

# 복수의 해쉬 함수를 이용한 병렬 IP 어드레스 검색 구조

준회원 정 여 진\*, 준회원 이 보 미\*, 정회원 임 혜 숙\*\*

## A Parallel Multiple Hashing Architecture for IP Address Lookup

Yeo-Jin Jung\* Associate Member, Bomi Lee\* Associate Member, Hyesook Lim\*\* Regular Member

### 요 약

IP 주소 검색은 인터넷 라우터의 필수적인 기능 중에 하나인 동시에 인터넷 라우터의 전체 성능을 결정하는 중요한 요소이다. 현재 인터넷 라우터에 연결된 네트워크 종류의 증가로 라우팅 테이블 엔트리 수가 급격히 증가하고 있으며, 인터넷 트래픽 역시 빠르게 증가하고 있어 효율적인 라우팅 테이블의 검색이 요구된다. 그 동안 빠른 주소 검색을 위해 다양한 알고리즘들과 검색 방식들이 제안되었지만 대부분 메모리 사이즈나 업데이트 등의 실용적인 측면에 대한 고려가 부족하였다. 본 논문에서는 IP 주소 검색을 위한 실용적인 하드웨어 구조를 제안한다. 제안된 구조는 multiple hashing을 적용한 병렬 IP 주소 검색 구조로, 메모리 사이즈나 메모리 검색 횟수, 업데이트에 있어서 장점을 가진다. 본 논문에서는 제안한 하드웨어 구조의 성능을 평가하기 위하여 MAE-WEST 라우터를 통과한 실제 데이터를 사용하여 시뮬레이션을 수행하고, 이를 통해 203kbytes의 메모리와 200여개의 엔트리를 저장할 수 있는 TCAM을 사용하여 한번의 메모리 접근으로 주소 검색이 가능함을 보였다.

Key Words : longest prefix match, IP address lookup, multiple hashing, CRC, parallel search

### ABSTRACT

Address lookup is one of the most essential functions of the Internet routers and a very important feature in evaluating router performance. Due to the facts that the Internet traffic keeps growing and the number of routing table entries is continuously growing, efficient address-lookup mechanism is indispensable. In recent years, various fast address-lookup schemes have been proposed, but most of those schemes are not practical in terms of the memory size required for routing table and the complexity required in table update. In this paper, we have proposed a parallel IP address lookup architecture based on multiple hashing. The proposed scheme has advantages in required memory size, the number of memory accesses, and table update. We have evaluated the performance of the proposed scheme through simulation using data from MAE-WEST router. The simulation result shows that the proposed scheme requires a single memory access for the address lookup of each route when 203kbytes of memory and a few-hundred-entry TCAM are used.

### I. 서 론

라우터에서 패킷을 적절한 출력 포트에 forwarding하기 위해서는 전달 테이블로부터 입력된 패킷의 network 부분, 즉 프리픽스와 일치하는 엔트리를 찾

아 출력 포트 정보를 알아내는 과정인 주소 검색을 수행해야 한다. Classless Inter-domain Routing (CIDR) 방식은 Classful Addressing 방식과 달리 주소의 network 부분이 고정되어 있지 않으므로 들어온 패킷의 프리픽스를 알 수 없다. 따라서 입력된 패킷의 목적지 주소와 일치하는 여러 엔트리 중에

\* 이화여자대학교 과학기술대학원 정보통신학과 Network SoC Design 연구실 \*\* (hlim@ewha.ac.kr)

논문번호 : 030348-0811 접수일자 : 2003년 8월 12일

\*\* 본 연구는 학술진흥재단 BK21 지원으로 수행되었습니다.



서 가장 길게 일치하는 프리픽스를 찾아내야 한다. 이를 Longest Prefix Matching (LPM)이라고 한다. 패킷의 프리픽스 길이가 고정되어 있지 않기 때문에, LPM은 exact matching보다 수행이 어렵고, 라우터에서의 병목점으로 작용하고 있다.

주소 검색 구조의 성능을 파악하는 여러 가지 기준 중에서 첫 번째로 들 수 있는 것이 메모리 검색 횟수이다. 링크 속도가 증가함에 따라 빠른 주소 검색에의 요구 또한 높아지면서 주소 검색 속도의 가장 큰 오버헤드로 작용하는 메모리 검색 횟수를 줄이는 것이 중요하게 되었다. 두 번째로는 메모리의 크기를 들 수 있다. 백본 라우터의 경우, 라우터에 연결된 네트워크의 수가 기하급수적으로 증가함에 따라 전달 테이블에 저장되어야 하는 엔트리의 수 또한 빠르게 증가하게 되었는데, 이들 엔트리를 메모리에 효율적으로 저장할 수 있어야 한다. 그 외에도 전달 테이블에 새로운 프리픽스를 추가하거나, 쓰이지 않는 프리픽스를 제거하기 위한 테이블 갱신의 용이성, 128bit 주소를 갖는 IPv6로의 확장성 등이 성능 평가의 기준이 된다. 본 논문에서는 이들 기준을 모두 만족하는, 효율적인 주소 검색 구조를 제안하고자 한다. 제안된 구조는 parallel search와 [1]에서 제안한 multiple hashing을 결합한 구조이다.

