韓國컴퓨터情報學會 論文誌
第15卷 第12號, 2010. 12.

2010-15-12-3-5

병렬 TCAM 기반의 IP 주소 검색에서 신속한 프리픽스 삭제

김 진 수*, 김 정 환*

Fast Prefix Deletion for Parallel TCAM-Based IP Address Lookup

Jinsoo Kim*, Junghwan Kim*

요 약

본 논문에서는 병렬 TCAM을 기반으로 한 IP 주소 검색 구조에서 프리픽스를 신속하게 삭제하는 기법을 제안한다. 기존의 삭제 기법들은, 프리픽스의 순서 유지와 가용 메모리 공간의 연속성 유지를 위해 한 차례 이상의 메모리 이동을 필요로 하고 있다. 본 기법에서는 삭제 시 TCAM에서 실제 메모리를 이동하지 않고, SRAM으로 구현된 스택을 사용하여 삭제된 프리픽스의 주소를 이 스택에 저장한다. SRAM은 TCAM에 비해 접근시간이 매우 짧기 때문에, 제안된 기법이 신속한 갱신을 수행할 수 있다. 그리고 실제 사용되는 포워딩 테이블과 갱신 내용을 적용한 실험을 통해, 프리픽스 삽입과 삭제에 따른 메모리 접근 시간의 관점에서 제안된 기법의 성능을 평가한다. 또한 상대적으로 매우 작은 크기의 스택을 사용해도 좋은 성능을 나타내고 있음을 실험 결과를 통해 제시한다.

Abstract

In this paper, we propose a technique which makes it faster to delete prefixes in an IP address lookup architecture based on parallel TCAMs. In previous deletion schemes, more than one memory movement is needed for the prefix ordering and keeping the available memory space consecutive. For deletion, our scheme stores the address of the deleted prefix in a stack implemented by SRAM instead of actual movement in TCAM. Since SRAM has very short latency compared to TCAM, the proposed scheme can accomplish fast updating. From the experiment with the real forwarding table and update trace, we evaluate the performance of our scheme in terms of the memory access time for the prefix insertion and deletion. The experiment result also shows good performance with considerably small size of stack.

▸ Keyword : IP 주소 검색(IP address lookup), 신속한 프리픽스 삭제(fast prefix deletion), 병렬 TCAM(parallel TCAM)

• 제1저자, 교신저자 : 김진수
• 투고일 : 2010. 08. 03, 심사일 : 2010. 08. 24, 게재확정일 : 2010. 09. 20.
* 건국대학교 컴퓨터응용과학부 교수
※ 이 논문은 2008년도 건국대학교 학술진흥연구비 지원에 의한 논문임.

Ⅰ. 서 론

인터넷 트래픽의 급속한 증가에 따라 고성능 라우터에 대한 필요성도 점점 증대하고 있다. 라우터의 핵심 기능은 패킷을 최종 목적지 방향으로 전달하는 것이다. 라우터는 입력 링크로 들어온 패킷을 다음에 어느 노드로 보낼 지를 결정해야 한다. 이 결정을 위해 입력 패킷의 목적지 IP 주소를 포워딩 테이블에서 검색하고, 검색된 정보를 바탕으로 다음 홉(next hop)과 출력 링크를 결정한다.

인터넷 초창기의 IP 주소 검색은 고정된 길이의 프리픽스를 비교하는 완전 일치(perfect matching) 방식을 사용하여 단순하게 처리될 수 있었다. 그러나 가변 길이의 프리픽스를 허용하는 CIDR(classless interdomain routing)[1]의 등장 이후, IP 주소의 검색은 목적지 주소에 매치(match)되는 여러 프리픽스 중 최장 매치 프리픽스(LMP, Longest Matching Prefix)를 가장 적합한 프리픽스로서 선택하는 방식으로 처리하게 되었다. 이에 따라 IP 주소 검색이 상당히 복잡하게 되었고, IP 주소 검색에 대한 성능 향상을 위한 많은 기법들이 소프트웨어와 하드웨어 접근방식으로 제안되어 왔다[2, 3].

하드웨어 접근 방식의 IP 주소 검색 기법은 대부분 TCAM (Ternary Content Addressable Memory)을 기반으로 제안되었다. TCAM은 각 메모리 셀에 0과 1의 상태뿐만 아니라 * (don't care) 상태도 저장 가능하도록 CAM을 확장한 완전 연관 기억장치이다[4]. TCAM은 이 don't care 상태를 활용하여 가변 크기의 프리픽스를 저장할 수 있고, 연관 기억장치의 특성을 통해 하나의 메모리 사이클(cycle) 동안에 LMP를 찾을 수 있다. TCAM은 다수의 메모리 사이클을 필요로 하는 다른 IP 주소 검색 방식 보다 고속의 검색을 제공할 수 있기 때문에 지속적인 관심을 받고 있다. 그러나 보통 TCAM은 LMP의 선택을 위해서, 프리픽스 길이의 역순으로 프리픽스를 저장하고, 우선순위 인코더(PE, Priority Encoder)가 있어 매치되는 프리픽스들 중 제일 상단에 위치한 것을 최종 선택한다.

