프리픽스 그룹화를 이용한 병렬 복수 해싱 IP 주소 검색 구조

준회원 김 혜 란, 정 여 진, 정회원 임 창 훈, 임 혜 숙*

A Parallel Multiple Hashing Architecture Using Prefix Grouping for IP Address Lookup

Hye ran Kim, Yeo jin Jung *Associate Member*, Chang hoon Yim, Hye sook Lim* *Regular Members*

요 약

라우터의 주요한 기능은 들어오는 패킷의 목적지 IP 주소를 참조하여 패킷을 최종 목적지를 향하여 내 보내는 것이다. 이것을 수행하기 위해서는 주소 검색 과정이 필요하며 이 작업은 모든 패킷에 대해 실시간으로 수행되어야 하므로 라우터의 성능을 결정하는 중요한 요소가 된다. 또한 CIDR(classless inter-domain routing) IP 주소 체계를 도입하게 되면서 라우터에서는 단순 exact 매치가 아니라 가능한 모든 프리픽스 중에 가장 길게 매치하는 프리픽스를 검색하는 longest prefix match가 필요하게 되었다. 이에 따라 IP 주소 검색을 위한 알고리즘 및 구조에 관한 연구가 널리 수행되고 있으며 본 논문에서는 병렬 복수 해싱 (parallel multiple hashing)과 프리픽스 그룹화 (prefix grouping)를 이용하여 한 번의 메모리 접근으로 IP 주소 검색을 수행할 수 있는 효율적인 하드웨어 구조를 제안한다.

Key Words : Multiple hashing; longest prefix match; IP address lookup; prefix grouping; parallel CRC.

ABSTRACT

The primary function of the Internet routers is to forward incoming packets toward their final destinations. IP address lookup is one of the most important functions in evaluating router performance since IP address lookup should be performed in wire-speed for the hundred-millions of incoming packets per second. With CIDR, the IP prefixes of routing table have arbitrary lengths, and hence address lookup by exact match is no longer valid. As a result, when packets arrive, routers compare the destination IP addresses of input packets with all prefixes in its routing table and determine the most specific entry among matching entries, and this is called the longest prefix matching. In this paper, based on parallel multiple hashing and prefix grouping, we have proposed a hardware architecture which performs an address lookup with a single memory access.

Ⅰ. 서 론

인터넷 라우터의 가장 기본적인 기능은 패킷을 목적지로 전달하는 것이다. 이 기능을 수행하기 위해 라우터는 들어오는 패킷의 목적지 IP 주소를 보고 그 패킷의 목적지로 전송 할 수 있는 아웃풋 포트로 패킷을 내 보내게 된다. 이 전체적인 과정을 주소 검색 이라고 하며 이 작업은 모든 패킷에 대해 실시

* 이화여자대학교정보통신학과 SoC Design 연구실 (hlim@ewha.ac.kr)
논문번호 : KICS2004-11-296, 접수일자 : 2004년 11월 30일

간으로 수행되어야 하므로 라우터의 성능을 결정하는 중요한 요소가 된다.

과거의 IP 주소는 4개의 클래스를 갖고 각 클래스에 따라 프리픽스의 길이가 정해져 있었기 때문에 단순 exact 매치 방식을 사용하여 주소 검색에 큰 어려움이 없었다. 그러나 클래스에 따라 생기는 계층 구조 때문에 IP 주소가 낭비되고, 또한 IP 주소의 통합 (aggregation)이 어려워 주소 검색을 위한 테이블 엔트리 수가 점차 증가하는 단점이 발생하였다.

이러한 단점을 보완하기 위해 CIDR (classless inter-domain routing) 이라는 IP 주소 체계를 도입하게 되었다. CIDR은 IP 주소의 클래스를 없애고 임의의 필요한 길이만큼의 프리픽스를 네트워크 부분으로 하게 함으로써 IP 주소를 효율적으로 할당할 수 있게 하였다. 그러나 IP 주소의 클래스가 없어짐에 따라 라우터에서는 단순 exact 매치가 아니라 가능한 모든 프리픽스 길이 중에 가장 길게 매치하는 프리픽스 (longest prefix matching)를 검색하여 포트를 결정짓게 된다. Longest prefix matching은 이전에 주소 검색에 이용되던 해싱이나 바이너리 검색 같은 방법으로는 할 수 없는 매칭 방식으로 새로운 IP 주소 검색 알고리즘이 필요하게 되었다.

