논문 04-29-11B-3 한국통신학회논문지 '04-11 Vol.29 No.11B

IP 주소 검색을 위한 가중 이진 프리픽스 트리

정회원 임 창 훈*, 임 혜 숙**, 준회원 이 보 미**

Weighted Binary Prefix Tree for IP Address Lookup

Changhoon Yim*, Hyesook Lim** *Regular Members* Bomi Lee** *Associate Members*

요 약

IP 주소 검색은 인터넷 라우터의 필수적인 기능의 하나로서, 라우터 전체의 성능을 결정하는 중요한 요소이다. 소프트웨어에 기반한 IP 주소 검색 방식의 성능 평가 기준 중 가장 중요한 것은 라우터의 처리 속도를 보장해 주는 의미를 갖는 최대 메모리 접근 횟수이다. 이진 프리픽스 트리 방식(BPT)은 최대 메모리 접근 횟수에 있어서 기존의 다른 소프트웨어에 기반한 방식 중 우수한 방식이지만, 트리의 구조가 불균형적이 되는 단점이 있다. 본 논문에서는 기존의 BPT 방식의 트리 생성 과정에 가중치 개념을 추가하여, 완전 균형 트리에 매우 근접하는 트리를 생성하는 가중 이진 프리픽스 트리 (WBPT) 방식을 제안한다. 제안하는 WBPT 방식은 기존의 소프트웨어에 근거한 방식들에 비교하여 최대 메모리 접근 횟수에 있어서 가장 적은 성능을 보인다. 또한 3만 개 정도의 프리픽스에 대해서 L2 캐쉬에 저장이 가능한 정도의 작은 메모리 크기를 요구하고, 프리픽스의 추가, 삭제가 용이하므로 실제적인 라우터의 IP 검색을 위하여 사용될 수 있는 방식이다.

Key Words : IP address lookup; longest prefix matching; binary prefix tree; weighted binary prefix tree; balanced tree; enclosure prefix; enclosed prefix

ABSTRACT

IP address lookup is one of the essential functions on internet routers, and it determines overall router performance. The most important evaluation factor for software-based IP address lookup is the number of the worst case memory accesses. Binary prefix tree (BPT) scheme gives small number of worst case memory accesses among previous software-based schemes. However the tree structure of BPT is normally unbalanced. In this paper, we propose weighted binary prefix tree (WBP) scheme which generates nearly balanced tree, through combining the concept of weight to the BPT generation process. The proposed WBPT gives very small number of worst case memory accesses compared to the previous software-based schemes. Moreover the WBPT requires comparably small size of memory which can be fit within L2 cache for about 30,000 prefixes, and it is rather simple for prefix addition and deletion. Hence the proposed WBPT can be used for software-based IP address lookup in practical routers.

I. 서 론

온라인 게임, 멀티미디어 스트리밍 등의 다양한 응용 프로그램이 개발되고 asymmetric digital subscriber line(ADSL), hybrid fiber coaxial cable(HFC)과 같은 인터넷 접근 기술의 발달로 네

* 건국대학교 정보통신대학 인터넷미디어공학부 (cyim@konkuk.ac.kr), ** 이화여자대학교 공과대학 정보통신학과 SoC 설계 연구실
논문번호 : 2004-07-117, 접수일자 : 2004년 7월 2일
* 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).

트워크 트래픽이 매우 빠르게 증가하고 있어 라우터에서의 고속 패킷 처리가 전체 인터넷 성능의 관건이 되고 있다[1].

라우터에서 고속 패킷 처리가 어려운 이유는 네트워크 다양화에 따라 라우팅 테이블의 크기가 급속히 증가하였으며, 링크 테크놀러지의 발달로 패킷의 도착율이 크게 증가한 상황에서, 들어오는 모든 패킷을 실시간으로 처리하여야 하기 때문이다.

인터넷 프로토콜 (Internet protocol - IP) 주소는 네트워크 파트(network part)와 호스트 파트(host part)의 계층 구조를 갖는데, 과거의 클래스를 갖는 어드레싱 (classful addressing) 방식에서는 IP 주소의 네트워크 파트, 즉 프리픽스 (prefix) 길이를 8, 16, 24로 고정시켜 사용하여 exact matching에 의한 주소 검색(address lookup)이 가능하였다. 그러나 제한된 자원인 IP 주소가 낭비되는 단점이 있어 이를 해결하기 위해 클래스를 갖지 않는 어드레싱 (classless inter-domain routing - CIDR) 방식이 등장하였다. CIDR은 네트워크 파트가 임의의 길이를 가질 수 있으므로, 네트워크에 붙는 호스트의 개수에 따른 다양한 크기의 네트워크 구성을 가능하게 하며, 또한 복수 개의 네트워크를 묶어 하나의 커다란 네트워크로 표현함으로써 백본 라우터(backbone router)에서 라우팅 테이블의 크기가 증가하는 것을 방지할 수 있는 장점이 있다. 그러나 CIDR 개념 하에서의 패킷은 네트워크의 계층 구조에 따라 여러 개의 프리픽스와 일치할 수 있으므로, 가장 정확한 네트워크를 찾기 위해, 입력 패킷의 목적지 주소와 가장 길게 매치하는 엔트리를 찾아야 하며 이를 longest prefix matching (LPM) 이라 한다[2]. LPM을 사용한 주소 검색은 과거의 <값>을 비교하는 일차원 검색에서, <값, 길이>를 비교하는 2차원 검색으로 전환 된 점에 그 어려움이 있다고 할 것이다.

