조를 제안하였다. 제안하는 구조는 4-way 검색을 수행함으로써 메모리 접근 횟수를 줄여 메모리 검색횟수가 $O(\log_4 N)$으로 줄어들게 되고 범위 테이블을 사용하여 검색 성능을 더욱 향상시킨 구조이다. 또한 incremental 업데이트가 가능하다. 향후에 IPv4와 IPv6 어드레스가 공존하게 되었을 때, 라우팅 테이블의 대부분의 양을 차지하게 될 IPv4 어드레스에 대해서는 4-way 검색 방식을 적용하고, IPv6 어드레스에 대해서는 바이너리 검색을 적용한다면, 어드레스 길이 특성을 살릴 뿐만 아니라, 비슷한 검색 시간을 갖게 한다는 측면에서도 매우 효율적이라 할 수 있을 것이다.

참 고 문 헌

[1] H.Jonathan Chao, "Next Generation Routers," Proceedings of the IEEE, Vol.90, No.9, pp. 1518-1558, Sep, 2002.

[2] 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.

[3] M.Degermark, A.Brodnik, S.Carlsson, S.Pink, "Small Forwarding Tables for Fast Routing Lookups," Proc. ACM SIGCOMM, pp. 3-14, 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] Hyesook Lim, Bomi Lee, Wonjung Kim "Binary Search on Multiple Small Trees for IP Address Lookup," IEEK SUMMER CONFERENCE 2004, pp. 175-178, 2004.

[6] B.Lampson, V.Srinivasan, and G.Varghese, "IP lookups using multiway and multicolumn search," Proc. IEEE INFOCOM'98, pp. 1248-1256, 1998.

[7] S. Nilsson and G. Karlsson, "IP-Address lookup using LC-tries," IEEE J. Select. Areas Commun, vol. 17, pp. 1083C92, June 1999.

[8] A-32 Intel® Architecture Software Developer's Manual, Intel, 2004.

김 원 정

2004년 2월 이화여자대학교 정보통신학과 학사. 2004년 3월~현재 이화여자대학교 정보통신학과 석사과정. 관심분야는 Router와 switch등의 Network 관련 SoC 설계, 멀티미디어 네트워크

이 보 미

2003년 2월 이화여자대학교 정보통신학과 학사. 2005년 2월 이화여자대학교 정보통신학과 석사. 2005년 3월~현재 LG 전자 디지털 미디어 연구소 MCT그룹 연구원. 관심분야는 Router와 switch등의 Network 관련 SoC 설계, 멀티미디어 네트워크

임 혜 숙

1986년 2월 서울대학교 제어계측공학과 학사. 1986년 8월 1989년 2월 삼성 휴렛팩커드 연구원. 1991년 2월 서울대학교 제어계측공학과 석사. 1996년 12월 The University of Texas at Austin, Electrical and Computer Engineering 박사. 1996년 11월~2000년 7월 Lucent Technologies Member of Technical Staff. 2000년 7월~2002년 2월 Cisco Systems Hardware Engineer. 2002년 3월~현재 이화여자대학교 공과대학 정보통신학과 조교수. 관심분야는 Router나 switch등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계