Binary Search on Multiple Small Trees for IP Address Lookup

이보미, 김원정, 임혜숙*

이화여자대학교 정보통신학과 SoC 설계 연구실
전화 : 02-3277-3403

Binary Search on Multiple Small Trees for IP Address Lookup

BoMi Lee, Won-Jung Kim, HyeSook Lim*

SoC Design Lab
Information Electronics Engineering , Ewha Womans University
*E-mail : hlim @ ewha.ac.kr

Abstract

This paper describes a new IP address lookup algorithm using a binary search on multiple balanced trees stored in one memory. The proposed scheme has 3 different tables; a range table, a main table, and multiple sub-tables. The range table includes 2^8 entries of 22 bits wide. Each of the main table and sub-table entries is composed of fields for a prefix, a prefix length, the number of sub-table entries, a sub-table pointer, and a forwarding RAM pointer. Binary searches are performed in the main table and the multiple sub-tables in sequence. Address lookups in our proposed scheme are achieved by memory access times of 11 in average, 1 in minimum, and 24 in maximum using 267 Kbytes of memory for 38,000 prefixes. Hence the forwarding table of the proposed scheme is stored into L2 cache, and the address lookup algorithm is implemented in software running on general purpose processor. Since the proposed scheme only depends on the number of prefixes not the length of prefixes, it is easily scaled to IPv6.

I. 서론

어드레스 검색은 목적지 어드레스의 네트워크 부분인 프리픽스와 일치하는 엔트리를 찾아 적절한 출력포트를 찾는 것이다. broadband access 기술이 발전하면서 라우터에서 패킷을 처리해야 하는 시간이 점점 짧아지게 되고 빠른 어드레스 검색 속도가 요구되었다. classful 어드레싱 방식은 IP 어드레스를 효율적으로 사용할 수 없고 라우팅 테이블이 무한히 커진다는 단점으로 인하여 CIDR(classless Interdomain Routing)방식으로 바뀌게 되었다. 이로 인해 어드레스 aggregation 이 가능하여 어드레스를 효율적으로 사용할 수 있고 테이블의 크기가 증가됨을 방지할 수 있게 되었다. 그러나 들어오는 패킷의 프리픽스가 고정되어 있지 않기 때문에 일치하는 여러 엔트리 중 가장 긴 프리픽스를 찾아야 하는 longest prefix matching 을 해야 한다. 이로 인해 테이블 업데이트, 메모리 검색 횟수, 메모리 접근 횟수, pre-processeing 측면에서 효율적인 어드레스 검색 알고리즘이 요구된다.

본 논문에서는 여러 개의 밸런스 트리를 하나의 메모리에 저장하고 바이너리 검색을 하는 소프트웨어에 기반한 새로운 어드레스 검색 구조를 제안한다. Ⅱ장에서는 기존의 어드레스 검색 구조를 살펴보고 Ⅲ장에서는 제안하는 구조에 대해 살펴본다. Ⅳ장에서는 다른 검색 구조들과 성능 비교를 하고 마지막으로 Ⅴ장에서는 결론을 맺는다.