IP 주소 검색을 수행하는 LPM을 위한 여러 가지 알고리즘과 구조가 제안되어 왔는데, 하드웨어에 기반한 방식과 소프트웨어에 기반한 방식으로 나눌 수 있다. 하드웨어에 기반한 방식은 백본 라우터 등과 같이 패킷의 고속 처리가 관건이 되는 곳에서 주로 사용되며, 패킷 포워딩을 위한 전용 하드웨어를 ASIC으로 구현하여 사용하는 방식이다. 소프트웨어에 기반한 방식은 소규모 엑세스 라우터에서 주로 사용되는 방식으로, 패킷 포워딩 알고리듬을 소프트웨어로 프로그램하여, CPU가 패킷 포워딩을 담당하도록 하는 방식이다. 저가의 스위칭 장비에 IP 어드레스 검색 기능을 추가하여 엑세스 라우터를 대치하고자 하는 노력들도 행하여지고 있는데, 이런 경우에도 낮은 가격으로 구현하기 위하여 고속의 하드웨어가 아닌, 소프트웨어가 처리하도록 설계된다.

IP 주소 검색 구조의 성능 평가 기준 중 가장 중요한 것은 두 방식 모두 처리 속도의 척도가 되는 메모리 접근(memory access) 횟수이다. 소프트웨어에 기반한 대부분의 기존의 방식들은 라우트의 개수가 아닌, 프리픽스의 길이에 따라 메모리 엑세스 횟수가 결정되는 방식이다. 라우팅 테이블을 저장하기 위해 필요로 하는 메모리의 크기도 중요한 요소이고, 새로운 네트워크 정보를 추가하거나 쓰지 않는 네트워크 정보 삭제의 용이성, 라우팅 엔트리 갯수의 확장성이나 IPv6로의 확장 가능성도 그 다음으로 중요한 척도가 된다.

본 논문에서 제안하는 방식은 소프트웨어에 기반한 방식으로, 메모리 접근 횟수에 있어 최소 한계(lower bound)에 거의 근접한 성능을 보이며, 비교적 적은 양의 메모리를 필요로 하는 방식이다. 그리고 이진 프리픽스 트리와 마찬가지로 전 처리(pre-processing) 과정을 필요로 하지 않기 때문에, 프리픽스의 추가 삭제가 용이하다.

본 논문의 구성은 다음과 같다. II 장에서는 IP 주소 검색을 위하여 기존에 제안된 방식들에 대해 간단히 살펴보고, III 장에서는 제안하는 방식을 설명한다. IV 장에서는 실제 라우터를 통과한 데이터를 사용하여 시뮬레이션 한 결과를 보여주고 다른 방식들과의 성능을 비교한다. 마지막으로 V 장에서 결론을 맺는다.

Ⅱ. 기존의 IP 주소 검색 방식

하드웨어에 기초한 방식은 주로 ternary content addressable memory (TCAM)나, 해싱(hashing)을 사용해 어드레스 룩업을 위한 전용 엔진을 구현하여 패킷 포워딩을 위해 특별히 제작된 칩안에 내장하는 방식이다[3],[4]. 이 경우 고속의 패킷 포워딩이 가능하나 알고리즘의 수정, 보완이 어렵다는 단점이 있다. 소프트웨어에 기초한 방식은 라우팅 테이블을 메모리에 저장하고, 어드레스 룩업 알고리듬을 프로그램으로 구현하여 CPU가 수행하도록 하는 방식으로서, 구현이 용이하고, 수정, 보완이 편리한 반면, 고속 라우터의 경우에는 패킷의 실시간 처리가 어렵다는 단점이 있다.

1. TCAM에 기초한 방식

현재 상용화된 라우터(router)나 네트워크 계층(network layer) 스위치에서 가장 많이 사용되는 방식으로서, TCAM이라는 특별히 제작된 메모리를 사용하여, 라우팅 테이블을 저장한다. 어드레스 검색은 입력된 패킷의 목적지 주소와, TCAM에 저장된 모든 엔트리의 프리픽스들이 동시에 비교되어, 일치하는 프리픽스 중에서 가장 길게 일치하는 엔트리의 주소가 선택되는 방식이다. 이 주소는 포워딩 메모리를 가리키는데 사용되어, 포워딩 메모리로부터 출력 포트 정보와 다음 홉 (hop) 주소의 정보를 얻게 된다. 구현이 간편하고, 매우 빠른 성능을 보이지만, TCAM은 일반 SRAM에 비하여 차지하는 면적이 6배 정도 크고, 가격도 비쌀 뿐만 아니라, 전력소모가 매우 커서 패킷 포워딩을 위해 제작된 칩 안에 내장되지 않는 단점이 있다. 또한 큰 라우팅 테이블이나, IPv6로의 확장에 적합하지 않은 문제가 있다[4].

