효율적인 IP 주소 검색을 위한 개선된 LC-trie

Improved LC-trie for Efficient IP Address Lookup

김정환, 김진수
건국대학교 컴퓨터 · 응용과학부

Junghwan Kim(jhkim@kku.ac.kr), Jinsoo Kim(jinsoo@kku.ac.kr)

요약

IP 주소 검색은 라우터에서 가장 중요하고 복잡한 기능중 하나이다. 본 논문에서는 고성능 라우터에서 IP 주소 검색의 성능을 향상시키기 위해 LC-trie를 개선하는 기법을 제안한다. TCAM(Ternary Content Addressable Memory)에서 테이블 압축을 위해 이용한 프리픽스 pruning 방법을 LC-trie에 효과적으로 적용한다. 이러한 기법은 메모리 참조 횟수를 감소시키고 검색의 속도를 높일 수 있다. 또한, 실제 사용되는 포워딩 테이블과 트래픽 분포를 사용하여 검색 시간과 메모리 참조 횟수 등의 관점에서 개선된 기법의 성능을 기존의 LC-trie 기법과 비교 평가한다.

■ 중심어 : | IP 주소 검색 | LC-trie | 고성능 라우터 | 프리픽스 Pruning |

Abstract

IP address lookup is one of the most important and complex functions in the router. In this paper, we propose an improved technique of LC-trie to increase the performance of IP address lookup in the high performance router. We effectively apply the prefix pruning method, which is used for the compression of the forwarding table in TCAM((Ternary Content Addressable Memory), to the LC-trie. This technique can decrease the number of memory accesses and upgrade the lookup speed. Moreover, through the real forwarding table and the real traffic distribution, we evaluate the performance of our scheme in terms of the lookup time and the number of memory access, comparing with that of the previous LC-trie.

■ keyword : | IP Address Lookup | LC-trie | High Performance Router | Prefix Pruning |

I. 서 론

인터넷에서 새로운 멀티미디어 서비스의 수요가 지속적으로 창출되고 사용자의 수가 늘어남에 따라 인터넷의 트래픽은 계속 증가하고 있다. 이와 같이 급증하는 트래픽을 처리하기 위해 인터넷의 고속화 요구는 꾸준히 증대되고 있다. 인터넷은 패킷을 포워딩하는 라우터와 이들 라우터를 연결하여 패킷을 전송하는 링크로 구성된다. 광섬유와 같은 전송 매체의 발달로 전송 링크는 높은 대역폭과 고속의 전송이 가능하게 됨에 따라서, 패킷을 처리하는 라우터가 인터넷 망의 성능에 있어 병목이 되고 있다. 이러한 병목 문제를 해결하기 위

* 본 논문은 2004년도 건국대학교 학술진흥연구비 지원에 의해 수행되었습니다.

접수번호 : #070116-001
접수일자 : 2007년 01월 16일
심사완료일 : 2007년 03월 20일
교신저자 : 김정환, e-mail : jhkim@kku.ac.kr

해서는 초고속으로 패킷을 처리할 수 있는 고성능 라우터의 개발이 필수적이다[1].

라우터의 기능은 패킷을 최종 목적지 방향으로 포워드(forward)하여 전달하는 것이다. 라우터는 이러한 기능을 실현하기 위해서, 입력 링크로 들어온 패킷을 다음에 어느 라우터로 보낼 지를 결정하여야 한다. 이 결정을 위해 입력 패킷의 목적지 IP 주소를 포워딩 테이블에서 검색하고, 검색된 정보를 바탕으로 다음 홉(next hop) 정보를 얻어 패킷의 출력 링크를 정할 수 있다. 따라서, 고속 라우터의 구현을 위해서는 패킷의 버퍼링, 스케쥴링, 스위칭 뿐만아니라 라우터의 핵심 기능인 IP 주소 검색이 효율적이고 빠르게 이루어져야 한다.

초기 인터넷의 IP 주소는 32비트 주소공간이 네트워크 주소 부분과 호스트 주소 부분으로 구성되고, 네트워크 규모에 따라 Class A (8비트의 네트워크 주소), Class B (16비트), Class C (24비트)로 구분하는 클래스 기반 주소방식을 사용하였다. 따라서 IP 주소 검색도 고정된 길이의 프리픽스를 비교하는 완전 일치(perfect matching) 방식을 사용하여 단순하게 처리할 수 있었다. 그러나 이 클래스 기반 주소방식은 융통성이 떨어지고 주소공간을 낭비하는 결과를 초래하였다. 포워딩 테이블의 효율성 제고와 엔트리 수의 감소를 위해 가변 길이의 네트워크 주소를 허용하는 CIDR (classless interdomain routing)가 대두되었다[2]. 이에 따라, IP 주소의 검색이 최장 프리픽스 일치(LPM, Longest Prefix Matching) 방식으로 처리하게 되었다.