이에 따라 IP 주소 검색을 위한 많은 알고리즘 및 구조에 관한 연구가 진행 되고 있는데 이러한 알고리즘 및 구조의 성능을 평가하기 위해서 몇 가지 척도가 사용된다. 가장 중요한 것으로는 검색 속도를 들 수 있다. IP 주소 검색은 들어오는 모든 패킷에 필요한 과정이므로 이것을 실시간으로 수행하지 못하면 라우터가 인터넷 통신의 병목지점이 되어 링크 속도가 아무리 빨라져도 원하는 속도로 통신할 수 없게 된다. 검색 속도는 실제 테이블 룩업 과정인 메모리 엑세스 횟수에 의해 좌우되므로, 적은 수의 메모리 엑세스를 요구하는 구조가 우수한 구조라고 할 수 있다. 그 다음 중요한 것은 테이블 저장에 필요한 메모리 크기이다. 네트웍이 점차 커짐에 따라 테이블 엔트리의 수도 매우 빠르게 증가하게 되는데 이러한 정보를 효율적으로 메모리에 저장하여 전체적인 메모리 사용에 낭비가 없어야 한다. 다음으로는 라우팅 테이블 업데이트 효율을 들 수 있다. 안정적인 라우팅 테이블을 가진 라우터는 큰 문제가 되지 않지만, 변화가 큰 에지 (edge) 라우터에서는 추가적 업데이트 (incremental update)를 할 수 있는 알고리즘이 필요하다. 또한 후에 IPv6 주소 체계를 사용하게 되더라도 IPv6로의 확장이 쉬워야 하며 구현 방식의 유연성도 갖춰야 한다.

본 논문에서 제안하는 구조는 병렬 복수 해싱 (parallel multiple hashing)과 프리픽스 그룹화 (prefix grouping)를 이용한 검색 방법으로 검색 시간이 빠르고 효율적인 메모리 크기를 갖는 구조이다. 본 논문의 구성은 다음과 같다. 2장에서는 기존의 검색 방법에 대해 살펴보고, 3장에서는 제안하는 구조에 대해 설명한다. 4장에서는 실제 라우터에서 사용된 라우팅 데이터를 사용하여 제안하는 구조의 성능을 평가해보고 5장에서 결론을 맺는다.

Ⅱ. 기존의 검색 방법

IP 주소 검색을 위해 많은 연구가 수행되고 있다. 트리 구조는 프리픽스 검색을 위해 가장 널리 사용되는 것으로 먼저 기본적인 바이너리 트리[1]를 들 수 있다. 0이면 왼쪽, 1이면 오른쪽으로 한 비트 (bit), 한 비트씩 가지를 뻗어 나가는 방식으로 가장 긴 프리픽스를 찾기 위해 가지의 끝까지 따라가면서 일치하는 노드가 있을 때까지 검색하는 방식이다. 이 방식은 가장 긴 프리픽스를 검색하는데 매우 유용한 방식이지만, 트리를 구성함에 있어 빈 노드가 있게 되어 메모리 효율이 떨어질 뿐 아니라, 메모리 검색 횟수가 트리의 최대 높이인 32까지 되는 단점이 있다. 이런 바이너리 트리의 단점을 개선하기 위해 트리의 높이를 줄일 수 있는 방법이 연구 되었는데, 하나의 차일드를 가지면서 비어 있는 노드를 제거한 방법[2]과 한 번에 한 비트 (bit) 씩 보는 것이 아니라 비트를 확장하여 저장한 후 한 번에 다수의 비트를 비교하면서 트리 검색을 진행하는 알고리즘, 또는 이 두 가지를 결합한 방식[3]이 제안되었다. 그러나 기본적으로 트리 구조는 빈 노드를 가지게 되는 단점을 극복할 수는 없다.

