Journal of Korea Multimedia Society Vol. 9, No. 2, February 2006(pp. 244-254)

비트-맵 기반의 혼합형 고속 IP 검색 기법

오 승 현†

요 약

본 논문은 고속 IP 검색을 위해 거대한 포워딩 테이블을 인덱싱하는 트라이(trie)를 캐시에 저장할 수 있는 작은 크기로 압축하는 복합적 기법을 제안한다. 본 논문의 복합적 기법은 bit-map과 controlled-prefix 기법을 복합한 것으로 저속의 주 메모리 검색을 약간의 계산을 포함한 고속 메모리 검색으로 대체한다. bit-map 트라이 압축 기법은 트라이의 인덱스와 자식 포인터를 각각 하나의 비트로 표시한다. 예를 들면 한 노드가 n bit를 대표할 때 bit-map은 노드에서 연결된 2^n개의 인덱스와 자식 링크를 2^{n-1} bit로 표시함으로써 높은 메모리 압축효과를 제공한다. controlled-prefix 기법은 주어진 트라이 계층 개수에 대해 각 계층의 깊이(stride) 즉, 트라이의 각 계층의 최상위 노드가 대표할 비트의 개수를 결정한다. 이때 controlled-prefix 기법은 주어진 트라이 계층 개수에 대해 최소의 트라이 크기를 구하기 위해 동적 프로그래밍(dynamic programming) 기법을 사용한다. 본 연구는 트라이 계층 개수에 따라 최적의 메모리 크기와 검색속도를 제시함으로써 시스템의 메모리 크기와 요구되는 검색속도에 맞추어 적절한 트라이 구조를 선택할 수 있는 기준을 제안한다.

Bit-Map Based Hybrid Fast IP Lookup Technique

Seung-Hyun Oh†

ABSTRACT

This paper presents an efficient hybrid technique to compact the trie indexing the huge forward table small enough to be stored into cache for speeding up IP lookup. It combines two techniques, an encoding scheme called bit-map and a controlled-prefix expanding scheme to replace slow memory search with few fast-memory accesses and computations. For compaction, the bit-map represents each index and child pointer with one bit respectively. For example, when one node denotes n bits, the bit-map gives a high compression rate by consumes 2^{n-1} bits for 2^n index and child link pointers branched out of the node. The controlled-prefix expanding scheme determines the number of address bits represented by all root node of each trie's level. At this time, controlled-prefix scheme use a dynamic programming technique to get a smallest trie memory size with given number of trie's level. This paper proposes standard that can choose suitable trie structure depending on memory size of system and the required IP lookup speed presenting optimal memory size and the lookup speed according to trie level number.

Key words: Routing(라우팅), Forwarding Table(포워딩 표), Prefix(프리픽스), Trie(트라이), Address Lookup(주소 검색), Next-Hop(다음-홉)

※ 교신저자(Corresponding Author) : 오승현, 주소 : 경북 경주시 석장동 707(780-714), 전화 : 054)770-2243, FAX : 054)770-2816, E-mail : shoh@dongguk.ac.kr
접수일 : 2005년 9월 15일, 완료일 : 2005년 10월 25일
† 정회원, 동국대학교 컴퓨터멀티미디어학부 조교수

1. 서 론

최근에 추진중인 광대역 통합망은 유선망과 무선망의 통합과 통신과 방송의 융합에 의해 최종적으로

광대역 기반 서비스의 통합적 제공을 목적으로 하고 있다. 광대역 통합망은 IT 신산업의 축으로써 각각의 망을 기반으로 보급된 모든 서비스를 통합하고, 최종 사용자에게 단절이 없는(seamless) 연속적인 서비스를 제공하고자 한다. 기존의 흩어져 있는 별개의 망들이 IP 네트워크를 중심으로 통합된 광대역 통합망은 멀티미디어 중심의 품질 보증형 서비스와 보안 서비스가 필수적으로 요구되며, 동시에 수많은 망과 멀티미디어 서비스가 정체 없이 제공되기 위해 넓은 대역폭이 요구된다. 또한 통합형 IP 네트워크에 모든 데이터 패킷이 처리 유통되어야 하므로 고속 패킷처리 서비스가 필수적으로 제공되어야한다. 본 연구는 이러한 추세에 부응하여 IP 패킷 즉, 프리픽스 정보의 고속 처리에 대한 bit-map 기반의 혼합형 IP 검색 기법을 제안하고자 한다. IP 프리픽스 검색은 거대한 크기의 포워딩(forwarding) 테이블에서 목적지 IP 프리픽스를 최장 프리픽스 검색(LPM: Longest Prefix Matching) 기법으로 검색한다.