2. 해쉬에 기초한 방식

해쉬(hashing)이란 메모리 사용 효율을 높이기 위하여 긴 길이의 IP 프리픽스를 짧은 길이의 스트링으로 매핑한 후, 짧은 길이의 스트링이 가리키는 메모리의 엔트리에 프리픽스를 저장하여 라우팅 테이블을 구성하는 방식이다. 서로 다른 프리픽스가 같은 스트링으로 매핑되는 경우가 발생할 수 있는데, 이를 충돌(collision)이라 하며, 충돌을 효율적으로 처리하여 주는 것이 관건이다. Exact matching을 사용하는 링크 계층 (link layer) 주소 검색을 위해 많이 이용되어 왔으며, LPM을 이용하는 네트워크 계층 주소 검색을 위해서는 각 프리픽스 길이별로 별도의 해쉬을 수행하게 된다[5]. 해쉬에 기초한 방식으로 프리픽스 길이별로 별도의 해쉬 테이블을 만들고, 복수개의 해쉬(multiple hashing)을 적용하여 충돌을 줄인 방식[6]과, 라우팅 테이블을 4개의 그룹(group)으로 나누어 별도의 메모리에 저장한 후, 프리픽스 길이별로 긴 프리픽스로부터 순차적인 해쉬을 수행하는 그룹 해쉬(group hashing) 방식[7] 등이 제안되었다.

3. Trie에 기초한 방식

Trie는 <길이>에 대해서는 선형 검색을 적용한 방식으로서, 모든 프리픽스는 trie에서 하나의 노드에 위치하게 되며, 길이가 i 인 프리픽스는 트리의 i 번째 레벨에 존재하게 된다. Trie는 root로부터 출발하여, 프리픽스의 한 비트씩 검사하여 값이 0이면 왼쪽, 1이면 오른쪽으로 이동하는 방식이다. 어드레스 검색은 입력된 어드레스를 한 비트씩 검사해가면서, 일치하는 프리픽스를 만날 때마다 매치 정보를 업데이트하고, trie의 leaf 노드에서 종료된다[2].

표 1은 프리픽스 검색 방식을 설명하기 위하여, 본 논문에서 사용할 23개로 구성된 프리픽스 집합의 예이다. 그림 1은 표 1의 프리픽스 집합에 대하여 이진 (binary) trie를 생성한 것을 나타낸다. 그림 1에서 흰색 노드는 빈 노드를 나타내고, 검은색 노드는 프리픽스가 존재하는 노드를 나타낸다. 표 1의 프리픽스는 최대 프리픽스의 길이가 7이기 때문에, 최대 7번의 검사가 필요하며 이진 trie가 7개의 레벨(level)로 구성된다.

이진 trie는 가장 직관적인 구조이나, 프리픽스가 존재하는 노드뿐만 아니라 프리픽스가 존재하지 않는 빈 노드를 저장해야 하는데서 오는 메모리 낭비가 초래되고, 메모리 검색 횟수는 O(W) (W는 프리픽스의 최대 길이)에 비례하는 단점이 있다.

표 1. 프리픽스 집합의 예

프리픽스 number	프리픽스	프리픽스 number	프리픽스
1	000001	13	010111
2	00001	14	0110010
3	0001	15	011011
4	001100	16	011010
5	0010111	17	0100
6	0011111	18	1011001
7	01	19	1011010
8	10	20	1011
9	110	21	00010
10	1110	22	010010
11	111101	23	010011
12	1111110		

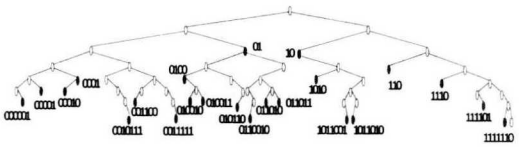


그림 1. 이진 trie

이러한 이진 trie의 단점을 해결하기 위해 여러 가지 방식들이 제안되었는데, 메모리 엑세스 횟수를 줄이기 위해, 두 개 이상의 가지를 갖도록 한 multi-bit trie[8], 하나의 가지를 갖는 비어있는 노드들을 없앤 Patricia trie 혹은 path-compressed trie[9],

이 두 가지 방식을 모두 결합한 level compressed trie[10] 등이 있다. Lulea 방식[11]은 trie를 저장하는데 있어 적은 양의 메모리를 사용하기 위해 제안된 방식으로, 프리픽스를 갖는 노드의 위치를 표시하는 비트와 노드 정보를 매우 축약하여 표현하였으나, 전 처리를 많이 요구하여 프리픽스의 추가, 삭제가 용이하지 않은 단점이 있다.

4. 이진 검색에 기초한 방식

Longest prefix match(LPM)를 수행하는 IP 주소 검색에 이진 검색을 적용하기 위한 새로운 개념의 트리(tree)가 도입되었다[12],[13]. Yazdani[12]는 길이가 다른 프리픽스(prefix)들 간의 크기 비교를 가능하게 하는 몇 가지 정의를 내렸는데, 다음과 같다.

정의 1 비교(comparison): 두 프리픽스의 길이가 같으면 숫자 값이 비교된다. 예) A=1001, B=1100인 경우, A<B. 두 프리픽스의 길이가 다른 경우 짧은 길이까지만 비교된다. 예) A=1001, B=110000인 경우, A<B. 짧은 길이까지의 값이 같은 경우, 다음 비트(bit)가 1이면 긴 프리픽스가 크고, 아니면 짧은 프리픽스가 크다. 예) A=1001, B=100110인 경우, A<B. 예) A=1001, B=100100인 경우, A>B.

정의 2 매치 (match): 하나의 프리픽스가 다른 프리픽스의 substring이면 두 프리픽스는 매치한다. 예) A=1001, B=100100인 경우, A와 B는 매치한다.

정의 3 디스조인트 (disjoint): 두 프리픽스가 매치하지 않으면 disjoint 하다. 예) A=1001, B=111인 경우, A와 B는 disjoint 하다.