또 다른 트리 구조는 프리픽스를 레인지로 전환하여 트리를 구성한 구조[4]이다. 프리픽스를 최대 길이로 확장하고, 한 프리픽스당 최대 두개의 엔트리를 차지하게 함으로서 IP 주소 검색에서의 길이 차원을 없앤 방식으로 매우 우수한 구조이다. 프릭픽스를 시작점과 끝점으로 확장한 후 크기 순으로 정렬하여 바이너리 검색을 수행하게 하였다. 이렇게 함으로써 바이너리 검색이 가능하여 검색 횟수를 줄 일 수 있지만, 모든 엔트리에 해당하는 best matching prefix (BMP)를 미리 계산하여 저장하여야 하는 단점이 있다.

[5]는 하드웨어를 이용한 구조로써 프리픽스의 대부분을 차지하고 있는 24 비트까지 프리픽스를 확장시켜 메인 메모리에 저장하고 그것보다 길이가 더

긴 프리픽스는 서브 테이블에 저장하는 방식을 제안하였다. 이렇게 함으로써 대부분의 주소는 한 번의 메모리 접근으로 검색을 가능하게 하고 그 이상의 길이에 대해서는 2번의 검색을 요구함으로써 좋은 성능을 갖고 있지만 프리픽스를 24비트로 확장함에 있어 매우 큰 메모리가 필요하게 된다. 이를 개선하기 위하여 테이블을 프리픽스 길이 16-8-8 등으로 나눠 여러 단계에 걸쳐 검색하는 방식[6]이 있다. 이것은 바로 이전의 구조보다 검색 횟수가 약간 증가하였지만 메모리 효율이 증가하였다.

그 다음 TCAM (ternary content addressable memory)을 이용한 방식[7]이 있다. TCAM은 주소 검색에 가장 널리 사용되는 방식으로써 프리픽스 매치를 위한 특수한 메모리 구조이다. 모든 엔트리를 동시에 비교하여 가장 길게 매치하는 프리픽스를 찾아 주는 구조로써 검색을 한 번에 끝낼 수 있다는 장점을 갖고 있다. 그러나 일반적인 메모리보다 같은 데이터를 저장하는데 더 큰 면적을 차지하고, 전력소모도 많을 뿐만 아니라 가격도 비싸다는 단점이 있다.

마지막으로 해쉉을 이용한 검색 구조가 있다. 해쉉은 exact 매치에 적합한 방식이므로 프리픽스에 적용하기 위해서는 프리픽스 길이에 대한 고려가 있어야 한다. [8]은 해쉉과 바이너리 검색을 결합한 구조로써 프리픽스 길이에 대해 바이너리 검색을 하고 같은 프리픽스 길이를 갖는 엔트리에 대해 해쉉을 수행하여 검색을 하는 방법이다. 그러나 프리픽스에 대해서는 일반적인 바이너리 검색이 불가능 하므로 그것을 위해 마커와 BMP를 계산하여 테이블 엔트리에 저장해 놔야 한다. 이것은 메모리 크기를 증가시키고 계산량을 커지게 하는 단점을 갖는다. [9]는 프리픽스 길이에 따라 테이블을 만들어 그것을 동시에 검색하는 방법을 제안하였다. 저장 시에 충돌(collision)이 발생한 경우는 그것을 보조 테이블에 저장하여 길이 별 테이블에서 매치하는 엔트리가 없는 경우 보조 테이블을 바이너리 검색하여 매치하는 엔트리를 찾게 하였다. 이 방법은 첫번째 테이블 검색에서 매치하는 엔트리가 없을 때 보조 테이블 검색을 진행하므로 검색 횟수가 증가하고 메인 테이블을 프리픽스 길이 별로 가져야 한다는 단점이 있다. [10]은 프리픽스 길이 별 테이블을 갖고 그것을 동시에 검색 하고 길이 별 테이블 저장시 충돌이 일어나면 그것을 작은 크기의 TCAM에 저장하는 방식을 제안하였다. TCAM에 저장함으로써 길이 별 테이블과 TCAM을 동시에 검색 할 수 있고 그리하여 한 번의 메모리 접근 시간으로 주소 검색을 마칠 수 있게 하였다. 그러나 이 방법은 가능한 프리픽스 길이 종류에 해당하는 개수만큼의 메모리를 가져야 한다는 단점이 있다.