LPM은 입력 패킷의 목적지 IP 주소와 일치하는 모든 프리픽스들 중에서 가장 긴 부분이 일치하는 프리픽스를 가장 적합한 프리픽스(BMP: Best Matching Prefix)로 선택하는 방식이다. 그러나 이 LPM은 완전 일치 방식에 비해 상당히 복잡하고 처리 시간이 오래 걸리기 때문에, 라우터의 고성능화를 저해하는 중요한 요인 중 하나이다. 고속의 라우터의 개발을 위해서 LPM의 IP 주소 검색에 대한 많은 기법들이 제안되어 왔고 [3][4], 대표적인 방식으로 TCAM을 이용하는 기법과 trie를 이용하는 기법을 들 수 있다.

TCAM(Ternary Content Addressable Memory)은 프리픽스의 각 비트에 대해 0과 1외에 추가적으로 *를 표현할 수 있도록 CAM을 확장한 것으로, 이를 이용해서 고속으로 LPM을 수행하도록 하는 여러 방식이 제안되었다[5-8]. TCAM 방식에서는 고가인 메모리의 소요량을 줄이기 위해 테이블 내용을 보다 더 함축시키고, 전력 소모량을 줄일 수 있도록 설계하는데 초점을 맞추고 있다. 특히 Liu[7]는 프리픽스 pruning과 논리 최소화 (logic minimization)를 이용해 테이블의 프리픽스 엔트리 수를 줄이고 있다.

Trie는 포워딩 테이블을 표현하기 적합한 트리 형태의 자료구조로서, IP 주소 검색을 위한 많은 기법들이 이를 이용해 제안되었다[9-11]. 이들 기법에서 IP 주소 검색 시간은 검색에 필요한 메모리 참조 횟수에 비례하고, TCAM 방식에 비해 시간이 많이 소요된다는 단점이 있다. 그러나 IP 주소 검색의 속도를 늘리기 위해 전통적인 trie를 효율적으로 변경하여 메모리 참조 횟수를 줄이도록 하는데 초점을 맞추어 연구가 진행되어 왔다. 제안된 기법들 중에서 LC-trie [10]는 메모리 참조 횟수가 상대적으로 적기 때문에 지속적인 관심을 받아 왔다[12-14].

본 논문에서는 LC-trie를 개선하여 IP 주소 검색의 성능을 향상시키는 기법을 제안한다. 제안된 기법은 TCAM에서 테이블 압축을 위해 이용한 프리픽스 pruning 방법을 LC-trie에 효과적으로 적용함으로써, LC-trie 생성시간, LC-trie 크기, 메모리의 참조 횟수 등을 줄인다. 논문의 구성은 2장에서 LC-trie와 관련된 기존의 연구 내용과 프리픽스 pruning에 대해 기술한다. 3장에서는 본 연구에서 제안한 기법을 설명하고 그 예를 보인다. 그리고 4장에서는 제안된 기법의 성능을 실제 사용되는 포워딩 테이블과 트래픽 분포를 사용하여 메모리 참조 횟수 등의 관점에서 기존의 LC-trie 기법과 비교 평가한다. 마지막으로 5장에서 결론을 맺는다.

II. 관련 연구

1. Trie 구조

이진 trie는 일종의 트리 구조로서, 1 비트로서 자식 노드를 선택하도록 하여, 루트로부터 해당 노드까지의

비트 스트링으로 프리픽스를 표현한다. 즉, 해당 비트의 값이 0이면 왼쪽 자식노드를, 1이면 오른쪽 자식노드를 선택한다. [표 1]의 6개 프리픽스를 갖는 포워딩 테이블은 [그림 1]과 같은 trie로 표시된다. 예를 들어 프리픽스 d는 0110*임을 알 수 있다. 그리고 프리픽스 d의 길이(length)는 4이다. 즉 프리픽스 길이는 루트로부터 그 노드까지의 경로 길이이다. 이진 trie에서 프리픽스 스트링의 각 비트를 비교하기 위해 매번 메모리를 참조해야 한다. 따라서 프리픽스의 길이, 즉 IPv4의 경우 32, IPv6의 경우 128, 만큼의 메모리 참조가 필요하다.

표 1. 포워딩 테이블의 예

프리픽스		다음 홉(next hop)
a	00*	N1
b	01*	N2
c	0100*	N2
d	0110*	N3
e	10*	N4
f	1101*	N5

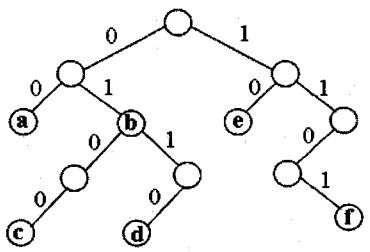


그림 1. 이진 trie의 구성 예

이진 trie는 [그림 1]의 c, d, f와 같이 연속해서 하나의 자식들로만 구성되는 경우가 생길 수 있다. 경로 압축(path compression) trie는 이와 같이 연속된 하나의 자식 노드들로 구성되는 경로를 생략하여 메모리의 참조 시간과 메모리의 사용 공간을 절약할 수 있도록 하는 기법이다. [그림 2]는 [그림 1]의 trie에 대한 경로 압축 trie를 보이고 있다. 그림에서 *skip*은 생략된 경로의 길이, 즉 생략된 노드의 수를 나타내고 괄호안의 값은 생략된 이진 스트링을 표시한다. 예를 들어 노드 f 는 프리픽스 1101* 이며 비트 3, 4인 스트링 01에 해당되는 경로가 생략되었으므로, *skip*은 2가 되고 괄호안의 값은 01임을 알 수 있다.