정의 4 포괄 프리픽스 (enclosure prefix): 프리픽스 A를 프리픽스로 갖는 다른 프리픽스가 하나라도 존재하면 A는 포괄 프리픽스이다. 예) A=1001, B=100100인 경우, A는 B의 포괄 프리픽스이다.

그림 2에서는 위의 정의를 사용하여 표 1의 프리픽스 집합에 대하여 이진 검색(binary search)이 가능하도록 생성된 이진 프리픽스 트리(binary prefix tree - BPT)를 보여준다. 그림 2에서 검은색 노드는 포괄 프리픽스(enclosure prefix)를 나타낸다. 길이가 다른 프리픽스들 간의 비교를 가능하게 하는 비교(정의 1)에 의하여 이진 trie와 달리 어떤 프리픽스도 임의의 레벨에 존재할 수 있는 특징이 있고, 빈 노드가 존재하지 않는 장점이 있다.

그림 2에서 보면 트리의 모양이 한 쪽으로 쏠린 불균형(unbalance) 트리를 구성한 것을 알 수 있는데, 여기에는 두 가지의 이유가 있다. 첫째는 LPM의 특성상 포괄 프리픽스는 자신을 프리픽스로 하는 더 긴 프리픽스보다 상위에 위치하여야 하는 제약점 때문이고, 둘째는 이진 트리를 구성하는 과정에서 반복적으로 선택되는 root 노드를 적절하게 선택하지 못했기 때문이다. BPT 방식으로 구성하는 불균형 트리는 프리픽스의 분포 특성에 따라 메모리 검색 횟수가 최대 W번까지 가는 단점이 있다.

본 논문에서 제안하는 구조는 포괄 프리픽스에 포함된 프리픽스의 개수를 고려하여, Yazdani의 BPT의 단점인 불균형 트리의 성질을 개선한 구조로서 가중 이진 프리픽스 트리(weighted binary prefix tree - WBPT)라 명명한다.

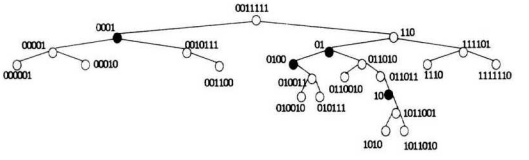


그림 2. 이진 프리픽스 트리 (BPT)

Ⅲ. 제안하는 가중 이진 프리픽스 방식

본 논문에서는 Yazdani가 제안한 정의에 가중치(weight) 개념을 도입하기 위하여 다음과 같은 새로운 몇 가지의 정의를 제안한다.

정의 5 예속 프리픽스 (enclosed prefix): 프리픽스 A가 프리픽스 B의 포괄 프리픽스라면, 프리픽스 B는 프리픽스 A의 예속 프리픽스로 정의한다. 예) A=1001이고 B=100100인 경우, B는 A의 예속 프리픽스이다.

정의 6 백 (bag): Bag은 포괄 프리픽스와 그 프리픽스의 예속 프리픽스들로 구성되어 있다.

정의 7 독립 (independent): 어떤 프리픽스가 리스트(list)에 있는 모든 다른 프리픽스와 디스조인트하다면 그 프리픽스는 독립 프리픽스이다.

Lemma: 어떤 프리픽스가 포괄 프리픽스도 아니고, 예속 프리픽스도 아니라면, 그 프리픽스는 독립 프리픽스이다.

정의 8 대표 프리픽스 (representative prefix): 포괄 프리픽스 혹은 독립 프리픽스는 대표 프리픽스이다. 예속 프리픽스는 대표 프리픽스가 아니다.

정의 9 가중치 (weight): 대표 프리픽스의 가중치는 대표 프리픽스에 포함된 프리픽스들의 개수로 정의된다. 즉, 대표 프리픽스가 포괄 프리픽스(정의 4)이면 가중치는 예속 프리픽스의 개수에 포괄 프리픽스를 고려하여 1을 더한 값이다. 만약 대표 프리픽

스가 독립 프리픽스(정의 7)이면 가중치는 1로 정의된다.

프리픽스 트리의 최대 레벨 수를 깊이(depth)라 하는데, 어떤 프리픽스 트리의 깊이를 L이라고 할 때, 그 트리에 포함되는 노드의 최대 갯수는 $2^0 + 2^1 + \cdots + 2^{L-1} = 2^L - 1$ 이다. 트리에 저장될 프리픽스의 개수를 N이라 하면 다음과 같은 관계가 성립한다.

$$2^L - 1 \geq N$$

그러므로

$$L \geq \log_2 (N+1) \quad (1)$$

$L_{\min}$을 프리픽스 트리 깊이의 최소 값이라고 한다면, $L_{\min}$은 정수 값이어야 하므로 식 (1)로 부터 다음과 같이 구할 수 있다.

$$L_{\min} = \lceil \log_2 (N+1) \rceil \quad (2)$$

여기서 $\lceil \bullet \rceil$는 ceil 연산으로서 $\bullet$ 보다 크며, 이 중 가장 작은 정수 값을 의미한다. 표 1의 프리픽스 개수 N은 23이므로 식 (2)에 의하여 $L_{\min} = 5$인데, 그림 2에 나타난 BPT는 깊이 L은 7이다.