IP 포워딩 테이블에는 모든 현행 프리픽스들에 대한 정보가 포함되어 있다. 인터넷 백본 라우터에서 포워딩 테이블은 1초에 보통 수백에서 수천 개의 프리픽스 정보가 변경될 수 있어 고속의 갱신 속도가 요구된다. TCAM 기반의 주소 검색 방식 대부분은 프리픽스 길이 등과 같은 특정한 순서를 지속적으로 유지해야만 하므로, 한 번의 포워딩 테이블 갱신이 프리픽스의 메모리 이동을 수차례 유발할 수 있다. 이러한 메모리 이동은 IP 주소 검색의 지연을 초래하여 성능에 부정적인 영향을 준다. 포워딩 테이블의 갱신은 프리픽스의 출력 링크 변경, 프리픽스의 삽입, 프리픽스의 삭제 등으로 구분할 수 있는데, 프리픽스의 출력 링크 변경은 일반적으로 인덱스 값만 변경하면 된다. 따라서 프리픽스 삽입과 삭제 등의 갱신에 따는 메모리 이동을 최소화하여 신속하게 처리하는 것이 고성능 라우터의 개발에 있어 매우 중요한 사안 중 하나이다.

본 논문에서는 병렬 TCAM을 기반으로 한 IP 주소 검색 구조에서 프리픽스를 보다 신속하게 삭제하는 기법을 제안한다. 제안된 기법은 프리픽스를 삭제할 때 가용 공간의 연속성 유지를 위한 메모리 이동을 제거하여, 빠른 갱신을 수행하도록 한다. 논문의 구성은 2절에서 TCAM 기반 IP 주소 검색에 대해 소개하고, 기존에 제안된 포워딩 테이블의 갱신 기법들에 대해 서술한다. 3절에서 병렬 TCAM 기반의 IP 주소 검색 구조에 대해 기술한다. 4절에서는 IP 포워딩 테이블의 갱신 방법으로서 프리픽스의 삽입과 삭제 알고리즘에 대해 설명한다. 그리고 5절에서는 제안한 기법의 성능을, 실제 사용되는 포워딩 테이블을 적용하여, 기존의 기법과 비교 평가한다. 마지막으로 6절에서 결론을 맺는다.

Ⅱ. 관련 연구

TCAM은 하드웨어적으로 병렬 검색을 할 수 있을 뿐 아니라 don't care 상태도 저장할 수 있어 IP 주소 검색 분야에 널리 사용되어 왔다. IP 포워딩 테이블에 들어 있는 다양한 길이의 프리픽스들은 이러한 TCAM에 효과적으로 저장된다.

그림 1은 단일 TCAM을 사용하여 프리픽스 검색을 하는 예를 보여준다. 100101011을 검색하는 경우 p0와 p2가 매치되며, 이중 가장 긴 프리픽스인 p0가 최종 선택된다. 이때 가장 긴 프리픽스, 즉 LMP는 PE(Priority Encoder)에 의해 TCAM 내에서 가장 위쪽에서 매치된 프리픽스로 결정된다. 따라서 이러한 방식의 TCAM은 동시에 매치되는 프리픽스들에 대해 항상 긴 것이 위쪽에 위치되어야 한다. 이러한 위치의 제한은 프리픽스 갱신 시 어려움을 초래한다.

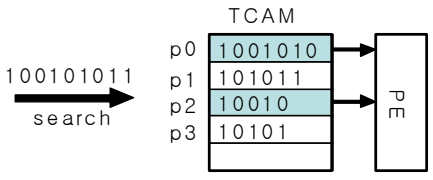

그림 1. 단일 TCAM 기반 구조
Fig. 1. Single TCAM-based Architecture

TCAM 기반 IP 주소 검색에서 검색 자체 속도의 향상 뿐만 아니라 포워딩 테이블의 효율적인 갱신을 위한 다양한 연구들이 진행되어 왔다[5-8]. 하드웨어의 추가 변경에 의해 프리픽스 길이와 무관하게 포워딩 테이블을 구성하도록 한 기법도 있지만[5], 대부분 연구에서는 프리픽스들을 TCAM에 효율적으로 저장하고 관리하여 갱신 과정에서 메모리 이동을 줄이는 방법을 제안하였다.