II. 기존의 방식들

트리(Trie) 구조는 프리픽스 간의 관계를 알기 쉽게 표현한 데이터 구조이다. Binary Trie 는 루트에서부터 검색이 시작되며 간단하고 업데이트를 비교적 쉽게 할 수 있으나, 빈 노드가 생겨 메모리가 낭비되고 최대 검색횟수가 O(W)인 단점이 있다.[2] SFT[3]는 큰 전달 테이블을 compact 하게 표현해 소프트웨어에 기반해서 빠른 검색이 가능한 구조이다. 그러나 캐쉬에 의존하므로 큰 테이블 사이즈에는 적합하지 않고 incremental update 가 어렵다. Yazdani[4]는 binary Trie 에서의 단점을 보완해 빈 노드가 없고 길이가 다른 프리픽스간의 크기비교 정의를 통해 바이너리 검색이 가능한 구조이다. 그러나 인클로저와 인클로저의 서브 프리픽스가 하나의 트리에 있기 때문에 언밸런스한 트리가 구성되어 최대 메모리 검색 횟수가 O(W)이고, 각 노드 당 왼쪽과 오른쪽 포인터를 다 기억해야 한다. Lim[5]은 이러한 단점을 보완하기 위해 디스조인트한 프리픽스들을 메인트리로 두고, 인클로저와 인클로저의 서브프리픽스는 메인트리에서 분리해서 독립된 트리로 만들어 검색공간을 줄인 구조를 제안하였다. 파이프라인을 적용해 한번의 메모리 접근으로 검색이 가능하나 SRAM 의 사이즈가 프리픽스 분포에 따라 결정되는 단점이 있다.

본 논문에서는 일반 프로세서로 처리가능하고 작은 메모리를 사용해 양쪽의 포인터를 기억할 필요없이 메모리의 인덱스를 저장해서 바이너리 검색이 가능한 구조를 제안한다.

III. 제안하는 구조

각 프리픽스들은 Yazdani[4]에서 정한 정의로 인클로저와 디스조인트로 나눌 수 있고, 어떤 인클로저의 서브 프리픽스들 중 인클로저가 있으면 이 인클로저는 다음 레벨의 인클로저가 된다. 그림 1 에서 보면 01 은 첫번째 레벨의 인클로저이고, 01 의 서브 프리픽스들인 0100,011010,0110010 등은 두번째 레벨로 들어가는데, 여기서 0100 이 인클로저이므로 두번째 레벨의 인클로저가 되고 0100 의 서브프리픽스들은 세번째 레벨로 들어가게 된다.

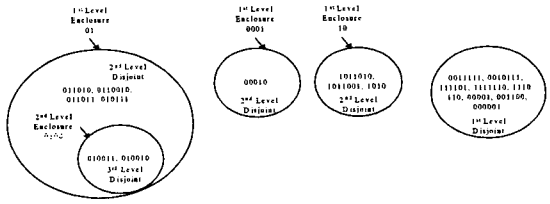

그림 1. 프리픽스 분류

Range table 에는 서브포인터와 개수를 가지고 있다. 서브포인터는 첫 번째 레벨의 인클로저와 메인트리의 디스조인트 프리픽스들의 앞의 8bit 으로, Main table 의 인덱스로 사용된다. Main table 은 디스조인트 프리픽스와 인클로저 프리픽스가 앞의 8bit 이 잘라져서 저장된다. 디스조인트 프리픽스일 때는 서브포인터와 개수가 없고, 인클로저 프리픽스일 때는 서브포인터와 개수가 저장된다. 첫 번째 레벨의 프리픽스가 다 저장되고 나면 업데이트를 고려해 어느 정도 공간을 비워놓고 두 번째 레벨의 프리픽스도 첫 번째 레벨의 프리픽스와 같은 방법으로 저장된다. 다음 레벨에 대해서도 앞과 같은 방법으로 저장된다. 제안하는 구조는 Range table 로 검색범위가 줄어들어 메모리 검색 횟수를 줄일 수 있고, Main table 에서는 프리픽스가 8bit 을 자른 24bit 만 저장하면 되므로 전체 메모리 사이즈도 줄어드는 장점이 있다. 그림 2 에서는 그림 1 의 prefix 들을 예로 하여 2bit 으로 구성한 Range table 과 Main table 을 나타내고 있다.