본 논문의 구성은 다음과 같다. II장에서는 기존 주소 검색 방법들에 대해 언급하고 III장에서는 multiple hashing을 이용한 주소검색에 대해서 서술한다. IV장에서 본 논문에서 제안하는 검색구조에 대하여 설명한 후, 이어 V장에서는 제안된 방식의 성능을 평가하고 기존의 다른 방법들과 비교 분석한 결과를 보여준다. 끝으로, VI장에서 결론을 맺는다.

## II. 기존의 검색 방법

최근 수년간 인터넷 라우터를 위해 다양한 주소 검색 방법들이 연구되어 왔다. 여러 가지 주소 검색 방법들을 분류하여 보면 다음과 같다.

첫 번째는 TCAM을 이용한 주소 검색 방법이다 [2]. TCAM은 테이블의 모든 엔트리에 대한 검색을 동시에 진행하여 한번의 검색으로 주소 검색을 가능하게 한다. 그러나 TCAM은 일반 메모리보다 비싸고 같은 크기의 RAM에 비하여 저장 공간이 작을 뿐 아니라 전력소모가 커서 칩 안에 내장되기 어려운 문제점을 지닌다. 따라서 수 만개가 넘는 엔트리를 가지는 백본 라우터의 전달 테이블이나

128bit 길이의 주소를 가지는 IPv6에 적용하는 것은 무리가 있다고 여겨진다.

두 번째는 trie 구조에 기초한 검색 방법이다. Trie는 프리픽스를 tree 형태로 저장한 것으로 주소 검색과 관련된 연구 중 가장 활발하게 연구되는 방법이다. Trie 구조에서의 각 노드는 주소의 한 bit에 해당하며, 각 노드에서 1 또는 0을 판단하여 매 bit에 대하여 반복적으로 검색을 진행한다 [3]. 이와 같은 검색 방법은 w(주소 길이 또는 trie의 높이)번의 메모리 검색이 요구되며, trie를 저장하기 위하여 많은 양의 메모리가 필요하게 된다 [3]-[8]. Prefix expansion을 이용하여 메모리 검색 횟수를 줄인 검색 방법들도 제안되고 있는데, D개의 프리픽스 길이의 집합들을 L개의 프리픽스 길이의 집합(D>L)으로 일부 프리픽스 길이를 확장하여 프리픽스 길이의 집합의 수를 줄이는 방법이다 [4].

[7]은 프리픽스의 길이가 24보다 짧은 모든 프리픽스들을 24로 확장하여  $2^{24}$ 개의 엔트리를 가지는 메모리에서 첫 번째 검색을 진행하는 방법을 제안하였다. 들어온 주소가 24보다 긴 프리픽스를 가지는 경우에는 24보다 긴 나머지 프리픽스를 저장한 다른 메모리로 이동하여 검색을 수행한다. 이는 최대 메모리 검색 횟수가 2번이라는 장점이 있지만  $2^{24}$ 개의 엔트리를 저장하기 위해서 33Mbytes라는 큰 메모리를 사용해야 한다.

[8]에서는 프리픽스의 길이가 16보다 짧은 프리픽스들을 16으로 확장하여  $2^{16}$ 개의 엔트리를 가지는 메모리로 전달 테이블을 구성하고 들어온 주소의 프리픽스가 16보다 긴 경우에는 [8]에서 제안한 알고리즘으로 구성된 sub-tree를 따라 검색을 수행한다. 이러한 검색 방식의 경우 사전 연산량이 많고 업데이트가 어렵다는 단점이 있다.

마지막으로 hashing을 이용한 검색 방법이 있다. Link Layer에서 exact match를 위해 사용되는 hashing을 IP 주소 검색에 적용시킨 것이다.

[9]는 프리픽스 길이에 따라 별도의 테이블과 hash 함수를 만들어 각각의 프리픽스 테이블에 대하여 병렬로 검색을 수행하는 방식을 제안하였다. 프리픽스를 저장하는 과정에서 충돌이 생기는 경우에는 보조 테이블에 저장하고, 보조 테이블에 대해서는 바이너리 서치를 적용하였다. 이러한 경우, 주 테이블에서 일치하는 프리픽스를 찾지 못했을 때에는 보조 테이블에서 검색을 다시 진행해야 하므로 메모리 검색 횟수가 증가하는 단점을 가지고 있다.

[10]은 프리픽스 길이 별로 분류된 하나의 테이블



블에 대하여 바이너리 검색을 수행하여 길이 레벨을 찾고, 각 길이 레벨에서의 검색에는 hashing을 적용하였다. W개의 프리픽스 길이 종류에 대하여 주소 검색을 하는 경우, 최대  $\log_2 W$ 번의 메모리 검색이 요구된다. 여기서 제안된 방식은 선처리 계산이 많고, 테이블 갱신이 어렵다. 또한 논문에서 가정한 perfect hash 함수를 찾아내는 데에 13분이라는 시간이 걸리게 되므로 이 방식은 프리픽스 정보가 자주 변화하지 않는 경우에만 유효하게 되는 단점이 있다 [4].