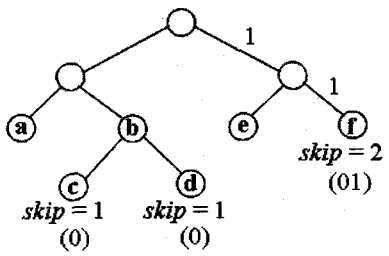


그림 2. 경로 압축 trie의 구성 예

다중 비트(multibit) trie 방식은 이진 trie의 단점을 개선하여 다수의 비트를 하나의 노드로서 표현함으로써 메모리의 참조 횟수를 줄이도록 개선하였다. 즉 프리픽스의 여러 비트를 한 번에 비교함으로써 전체 비교 횟수, 즉 메모리의 참조 횟수를 감소시키고 있다. Nilsson[10]은 LC-trie(Level Compressed trie)을 제안하여 메모리 참조 횟수를 줄였고, IP 주소 검색의 효율을 향상시켰다. LC-trie는 프리픽스 수가 적은 경우에 적합한 경로 압축과 프리픽스 수가 많은 경우에 적합한 레벨 압축을 동시에 수행한다.

2. LC-Trie

LC-trie는 경로 압축과 다중 비트 trie의 장점을 혼합한 기법이다[10]. 즉, LC-trie는 이진 trie에서 경로 압축 trie로 개선한 후, 레벨 압축하여 다중 비트 trie로 변환함으로써 생성할 수 있다. 노드 n에서의 레벨 압축은 n으로부터 레벨 i 아래에 있는 최대 2^i 개의 후손 노드들을 n의 2^i 개의 자식 노드들로 변환하는 것이다. 즉 하나의 레벨로 압축시킨 i-비트 sub-trie를 생성하는 것을 의미한다. 이 때 효율성을 높이기 위해, fill factor라는 개념을 사용하는데, 해당 레벨의 후손 노드들이 fill factor 비율 이상 존재할 경우에만 레벨 압축을 수행한다.

[그림 3]은 [그림 1]의 trie에 대한 LC-trie를 보이고 있다. [그림 3]에서 보는 바와 같이, 레벨 2의 노드 a, e, f가 2-비트 sub-trie로 구성되어 있음을 알 수 있다. 그

리고 LC-trie에서는 leaf 노드가 아닌 내부 노드들은 prefix vector에 저장되고 trie에 표시를 하지 않기 때문에 [그림 3]에서 b가 생략되어 있다. 또한, [그림 3]과 같이 LC-trie에서 skip은 존재하지만 생략된 이진 스트링 값은 저장하지 않는다.

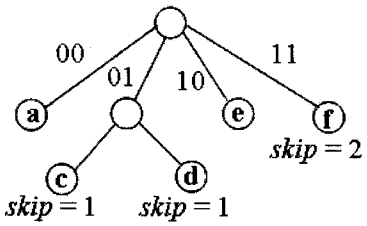


그림 3. LC-trie의 구성 예

LC-trie에서 필요한 자료 구조는 packed array (배열로 표현된 LC-trie), base vector, next hop table, prefix vector 등의 4개로 구성된다. LC-trie의 노드들은 각 레벨 순서로, 즉 레벨 1 노드(루트 노드), 레벨 2 노드들, 레벨 3 노드들과 같은 순서로, 연속된 메모리에 배열로서 저장된다. Base vector는 trie의 leaf 노드들에 해당되는 실제 프리픽스 스트링을 저장하고 있다. Prefix vector는 다른 스트링의 proper prefix인 스트링에 대한 정보를 연결 리스트 형태로 나타내며, 내부 노드들을 위한 것이다. [그림 4]는 [그림 3]의 LC-trie와 관련된 자료 구조들을 나타내고 있다.

(a) packed array (trie)

node	branch	skip	ptr
0	2	0	1
1	0	0	a
2	1	0	5
3	0	0	e
4	0	2	f
5	0	1	c
6	0	1	d

(b) base vector

	prefix	ptr	next hop
a	00*	-	0
c	0100*	b	1
d	0110*	b	2
e	10*	-	3
f	1101*	-	4

(c) prefix vector

	len	ptr	next hop
b	2	null	1

(d) next hop

N1
N2
N3
N4
N5

그림 4. LC-trie 관련 자료구조의 예

3. LC-trie 개선에 관한 기존 연구

LC-trie 개선에 관한 기존 연구로는 검색 성능의 향상을 위한 Ravikumar의 연구[13]와 점진적 갱신에 관한 Pao의 연구[12]를 들 수 있다.