프리픽스들을 TCAM에 저장하는 가장 단순한 방법으로 모든 프리픽스에 대해 길이 순서를 적용하는 PLO(Prefix Length Ordering)와 프리픽스 간의 체인 순서에 의한 CAO (Chain Ancestor Ordering)가 제안되었다[6].

TCAM 내에서 프리픽스 간의 순서 유지는 테이블 갱신 시에 어려움을 가져온다. 즉, 새로운 프리픽스 삽입 시에 해당하는 위치를 다른 프리픽스가 이미 점유하고 있으면, 연쇄적인 메모리 이동이 발생할 수 있다. 삭제 시에도 해당 위치를 그대로 비워둘 경우 가용 공간 관리에 어려움이 있으며, 따라서 1회 이상의 메모리 이동이 불가피하다. L을 프리픽스 최대길이, D를 체인의 길이라고 할 때, PLO와 CAO 기법에서 프리픽스 갱신에 따른 메모리 이동 회수는 최악의 경우 각각 $L/2$와 $D/2$이다. [6]의 실험을 통해, PLO와 CAO 알고리즘에서 프리픽스 갱신은 각각 4.1과 1.02번 메모리 이동을 유발하는 것으로 평가되었다. [7]에서는 프리픽스들의 레벨별 특성을 활용하고 가용 공간을 효율적으로 배치하여, D를 체인의 길이라고 할 때, 프리픽스 갱신에 따른 최악의 경우 메모리 이동 회수를 $(D-2)/2$로 개선하였다. 또한 가용 공간 관리를 위해 프리픽스 삭제 시 최소 1회의 메모리 이동이 필요하다.

서로 중첩되는 프리픽스를 서로 다른 TCAM에 배치하도록 테이블을 분할하여 저장하는 방식[8]은 순서 유지에 따른 문제를 피할 수 있다. 프리픽스를 삽입할 경우, 그 프리픽스와 분할 내의 모든 프리픽스들이 서로 중첩되지 않는 분할에 삽입한다. 이 방식에서는 프리픽스 삽입 시 분할들 간의 메모리 이동이 필요 없기 때문에 상당히 효율적이다. 그러나 프리픽스 삭제 시, 가용 공간의 연속성을 유지하기 위해 한 번의 메모리 이동이 수반된다. 즉 삽입 시 메모리 이동은 0회, 삭제 시 메모리 이동은 1회 발생한다. 본 논문은 [8]에서 제안된 병렬 TCAM 기반의 IP 주소 검색 방식을 개선하여, 프리픽스 삭제에 따른 메모리 이동을 제거함으로써 보다 신속하게 갱신을 수행한다.

III. 병렬 TCAM 기반 IP 주소 검색 구조

병렬 TCAM 기반의 IP 주소 검색 구조[8]는 여러 개의 TCAM으로 구성되며, 각각의 TCAM에는 서로 중첩되지 않는 독립적인 프리픽스들로만 저장하도록 하여 순서 유지의 필요성을 배제한다. 예를 들어, 그림 1의 예에서 p0와 p2 프리픽스는 서로 다른 TCAM에 저장하며, 마찬가지로 p1과 p3도 중첩되므로 서로 다른 TCAM에 저장하는 방식이다.

이러한 병렬 TCAM 기반의 IP 주소 검색 구조는 그림 2와 같다. T0~T6의 7개 TCAM 각각은 중첩되지 않는 프리픽스만을 저장한다. 대부분의 프리픽스들은 이들 7개의 TCAM에 저장 가능하지만, 어떤 프리픽스들은 이들 TCAM 모두에 중첩된 프리픽스가 존재하여 어디에도 저장할 수 없는 경우가 생긴다. 그러한 프리픽스의 경우 T7에 저장하며, T7은 기존 단일 TCAM 방식처럼 중첩된 프리픽스의 저장을 허용한다. T7에 저장되는 프리픽스는 수백 개 이내의 극소수인 것으로 실험에서 평가되었다.

검색하고자 하는 IP 주소가 들어오면 매치되는 프리픽스가 어느 TCAM에 들어있는지 알 수 없기 때문에 모든 TCAM에 대해 병렬적으로 검색이 수행된다. 다수의 TCAM에서 매치가 발생할 수 있으며, 이 때 최적의 매치인 LMP을 고르기 위해 selector 하드웨어가 존재한다. selector는 매치된 프리픽스의 길이 정보를 비교하여 가장 긴 프리픽스를 선택한다. 비교 대상은 최대 8개이므로 단일 TCAM의 PE에 비해 매우 적은 동작 지연시간을 갖는다.