### III. Multiple Hashing을 이용한 주소 검색

Hashing은 프리픽스를 보다 짧은 bit으로 표현하므로 충돌, 즉 서로 다른 프리픽스가 같은 hash 값을 가지는 현상이 발생할 수 있다. [4]에서는 MAE-EAST 데이터에서 충돌이 거의 발생하지 않는 semi-perfect hash 함수를 찾는데 13분의 시간이 걸림을 언급하고 있다.

Semi-perfect hash 함수를 찾아내는 데 긴 시간이 걸리는 단점을 보완하기 위해 [1]에서는 perfect hash 함수 대신, multiple hash 함수를 이용하는 방법을 제안하였다. 각 프리픽스에 대하여 multiple hash 함수를 적용시킴으로써 semi-perfect hashing이 가능하게 하겠다는 것이다. [1]은 여러 개의 hash 함수를 사용하는 것이 하나의 hash 함수를 사용할 때보다 성능이 크게 개선되는 것을 분석을 통해 예측하고, multiple hashing에 [10]의 방식을 접목한 주소 검색 시뮬레이션을 수행하여 그 사실을 증명하였다. 그림 1은 [1]에서 제안하는 테이블 구성 방식에 대한 그림이다. 그림 1에서 볼 수 있듯이 라우팅 테이블을 만들 때 하나의 프리픽스가 여러 개의 다른 hash 함수를 통과하고 함수에서 얻은 hash 값을 각 테이블의 포인터로 하여 테이블 중 load가 적은 쪽의 테이블에 채워지게 된다. 여러 개의 hash 함수를 사용함으로써 hashing으로 인한 충돌 발생 가능성이 줄어들고 데이터들이 테이블에 균등하게 분포하게 된다. [1]에서는 프리픽스 길이를 16, 26, 32로 expansion 한 후 multiple hashing 방식으로 구성된 테이블에 대해 [10]에서 제안한, 프리픽스 길이 level에 따른 binary search 방식을 적용하여 주소 검색을 수행하였다. 그 결과 4Mbytes의 메모리가 사용되었으며 최대 2번의 메모리 접근이 소모되었다.

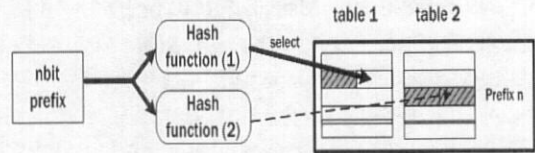


그림 1. 여러 개의 hash함수를 이용한 라우팅 테이블 구성

그러나 [1]에서 제안하는 주소 검색은 exact matching을 기본으로 하고 있어 LPM에 적용하기 위한 구체적인 방안이 없고, 충돌에 대한 고려가 없어 multiple hash 함수를 사용하기 위해서도 데이터의 분포에 따라 적합한 hash 함수를 찾아내야 하는 단점이 있다. 이 때문에 테이블 업데이트를 위한 과정이 있어서도 hash 값이 가리키는 테이블의 bucket이 꼭 차 있는 경우 현재의 데이터 분포에 맞는 multiple hash 함수를 찾아내는 과정이 다시 수행되어야 한다.

### IV. 제안하는 주소 검색 구조

본 논문에서는 parallel search와 [1]에서 제안한 multiple hashing을 접목시킨 새로운 구조를 제안한다. 프리픽스들을 길이별로 분류하여 각 길이마다 복수의 테이블을 구성하고 각 테이블에 대해서는 CRC 해쉬 함수를 사용하여 multiple hashing을 적용하는 방법이다. 그림 2는 본 논문에서 제안하는 하드웨어의 구조를 나타낸 것이다.

#### 1. Parallel Lookup

LPM가 어려운 이유는 프리픽스의 길이가 고정되어 있지 않기 때문에 전달 테이블에 복수의 매칭 엔트리가 저장되어 있을 수 있고 이 중에서 가장 긴 엔트리를 찾아야 하기 때문이라고 앞에서 언급하였다. 본 논문에서 제안된 구조에서는 고정되어 있지 않은 프리픽스 길이로 인한 메모리 검색 횟수의 증가라는 문제점을 해결하기 위하여 프리픽스 길이별로 별도의 테이블을 구성, 다른 메모리에 저장함으로써 모든 길이에 대하여 병렬 검색을 가능하게 하였다. 즉 LPM의 문제를 exact matching의 문제로 전환하여 접근하는 방법이다. 따라서 검색 시 프리픽스 길이별로 구성된 각 프로세스는 해당 길이에 대하여 하나의 일치하는 엔트리만을 찾으므로, 이렇게 각 프로세스에서 병렬로 찾아진 엔트리 중에서 가장 긴 엔트리가 매칭 엔트리로 결정되



는 것이다.