본 논문에서는 검색 시간에 있어서는 TCAM 과 같은 성능을 보이면서도, 요구되는 메모리 크기에 있어서나, 메모리의 수에 있어서, 전적으로 TCAM을 사용하는 것 보다 매우 우수한 성능을 보이는 구조를 제안한다.

Ⅲ. 제안하는 검색 구조

복수 해쉉 (multiple hashing)은 해쉉 적용시 하나의 해쉬 인덱스 (hash index) 대신 복수개의 해쉬 인덱스 (hash index)를 사용함으로써 해쉉시 발생할 수 있는 충돌을 여러 테이블에 분산시키는 방식이다.

3.1 프리픽스 그룹화 (Preifx Grouping)

해쉉은 고정된 길이에 대해 검색하는 방식으로 주소 검색에 해쉉을 이용하기 위해서는 프리픽스의 길이를 미리 알아야 한다. [10]에서는 프리픽스 검색에 해쉉을 적용하기 위해 각 프리픽스 길이 마다 테이블을 할당하였다. 그러나 그렇게 하면 테이블 수가 너무 많아지게 된다. 게다가 멀티 해쉉을 적용하면 사용하는 해쉉 함수의 수에 따라 테이블의 수가 2배, 3배로 증가하기 때문에 더 큰 문제가 된다. 따라서 본 논문에서는 프리픽스 그룹화를 제안하였다. 그림 1은 라우터의 프리픽스 분포를 나타낸 표로서, 본논문에서는 대부분의 라우터 데이터가 이와 같은 분포를 보임을 이용하여 여러 프리픽스를 적당한 길이로 묶어서 하나의 테이블에 저장함으로써 테이블 수를 줄 일 수 있게 하였다. 다음의 표를 보면 프리픽스가 특정 프리픽스 길이에 대해 많이 분포해 있는 것을 볼 수 있다. 이것을 바탕으로 프리픽스를 묶어

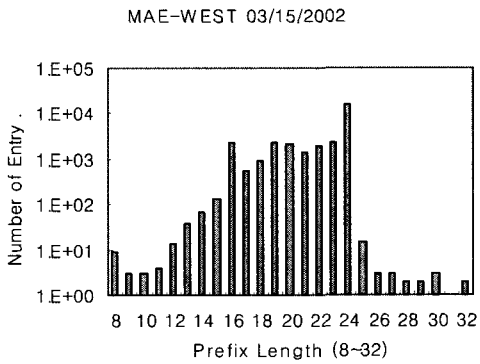


그림 1. 프리픽스 분포

그룹을 나누었다. 그룹은 4개로 나누었는데 프리픽스 길이가 8~15, 16~19, 20~23, 24~32인 그룹으로 나누어 각 그룹의 첫번째 프리픽스 길이를 그 그룹의 해쉬 함수의 입력으로 사용한다. 프리픽스 분포가 많은 길이를 그룹의 시작 프리픽스로 함으로써 해쉬 적용 시 모든 길이에 대해 인풋으로 들어가게 하여 해쉬의 효율을 높이게 하였다. 예를 들어 프리픽스의 길이가 21이라면 이 프리픽스는 3번째 그룹인 20~23 테이블에 들어가게 되고 해쉬 시에는 20 비트 까지만 해쉬의 인풋으로 들어간다. 이렇게 그룹을 나누어 해쉬을 하게 되면 프리픽스 길이들 마다 각각 해쉬 함수를 통과시키는 것보다 더 많은 충돌이 발생하게 되는데, 4장에서 보이려는 바와 같이 복수의 해쉬을 이용하기 때문에 오버플로우 수는 크게 증가하지 않음을 알 수 있었다.

3.2 병렬 CRC를 사용한 복수 해쉬

복수 해쉬은 해쉬에서 피할 수 없는 충돌을 여러 테이블에 분산시킴으로써 충돌을 줄이는 역할을 수행한다. 그러나 복수 해쉬은 해쉬에서 발생하는 충돌을 완화시키는 것에 국한되므로, 해쉬 함수의 선택은 매우 중요한 문제이다.