단일 TCAM 기반 구조는 PE에서 상당한 지연 시간을 갖는 것으로 알려져 있으며, 이에 비해 병렬 TCAM 기반 구조는 PE가 필요하지 않으므로 매우 빠른 동작 시간을 갖는다고 할 수 있다. T7의 경우 PE를 필요로 하지만, 앞서 언급한 것처럼 저장되는 프리픽스가 극소수이기 때문에 매우 작은 용량으로 만들 수 있으며, 따라서 PE의 크기도 작다.

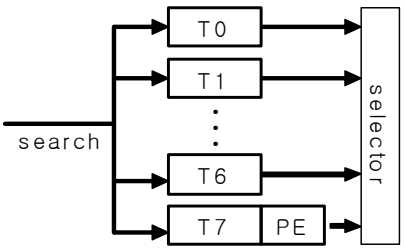

그림 2. 병렬 TCAM 기반 구조
Fig. 2. Parallel TCAM-based Architecture

병렬 TCAM 기반 구조는 또한 테이블의 갱신 속도가 매우 빠른데, 이는 앞서 언급한 것처럼 TCAM 내의 프리픽스 간의 순서 유지가 필요치 않기 때문이다.

새로운 프리픽스의 삽입 시 해당 프리픽스로 T0~T6를 검색하여 매치가 발생하지 않는 TCAM을 찾는다. 매치가 발생하지 않았다는 것은 중첩된 프리픽스가 없다는 의미로써 그러한 TCAM 중 어느 하나에 새 프리픽스를 삽입하면 된다. 이때 순서 유지는 필요하지 않으므로 임의의 위치, 즉, 가용 공간의 첫부분에 넣으면 되며, 따라서 이로 인한 추가적인 메모리 이동은 발생하지 않는다. 만일 T0~T6 모두에서 매치가 발생하였다면, T7에 삽입하게 되며 이때는 기존 방식과 마찬가지로 추가적인 메모리 이동이 발생할 수 있다.

삭제의 경우 추가적인 메모리 이동이 발생한다. 삭제하고자 하는 프리픽스가 T0~T6 중 어느 하나에 들어 있다면 프리픽스 삭제와 함께 해당 공간을 가용 공간으로 이동시키는데 1번의 메모리 이동이 필요하다. T7이라면 기존 방식과 마찬가지의 메모리 이동이 필요할 것이다.

종합해보면, T0~T6 TCAM에 대해 삽입 시 메모리 이동 0회, 삭제 시 메모리 이동은 1회 발생한다고 할 수 있다. 다음 절에서는 삭제에 따른 메모리 이동을 줄이기 위한 구체적인 방법을 기술한다.

IV. IP 포워딩 테이블의 갱신 알고리즘

1. TCAM에서 테이블 갱신 기법 비교

이 절에서는 프리픽스의 삽입과 삭제에 대해 기존 방식[8]과 제안하는 방식을 비교 설명한다. 그림 3은 분할된 하나의 TCAM(그림 2에서 T0~T6 중 하나)에 대한 포워딩 테이블의 구조를 기존 방식과 개선 방식에 대해 각각 보여주고 있다. 개선 방식은 기존 방식과 같이 프리픽스 정보를 저장하는 TCAM 뿐만 아니라 스택(stack)을 갖고 있다. 스택은 사용 가능한 영역 안에서 삭제로 인해 발생한 가용 엔트리의 주소들을 보관한다. 기존 방식은 TCAM에서 프리픽스 삭제가 발생하면, 사용 중인 영역의 최상위 주소에 있는 프리픽스를 삭제된 주소로 이동시키는 추가 작업을 한다. 개선 방식에서는 TCAM에서 삭제 후의 공백 주소를 스택에 보관하여, 추후 프리픽스 삽입이 발생하면 스택에서 TCAM의 공백 주소를 찾아 그 주소에 프리픽스를 삽입하도록 한다.