Ravikumar 등은 효율적인 주소 검색을 위해 LC-trie의 내부 노드에 skip string을 저장하는 방법을 제안하였다. 기존 LC-trie는 내부 노드에 skip string을 저장하지 않고 prefix vector를 역추적하는데, 그들의 연구는 이 부담을 줄이고자 한 것이다. 그러나 이 방법은 skip string의 저장을 위해 엔트리 크기가 상당히 증가하는 또 다른 문제가 발생한다. 더우기 최근의 LC-trie 분석 연구[14]를 보면 실제 트래픽에서 약 71.8%의 주소 검색이 단말 노드에서 이루어지는 것을 알 수 있으며, 이는 Ravikumar 방법의 개선 정도가 실제로는 크지 않다는 것을 시사한다.

LC-trie는 레벨 압축과 경로 압축이 이루어진 상태이고 단말 노드가 아닌 경우 prefix vector에 저장되기 때문에 포워딩 엔트리가 추가되거나 삭제되는 경우 갱신의 어려움이 있다. Pao 등은 내부 노드에 skip string을 저장하고 이를 이용해 갱신을 용이하게 하는 방법을 제안하였다. 그러나 내부 노드에 skip string을 저장하는 경우 LC-trie의 크기가 증가하는 부담이 있다는 단점이 있다.

4. 프리픽스 pruning

프리픽스의 pruning은 TCAM 메모리의 절약을 위해 테이블 압축에 이용한 방식이다[7]. 먼저 P_x와 P_y는 상이한 프리픽스이고, P_x가 P_y의 proper prefix임을 $P_x < P_y$ 로, 테이블에서 P_i의 next hop을 N_i로 표시한다고 하자. P_x와 P_y ($x \neq y$) 간에 다음과 같은 조건을 만족할 경우, P_y를 삭제할 수 있으며, 이것을 프리픽스 P_y의 "pruning" 이라고 한다.

1. $|P_x| < |P_y|$ 이다.
2. $N_x = N_y$ 이다.
3. $P_x < P_i$, $P_i < P_y$이고 $N_x \neq N_i$를 만족하는 프리픽스 P_i는 존재하지 않는다.

입력	entry[] (정렬된 포워딩 테이블) size (포워딩 테이블 크기)
출력	p (pruning 된 prefix vector) b (pruning 된 base vector)

```
for (i = 0; i < size; i++) {
  /* entry[i]->pre는 entry[i]의 proper prefix가 저장되어 있는 p[] 안의 위치 */
  if ( (j = entry[i]->pre) != NOPRE &&  p[j]의 next hop == entry[i]의 next hop) {   /* prune entry[i] */
       /* 즉, entry[i]는 p[]와 b[] 둘 중 어디에도 삽입되지 않음. */
       p[j]->refcnt--;  /* p[j]->refcnt는 p[j]의 후손 prefix의 개수 */
       if (p[j]->refcnt == 0)  p[j]의 후손이 더 이상 없으므로 p[j]를 b[]로 이동;
  } else
     if (i < size-1 && isprefix(entry[i], entry[i+1])) {  /* p[]에 삽입 */
        p[nprefs] = entry[i];
        for (j = i + 1; j < size && isprefix(entry[i], entry[j]); j++) {
           entry[j]->pre = nprefs;
           p[nprefs]->refcnt++;
        }
        nprefs++;
     } else     /* b[]에 삽입 */
       b[nbases++] = entry[i];
} /* for */
```

그림 5. prefix vector와 base vector 생성

예를 들어 프리픽스 01*, 0100*, 01001* 의 next hop이 각각 N1, N1, N2 라고 할 때, 프리픽스 0100*는 pruning된다. 만일 프리픽스 01*, 0100*, 01001* 의 next hop이 각각 N1, N2, N1 이었다면, pruning이 불가능하다.

프리픽스의 pruning을 trie 상에서 살펴보면, 동일한 next hop을 갖는 두 개의 프리픽스 노드가 존재할 때, 후손에 해당하는 노드는 제거할 수 있다는 것이다. 이 때 두 노드 간의 경로에 다른 next hop을 갖는 프리픽스 노드가 없어야 한다.

III. 개선된 LC-trie

1. Pruning의 필요성

LC-trie는 내부 노드에 대한 프리픽스 비교가 필요 없고, 레벨 압축을 통해 짧은 탐색 경로로 leaf 노드에 도달할 수 있는 장점을 갖고 있다. 그러나 레벨 압축이 메모리 참조 횟수는 줄여주는 대신, 프리픽스 확장으로 인해 메모리 공간의 크기는 늘어나게 된다. 가령, 루트 노드에서는 보통 branch factor를 16으로 하여 레벨 압축을 하게 되는데, 이때 첫 번째 레벨의 노드 수는 2^{16} = 65536이 된다.