[8]에서는 완전 해쉬 함수 (perfect hashing function)를 가정하였지만 완전 해쉬 함수를 찾는 것은 시간이 매우 많이 걸리기 때문에, 실제적으로 이용하기는 어렵다고 볼 수 있다. 본 논문에서는 해쉬 함수로 완전 해쉬 함수에 가까운 성능을 낸다고 알려진 CRC 함수를 이용하였다. [10]에서 CRC-32 해쉬 함수 하나를 이용하여 모든 테이블의 인덱스를 얻는 방법을 소개하였다. 목적지 주소를 한 클럭 (clock) 마다 상위 비트부터 한 비트씩 CRC 하드 웨어에 들어가게 하고 D 클럭 사이클마다 CRC 레지스터로부터 프리픽스 길이 D에 해당하는 해쉬 값을 구하게 하였다. 그러나 이런 방식으로 해쉬 값을 구하면 로직 구현이 간단하다는 장점이 있지만, CRC 함수에 한 비트씩 들어가기 때문에, 해쉬 입력이 모두 들어갈 때까지 여러 클럭 사이클 (clock cycle)이 필요하게 된다. 때문에 본 논문에서는 모든 비트가 동시에 함수의 인풋으로 들어 갈 수 있는 병렬 CRC 함수를 이용하여 거기에 필요한 시간을 단축 할 수 있게 하였다. 병렬 CRC는 정해진 비트 수에 맞게 구현하였다.

병렬 CRC를 이용하면 정해진 길이의 비트 수에 따라 회로가 다르게 구성 되므로 하나의 CRC 함수를 여러 길이에 대해 쓸 수 없고 회로가 좀 더 복잡해진다는 단점이 있지만 모든 결과 값이 동시에 나온 다는 장점을 가진다.

그림 2는 본 논문에서 사용된 CRC 함수의 polynomial 이다. 프리픽스 검색을 위해 길이에 따라 몇 개의 그룹으로 해쉬 그룹을 나누었는데 병렬 CRC 함수를 사용함으로써 각 해쉬 그룹마다 하나의 CRC 함수가 필요하게 되고 이 그룹들에 대해 동시에 해쉬을 하여 검색하는 병렬 검색을 적용하였다. 그룹은 프리픽스 길이에 따라 8, 16, 20, 24의 4개로 나누었으며 그룹 8, 16, 24는 길이에 맞는 CRC-8, CRC-16, CRC-24 함수를 이용하였고, 그룹 20의 경우는 길이에 맞는 CRC함수가 없으므로 CRC-24를 이용하여 20 비트까지만 들어간 결과를 구하였다. 그림 3은 parallel CRC-8을 사용한 경우 해쉬 함수 연산식을 나타낸 것이다. 32bit의 주소가 입력으로 오면 그림과 같은 연산을 통해서 해쉬 함수의 결과를 구하고, 해쉬 인덱스로 7bit을 사용하는 경우 그림 3에서 나타내듯이 해쉬 함수 결과중 7비트를 선택하여 해쉬 인덱스로 이용한다.

3.3 오버플로우 테이블

오버플로우 테이블은 메인 테이블이 꽉 찬 경우 프리픽스를 저장하는 테이블로 TCAM으로 구성된다.