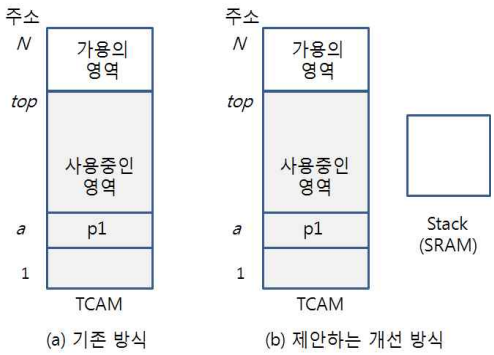

그림 3. TCAM 기반의 포워딩 테이블 비교
Fig. 3. Comparison of TCAM-based Forwarding Tables

기존 방식과 제안하는 방식에 대해, 구체적인 예를 들어 갱신하는 방법을 각각 설명한다. 그 예는 프리픽스 p2의 삽입, 프리픽스 p1의 삭제, 그리고 프리픽스 p3의 삽입 등과 같은 순서로 3번의 테이블 갱신이 발생했을 경우이다. 기존방식에서 갱신 처리 예는 그림 4와 같다. 첫 번째, 초기 상태인 그림3(a)에서 p2를 삽입하면, 그림 4(a)와 같이 현재 사용 중인 영역의 최상위 주소인 *top*을 1 증가시킨 주소에 p2를 삽입한다. 두 번째, 프리픽스 p1이 삭제되면 해당 주소 a로 프리픽스 p2를 이동시킨다(그림 4(b)). 따라서 삭제 시 TCAM에서 1회의 메모리 이동이 발생한다. 프리픽스 p3를 삽입하는 경우에 대해서는 p2의 삽입 경우와 유사하게 처리된다(그림 4(c)).

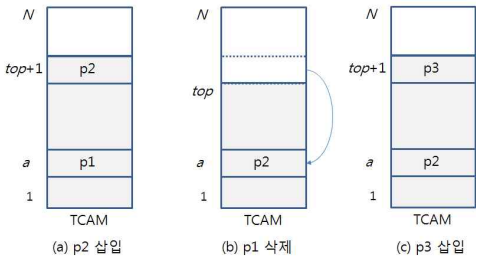

그림 4. 기존 방식에서 갱신 예
Fig. 4. An Update Example for Previous Scheme

제안한 개선 방식에서의 갱신은 그림 5와 같이 수행된다. 첫 번째, 프리픽스 p2의 삽입을 위해서 먼저 스택을 조사하는데, 현재 스택이 비어 있으므로 그림 5(a)와 같이 기존 방식과 동일하게 처리됨을 알 수 있다, 이 경우, 만일 스택에 공백(가용 엔트리)의 주소가 있다면, 스택에서 그것을 꺼내서

(pop), TCAM의 해당 주소에 프리픽스 p2를 삽입한다. 두 번째, 프리픽스 p1이 삭제되면 TCAM에서 메모리 이동 등의 추가 작업 없이, 그림 5(b)와 같이 p1이 삭제되어 공백이 된 주소(그림에서 *a*)를 스택에 넣는다(push). 따라서 TCAM에서 메모리의 이동 없이 스택에 쓰기만으로 삭제가 완료된다. 세 번째, 프리픽스 p3를 삽입하는 경우는 그림 5(c)와 같이 처리된다. 스택을 조사해서 내용이 들어 있으므로, 스택에서 그 내용(주소 *a* 정보)를 꺼낸다. 그리고 TCAM의 주소 *a*에 프리픽스 p3가 삽입된다.

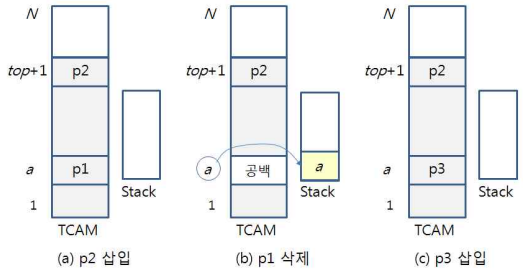

그림 5. 개선 방식에서 갱신 예
Fig. 5. Update Example for Improved Scheme

2. 프리픽스의 삽입 방법

그림 6은 *TCAM*에 신규 프리픽스 *prfx*를 삽입하는 알고리즘을 기술하고 있다. *TCAM*에 프리픽스를 삽입하려면 먼저 스택이 비어 있는지 확인한다. 스택이 비어 있지 않으면, 스택에서 내용을 꺼내고 그곳에 삽입을 한다(과정 1). 만일 스택이 비어 있으면, 기존의 방식과 동일하게 *TCAM*의 가용 영역(free area)에 *prfx*를 삽입한다(과정 2).

```
Insert(prfx, TCAM)
/* prfx: 프리픽스, TCAM: 분할 TCAM */
1. if (stack is not empty)
   /* 스택에서 주소를 꺼냄 */
   addr ← pop()
   TCAM[addr] ← prfx
2. else
   /* stack이 empty이면 TCAM의 가용 영역에
      프리픽스 prfx를 삽입함 */
   top ← top + 1
   if (top > N)  /* N: TCAM의 크기 */
        TCAM is full; return
   else TCAM[top] ← prfx
```