제안하는 구조로 검색을 하려면, 먼저 Range table 로 가서 프리픽스의 앞의 8bit 을 보고 해당하는 엔트리

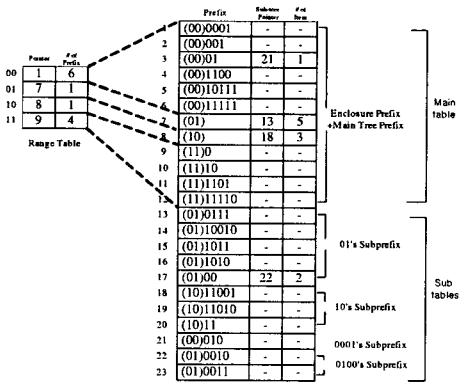

그림 2. 프리픽스 저장 방식

로 가서 검색범위의 정보인 Main table 포인터와 개수를 얻는다. 개수가 0 이라면 검색이 끝나고 디폴트 포워딩 램 포인터를 출력한다. 그렇지 않으면 Enbit table 포인터가 가리키는 메모리의 인덱스부터 개수까지 바이너리 검색을 하면서 매치하는 프리픽스를 찾는다. 매치한 프리픽스가 서브포인터와 개수를 가지고 있지 않은 디스조인트 프리픽스라면 해당하는 포워딩 램 포인터를 출력하고 검색이 끝난다. 매치하는 프리픽스가 서브포인터와 개수를 가지고 있는 인클로저 프리픽스라면, 우선 포워딩 램 포인터를 저장한다. 그 다음 서브포인터가 가리키는 메모리의 인덱스로 가서 개수만큼 바이너리 검색을 한다. 매치하는 것이 없으면 검색이 끝나고 저장되어 있던 포워딩 램 포인터를 출력한다. 매치하는 경우에 디스조인트 프리픽스라면 검색을 끝내고 해당하는 포워딩 램 포인터를 출력하고, 인클로저 프리픽스라면 우선 저장되어 있는 포워딩 램 포인터를 업데이트시키고, 해당하는 인덱스부터 바이너리 검색을 한다.

업데이트는 업데이트 할 프리픽스를 가지고 먼저 Range table 을 보고 해당하는 검색 범위를 찾는다. 그 범위 안에 있는 프리픽스들과 디스조인트하다면 크기 비교 후 해당하는 엔트리에 앞의 8bit 을 자른 프리픽스를 넣고 Main table 의 정보를 업데이트한 후, Range table 의 정보를 업데이트한다. 만약, 업데이트 할 프리픽스가 그 범위 안에서의 프리픽스의 인클로저라면, 업데이트 할 프리픽스는 첫 번째 레벨의 해당하는 엔트리에 저장되고 이 프리픽스의 서브프리픽스들은 두 번째 레벨로 내려가게 된다. 그리고 Range table 을 업데이트 한다. 마지막으로 업데이트 할 프리픽스가 그 범위 안에서의 프리픽스의 서브프리픽스라면 해당하는 서브프리픽스들의 검색범위로 가서 크기를 비교한 후 저장한 후 Range table 을 업데이트한다.

엔트리 구조는 그림 3 에서 볼 수 있다. Range table 엔트리에는 해당하는 범위가 시작하는 서브포인터와 개수가 저장되어 있다. Main table 의 엔트리에서는 인클로저인 경우 서브트리가 시작하는 서브포인터와 개수를 가지고 있다.

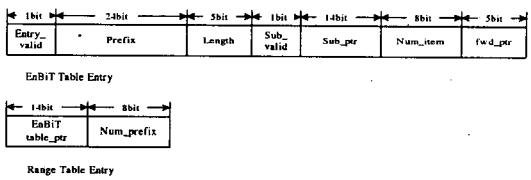

그림 3. 엔트리 구조

IV. 성능 비교

본 논문에서 제안한 구조의 성능을 C 언어를 사용하여 실험하였다. 그림 4 과 그림 5 을 보면 [4]에서 제안한 트리와 Range table 을 사용하지 않은 Pure binary 와 Range table 을 사용한 8+binary 의 메모리 사이즈를 비교하였을 때 8+binary 방식이 가장 작은 것을 볼 수 있고 평균 메모리 검색 횟수도 4 만여개의 라우팅 엔트리를 갖는 라우팅 테이블에 대해 [4]에서 제안한 구조는 평균 16.5 번, 본 논문에서 제안한 8+binary 구조는 평균 11.3 번으로 나타났다.