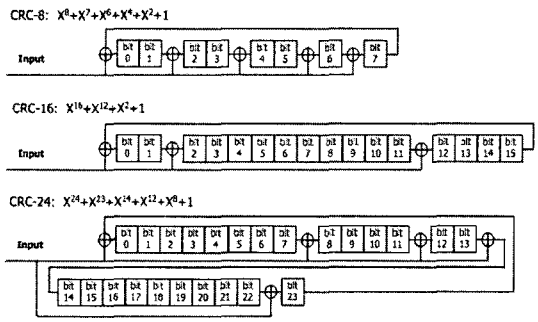


그림 2. 여러가지 CRC 함수들

```
hashout[0]=adr[24]^adr[25]^adr[27]^adr[30]^adr[31];
hashout[1]=adr[25]^adr[26]^adr[28]^adr[31];
hashout[2]=adr[24]^adr[25]^adr[26]^adr[29]^adr[30]^adr[31];
hashout[3]=adr[25]^adr[26]^adr[27]^adr[30]^adr[31];
hashout[4]=adr[24]^adr[25]^adr[26]^adr[28]^adr[30];
hashout[5]=adr[25]^adr[26]^adr[27]^adr[29]^adr[31];
hashout[6]=adr[24]^adr[25]^adr[26]^adr[28]^adr[31];
hashout[7]=adr[24]^adr[26]^adr[29]^adr[30]^adr[31];

Ex) hash_index1[6:0]={hashout[6],hashout[5],hashout[4],hashout[3],hashout[2],hashout[1],hashout[0]};
    hash_index2[6:0]={hashout[7],hashout[6],hashout[5],hashout[4],hashout[3],hashout[2],hashout[1]};
```

그림 3. CRC-8 경우의 예

TCAM으로 구성하기 때문에 이 테이블의 검색은 한 번의 메모리 접근으로 가능하게 된다. TCAM은 특수한 메모리로 가격이 비싸고 저장 용량도 크기에 비해 효율적이지 않지만 본 논문에서 제안한 구조는 오버플로우가 발생한 엔트리만을 저장함으로 효율적이다. 다음 장에서 보이려는 바와 같이 41584개의 프리픽스를 저장하기 위해 약 389개의 TCAM 엔트리 정도만 있으면 오버플로우 테이블을 구성 할 수 있다.

3.4 라우팅 테이블 구성

제안하는 구조는 프리픽스를 저장하기 위해 복수개의 테이블을 가지며 그림 4 와 같은 버켓으로 구성된다. 먼저 한 버켓에 몇 개의 엔트리가 저장되어 있는지가 표시되고, 각 엔트리에는 저장된 프리픽스 길이와 프리픽스, 그리고 그에 해당하는 아웃 포트 포인터 (out port pointer) 가 그 버켓에 저장되어 있는 로드의 수만큼 반복되어 저장된다. 그림 4는 버켓 당 3개의 엔트리를 가질 경우의 버켓 구조이다.

테이블을 구성하기는 방법은 그림 5과 같다. 그림 5는 두개의 해쉬 인덱스를 갖는 경우의 구조이다. 먼저 목적지 프리픽스를 그에 해당하는 그룹의 CRC 함수를 통과시켜서 두개의 해쉬 인덱스를 얻는다. 그렇게 얻어진 해쉬 인덱스를 통해 각 각의 테이블에 접근하고, 두 개의 테이블의 Number of Items 필드를 비교하여 더 적은 수의 로드가 걸려 있는 테이블에 프리픽스를 저장한다. 만약 두개 테이블에 모든 엔트리가 저장되어 있다면 그 프리픽스는 오버플로우 테이블에 저장된다.