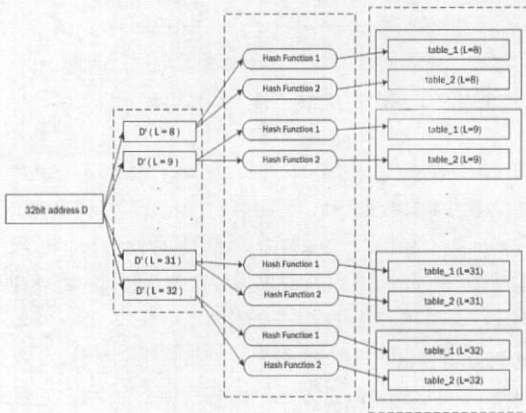


그림 2. 제안하는 구조

## 2. CRC를 이용한 Multiple Hashing

Hashing은 입력된 값을 hash 함수를 통과시켜 더 짧은 값에 매핑시키는 방식이다. Hashing은 입력된 값을 더 짧은 값에 매핑시키므로 서로 다른 여러 개의 입력이 같은 출력 값에 매핑되게 된다. 이러한 현상을 충돌이라고 하며, hashing을 이용한 구조의 경우 충돌을 최소화하는 것이 관건이 된다. 따라서 좋은 hash 함수는 임의의 입력 집합에 대하여 균일한 출력 값을 가져야 한다 [11].

IP 주소 검색에 있어 hashing을 적용하게 되면 프리픽스들을 전달 메모리에 좀더 효율적으로 저장할 수 있고, 검색 시, 한번의 검색으로 일치하는 엔트리를 찾을 수 있으므로 우수한 성능을 보이지만 이때 반드시 고려해야 하는 점이 hashing으로 인한 충돌을 어떻게 처리할 것인가 하는 문제이다. 본 논문에서는 충돌을 최소화하기 위하여 hash 함수 중에서 perfect hashing에 가까운 성능은 낸다고 평가되는 CRC를 사용하여 [12], 여기에 [1]에서 제안한 multiple hashing을 적용하였다. 그러나 [1]이 데이터 분포에 따라 hash 함수를 찾는 software 방식에 기반을 두었다면, 제안된 구조에서는 고정된 CRC hash 함수를 사용하여 하드웨어 구현 및 전달 테이블의 갱신을 용이하게 하였다. 또한, 각 테이블을 위한 모든 hash 값들은 하나의 hashing 하드웨어에서 얻도록 하여 각 프리픽스별로 별도의 hashing 하드웨어를 사용하지 않음으로써 병렬 처리 구조에 대한 부담을 줄였다. 그림 3은 본 논문에서 사용한 CRC를 계산하는 하드웨어를 나타내고 있다 [11].

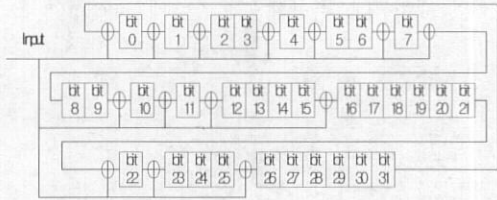


그림 3. CRC-32

CRC hashing 하드웨어로부터 hash 값들을 얻어내는 방법은 다음과 같다. 우선, 목적지 IP 주소가 매 cycle마다 상위 bit부터 한 bit씩 CRC hashing 하드웨어에 들어가게 된다. 목적지 주소가 CRC hashing 하드웨어에 들어가기 시작하지 D cycle 뒤에, CRC 레지스터로부터 특정 길이의 hash 값을 2개 뽑아서 프리픽스 D의 hash 값으로 사용하게 되는 식으로, 각 프리픽스들의 hash 값은 시간차를 두고 하나의 하드웨어로부터 얻어지게 된다. 예를 들면 IP 주소의 첫 8bit이 CRC hashing 하드웨어에 들어가면 CRC 레지스터에서 프리픽스 8에 대한 hash 값을 2개 뽑아내고 그 후 한 cycle 뒤에는 한 bit이 더 들어가서 9bit이 들어갔으므로 프리픽스 9에 대한 hash 값 2개를 뽑아내는 식으로, 프리픽스 32에 대한 hash 값까지 얻는 데에는 32cycle이 소요된다.

위에 설명한 바와 같이 IP 주소의 모든 프리픽스에 대하여 hash 값이 나오기까지 32 cycle이 걸리게 되는데, 이 시간이 실시간으로 주소 검색을 수행하는 데 있어 문제가 없는 지를 분석하여 보면 다음과 같다. 최소 길이의 패킷은 preamble과 SFD (Start of Frame Delimiter)를 포함한 72byte로 가정하고, 패킷과 패킷 사이에는 96bit의 IFG (inter-frame gap)를 가정한다. 패킷을 100MHz cycle로 처리한다면 모든 hash 값을 얻는 데 요구되는 시간은 320ns이다. 따라서 라우터는 2.1Gbps까지 line rate로 동작이 가능하다. 200MHz cycle로 패킷을 처리하는 경우에는 4.2Gbps의 line speed까지 가능하다. 라우터의 aggregated bandwidth가 4.2Gbps 이상 요구될 경우 그림 2와 같이 프리픽스 길이별로 별도의 hash 함수를 사용하면 된다.