어떤 프리픽스 트리의 깊이가 프리픽스 개수에 의하여 결정되는 최소 값과 같다면, 즉 $L = L_{\min}$이면 그 프리픽스 트리는 균형(balance) 트리로 정의한다. 프리픽스 트리의 깊이는 요구 되는 최대 메모리 엑세스(memory access) 횟수와 직접적인 관련을 갖는데, 다음 장의 실험에서 보인바와 같이 Yazdani의 BPT는 불균형적 특성 때문에 $L_{\min}$보다 훨씬 큰 깊이 L을 갖는다.

본 논문에서 제안하는 WBPT는 최소 깊이에 매우 근접하는 트리를 구성하여, Yazdani의 BPT와 비교하여 매우 적은 수의 최대 메모리 엑세스를 요구한다. 같은 수의 프리픽스에 대한 트리의 깊이가 깊어진다면, 트리는 점점 더 불균형이 됨을 의미한다. 여기에서 트리의 균형된 정도를 정량적으로 표현하기 위하여 균형도 (balance degree) D_b를 다음과 같이 정의한다.

$$D_b = \frac{W - L}{W - L_{\min}} \quad (3)$$

여기에서 W는 프리픽스의 최대 길이를 나타낸다.

식 (3)에서 균형도 D_b는 0과 1사이의 값을 갖는다. 트리가 균형 되었다면, 즉 $L = L_{\min}$이라면 이 경우 균형도는 식 (3)으로부터 $D_b = 1$이고, D_b 값이 클수록 트리가 잘 균형 된 것을 의미한다. 이와 반대로 D_b 값이 작아서 0에 가까울수록 트리가 불균형 된 것을 의미한다.

이진 프리픽스 트리를 구성하는 과정에서, 정렬 과정(sort step)을 거쳐 정렬된 리스트는 포괄 프리픽스(정의 4)들과 독립 프리픽스(정의7)들, 즉 대표 프리픽스(정의 8)들로 구성된다. 정렬된 리스트에서 i번째 대표 프리픽스는 포괄 프리픽스일 수도 있고 독립 프리픽스일 수도 있다. i번째 대표 프리픽스의 가중치(정의 9)를 w_i로 나타내기로 한다. 만약 i번째 대표 프리픽스가 포괄 프리픽스라면, w_i는 bag (정의 6)의 가중치이고, 만약 i번째 대표 프리픽스가 독립 프리픽스라면, w_i는 1이다.

리스트가 정렬되고 나면 각 대표 프리픽스들의 가중치를 알 수 있기 때문에, 각 대표 프리픽스에 해당되는 가장 작은 프리픽스 순위와 가장 큰 프리픽스 순위를 구할 수 있다. i번째 대표 프리픽스에 해당되는 프리픽스 중 가장 작은 프리픽스의 순위를 a_i, 가장 큰 프리픽스의 순위를 b_i라 하면, a_i와 b_i는 다음과 같이 구할 수 있다.

$$\begin{aligned} a_i &= b_{i-1} + 1 \\ b_i &= b_{i-1} + w_i \end{aligned} \quad (4)$$

여기에서 초기 값을 위하여 $b_0 = 0$으로 정의한다.

i번째 대표 프리픽스가 포괄 프리픽스인 경우에는 식 (4)의 w_i가 bag의 가중치(정의 9)로 정의되며, 여기에 예속된 프리픽스들의 순위는 $[a_i, b_i]$에 존재한다. 만약 i번째 대표 프리픽스가 독립 프리픽스인 경우에는 $w_i = 1$이므로, $a_i = b_i = b_{i-1} + 1$ 이 된다. 이와 같이 구해진 $[a_i, b_i]$는 i번째 대표 프리픽스에 해당되는 프리픽스들의 순위 범위(range)를 나타낸다.

리스트의 전체 가중치는 대표 프리픽스 가중치들의 합으로 정의할 수 있으며 w_t로 나타내기로 한다. 전체 가중치 w_t는 다음과 같이 구할 수 있으며

$$w_t = \sum_{i=1}^{n} w_i \quad (5)$$

실제적으로 리스트의 전체가중치는 리스트의 프리픽스 개수와 같다. 즉, $w_t = N$이다.

전체 가중치 w_t의 중간 값 (median number)을 M으로 나타내기로 하면, M은 다음과 같이 구할 수 있다.

$$M = round\left(\frac{w_t}{2}\right) \quad (6)$$

$w_t = N$이므로, M번째 순위의 프리픽스는 전체 N개의 프리픽스 중, 중간 순위의 프리픽스를 의미한다.

대표 프리픽스의 개수를 n으로 나타내기로 한다. 대표 프리픽스 n은 전체 프리픽스 개수를 나타내는 N 보다 작다. 대표 프리픽스 중 포괄 프리픽스가 하나도 없을 경우에만, 즉 모든 대표 프리픽스가 독립 프리픽스일 경우에만 n이 N과 같아진다. 정렬된 리스트의 중간 인덱스(median index)를 m으로 나타내기로 한다. 이 중간 인덱스는 이진 프리픽스 트리를 구성하는 과정에 있어서, 그 다음 레벨의 루트에 해당되기 때문에 매우 중요한 의미를 갖는다. 기존의 BPT 방식[12]에서는, 중간 인덱스를 단순하게 대표 프리픽스의 수 n을 반으로 나눈 값으로 정의하였다. 식으로 표현하면,

$$m = round\left(\frac{n}{2}\right) \quad (7)$$

이와 같이 구해진 m번째 대표 프리픽스가 그 다음 레벨의 root 노드가 되므로 트리가 불균형이 되는 것이다.