본 논문은 트라이의 모양을 결정하는 controlled-prefix 기법[1]과 트라이 압축기법인 bit-map[2,3]이 결합된 복합 기법을 제안한다. 본 논문의 복합 기법은 약간의 추가적인 선행 계산 테이블과 함께 트라이의 노드와 링크를 비트로 표시하는 bit-map을 이용해서 트라이를 압축한다. bit-map 압축 후 controlled -prefix 기법은 트라이의 모양을 결정한다. 트라이 모양 결정은 트라이 검색 시간의 단축을 위해 한 번에 검색할 수 있는 비트열의 길이 즉, [1]에서 stride로 불리는 트라이의 분기도를 결정하는 것이다. [1]은 모든 검색단위가 동일하게 결정되는 정적기법과 다양한 검색단위를 가질 수 있는 동적기법을 제안하였다. 본 논문에서는 계산량 감소와 메모리 절약을 위해 마지막 계층의 길이만 다르게 구성되는 변형된 동적기법을 사용한다. 참고로, 트라이 분기도(degree of branch)는 한 번에 검색할 수 있는 트라이 계층의 깊이를 말한다. 만일에 검색단위가 1비트라면 깊이가 32인 트라이는 32번의 검색연산이 필요하고, 분기도가 8비트라면 4번의 검색연산이 발생한다.

이 논문은 구성은 다음과 같다. 2장에서 LPM 검색에 대한 기존 연구 결과를 소개하고, 3장에서는 bit-map 트라이와 bit-map 모양을 결정하는 정적, 동적기법에 대해 소개한다. 4장에서는 백본 라우터에서 CPU 성능 변화에 따른 복합 기법의 성능을 제시하고, 마지막으로 5장에서 요약과 결론을 제시한다.

2. 관련연구

포워딩 검색은 최장 길이의 프리픽스가 최종적으로 선택(LPM)되는 독특한 제한사항을 가진 검색으로 키의 길이가 고정되어 있지 않으므로 이진검색 등의 일반적인 검색방법을 사용할 수 없다. 포워딩 검색은 하드웨어 기반[4,5], 프로토콜-기반[6], 그리고 소프트웨어 기반[7,8] 검색의 3가지로 구분할 수 있다. 포워딩 테이블을 SRAM, CAM 등 속도가 빠른 특별한 하드웨어에 저장하고 검색하는 방법은 주기억장치의 저속 메모리 접근을 회피하려는 것이고, 새 프로토콜을 고안하려는 방법은 포워딩 검색정보를 경로를 구성하는 노드들에 분산함으로써 간단하게 동작하는 일치검색을 사용하거나 거대한 포워딩 테이블 검색범위를 제한함으로써 상대적으로 복잡한 LPM 검색을 대치하려는 시도이다. 이러한 두 가지 기법들은 비용이 더 투자되어야 하거나 새 프로토콜을 보급하는 것이 매우 어려운 측면이 있다.