## 3. 테이블을 구성하는 방법

본 논문에서는 각 테이블의 bucket 수와 bucket 당 엔트리 수를 지정하기 위하여 [1]의 data 분석을 참고하였다. [1]의 분석에 따르면, 2개의 hash 함수를 사용하고 N개의 프리픽스를 N/2개의 bucket을



가지는 테이블에 저장하고자 할 때 bucket당 5번 이상의 충돌이 발생할 확률은 optimal한 hash 함수를 사용하였을 때,  $5.0 \times 10^{-7}$ 으로 매우 낮았다. 이 분석을 바탕으로 각각의 프리픽스 길이에 대하여 두 개의 hash 함수와 bucket당 2개의 load를 가지는 경우를 예로 하여 제안된 구조를 설명하면 다음과 같다.

테이블의 각 엔트리는 그림 4와 같이 엔트리에 저장된 유효한 load 수를 나타내는 field와 최대 2개의 load에 대한 정보들을 저장하는 field들을 가지게 된다. 각 load는 프리픽스 field와 forwarding RAM pointer field를 가지고 있다.

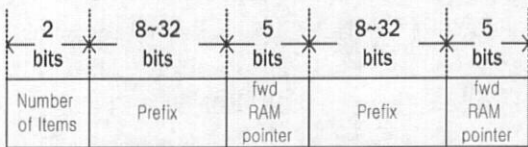


그림 4. 제안된 구조의 전달 테이블의 엔트리 구조

Hash 값의 길이는 bucket의 수에 따라 결정되는데, 그 과정은 다음과 같이 이루어진다. 각 프리픽스 길이별로 저장하고자 하는 프리픽스가 N개일 때, 한 bucket당 load의 수를 4로 하고, 프리픽스를 저장할 수 있는 총 load의 수를 2N으로 하면, 테이블의 bucket의 수는  $N/2$ 이 되고 hash 값은 bucket을 가리키는 인덱스가 되므로, 이에 필요한 hash 값의 길이는  $\lceil \log_2(N/2) \rceil$ 가 된다. 저장하고자 하는 프리픽스의 수가 2개 이하인 경우에는 hash bit을 2로 하였다.

테이블을 구성하는 알고리즘은 그림 5와 같이 표현될 수 있다. 우선, 목적지 IP주소가 CRC hashing 하드웨어에 한 bit씩 들어간다. 그 후 L(프리픽스 길이) cycle 후에 CRC 레지스터로부터 hash 값을 뽑아낸다. 테이블에 저장되는 프리픽스는 두 개의 hash 값이 가리키는 bucket 중 load를 적게 가지는 bucket에 저장된다. 두 bucket이 같은 load를 가지는 경우에는 테이블 1에 선 저장되는 것으로 하였고, 두 테이블이 꼭 차서 오버플로우가 난 경우에는 별도로 구성된 오버플로우 테이블에 저장하였다. 제안된 구조에서는 각 프리픽스마다 2개씩 총 48개의 주 테이블(prefix 8~32, 31제외)과 오버플로우 테이블이 필요하게 된다. 그림 6은 전달 테이블을 구성하는 방법에 대한 블록다이어그램이다.

#### 4. 테이블을 검색하는 방법

검색 과정은 CRC hashing 하드웨어에서 얻어진 hash 값을 이용하여 병렬로 행해진다. 검색 과정에 필요한 hash 값은 전달 테이블을 구성할 때와 비슷한 방법으로 얻어지는데, 한 가지 다른 점은 하나의 프리픽스에 대한 hash 값이 아니라 모든 프리픽스에 대한 hash 값을 얻어내야 한다는 것이다. 그림 7의 알고리즘에 설명된 것처럼, hash 값을 테이블의 인덱스로 하여 각 프리픽스 테이블에 대하여 병렬 검색을 수행하며, 이 때 오버플로우 테이블에 대해서도 동시에 검색을 수행하게 된다. 그림 8의 priority encoder에서는 검색 과정에서 얻어진, 일치하는 엔트리들 중 가장 길게 일치하는 엔트리를 뽑아내며, 이 엔트리의 forwarding RAM pointer 정보를 이용하여 패킷을 출력 포트로 전달하게 된다.

```

For prefix length L, P[L:1:0]
P[L:1:0] serially entered to CRC hash function
Extract H1(L), H2(L) from CRC registers after L cycles
Do
    table1_ptr=H1(L)
    table2_ptr=H2(L)
    If ((# of load(table1_ptr) == (# of load(table2_ptr) == 3)
        Then put P[L:1:0] to overflow table
    Else if (# of load(table1_ptr) > # of load(table2_ptr))
        Then put P[L:1:0] to table2
    Else put P[L:1:0] to table1
End Do
    
```