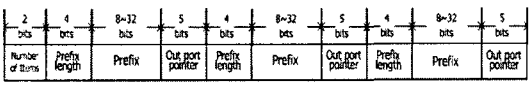


그림 4. 버켓 구조

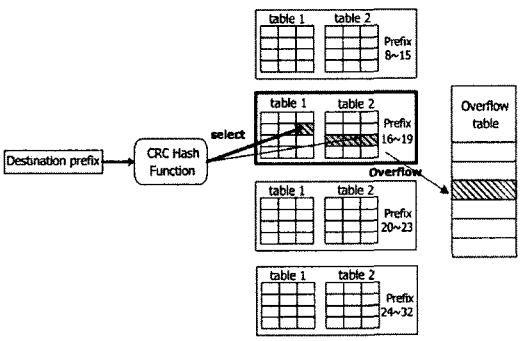


그림 5. 라우팅 테이블 구성 과정

3.5 라우팅 테이블 검색

검색은 각 그룹과 오버 플로우 테이블에 대해 병렬로 수행된다. 그림 6은 검색 과정을 나타낸 그림이다. 먼저 들어오는 패킷의 목적지 주소를 해쉬 함수에 통과시킨다. 해쉬 함수를 통과시키면 두개의 테이블 인덱스가 나오고 그 인덱스에 따라 두개의 테이블에 접근하여 엔트리를 비교한다. 엔트리를 비교하여 매치하는 엔트리가 있으면 그 곳에 저장된 포트 넘버가 priority 인코더로 전해진다. 4개의 그룹을 검색하는 것과 동시에 오버플로우 테이블도 검색한다. 오버 플로우 테이블에 매치되는 검색 결과가 있으면 그 결과도 priority 인코더에 전달한다. 모든 테이블의 검색이 끝나면 priority 인코더에서는 각 테이블의 결과 중에서 가장 프리픽스가 길게 매치되어 나온 결과를 최종적으로 선택하고 이렇게 함으로써 검색을 끝마치게 된다.

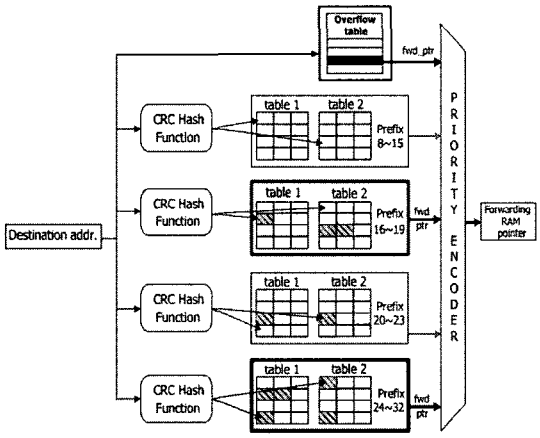


그림 6. 라우팅 테이블 검색 과정

Ⅳ. 시뮬레이션 결과

본 논문에서는 실제 라우터의 8개 데이터를 이용해 시뮬레이션을 수행 하였다. 각 데이터에 대해 3가지 경우를 실험하였다. 테이블 수를 1, 2, 3개로 했을 경우에 대해 실험하였는데, N 개의 프리픽스를 저장하기 위해 N개의 버켓을 사용하였고 각 버켓에는 3개의 로드를 저장하므로 각 실험의 메모리 효율은 1/3로 같게 하였다.

표 1은 다음과 같은 엔트리 수를 가지는 데이터에 대해 실험 한 결과 필요한 메모리 크기와 발생한 오버플로우 수를 나타낸 표이다. 엔트리 수에 따라 메모리 크기는 증가하며 같은 메모리 효율을 갖도록 테이블 엔트리 수를 정했음에도 테이블 수에 따라

오버 플로우 수가 크게 차이가 나는 것을 볼 수 있다. 이 결과는 복수 해쉬이 효과적임을 보여주고 있다.

표 1. 시뮬레이션 결과

라우팅 데이터	엔트리수	메모리 크기 (KB)	Table 수		
			1	2	3
Funet	41584	724	1733	389	441
MaeEast1	39902	724	612	41	14
MaeEast2	38470	462	4541	37	9
MaeWest1	29584	459	1946	414	167
PacBell	20637	361	342	33	3
Aads	20328	361	345	10	1
MaeWest3	15050	231	291	1	0
MaeWest2	14636	205	379	1	0

표 2는 본 논문에서 제안하는 구조와 다른 알고리즘들의 성능을 비교한 표이다. 30000여 개의 프리픽스를 저장 한 것에 대해 실험하였으며 제안하는 알고리즘은 MaeWest1 데이터를 비교하였다. 메모리 접근 횟수와 메모리 크기를 비교 하였는데 본 논문에서 제안한 구조는 메모리 접근 횟수에서 TCAM과 같은 1,1로 매우 우수한 성능을 보이며 메모리 크기도 다른 알고리즘과 비교했을 때 비슷한 크기를 필요로 한다. 단 본 논문에서 제안하는 구조는 약 414개의 TCAM 엔트리를 필요로 하게 된다.