소프트웨어 기반 연구는 포워딩 테이블의 자료구조 개선을 통해 LPM 검색을 빠르게 하려는 시도로 포워딩 테이블을 SRAM과 같은 고속 메모리에 저장할 수 있는 작은 크기로 압축하거나 저속 메모리에 저장되더라도 접근횟수를 줄일 수 있는 자료구조 개발을 목표로 한다. 전통적으로 자료구조 개선을 시도하는 많은 연구들에서 이진 트라이[8] 구조와 래딕스 트리(radix tree)[9]에 대해 많은 관심을 기울여 왔다. 트라이를 이용한 프리픽스 검색에서 트라이의 링크는 IP 프리픽스의 조각(segment)을 표시하며, 이러한 링크의 연결은 목적지 IP 주소와 비교될 프리픽스 주소로 간주될 수 있다. 트라이를 이용한 검색에서는 루트노드에서 출발해서 단말노드 방향으로 검색하던 중 도착한 임의의 노드가 더 이상 목적지 IP 주소의 일부와 일치하는 링크를 갖고 있지 않다면 그 노드는 패킷이 전송될 출력 링크의 주소를 가진 포워딩 테이블의 엔트리 색인 값을 갖고 있게 된다.

패트리샤 트라이(Patricia trie)[10]는 이진 트라이를 개량한 것으로 트라이의 계층을 압축하고 불필요한 중간노드를 생략하여 메모리를 압축한다. 패트리샤 트라이는 BSD 커널[11]에서 채택되었으며 다양한 IP 주소 검색연구의 출발점으로 사용되고 있다. [12]는 동일한 길이의 프리픽스로 그룹을 구성하고 각 그룹을 해시 테이블에 저장한다. 가장 긴 프리픽

스의 그룹부터 해시 검색을 하고 차례로 짧은 길이의 프리픽스 그룹으로 검색을 진행한다. [7]은 프리픽스 검색에 이진검색을 적용할 수 있도록 프리픽스를 범위의 개념으로 전환하여 Low, High 프리픽스로 확장하였고, 캐시 워드 크기를 반영하여 multiway 이진검색을 제안하였다. 프리픽스 분포가 균등하다고 가정할 때 검색속도는 Log_2N, multiway 검색속도는 log_mN이 된다. IP 주소검색에 사용된 다른 시도는 해시 테이블을 이용하는 것이다[13]. 해시 테이블은 프리픽스 전체를 저장하여야하고, 캐시에 저장되는 것을 전제로 하지만 백본 라우터의 해시 적중률이 좋지 않음이 알려져 있다[14].

Degermark[8]의 트라이 기반 연구는 IP 프리픽스로 구성된 트라이를 특정한 길이로 정렬(align)하고, 각 노드를 표현하는데 필요한 공간을 압축하기 위해 프리픽스 정보를 가진 노드에 비트 1을 할당(assign)하고 그 외의 경우에는 0을 할당했다. 이렇게 할당된 비트정보를 넓이우선탐색 순서로 정렬함으로써 비트 벡터(bit vector)를 만든다. 노드의 프리픽스 정보 즉, 포워딩 테이블 엔트리의 색인 값을 계산하기 위해서는 비트 벡터의 처음부터 목적지 IP 주소와 일치하는 노드까지 비트 값 1의 개수를 카운트 하면 된다. 그러나 비트 벡터 기법은 3 단계 트라이 구조만 지원하고, 트라이 링크를 저장할 때 많은 메모리 공간이 필요하다. Varghese[1]의 controlled-prefix 기법은 트라이가 다양한 인코딩 기법을 통해 압축될 수 있다는 점은 고려하지 않고 단지 트라이의 계층 개수를 압축함으로써 늘어난 연산량이 캐시 접근을 대신하도록 한다.

본 연구는 트라이 압축에 의한 메모리 공간과 검색시 필요한 연산량의 조절을 통해 최적의 프리픽스 검색 성능을 제공할 수 있는 하이브리드 기법을 제안하며, 프로세서 고속화가 급진전됨에 따라 연산량의 증가가 전체 검색비용에서 차지하는 비중이 낮아져 더 많은 연산이 필요한 압축 기술이 사용되어도 문제가 되지 않음을 확인할 수 있었다.

3. Bit-Map 트라이

트라이는 서로 길이가 다르지만 동일한 프리픽스를 공유하는 문자열을 저장하는데 널리 사용되는 자료구조의 하나이다. 예를 들어 사전을 트라이로 구성하면 단어검색은 문자가 할당된 트라이 링크를 따라 내려가는 것이 된다. 검색할 단어와의 비교는 더 이상 따라갈 링크가 없거나 비교할 문자가 없을 때까지 계속된다. 마지막에 도착한 트라이 노드는 특정한 위치를 가르치는 색인을 갖고 있는데, 바로 이 특정한 위치에 사전에 기술된 단어의 정의가 저장되어 있다.