그림 5. 제안된 구조의 전달 테이블 구성 알고리즘

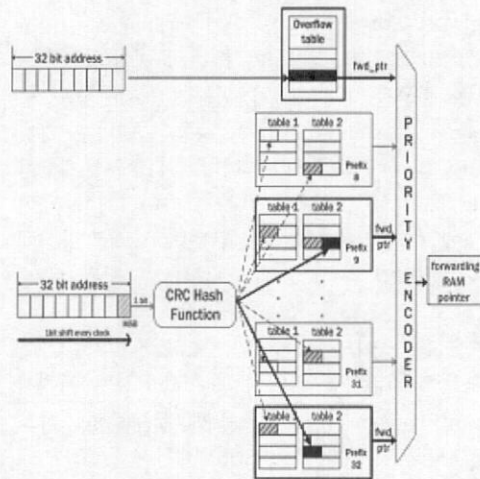


그림 6. 제안된 구조의 전달 테이블 구성 방법



```

At cycle L (for L=8-32), let D[31:31-L+1] is L bits of destination address D.
D[31:31-L+1] serially entered to CRC hash function
Extract H1(L), H2(L) from CRC registers
Do Parallel (L=8-32)
    table1_ptr=H1(L)
    table2_ptr=H2(L)
    If(D[31:31-L+1]=prefix(table1_ptr))
        Then fwd_ptr=fwd_ptr(table1_ptr)
    Else if(D[31:31-L+1]=prefix(table2_ptr))
        Then fwd_ptr=fwd_ptr(table2_ptr)
End Do Parallel
Search from overflow CAM
Determine LPM among matching entries
    
```

그림 7. 제안된 구조의 전달 테이블 검색 알고리즘

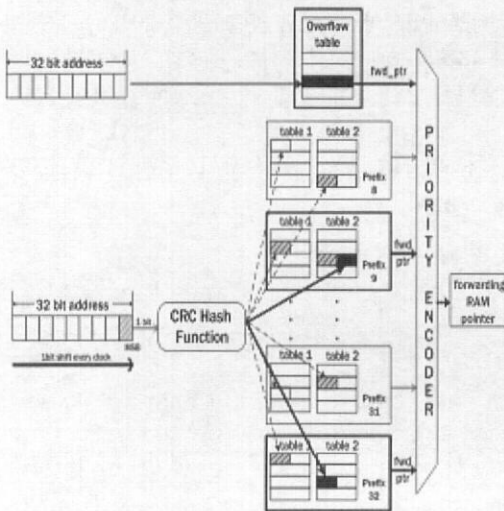


그림 8. 제안된 구조의 전달 테이블 검색 방법

### 5. Overflow 테이블

테이블 1과 테이블 2가 꽉 차서 새로 추가되는 프리픽스를 저장할 공간이 없는 경우에는 별도로 구성된 오버플로우 테이블에 저장되게 된다. V 장의 표 1에서 보이는 바와 같이 두개의 hash 값과 203Kbytes 크기의 메모리를 사용한 경우 전체 프리픽스의 0.5%에 해당하는 150여개의 오버플로우가 발생하였는데 이는 작은 TCAM을 이용하여 구현될 수 있다. TCAM으로 구현된 오버플로우 테이블의 경우도 다른 테이블들의 검색과 병렬로 이루어지게 된다.

### 6. Update 및 IPv6로의 확장성

제안된 구조의 업데이트는 매우 쉽다. 업데이트

과정은 전달 테이블 구성 과정과 동일하게 이루어진다. 새로운 프리픽스를 추가하는 경우, 저장할 프리픽스를 hash 값이 가리키는 bucket 중 load가 적게 걸린 bucket에 저장하면 되는 것이다. 테이블에 오버플로우가 생기는 경우에는 오버플로우 TCAM에 저장된다. 기존의 프리픽스를 테이블에서 삭제하는 과정도 새로운 프리픽스를 추가하는 과정과 유사하게 이루어진다. hash 값이 가리키는 bucket에서 해당 프리픽스를 찾아 삭제하고 나머지 유효한 프리픽스들을 중간에 무효한 load가 있지 않도록 재배열한 뒤, 그 bucket에 저장된 유효한 load의 수를 나타내는 값을 하나 감소시킨다. 만약 삭제하고자 하는 프리픽스가 hash 값이 가리키는 bucket에 존재하지 않는 경우에는 오버플로우 테이블에서 해당 프리픽스를 찾아 삭제한다. 이와 같은 방식은 긴 선처리 시간이 요구되지 않기 때문에 빠른 업데이트가 가능하다. 제안하는 구조는 존재하는 프리픽스 길이별로 별도의 메모리를 사용하므로 IPv6로의 확장을 위해서는 매우 많은 수의 메모리가 요구될 것이나 IC 기술의 발달로 큰 문제가 되지 않을 것으로 생각된다.