그림 6. 프리픽스 삽입 알고리즘
Fig. 6. Algorithm for the Prefix Insertion

3. 프리픽스의 삭제 방법

그림 7은 *TCAM*에 있는 프리픽스 *prfx*를 삭제하는 알고리즘이다. 스택이 다 차지 않았으면, 삭제된 *prfx*의 주소를 스택에 넣는다(과정 3). 만일 스택이 가득 찼으면, 기존의 방식처럼 *TCAM*의 *top*에 있는 프리픽스를 *prfx*의 주소로 이동시킨다(과정 4). 만일 *prfx*의 주소와 *top*이 같으면, 사용 영역 중간에 공백이 있는 것이 아니므로 *top* 포인터만 감소시킴으로써 삭제를 수행한다(과정 2).

```
Delete(prfx, TCAM)
/* prfx: 프리픽스, TCAM: 분할 TCAM */
1.  addr ← TCAM에서 prfx의 주소
2.  if (addr = top)  top ← top - 1; return
3.  if (stack is not full)
    /* 스택에 주소를 넣음 */
      push(addr)
4.  else
    /* stack이 full이면, TCAM에서 최상위
    주소(top)의 프리픽스를 addr로 이동시킴 */
    TCAM[addr] ← TCAM[top]
    top ← top - 1
```

그림 7. 프리픽스 삭제 알고리즘
Fig. 7. Algorithm for the Prefix Deletion

V. 실험 및 성능 평가

1. 실험 환경

실험에는 [8]과 본 논문에서 제안한 기법에 대해 자체적으로 구현한 시뮬레이터가 사용되었다. 포워딩 테이블은 Route Views[9]에 올라와 있는 데이터를 시뮬레이터에 맞게 가공하여 사용하였으며, 갱신 메시지는 2007년 7~10월의 각 첫 번째 1주일치 데이터를 사용하였다. 사용된 포워딩 테이블의 프리픽스 개수와 갱신 회수는 다음 표 1과 같다. 여기서 갱신 회수는 프리픽스의 삽입과 삭제 만 포함하며, 메모리 이동과 무관한 프리픽스의 출력 링크 변경을 위한 갱신은 대상에서 제외하였다.

표 1. 실험에 사용된 데이터
Table 1. Data used in the Experiment

	7월	8월	9월	10월
프리픽스 수	243,511	244,095	242,635	248,389
갱신 회수	164,467	527,204	787,944	651,771

시뮬레이터로부터 프리픽스 삽입 및 삭제에 소요되는 TCAM 접근 회수와 SRAM 접근 회수를 얻었다. SRAM은 스택을 구성하는 메모리이고, 이의 접근 회수는 스택 접근회수와 같다.

TCAM 모델[10]과 Cacti[11, 12]의 시뮬레이터를 사용하여 TCAM과 SRAM의 접근 시간을 각각 산출하였다. 이때 시뮬레이션의 입력 값은 90*nm* 공정을 기준으로 동일하게 적용하였다. 산출된 TCAM의 접근 시간은 13.324*ns*이고, SRAM의 접근시간은 크기에 따라 0.586~0.769*ns*이다.

2. 성능 평가

표 2는 2007년 7월 데이터에서 삽입 시에 소요된 메모리별 접근 회수이이다. 접미사 _R은 읽기 접근을, _W는 쓰기 접근을 각각 의미한다. 처리된 삽입 메시지의 개수는 84,014개이다. TCAM에 대한 쓰기 접근은 84,016회로 거의 삽입 1회당 1번의 쓰기가 이루어졌다고 볼 수 있으며, 이는 추가적인 메모리 이동은 거의 없다는 의미이다.

한편, 삽입 시에는 빈 공간이 존재하는 지 여부를 스택 읽기를 통해 확인하게 되며, 따라서 SRAM에 대한 읽기 접근(SRAM_R)이 발생한다. 기존 병렬 TCAM 기반 구조는 스택이 없었으므로 SRAM 접근은 제안된 기법에 의해 유발된 추가적인 것이지만, SRAM 접근 시간이 TCAM 접근 시간에 비해 매우 작으므로 평균 메모리 접근 시간에 미치는 영향은 미미하다.