3.1 Bit-Map 트라이의 구조

IP 프리픽스의 비트 혹은 일부분을 문자처럼 생각한다면 사전검색과 같은 방법으로 트라이가 포워딩 테이블을 색인할 수 있다. 예를 들어 어떤 링크가 IP 프리픽스의 2비트를 표시한다고 하면 노드로부터 4개의 링크가 발생하고 각 링크에는 좌측부터 00, 01, 10, 11 비트가 할당된다. 사전검색과 마찬가지로 포워딩 테이블의 모든 프리픽스 엔트리는 각 링크에 할당된 비트열의 집합이 되고 각 노드들은 일치하는 포워딩 테이블 엔트리를 가리키는 색인 포인터를 저장하게 된다. 이때 노드와의 일치는 검색할 IP 프리픽스와 루트노드부터 해당노드까지 연결된 링크에 할당된 비트의 연결 합과 일치 여부로 판단한다. 물론 어떤 노드까지 경로에 할당된 비트열의 조합과 일치하는 포워딩 엔트리가 없다면 그 노드는 색인을 갖지 않는다.

트라이 검색의 성능개선을 위해서는 노드에서 분기하는 링크의 개수 즉, 분기도(degree of branch)를 높임으로써 목적지 노드까지 탐색하면서 링크에 할당된 비트열과 비교하는 횟수를 줄일 필요가 있다. 참고로 분기도를 높이는 것은 하나의 링크가 표시하는 비트수를 늘임으로써 깊이를 얕게 하는 것으로 분기도에 포함된 비트수는 하나의 트라이 계층을 형성한다. 예를 들어 32비트 주소에서 8비트씩으로 분기도가 정해지면 하나의 계층은 8비트, 8개의 트라이 링크로 구성되고 트라이는 4개의 계층이 된다. 분기도를 높일 때는 링크별로 표시할 비트 길이에 맞도록 기존 프리픽스를 확장(subnetting)해서 짧은 프리픽스를 더 긴 프리픽스로 대체하는 효과를 갖는다. 예를 들어 포워딩 테이블에 010* 프리픽스가 있고, 각 링크가 2비트를 표시하면 링크경로 0100*와 0101*는 010* 엔트리를 가리키게 된다. 이것은 각 링크가 4개의 비트열 조합중의 하나를 표시함으로써 트라이가 짝수 프리픽스만을 표시할 수 있기 때문이다.

그러나 깊이가 얕아진 트라이는 색인 포인터를 포

함한 많은 수의 노드 때문에 더 많은 저장 공간을 사용하게 된다. 트라이의 크기가 캐시보다 커지면 트라이는 저속의 주기억장치에 저장되고, 얕은 깊이의 트라이 검색에 필요한 비트열 비교횟수가 줄어듦에도 불구하고 성능이 나빠질 수 있다. 주기억장치의 접근속도가 캐시보다 수배이상 늦다는 것을 감안하면 좋은 검색성능을 얻기 위해서는 LPM 검색이 캐시검색 위주로 수행되어야 한다.

본 연구의 트라이 기반의 포워딩 검색 구조는 트라이를 L2 캐시에 저장할 수 있는 크기로 압축하기 위해 트라이의 자식 포인터(child pointer)와 프리픽스 엔트리 색인을 각 1비트씩으로 인코딩함으로써 2개의 비트열 즉, cBM(child Bit-Map)과 pBM(prefix Bit-Map)을 생성한다. 예를 들어 트라이의 분기도가 8비트이면 링크에 연결된 노드는 2^8비트의 pBM과 cBM을 갖고, 각 비트열에서 비트 1은 자식 포인터 혹은 프리픽스 엔트리 색인의 존재를 의미한다. 참고로 우리가 사용하는 트라이가 bit-map으로 트라이 정보를 표현하므로 이 연구에서 사용하는 트라이를 BM 트라이로 부른다. BM 트라이는 자식 포인터와 프리픽스 엔트리의 색인을 직접 제공하지 않고 비트열이 포함하고 있는 1의 개수로 간접적으로 표현한다. cBM의 1의 개수는 cBM이 저장된 노드 테이블의 색인으로 다음에 검색할 트라이 링크와 노드의 위치를 지정하며, pBM의 경우에는 포워딩 테이블의 색인이다. 참고로 pBM과 cBM 배열은 넓이 우선 탐색 순서로 노드 테이블에 저장된다.