메모리 크기의 증가는 LC-trie를 표현하는 packed array 전체를 캐쉬에 효과적으로 적재하기 어렵게 만들며, 감소된 캐쉬 적중률이 검색 성능을 떨어뜨린다. 한편, 최근의 증가된 포워딩 테이블은 약 20만개 이상의 엔트리를 보유하고 있으며, 이로부터 생성된 LC-trie의 노드 수는 약 40만개에 이른다. 포워딩 테이블 크기는 계속적으로 증가하는 추세에 있으므로, LC-trie의 크기에 대한 효과적인 제어는 검색 성능에 중요한 요소라고 할 수 있다.

또 LC-trie는 leaf 노드에 신속히 도달하는 대신, 중간에 존재하는 프리픽스의 경우 prefix vector를 통해 다시 거슬러 올라가는 부담이 존재한다. 따라서 이러한 전체 탐색 경로 길이에 대한 효과적인 제어가 없다면, 좋은 검색 성능을 보장하기 어렵다.

본 논문에서는 LC-trie의 크기를 줄이고, 탐색 경로도 단축할 수 있도록 개선하기 위해 pruning 기법을 적용한다.

2. 개선된 LC-trie 생성 기법

제안하는 LC-trie 생성 기법은 LC-trie 생성 과정에서 효과적으로 pruning을 수행함으로써, 전체적으로 생성되는 trie 노드 수, base vector 및 prefix vector 엔트리 수를 줄이는 것이다.

LC-trie의 생성 단계를 개략적으로 보면 다음과 같다.

단계 1. next hop table 생성
단계 2. prefix vector 및 base vector 생성
단계 3. trie (packed array) 생성

제안하는 방법은 단계 2에서 pruning과 함께 prefix vector와 base vector를 생성함으로써 효과적으로 LC-trie를 생성하는 것이다. 제안하는 기법의 단계 2 알고리즘에 대해 기술하면 [그림 5]와 같다. pruning되는 엔트리에 대해서는 후손 엔트리를 찾아 포인터로 가리키게 할 필요가 없기 때문에, 소요 시간은 줄어든다. 즉, pruning을 함에도 불구하고 단계 2의 소요 시간은 오히려 감소한다. 이는 4장의 실험 결과에서도 확인할 수 있다.

3. 개선된 LC-trie의 예

[표 1]의 포워딩 테이블에 대해 pruning을 적용하여 생성한 LC-trie와 관련 자료구조는 각각 [그림 6]과 [그림 7]과 같다. 프리픽스 b=01*가 프리픽스 c=0100*의 proper prefix이고 두 프리픽스의 next hop이 N2로 동일하므로 프리픽스 c가 pruning되었음을 알 수 있다. 그리고 c가 pruning되므로, 프리픽스 d에서 추가적인 경로 압축이 행해졌다. 이와 관련하여 자료구조도 [그림 7]과 같이 변경되었다.

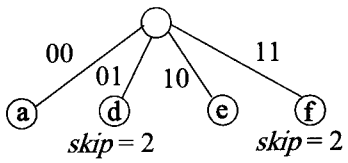


그림 6. 개선된 LC-trie의 구성 예

(a) packed array (trie)

node	branch	skip	ptr
0	2	0	1
1	0	0	a
2	0	2	d
3	0	0	e
4	0	2	f

(b) base vector

	prefix	ptr	next hop
a	00*	–	0
d	0110*	b	2
e	10*	–	3
f	1101*	–	4

(c) prefix vector

	len	ptr	next hop
b	2	null	1

(d) next hop

N1
N2
N3
N4
N5

그림 7. 개선된 LC-trie 관련 자료구조의 예

Ⅳ. 실험 및 성능 평가

1. 실험 환경

실험은 LC-trie와 개선된 LC-trie 각각에 대해 생성 및 검색 알고리즘을 구현한 후, 비교 평가를 수행하였다. 실험 환경은 x86 기반의 플랫폼에서 구축된 Linux 시스템이 사용되었다. 사용된 시스템의 사양은 다음과 같다.

- Pentium 4 3.2GHz
- 1MB Cache
- 1GB Main Memory
- Linux 2.4.20

생성 및 검색 성능의 비교를 위해서는 포워딩 테이블과 트래픽 데이터가 필요하다. 본 실험에서는 포워딩 테이블로 KPN Internet Backbone의 2007년 1월 데이터를 사용하였다[15]. 메시지 트래픽의 경우 보안 문제가 있기 때문에 실제 트래픽이 공개되어 있지는 않다. 대부분의 연구는 균일 분포 또는 특정 분포의 랜덤 트

래픽을 사용하고 있다. 본 실험에서는 균일 분포의 랜덤 트래픽 뿐 아니라, 실제 트래픽을 사용하여 분석한 데이터를 바탕으로 트래픽을 발생시켜 보다 실제 상황에 맞도록 하였다[14]. 본 장에서 전자와 후자를 각각 랜덤(random) 트래픽과 가상(virtual) 트래픽이라는 용어로 사용한다.