### V. 시뮬레이션 결과 및 비교

본 논문에서는 IV장에서 제안된 하드웨어를 2002년 3월 15일자 MAE-WEST 라우터를 통과한 실제 데이터를 사용하여 시뮬레이션을 수행하였다. 그림 9은 MAE-WEST를 통과한 프리픽스들의 분포를 나타내고 있다.

본 시뮬레이션은 엔트리 효율과 오버플로우 발생 비율을 테스트하기 위하여, 라우터의 프리픽스 분포에 바탕을 두고 bucket의 수, hash 값에 대하여 테스트 케이스들을 결정하였다. 여기서 엔트리 효율이란, 전체 테이블 엔트리에 대하여 프리픽스를 저장하고 있는 엔트리의 비율을 의미한다.

표 1은 N개의 프리픽스에 대하여 bucket 수, hash 값의 수를 달리하며 수행한 시뮬레이션의 결과 얻어진 엔트리 효율과 오버플로우 발생 비율을 나타낸 표이다. 케이스 1은 hash 값을 1개 사용한 경우이다. 즉, 24개의 테이블(prefix길이 31 제외)을 사용하여 프리픽스가 길이별로 저장되었다. N개의 프리픽스를 프리픽스 길이에 따라 각 bucket당 4개의 load를 가지는 N/2개의 bucket에 저장하였을 때, 약 203Kbyte의 메모리가 소요되었으며 이 경우,



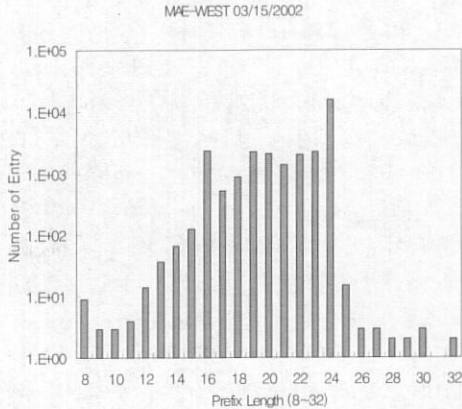


그림 9. MAE-WEST 라우터의 프리픽스 길이에 따른 라우트 분포

3.4%의 오버플로우가 발생하였다. 그러나 다른 조건은 케이스 1의 경우와 같게 하고 hash 값의 수만 2로 늘린 케이스 2(48개의 테이블)의 경우에는 오버플로우 비율이 약 0.5%(154개)로 현저히 줄어드는 것을 볼 수 있었다. 이는 하나의 hash 값을 이용하였을 때에 비하여 두개의 hash 값을 사용함으로써 각 프리픽스가 두 테이블에 고루 분포되기 때문에 오버플로우의 갯수가 줄어들게 되는 것이다. 세 번째 케이스는 오버플로우를 완전히 없애기 위하여 bucket당 저장될 수 있는 load의 수를 2개에서 3개로 늘려 앞의 두 경우보다 약 33%의 메모리를 더 할당한 케이스로 오버플로우를 완전히 없앨 수 있었다. 마지막으로, 케이스 4는 N개의 프리픽스를 bucket당 2개의 load를 가지는 N/4개의 bucket에 저장하고 3개의 hash 값을 사용하여 메모리를 가장 효율적으로 사용한 경우이다. 이 경우, 약 152Kbytes의 메모리가 소요되었으며 136개의 오버플로우가 발생하였다. 위의 4개 케이스의 비교를 통하여, hash 값을 하나 사용하여 시뮬레이션한 경우보다 여러 개의 hash 값을 사용한 경우, 충돌로 인한 오버플로우가 급격히 줄어들음을 보였다. 또, 메모리 효율과 오버플로우의 발생 비율은 trade-off 관계에 있는 것을 알 수 있다.

표 2는 기존의 다른 검색 구조들과 본 논문에서 제안된 검색 구조와의 성능을 비교한 것이다. 표 2에서 볼 수 있듯이 본 논문에서 제안된 구조는 다른 구조들보다 메모리 검색 횟수나 메모리 크기 면에서 매우 우수한 성능을 나타내었다.

## VI. 결 론

본 논문에서는 효율적이고 실용적인 하드웨어 구조를 제안하였다. 본 구조의 핵심은 parallel search와 multiple hashing이다. 프리픽스들을 프리픽스 길이에 따라 분류하고 각각 별도의 테이블을 구성함으로써 모든 프리픽스를 길이별로 병렬 검색이 가능하게 하였다. 이를 통하여 IP 주소에 hashing을 적용할 수 있도록 하였으며, 검색 시간을 줄일 수 있었다. 여기에 충돌로 인한 오버플로우의 문제를 해결하기 위하여 perfect hashing과 비슷한 성능을 보이는 multiple hashing을 접목하였다. 그 결과, 한번의 메모리 검색으로 모든 프리픽스에 대한 검색이 가능하였으며, 제안된 구조는 총 203kbytes를 갖는 48개의 메모리와 적은 수의 엔트리를 갖는 TCAM을 사용하여 구현되었다. 게다가 본 구조는 프리픽스 업데이트 및 IPv6로의 전환이 간단하며, 그 기능의 반복성으로 하드웨어 구현이 쉽다는 장점을 가진다.