표 2. 삽입 시 메모리 접근 회수
Table 2. The Number of Memory Accesses for Insertion

스택	SRAM_R	SRAM_W	TCAM_R	TCAM_W
128	79,252	0	0	84,016
256	80,085	0	0	84,016
512	80,263	0	0	84,016
1,024	80,235	0	0	84,016
2,048	80,253	0	0	84,016

표 3은 동일 데이터에 대해 삭제 시의 메모리 접근 회수이다. 처리된 삭제 메시지의 총 개수는 80,453이다. 제안된 기법은 TCAM내에서의 메모리 이동 대신 스택에 삭제 위치를 기록하므로 SRAM에 대한 쓰기 접근이 발생한다. 반면, TCAM 내의 메모리 이동은 매우 제한적으로 발생하기 때문에 TCAM 쓰기 접근이 극적으로 작아졌음을 볼 수 있다. TCAM에 비해 SRAM의 접근 시간이 매우 작으므로 전체적인 평균 접근 시간은 매우 줄어들 것이다.

TCAM 읽기는 쓰기와 함께 스택이 가득 찬 경우에 발생하게 된다. 이러한 회수는 스택의 크기를 늘임에 따라 감소함을 알 수 있다.

표 3. 삭제 시 메모리 접근 회수
Table 3. The Number of Memory Accesses for Deletion

스택크기	SRAM_R	SRAM_W	TCAM_R	TCAM_W
128	0	79,362	1,416	1,093
256	0	80,165	174	290
512	0	80,348	0	107
1,024	0	80,332	0	123
2,048	0	80,330	0	125

표 4와 5는 삽입과 삭제 시의 평균 메모리 접근 시간을 각각 보여준다. 이는 SRAM과 TCAM에 대한 접근 회수와 접근 시간 각각을 곱한 후 모두 합한 후 평균을 낸 것이다.

기존 병렬 TCAM 기반 구조[5]는 삽입 시 약 1회의 TCAM 쓰기를 수행하고 이의 접근 시간은 13.32*ns*로써, 제안된 기법의 삽입 소요 시간이 근소하게 증가함을 보여준다. 이는 삽입 시에 스택 읽기를 하기 때문인데, SRAM의 접근 시간은 상대적으로 매우 작기 때문에 미치는 영향은 미미하다.

한편 기존 병렬 TCAM 기반 구조에서 삭제 시 1회의 TCAM 읽기와 쓰기가 발생하며, 이의 접근 시간은 26.649*ns*에 해당한다. 표 5의 삭제 접근 시간은 이에 비해 매우 작은 값으로 제안된 기법이 효과적으로 삭제 시간을 감소시켰음을 보여준다.

표 4. 삽입 시 평균 메모리 접근 시간(*ns*)
Table 4. Average Access Time for Insertion

스택 크기	128	256	512	1024	2048
7월	13.88	13.97	13.96	14.01	14.06
8월	13.88	13.97	13.97	14.03	14.07
9월	13.89	13.98	13.97	14.03	14.07
10월	13.89	13.98	13.97	14.03	14.08

표 5. 삭제 시 평균 메모리 접근 시간(*ns*)
Table 5. Average Access Time for Deletion

스택크기	128	256	512	1024	2048
7월	0.99	0.75	0.68	0.74	0.79
8월	1.70	1.29	0.97	0.89	0.78
9월	0.78	0.72	0.67	0.73	0.78
10월	1.38	1.04	0.69	0.73	0.78

표 3에서 살펴본 것처럼 스택이 가득 차면 추가적인 TCAM 접근을 유발한다. 스택의 크기가 클수록 TCAM에 대한 접근을 줄일 수 있는데, 그림 8은 요구되는 최대 스택 크기를 실험을 통해 얻은 것이다. 입력 데이터와 TCAM별로 차이가 있지만, 최대 1761개의 엔트리가 필요한 것으로 나타났

으며, 이는 2K×16bits이하의 작은 스택으로도 좋은 성능을 얻을 수 있음을 보여준다.

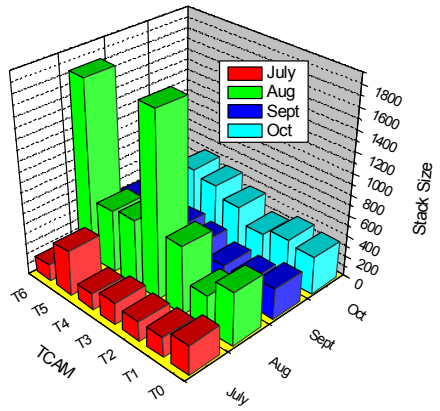

그림 8. 요구되는 최대 스택 크기
Fig. 8. The Maximum Size of the Required Stack

Ⅵ. 결 론

TCAM을 사용하는 IP 주소 검색의 성능을 향상시키기 위해서, 신속한 주소 검색뿐만 아니라 프리픽스의 신속한 갱신도 상당히 중요하다. TCAM을 이용한 대부분의 기존 방식에서는 프리픽스 갱신 속도의 향상을 위해 프리픽스의 삽입 개선에 초점을 맞추었다. 그리고 프리픽스를 삭제할 경우, 프리픽스의 순서 유지나 가용 공간의 연속성 관리를 위해 최소 한 번 이상의 메모리 이동이 수반되었고, 이에 따라 갱신 시간이 지연되었다.