2. 성능 평가

2.1 생성된 LC-trie 비교

총 202444 엔트리를 갖는 포워딩 테이블에 대해 LC-trie을 생성한 결과 개선된 LC-trie는 70666개의 pruning을 통해 크기가 감소하였다. 기존 LC-trie에 비해 pruning 기법이 적용된 LC-trie의 trie 노드 수와 base vector 및 prefix vector의 엔트리 수는 [표 2]와 같다. trie 노드 수의 경우 약 1.41배, base vector 및 prefix vector는 1.48배, 2.69배 각각 향상된 것을 알 수 있다. 엔트리 수의 감소는 LC-trie의 메모리 소요량을 줄임으로써, 캐쉬 적중률 향상에 기여하게 된다.

표 2. 생성된 LC-trie 비교

	LC-trie	개선 LC-trie
엔트리수(포워딩테이블)	202444	202444
생성 시간(sec)	0.104	0.073
leaves	350468	251511
internal nodes	44915	29768
total (leaves+internals)	395383	281279
trie-level 1	55583	57656
trie-level 2	138352	91680
trie-level 3	146610	95056
trie-level 4	9851	7067
trie-level 5	72	52
max level (path length)	5	5
avg. level (path length)	2.317	2.206
base vector	185383	125424
prefix vector	17061	6354
max path in prefix vector	6	3
avg. path in prefix vector	0.398	0.119
next hops	306	306
# of pruning	0	70666

또한 3장 알고리즘에서 제안한 바와 같이 prefix vector와 base vector의 생성과 pruning을 동시에 하는 것은 생성 소요 시간을 오히려 단축할 수 있다. [표 2]에서 보듯이 기존 LC-trie의 생성 시간은 0.104이지만, 개선된 방법의 LC-trie의 생성 시간은 pruning을 수행함에도 불구하고 0.073으로 단축되었다.

위 생성 결과를 보면, trie-level 1에 있는 노드 수의 경우 기존 LC-trie보다 제안한 LC-trie가 오히려 증가하고 trie-level 2 이후 감소하는데, 이는 검색 시간 단축에 유리하게 작용할 것으로 생각된다.

가장 획기적인 감소, 즉 메모리 절약과 검색 시간 개선이 발생한 부분은 prefix vector로서 약 2.69배 향상되었다. 특히 prefix vector에 도달하는 검색은 가장 메모리 참조 수가 많기 때문에 prefix vector 크기의 감소는 최대 소요 시간에 많은 영향을 미칠 것으로 생각된다. 한편, prefix vector 내에서의 최대 경로 길이도 6에서 3으로 감소하였다.

2.2 검색 성능 비교

검색 성능은 실험 환경에 따라 달라지지만, 동일 조건 하에서 두 개 LC-trie의 상대적 비교는 가능하다. 균일 분포의 랜덤 트래픽을 사용한 경우 약 1.15배 향상되었으며, FUNET trace의 가상 트래픽을 발생시킨 경우 약 1.3배 향상되었다. [그림 8]에서 볼 수 있듯이 두 LC-trie 모두 랜덤 트래픽보다 가상 트래픽([그림 8]의 "virtual")에서 더 좋은 성능을 보여준다. 이에 대한 이유는 [14]의 분석에서 확인할 수 있으며, 2.3절에서 보다 자세하게 논의한다.

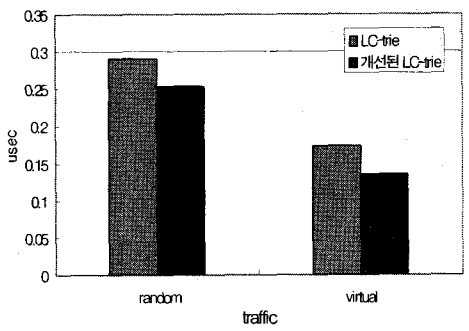


그림 8. 검색 성능 비교

2.3 탐색 깊이 비교

IP 주소 검색 시 trie 탐색이 수행되고, 경우에 따라서는 prefix vector로 거슬러 올라가 탐색이 수행되기도 한다. 본 절에서는 trie 탐색 깊이와 prefix vector 탐색 깊이로 구분하여 비교해보기로 한다.

• Trie 탐색 깊이

[그림 9]는 랜덤 트래픽에 대한 결과인데, 랜덤 트래픽을 사용했을 때 LC-trie에서 가장 빈도가 높은 탐색 깊이는 3이다. 이에 비해 개선된 LC-trie에서는 가장 빈도가 높은 탐색 깊이는 2이며, 전반적으로 탐색 깊이가 감소하였음을 확인할 수 있다. 그림에서 y축은 해당 탐색 깊이에 대한 주소 검색 수를 1000으로 나눈 값을 의미하며, 이하 그림들에서 동일하다.