표 1. 제안된 구조의 메모리 크기와 엔트리 효율 분석

case	Prefix의 수	Bucket의 수	Hash함수의 수	엔트리 수/bucket	메모리 사이즈	엔트리 효율	Overflow 비율
1	N	N/2	1	4	203KB	49.85%	3.4%
2	N	N/2	2	2	203KB	49.85%	0.52%
3	N	N/2	2	3	303KB	33.41%	0%
4	N	N/4	3	2	152KB	66.35%	0.46%

표 2. 제안된 구조와 기존의 검색 구조와의 성능 비교

Address Lookup Scheme	Number of Memory Accesses (Minimum, Maximum)	Forwarding Table Size
Huang's scheme [8]	1, 3	450KB~470KB
DIR-24-8 [7]	1, 2	33MB
DIR-21-3-8 [7]	1, 3	9MB
SFT [13]	2, 9	150KB~160KB
Parallel hashing [9]	1, 5	189KB
Proposed Architecture	1, 1	203KB+154 entry TCAM



## ACKNOWLEDGEMENT

본 연구의 시뮬레이션을 위해 도움을 준 윤명희 학생에게 감사합니다.

### 참 고 문 헌

- [1] A. Broder and M. Mitzenmacher, "Using Multiple Hash Functions to Improve IP Lookups", *IEEE INFOCOM*, pp. 1454-1463, 2001.
- [2] A. McAuley and P. Francis, "Fast routing lookup using TCAM's", in Proc. IEEE INFOCOM, pp.1382-1391, 1993.
- [3] M. A. Ruiz-Sanchez, E. W. Biersack and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE Network*, pp. 8-23, March/April 2001.
- [4] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," in Proc. ACM Sigmetrics'98 Conf., Madison, WI, pp. 111.
- [5] W. N. Eatherton, "Hardware-based Internet protocol prefix lookups", M.S. thesis, Washington Univ., St. Louis, MO, 1998. [Online]. Available at [www.arl.wustl.edu](http://www.arl.wustl.edu).
- [6] David E. Taylor, Jonathan S. Turner, John W. Lockwood, Todd S. Sproull and David B. ParLOUR, "Scalable IP Lookup for Internet routers", *IEEE Journal on Selected Areas in Communications*, Vol.21, NO.4, pp.522-533, May 2003.
- [7] N. McKeown, P. Gupta and S. Lin, "Routing lookups in hardware at memory access speeds," in Proc. IEEE INFOCOM'98 Conf., pp.1240-1247.
- [8] Nen-Fu Huang and Shi-Ming Zhao, "A Novel IP Routing Lookup Scheme and Hardware Architecture for Multigiga bit Switching routers", *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 6, pp. 1093-1104, June 1999.
- [9] Ji-Hyun Seo, Hyesook Lim, Yeo-Jin Jung and Seung-Jun Lee, "Parallel IP Address Lookup

Using Hashing with Multiple SRAMs", *한국통신학회논문지*, 28권, 2B, p138-p143, 2003

- [10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP Routing Lookups", in Proc. ACM SIGCOMM'97 Conf., Cannes, France, pp. 2535.
- [11] Rich Seifert, "The Switch book", wiley, 2000
- [12] Raj Jain, "Comparison of Hashing Schemes for Address Lookup in Computer Networks", in *IEEE Transactions on Communications*, 1989.
- [13] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, "Small Forwarding Tables for Fast Routing Lookups", Proc. ACM SIGCOMM, pp.3-14. 1997

정 여 진(Yeo-Jin Jung)

준회원



2002년 2월 : 이화여자대학교  
정보통신학과 학사  
2002년 3월~현재 : 이화여자  
대학교 정보통신학과 석사  
과정

<관심분야> 컴퓨터 네트워킹을 위한 스위치/라우터 칩의 설계, TCP/IP 관련 하드웨어 설계 등

이 보 미(Bomi Lee)

준회원



2002년 2월 : 이화여자대학교  
정보통신학과 학사  
2002년 3월~현재 : 이화여자  
대학교 정보통신학과 석사  
과정

<관심분야> 컴퓨터 네트워킹을 위한 스위치/라우터 칩의 설계, TCP/IP 관련 하드웨어 설계 등



임 혜 숙(Hyesook Lim)

정회원



1986년 2월 : 서울대학교 제어  
계측공학과 학사

1986년 8월~1989년 2월 : 삼  
성 휴렛 팩커드 연구원

1989년 3월~1991년 2월 : 서  
울대학 제어계측공학과 석  
사, 신호처리 전공

1992년 1월~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> 컴퓨터 네트워킹을 위한 스위치/라우터  
칩의 설계, TCP/IP 관련 하드웨어 설계, MPLS 등