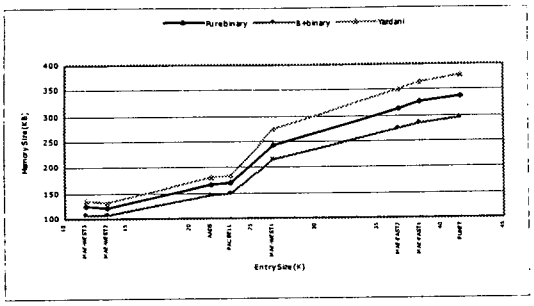

그림 4. [4]와 제안하는 구조의 메모리 사이즈 비교

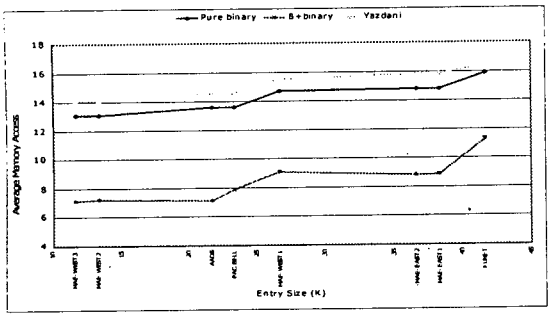

그림 5. [4]와 제안하는 구조의 메모리 access 횟수

표 1 은 소프트웨어에 기반한 방식들과 본 논문에서 제안한 구조를 비교하였다. 제안하는 구조가 작은 메모리 사이즈를 요구하며 평균 메모리 검색 횟수도 가장 작은 것을 볼 수 있다. 그리고 267.2Kbyte 의 메모리 사이즈가 필요해 L2 cache 에 저장할 수 있다. 여기서 Range table 의 사이즈는 0.7Kbyte 로 L1 Cache 에 저장 가능하므로 전체 메모리 사이즈에서 제외되었다.

		SFT [3]	Yu [6]	Yazdani [4]	Proposed Scheme
# of memory access	Avg	Not available	25.6	16.4	11.3
	Min	5	8	14	1
	Max	40	104	31	24
Storage		160KB	1500 KB	342.5 KB	267.2 KB
Pre-processing		Required	Not Required	Not Required	Not Required
Incremental Update		Not Possible	Possible	Not Possible	Possible

표 1. 소프트웨어에 기반한 방식 비교

V. 결론

본 논문에서는 IP 어드레스 룩업을 위한 새로운 binary search 알고리듬을 제안한다. 제안된 구조는 하나의 routing table 을 위하여, 여러개의 balanced tree 를 구성하고, 각각의 tree 에 대해 binary search 를 적용하는 방식이다. 라우터를 통과한 실제 데이타를 사용하여 실험하여 본 결과, 약 4 만여개의 prefix 를 저장하기 위하여 267 KByte 의 메모리를 사용하였으며, 요구되는 검색 횟수는 평균 11 번으로 매우 우수한 성능을 보였다. 제안된 구조는 저장되어질 프리픽스의 갯수에 따라 요구되는 메모리 크기및 메모리 엑세스 횟수가 결정되므로, 128 bit 어드레스를 갖는 IPv6 로의 확장이 용이하며, general purpose CPU 를 사용하여 software 방식으로 구현될 수 있다.

References

[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 and Bomi Lee, "A New Pipelined Binary Search Architecture for IP Address Lookup", Proc. IEEE HPSR2004, pp.86-90, Apr. 2004

[6] Daxiao Yu, Brandon C. Smith and Belle Wei, "Forwarding Engine for Fast Routing Lookups and Updates", Proc.IEEE GLOBECOM'99, pp.1556-1564, 1999