이에 비하여, 제안하는 가중 이진 프리픽스 트리 (WBPT) 방식에서는 중간 인덱스 m을 대표 프리픽스들의 가중치(정의 8)를 고려하여 계산하는 것을 제안하다. 중간 인덱스에 해당되는 대표 프리픽스가 전체 프리픽스 순위의 중간 값 M에 해당하는 프리픽스를 포함하도록 구하는 것이다. 다시 말하면, 중간 인덱스 m이 현재 정렬된 리스트에서의 대표 프리픽스들의 가중치의 합의 중간이 되도록 단계별로 정렬된 리스트에 대하여 식 (6)을 반복적으로 사용한다. 단계별로 정렬된 리스트에 대해 선택된 중간 인덱스 m과 그 리스트의 대표 프리픽스들의 가중치의 합의 중간인 M과는 다음과 같은 관계가 성립한다.

$$a_m \leq M \leq b_m \quad (8)$$

식 (8)에서 m번째 대표 프리픽스가 포괄 프리픽스라면, 전체 프리픽스 순위의 중간 값 M에 해당되는 프리픽스가 여기에 포함된 예속 프리픽스 중에 존재한다는 것을 의미한다. 만약 m번째 대표 프리픽스가 독립 프리픽스라면, 간단하게 이 프리픽스가 중간 값 M에 해당되는 프리픽스라는 것을 의미한다. 이와 같은 방법에 의하여, 대표 프리픽스들의 가중치를 고려한 중간 인덱스 m을 구한다.

이상에서 설명한 WBPT 알고리즘을 Sort 부분과 BuildTree 부분으로 나누어 의사 코드 (pseudo code)로 표현할 수 있다. 그림 3과 그림 4는 각각 Sort 부분과 BuildTree 부분의 의사 코드를 나타낸다. Sort 부분은 전체 프리픽스 리스트에서 포괄 프리픽스(정의 4)와 그 프리픽스의 예속 프리픽스(정의 5)를 구하고, 대표 프리픽스(정의 8)들에 대해서 비교 방법(정의 1)을 적용하여 정렬한다. 그리고 각 대표 프리픽스의 가중치(정의 9)를 구하고, 식 (4)를 이용하여 각 대표 프리픽스에 해당하는 순위 범위 $[a_i, b_i]$를 구한 다음, 식 (5)와 (6)에 의하여 프리픽스 순위의 중간 값 M을 구하고 식 (8)을 만족하는 가중 중간 인덱스 (weighted median index) m을 구한다. BuildTree 부분은 Sort 부분에서 구한 가중 중간 인덱스 m을 root로 잡고 root를 제외한 나머지 프리픽스들을 leftList와 rightList에 분류한 다음, leftList와 rightList를 다음 레벨에서 각각 recursive하게 BuildTree 과정을 수행한다. 이와 같은 recursive 과정을 통하여 모든 프리픽스가 leaf 노드가 될 때까지 하위 레벨들의 노드들을 구하고 가중 이진 프리픽스 트리 (WBPT)를 생성한다.

그림 5는 이와 같은 방법으로 표 1의 프리픽스 집합에 대하여 생성한 가중 이진 프리픽스 트리 (WBPT)를 나타낸다. 이 경우의 트리 깊이 L은 5로서 트리의 최소 깊이 $L_{\min}$인 5와 같다. 따라서 이 경우는 균형 트리이며, 균형도 D_b는 1이다.

Ⅳ. 시뮬레이션 결과 및 성능 비교

본 논문에서 제안하는 구조를 C언어를 이용하여 시뮬레이션을 수행하였다. 시뮬레이션을 위해 사용된 데이터는 실제 라우터를 거쳐 간 패킷들의 정보를 이용하였는데, 표 2는 제안하는 가중 이진 프리픽스 트리(weighted binary prefix tree - WBPT) 방식과 기존의 이진 프리픽스 트리(binary prefix tree - BPT) 방식[12]의 깊이 L과 균형도 D_b를 비교한 것이다. 제안하는 WBPT 방식은 최소 깊이 $L_{\min}$에 매우 근접하는 성능을 보이며, BPT 방식보다 균형도에 있어 훨씬 우수함을 알 수 있다.

```
Sort(List)
/* First find enclosure prefixes */
for all i in List do
 compare i with all j in List where j ≠ i
  if i matches j then
        if i is shorter than j, then
          put j in i's bag
          delete j from List
        else
          put i in j's bag
          delete i from List
 end compare
end for

/* Sort representative prefixes*/
for all i in List do
 compare i with all j in List where
   if j < i, then
        exchange i with j
 end compare
end for

/* Find the weight and range of each representative prefix */
for all i in List do
   number ← 0    /* number of elements in List */
  if i is an enclosure, then
     w_i is the number of prefixes in the bag
  else
     w_i ← 1;
  a_i ← b_{i-1} + 1
  b_i ← b_{i-1} + w_i
  number ← number + w_i
end for
w_t ← number

/* Find the median index including weight in List */

for all i in List do
  if (a_m ≤ M ≤ b_m )
        break;
end for
M ← round (w_t / 2)    /* weighted median index */
return m
end Sort
```

그림 3. WBPT 알고리즘 중 Sort 부분 의사 코드

```
BuildTree(List)
 if List is empty, return;
 m<-Sort(List);
 root<- m
 let leftList and rightList contain all elements in the Left and right of m
 if m is an enclosure, then
   distribute elements in m's bag into leftList and rightList
 leftChild(root)<-BuildTree(leftList);
 rightChild(root)<-BuildTree(rightList);
 return the address of root
end BuildTree
```