[그림 10]은 가상 트래픽에 대한 결과이다. 가상 트래픽을 사용하면 전반적으로 두 LC-trie의 탐색 깊이가 모두 향상됨을 알 수 있다. 또한 대부분의 검색은 trie 탐색 깊이가 1인데, 이는 LC-trie에서 16 비트가 레벨 압축된 첫 번째 단계에서 trie 탐색이 끝남을 의미한다. 그리고 trie 탐색 깊이가 2, 3, 4로 증가함에 따라 검색 빈도는 급격히 감소한다. 이러한 특성은 [14]에서 FUNET의 실제 trace를 분석의 경우와 동일하다.

한편, trie 탐색 깊이가 1인 검색 빈도는 원래의 LC-trie보다 개선된 LC-trie에서 더 높으며, 이는 개선된 LC-trie가 더 좋은 성능을 내는데 기여하는 것으로 생각된다. 또한 trie 탐색 깊이가 2 이상인 경우는 반대로 개선된 LC-trie에서의 빈도가 낮다. 이러한 현상은 pruning 기법을 통해 탐색 깊이가 줄어들었기 때문인 것으로 해석된다.

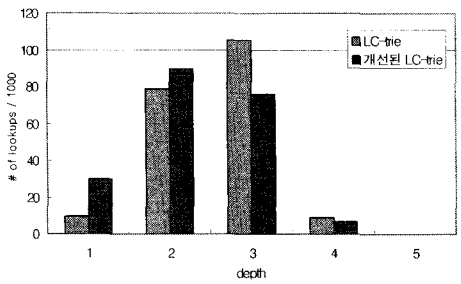


그림 9. Trie 탐색 깊이(랜덤 트래픽)

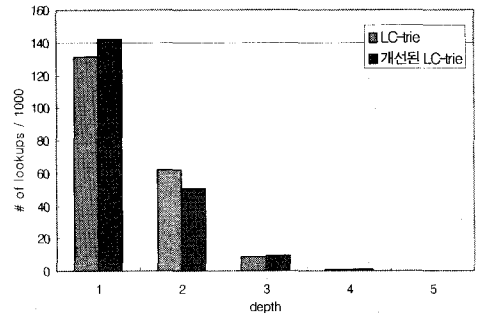


그림 10. Trie 탐색 깊이(가상 트래픽)

• Prefix vector 탐색 깊이

[그림 11]과 [그림 12]는 각각 랜덤 트래픽과 가상트래픽에 대해 prefix vector상의 탐색 깊이 결과를 보이고 있다. 그림에서 prefix vector의 탐색 깊이가 0이라는 것은 trie탐색만으로 주소 검색이 종료된다는 것을 의미한다. 랜덤 트래픽에서 prefix vector 탐색 깊이는 대부분 0이지만, 가상 트래픽에서는 탐색 깊이가 다소 증가한다.

한편, 랜덤 트래픽에서는 원래의 LC-trie가 개선된 LC-trie보다 탐색 깊이 0인 경우가 더 많음으로써 좋은 특성을 보이지만, 가상 트래픽에서는 상황이 반전되어 개선된 LC-trie가 보다 좋은 특성을 보인다.

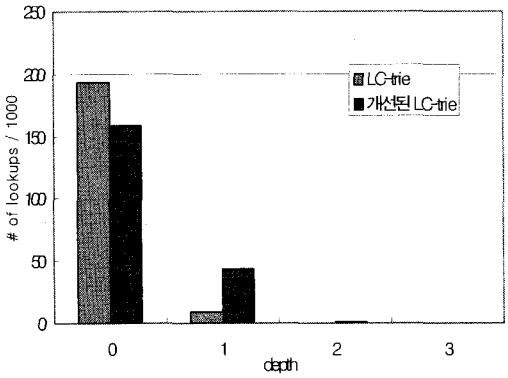


그림 11. Prefix vector 탐색 깊이(랜덤 트래픽)

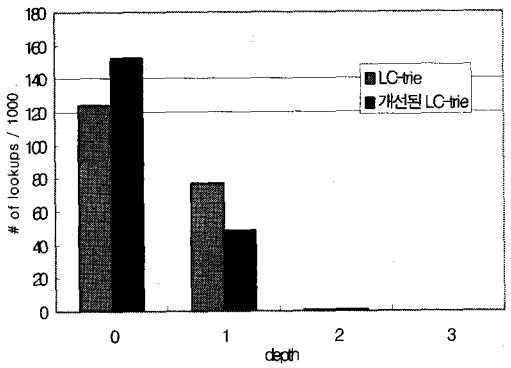


그림 12. Prefix vector 탐색 깊이(가상 트래픽)

V. 결 론

LC-trie는 레벨 압축과 경로 압축을 동시에 적용함으로써, 소프트웨어 기반의 다른 IP 주소 검색 기반에 비해 메모리 참조 횟수를 획기적으로 줄일 수 있다. 또한 중간 노드에 대한 프리픽스 비교 없이 leaf 노드에 신속히 도달할 수 있는 것도 LC-trie의 장점이다. 그러나 leaf 노드가 찾고자 하는 프리픽스가 아닌 경우 prefix vector에 있는 포인터를 추적하여 거슬러 올라가야 하는 부담이 있다.