그림 1은 간단한 BM 트라이에서 비트 1 개수를 세는 방법을 보여준다. BM 트라이의 각 노드는 4비트의 pBM과 cBM 배열로 구성되어 있으며 가상의 목적지 주소 100111에 맞는 프리픽스 색인을 검색한다. BM 트라이는 3개의 2비트 세그먼트 10, 01, 11을 cBM에 대한 오프셋으로 차례로 적용해서 단말노드까지 검색한다. 루트의 cBM 배열에서 세 번째 비트가 1이고, 배열의 왼쪽부터 세 번째 비트사이에 1의 개수가 3이므로 검색은 네 번째 노드(세그먼트 10에 해당)로 이동한다. 참고로 숫자들은 0부터 시작하고 노드들은 넓이우선 탐색 순서로 배열되어 있다. 네 번째 노드의 cBM에서 두 번째 비트 역시 1이므로 루트부터 세 번째 노드의 모든 비트 1개수와 네 번째 노드의 첫 번째 비트까지 1의 개수를 세면 그 값은 8이다. 최종적으로 단말노드 8에 도달하고, 네 번째 cBM 비트

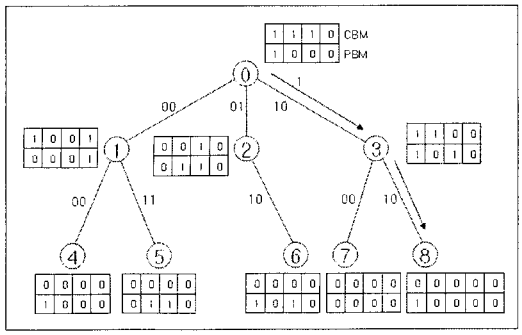


그림 1. BM 트라이의 활용 예제

는 0이므로 더 이상의 자식 포인터는 없고 검색은 종료된다. 마지막으로 프리픽스 색인 계산을 위해 0~7번 노드의 pBM에서 1의 개수를 세고, 8번째 노드의 네 번째 비트까지의 1 개수를 더하면 10이 된다. 따라서 100111 프리픽스에 알맞은 출력 링크의 정보는 포워딩 테이블의 10번째 엔트리에서 찾을 수 있다.

비트열 정보를 사용하는 BM 트라이는 프리픽스의 범위가 겹칠 때 구분을 위해서 추가적인 프리픽스 엔트리가 필요하다. 앞에서 가정한 01* 프리픽스 이외에 0101*이 포워딩 테이블에 들어 있고 0111을 검색한다고 가정해보자. 만약 BM 트라이가 pBM배열에서 0100과 0101 비트만 1로 설정했다면 0111은 pBM의 0111 비트가 0 이고 pBM의 좌측으로부터 센 1의 개수가 2이므로 01* 대신 포워딩 테이블의 두 번째 엔트리인 0101*을 선택할 것이다. 0111이 올바른 프리픽스를 선택하도록 하려면 pBM에서 0111 비트 위치에 1을 설정하고, 포워딩 테이블에 01*과 같은 출력포트를 갖는 새 엔트리를 삽입해야한다. 그러나 추가되는 엔트리들 때문에 포워딩 테이블이 계속 커지는 것을 방지하기 위해 BM 트라이는 계산된 색인을 실제 오프셋으로 변환해주는 변환 테이블을 사용한다. 예를 들어 pBM의 0100, 0101, 0111 위치까지 1의 합이 각각 7,8,9라고 가정하면 변환 테이블은 7, 8, 9 번째 엔트리에 7, 8, 7 색인을 갖고 있다.