그림 4. WBPT 알고리즘 중 BuildTree 부분 의사 코드

표 3는 약 3만개 정도의 프리픽스를 갖는 MAE-WEST-3 라우팅 테이블에 대하여 기존에 제안된 소프트웨어에 기반을 둔 방식들과의 성능을 비교한 결과이다. 소프트웨어에 기반을 둔 IP 주소 검색 방식의 평가 기준 중 가장 중요한 메모리 접

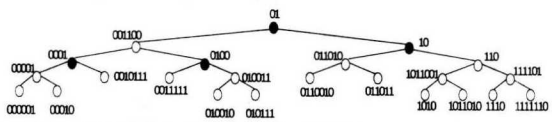


그림 5. 가중 이진 프리픽스 트리 (WBPT)

근 횟수에 대해서는 평균, 최소, 최대 값을 각각 비교하였는데, 이 중 라우터의 처리 속도를 보장(guarantee)해 주는 의미를 갖는 최대(worst case) 메모리 접근 횟수가 실제적으로 가장 중요한 의미를 갖는다. Lulea 방식[11]은 캐쉬 사이즈에 맞는 작은 사이즈의 메모리를 사용하지만 이를 위하여 전처리(pre-processing) 과정이 필요하며 최대 메모리 접근 횟수가 많은 단점이 있다. 그룹 해싱(group hashing) 방식[7]의 경우 네 개의 메모리에 대해 병렬 검색을 수행한 경우 최대 검색 횟수는 24번이나 본 논문에서 제안하는 방식처럼 하나의 메모리에 저장한 경우 병렬 검색이 불가능 하여 최대 104번의 메모리 접근이 요구되어 실제적이지 못하며, 1500Kbyte의 매우 큰 사이즈 메모리를 필요로 함을 알 수 있다. BPT 방식[12]의 경우 평균 메모리 접근 횟수는 16.4로 제안하는 방식의 15.4와 큰 차이를 보이지 않지만 BPT의 불균형 특성으로 최대 메모리 접근 횟수가 31로서 제안하는 WBPT 방식의 17에 비하여 훨씬 큰 것을 알 수 있다. 메모리 사이즈에 있어서 제안하는 구조는 343Kbyte의 작은 사이즈 메모리를 필요로 하며 이는 L2 cache의 최대 사이즈인 512Kbyte보다 작은 사이즈로 L2 cache에 저장이 가능하다. 그리고 Lulea 방식과 달리 전처리 과정이 필요로 하지 않아, 그룹 해싱이나 BPT 방식과 마찬가지로 프리픽스의 추가, 삭제가 용이하다.

표 2. BPT 방식[12]과 제안하는 WBPT 방식의 트리 깊이 및 균형도 비교

	프리픽스 개수 N	최소 깊이 L_{min}	BPT		WBPT	
			깊이 L	균형도 D_b	깊이 L	균형도 D_b
MAE-WEST-1	14553	14	23	0.5	16	0.89
MAE-WEST-2	14937	14	22	0.56	16	0.89
AADS	20204	15	22	0.59	16	0.94
PAC BELL	20519	15	22	0.59	16	0.94
MAE-WEST-3	29584	15	31	0.06	17	0.88
MAE-EAST-1	37993	16	25	0.44	17	0.94
MAE-EAST-2	39464	16	26	0.38	17	0.94
FUNET	40905	16	31	0.06	17	0.94

표 3. 소프트웨어 기반한 기존의 방식들과의 성능 비교

	메모리 접근 횟수			메모리 (Kbyte)	전처리
	평균	최소	최대		
Lulea[11]	Not Available	5	40	160	필요함
Group hashing[7]	25.6	8	104	1500	필요하지 않음
BPT[12]	16.4	14	31	343	필요하지 않음
WBPT	15.4	12	17	343	필요하지 않음

그림 6은 최대 메모리 접근 횟수에 있어서, 기존의 소프트웨어에 기반한 IP 주소 검색 방식 중 가장 좋은 성능을 보이는 BPT 방식[12]과 본 논문에서 제안하는 WBPT 방식의 최대 메모리 접근 횟수를 비교한 그래프이다. 제안하는 WBPT 방식은 기존의 BPT 방식에 비해 최대 메모리 접근 횟수가 크게 줄어든 것을 볼 수 있다. 따라서 본 논문에서 제안하는 방식은 다른 소프트웨어에 기반한 기존의 방식들에 비하여, 가장 중요한 성능 평가 기준인 최대 메모리 접근 횟수에 있어서 매우 좋은 성능을 보이는 것을 알 수 있다.

V. 결 론

본 논문에서는 IP 주소 검색을 위한 라우팅 테이블을 구성함에 있어, 최적 균형 트리에 근접하는 성능을 보이는 가중 이진 프리픽스 트리(WBPT) 방식을 제안하였다. WBPT는 포괄 프리픽스의 경우, 예속 프리픽스의 개수를 고려한 가중치(weight)를 부과하여, 기존의 BPT 방식에 비하여 프리픽스 트리가 보다 균형된 성질을 갖도록 개선한 방식이다. 프리픽스 트리의 균형도는 패킷 처리 속도의 관건이 되는 메모리 접근 횟수와 매우 밀접한 관련을 갖는데, 본 논문에서 제안된 WBPT 방식은 기존의 BPT 방식과 비교하여, 트리의 균형도에 있어 매우 우수한 성질을 보였다. 실제 라우터를 통과한 데이터를 가지고 시뮬레이션을 수행한 결과, 라우터의 주소 검색을 위한 소프트웨어 기반의 기존의 주소 검색 방식과 비교하여 WBPT 방식은 성능 평가 기준 중 가장 중요한 의미를 가지는 최대 메모리 접근 횟수에 있어서 가장 우수한 성능을 보였다. 그리고 3만 개 정도의 라우터 데이터에도 L2 cache에 저장이 가능한 정도의 메모리 크기를 요구하고, 전처리가 필요하지 않아 프리픽스의 추가, 삭제가 용이한 방식이다.