본 연구에서는 pruning 기법을 통해 LC-trie를 개선하여 IP 주소 검색의 성능을 향상시키는 방법을 제안하였다. 개선된 LC-trie는 IP 주소 검색 시 메모리 참조 횟수를 줄여 검색 시간을 감소시키고, 메모리 크기를 줄임으로써 전체 성능을 높일 수 있었다.

개선된 LC-trie의 성능을 실제 포워딩 테이블과 트래픽 분포를 사용하여 기존의 LC-trie 성능과 비교하여 실험하였다. 실험 결과를 보면, 개선된 LC-trie는 trie 탐색 깊이가 1인 주소 검색의 비중이 기존 LC-trie에 비해 높아졌고, 이에 따라 전체적인 탐색 길이가 감소하였다. 또한, 개선된 LC-trie의 성능은 기존의 LC-trie와 비교했을 때, 랜덤트래픽에서 1.15배, 가상트래픽에서 1.3배 향상되었다.

대부분의 IP 주소 검색 기법에서 자료구조의 갱신은 어려운 문제로 남아 있다. LC-trie의 경우도 검색 성능에 부담을 주지 않으면서 이를 갱신하는 방법이 해결하기 힘든 문제이며, 개선된 LC-trie 또한 동일한 어려움을 지니고 있다. 향후에, 개선된 LC-trie에서 효율적으로 자료구조를 갱신할 수 있는 방법에 대한 연구와 분석이 필요하다고 생각된다.

참 고 문 헌

[1] H. J. Chao, "Next Generation Routers," Proc. of IEEE, Vol.90, No.9, pp.1518-1558, Sep. 2002.

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

[3] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," IEEE Network, Mar./Apr. 2001.

[4] G. Varghese, *Network Algorithmics*, Morgan Kaufmann Pub., 2005.

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

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

[7] H. Liu, "Routing table compaction in Ternary CAM," IEEE Micro, Vol.22, No.1, pp.58-64, Jan. 2002.

[8] V. C. Ravikumar and R. N. Mahapatra, "TCAM architecture for IP lookup using prefix properties," IEEE Micro, Vol.24, No.2, pp.60-69, Mar. 2004.

[9] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest Matching Prefixes," IEEE/ACM Trans. on Net., Vol.4, pp.86-97, Feb. 1996.

[10] S. Nilsson and G. Karlsson, "IP-Address

Lookup using LC-Tries," Journal of Selected Areas in Communications, Vol.17, pp.1083-1092, Jun. 1999.

[11] S. Sahni and K. S. Kim, "Efficient construction of mutibit tries for IP lookup," IEEE/ACM Tran. on Networking, Vol.11, No.4, pp.650-662, 2003.

[12] D. Pao and Y. K. Li, "Enabling Incremental Updates to LC-Trie for Efficient Management of IP Forwarding Tables," IEEE Communications Letters, Vol.7, No.5, pp. 245-247, May 2003.

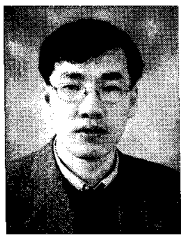
[13] V. C. Ravikumar, R. Mahapatra, and J. C. Liu, "Modified LC-trie based efficient routing lookup," Proc. on 10th IEEE International Symposium on MASCOTS, pp.177-182, Oct. 2002.

[14] F. Jing, O. Hagsand, and G. Karlsson, "Performance evaluation and cache behavior of LC-trie for IP-address lookup," Proc. of 2006 Workshop on HPSR, pp.29-35, June 2006.

[15] http://bgp.potaroo.net/286/bgp-active.html

저 자 소 개

김 정 환(Junghwan Kim) 정회원



- 1991년 2월 : 서울대학교 계산통계학과 (이학사)
- 1993년 2월 : 서울대학교 대학원 전산과학과(이학석사)
- 1999년 2월 : 서울대학교 대학원 전산과학과(이학박사)
- 1999년 ~ 2000년 : 삼성전자 통신연구소 연구원
- 2001년 ~ 현재 : 건국대학교 컴퓨터 · 응용과학부 교수

<관심분야> : 병렬처리, 분산시스템, RFID 시스템

김 진 수(Jinsoo Kim) 정회원

- 1983년 2월 : 서울대학교 컴퓨터공학과(공학사)
- 1985년 2월 : 한국과학기술원 전산학과(공학석사)
- 1998년 8월 : 한국과학기술원 전산학과(공학박사)
- 1985년 4월 ~ 2000년 2월 : KT 선임연구원
- 2000년 3월 ~ 현재 : 건국대학교 컴퓨터 · 응용과학부 교수

<관심분야> : 병렬처리, 고속통신망, 센서네트워크