본 논문에서는 병렬 TCAM 기반의 IP 주소 검색 구조[8]을 개선하여 프리픽스를 신속하게 삭제할 수 있는 기법을 제안하였다. 제안 기법에서 프리픽스를 삭제할 때, TCAM에서 메모리 이동을 하지 않고, 삭제되는 프리픽스의 TCAM 상 주소를 스택에 보관한다. 이후 해당 TCAM에 프리픽스를 삽입할 때, 이미 스택에 보관된 주소를 사용할 수 있다. 이 때 스택은 TCAM보다 훨씬 접근속도가 빠른 SRAM를 사용하여, 프리픽스 삭제에 따른 전체 메모리 접근시간을 감소시켜 성능을 향상시켰다.

논문에서 제안된 기법의 성능을 평가하기 위해, 실제 포워딩 테이블과 갱신 내용을 적용하여 프리픽스 삽입과 삭제 시 메모리 접근 회수와 시간에 대해 각각 실험결과를 구하였고, 기존 병렬 TCAM 기반의 기법[8]과 비교하였다. 제안된 기법의 성능을 기존 기법과 비교하면, 프리픽스의 삽입 시 메모리 접근시간은 거의 비슷한 반면, 프리픽스의 삭제 시 메모리 접근시간은 스택의 크기에 따라 다소 차이가 있으나 1/15~1/40 배로 감소되었다. 프리픽스 갱신에서 삽입이 차지하는 비율이 삭제 보다 약간 높지만 거의 비슷한 규모임을 고려하면, 전체 갱신 속도에 있어 상당한 성능이 개선되었음을 알 수 있다.

참고문헌

[1] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy," RFC1519, Sep. 1993.

[2] Mi. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," IEEE Network, March/April 2001.

[3] H. J. Chao and B. Liu, "High Performance Switches and Routers," Wiley Interscience, 2007.

[4] A. J. McAuley and P. Francis, "Fast Routing Table Lookup using CAMs," Proc. of INFOCOM, pp.1382-1391, 1993.

[5] M. Kobayashi, T. Murase, and A. Kuriyama, "A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing," Proc. of 2000 International Conf. on Communications, pp. 1360 - 1364, 2000.

[6] D. Shah and P. Gupta, "Fast Updating Algorithm for TCAMs," IEEE Micro, Vol. 21, No. 2, pp. 36-47, Jan. 2001.

[7] Y.-M. Hsiao, M.-J. Chen, Y.-J. Hsiao, H.-K. Su, and Y.-S. Chu, "A Fast Update Scheme for TCAM-Based IPv6 Routing Lookup Architecture," Proc. of 15th Asia-Pacific conference on Communications, pp. 779-783, 2009.

[8] J. Kim and J. Kim, "An Efficient IP Lookup Architecture with Fast Update Using Single-Match TCAMs," Proc. of WWIC, LNCS 5031, pp. 104-114, May 2008.

[9] University of Oregon Route Views Project, *http://www.routeviews.org/*

[10] B. Agrawal and T. Sherwood, "Ternary CAM Power and Delay Model: Extensions and Uses," IEEE Tr. on Very Large Scale Integration (VLSI) Systems, Vol. 16, pp.554-564, 2008.

[11] P. Shivakumar, N. P. Jouppi, "Cacti 3.0: An Integrated Cache Timing, Power and Area Model," Technical Report Western Research Lab (WRL) Research Report. 2001/2.

[12] *http://www.hpl.hp.com/research/cacti/*

저 자 소 개

김 진 수

1983년: 서울대학교 컴퓨터공학과 공학사

1985년: KAIST 전산학과 공학석사

1998년: KAIST 전산학과 공학박사

1985년 ~ 2000년: KT 선임연구원

2000년 ~ 현재: 건국대학교 컴퓨터응용과학부 교수

관심분야: 정보통신 네트워크, 병렬처리, 센서네트워크

김 정 환

1991년: 서울대학교 계산통계학과 이학사

1993년: 서울대학교 대학원 전산과학과 이학석사

1999년: 서울대학교 대학원 전산과학과 이학박사

1999년 ~ 2000년: 삼성전자 통신연구소 연구원

2001년 ~ 현재: 건국대학교 컴퓨터응용과학부 교수

관심분야: 병렬처리, 컴퓨터네트워크, GPU 컴퓨팅