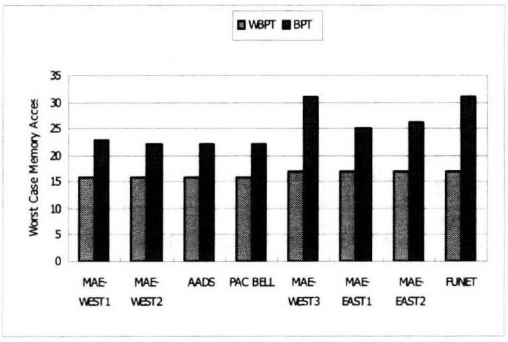


그림 6. 제안하는 WBPT 구조와 기존의 BPT 구조[12]의 최대 메모리 접근 횟수 비교

참 고 문 헌

[1] H.J. 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] P. Gupta, S. Lin, and N. McKeown, " Routing lookups in hardware at memory access speed," Proc. IEEE INFOCOM'98, pp 1240-1247, ARP 1998.

[4] Mohammad Peyravian, Gordon Davis, and Jean Calvignac, "Search Engine Implications for Network Processor Efficiency," IEEE Network, pp.12-20, July/August 2003.

[5] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookups," in Proc. ACM SIGCOMM'97 Conf., pp. 25-35, 1997.

[6] 정여진, 이보미, 임혜숙, "복수의 해쉬 함수를 이용한 병렬 IP 어드레스 검색 구조," 한국통신학회 논문지, 29권, 2B호, pp.158-166, 2004.

[7] D. Yu, B. C. Smith, and B. Wei, " Forwarding Engine for Fast Routing Lookups and Updates," Proc.IEEE GLOBECOM'99, pp.1556-1564, 1999.

[8] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups Using Multiway and Multicolumn Search", IEEE/ACM

Transactions on Networking, pp 324-334, Vol.7, No.3, Jun. 1999.

[9] V. Srinivasan and G. Varghese, "Faster IP Lookup using Controlled Prefix Expansion," in Proc. ACM SIGMATICS'98, pp.1-10, June 1998.

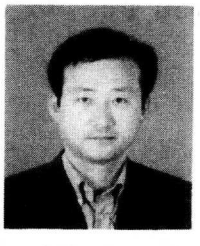
[10] S. Nilsson and G. Karlsson, "IP-Address Lookup using LC-tries," IEEE Journal on Selected Area in Communication, Vol.17, pp.1083-1092, June 1999.

[11] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," Proc. ACM SIGCOMM, pp.3-14, 1997.

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

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

임 창 훈(Changhon Yim) 정회원



1986년 2월 : 서울대학교 제어계측공학과 학사

1988년 2월 : 한국과학기술원 전기 및 전자공학과 석사

1996년 12월: The University of Texas at Austin, Electrical and Computer Engineering 박사

1988년 3월 ~ 1991년 6월: 한국방송공사 기술연구소, 연구원

1996년 12월 ~ 1999년 3월: Sarnoff Corporation, 연구원

1999년 3월 ~ 2000년 7월: Lucent Technologies, Bell Labs, 연구원

2000년 8월 ~ 2002년 3월: KLA Tencor Corporation, Sr. Software Engineer

2002년 5월 ~ 2003년 8월: 삼성전자 디지털미디어 연구소, 수석연구원

2003년 9월 ~ 현재: 건국대학교 인터넷미디어공학부, 조교수

<관심분야> 멀티미디어 통신, 멀티미디어 네트워크, 비디오/영상 압축

임 혜 숙(Hyesook Lim) 정회원

1986년 2월 : 서울대학교 제어계측공학과 학사

1986년 8월 ~ 1989년 2월 : 삼성 휴렛 팩커드 연구원

1991년 2월 : 서울 대학교 제어계측공학과 석사

1996년 12월: The University of Texas at Austin, Electrical and Computer Engineering 박사

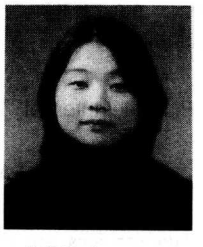
1996년 11월 ~ 2000년 7월 : Lucent Technologies, Member of Technical Staff

2000년 7월 ~ 2002년 2월 Cisco Systems, Hardware Engineer

2002년 3월 ~ 현재 : 이화여자대학교 정보통신학과, 조교수

<관심분야> Router나 switch등의 Network 관련 SoC설계, 통신관련 SoC 설계

이보미 (Bomi Lee) 준회원



2003년 2월 : 이화여자대학교 정보통신학과 학사

2003년 3월 ~ 현재 : 이화여자대학교 정보통신학과 석사과정

<관심분야> Router나 switch등의 Network 관련 SoC설계, TCP/IP관련 하드웨어 설계