표 2. 다른 알고리즘과의 비교

Address lookup scheme	Number of memory accesses (Minimum, Maximum)	Required SRAM (Kbytes)	Required TCAM (number of entries)
Range search [4]	1, 16	376	0
DIR-24-8 [5]	1, 2	33,000	0
DIR-21-3-8 [5]	1, 3	9,000	0
Huang [6]	1, 3	450 ~ 470	0
TCAM-based [7]	1, 1	0	30,000
Parallel Hashing [9]	1, 5	189	0
SFT [11]	2, 9	150 ~ 160	0
Proposed Architecture	1, 1	459	414

V. 결 론

본 논문에서는 효율적인 주소 검색 구조를 제안하였다. 멀티플 해쉬를 이용하여 충돌 현상을 줄이고, 프리픽스 그룹화를 이용하여 적은 수의 테이블을 요구하는 구조이다. 본 구조는 메모리 접근 횟수가 1로 매우 우수한 성능을 보이며 메모리 크기 면에서도 459KB로 좋은 성능을 보인다. 본 구조에서는 적은 엔트리의 TCAM이 사용되는데, 고속 포워딩을 위해 설계된 스위치 ASIC 에 매크로 셀의 형태로 들어갈 수 있는 크기이다. 또한 업데이트를 프리픽스 검색과 같은 방식으로 진행 할 수 있어 추가적 업데이트 (incremental update)가 가능한 구조이다.

참 고 문 헌

[1] 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.

[2] D. R. Morrison, "PATRICIA Practicla Algorithm to Retrieve Information Coded in Alphanumeric." J.ACM, vol. 15, no. 4, Oct. 1968, p.514-34.

[3] S. Nilsson and G. Karlsson "IP-Address Lookups using LC-Tries," IEEE JSAC, June 1999, vol. 17, no.6, pp 1083-92.

[4] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway multicolumn search," IEEE/ACM Transactions on Networking, Vol. 7, No. 3, pp.650-662, June 1999.

[5] N. McKeown, P. Gupta, and S. Lin, "Routing lookups in hardware at memory access speeds," in Proc. IEEE INFOCOM, 1998, pp.12401247.

[6] N.-F. Huang and S.-M. 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.

[7] A. McAuley and P. Francis,"Fast routing lookup using CAM's", in Proc. IEEE INFOCOM, 1993, pp. 13821391.

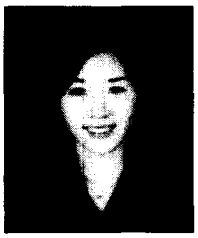
[8] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in Proc. ACM SIGCOMM Conf., Cannes, France, 1997, pp. 2535

[9] Hyesook Lim, Ji-Hyun Seo and Yeo-Jin Jung, "High Speed IP Address Lookup Architecture Using Hashing", IEEE Communications Letters, Oct. 2003, pp.502-504

[10] 정여진, 이보미, 임혜숙, "A Parallel Multiple Hashing Architecture for IP Address Lookup",한국통신학회 논문지, Vol.29 , No.2B, pp.15-166, Feb 2004

[11] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups", in Proc. ACM SIGCOMM, 1997, pp.3-14.

김 혜 란(Hye ran Kim) 준회원

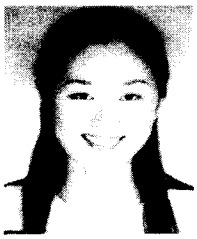


2004년 2월 이화여자대학교 정보통신학과 학사

2004년 3월~현재 이화여자대학교 정보통신학과 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

정 여 진(Yeo jin Jung) 준회원



2003년 2월 이화여자대학교 정보통신학과 학사

2005년 2월 이화여자대학교 정보통신학과 석사

2005년 3월~현재 삼성전자

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

임 창 훈(Chang hoon 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 디지털미디어연구소, 수석연구원

2003년 9월~현재 건국대학교 인터넷미디어공학부, 조교수

<관심분야> 멀티미디어 통신, 비디오 압축, 디지털 영상처리, 멀티미디어 네트워크

임 혜 숙(Hye sook Lim) 정회원

1986년 2월 서울대학교 제어계측공학과 학사

1986년 8월~1989년 2월 삼성휴렛 팩커드 연구원

1991년 2월 서울대학교 제어계측공학과 석사

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

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

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계