비트열의 1을 세는 연산에서 발생하는 부담을 줄이기 위해 BM 트라이에서는 미리 계산된 1의 누적 개수를 함께 저장하고 있다. 더 자세히 말하면 BM 배열을 몇 개의 비트 조각(chunk)으로 구분한 후 각 조각별로 미리 계산된 1의 개수(PreCnt)를 함께 저장한다. 참고로 분할된 비트 조각의 크기가 크다면 시프트-증가(shift-and-increment) 연산 대신 십진

수-1 개수(decimal-to-1's count) 변환표를 사용해서 1의 개수를 세는 속도를 증가시킬 수 있다.

3.2 Bit-Map 트라이의 복잡도

BM 트라이 검색 알고리즘은 2단계로 구성되어있다. 먼저 cBM 배열을 이용해서 단말노드까지 검색하고 pBM 배열을 이용해서 색인을 계산한다. 따라서 BM 트라이의 최악의 시간 복잡도는 식 (1)과 같으며 n은 BM 트라이의 계층수이다. 식 (1)에서 T_{chunk}는 cBM 배열 선정 시간, T_{bit}는 선정된 cBM에서 비트 위치를 결정하는 시간, $T_{mBitCnt}$는 m 비트열에서 1을 세는 시간이고, T_{sum}은 PreCnt에 저장되어 있던 누적치를 합산하는 시간이다. T_{inst}는 하나의 명령어를 실행하는 시간이다.

$$\begin{aligned}&(n-1)\times O(one-step-traversal)\\&\quad+O(prefix-computation)\\&=n\times O(one-step-traversal)\\&=T_{chunk}+T_{bit}+T_{mBitCount}\times T_{inst}+T_{preCnt}\end{aligned}\quad(1)$$

앞에서 설명한바와 같이 트라이 검색 복잡도 n×O(one_step_traversal)은 BM 배열의 비트 조각 크기와 트라이 계층 수에 의존적이다. 따라서 BM 트라이의 검색성능을 높이기 위해서는 BM 트라이의 크기가 캐시 크기보다 작아지도록 트라이의 계층수와 비트 단위의 크기를 줄여야한다. 식 (2)는 Varghese[1]의 연구에서 W-bit 높이의 트라이를 k 계층으로 구성할 때 필요한 메모리 크기를 계산하는 식을 BM 배열구조를 반영하여 수정된 것으로 트라이 계층의 깊이에 따라 달라지는 메모리 크기를 계산할 수 있으며, 이 식을 이용하여 L2 캐시에 저장할 수 있고 동시에 최소 크기의 메모리를 사용하는 계층의 크기를 결정할 수 있다. 식 (2)는 그림 2의 트라이 분할구조를 기반으로 트라이 전체가 1 계층일 때부터 k 계층까지 재귀적으로 트라이 크기를 계산한다.

$$\begin{aligned}&T[W,k]=min_{h\in\{k-1..W-1\}}\\&T[W-h,k-1]+nChild_{W-h}\times 2^{h}\\&T[W,1]=2^{W}\end{aligned}\quad(2)$$

그림 2의 *cost*는 W bit의 트라이를 k 계층으로 분할하여 저장할 때 필요한 메모리 크기을 (W-h) bit 트라이가 k-1개 계층이 될 때의 메모리 크기와 h 높이의 k번째 계층이 (W-h+1)부터 W 깊이까지 확장된 노드의 개수를 합산한 것이다. 따라서 메모리 크기가 최소가 되기 위해서는 각 계층에서 확장되는 노드의 개수가 최소화 되는 깊이에서 계층의 위치를 결정한다. $nChild_{W-h}$는 트라이 깊이 W-h에서 자식 링크를 갖는 노드의 개수이고, 2^h는 h깊이에 형성되는 비트열의 길이이다.

식 (3)은 노드의 개수를 사용하는 대신 이 연구에서 제안하는 BM 배열의 크기를 반영하여 트라이의 메모리 크기를 계산한다. 트라이 한 노드는 2개의 2^n-bit BM 배열을 가지며, BM 배열은 m-bit 단위로 분할되고 각 단위는 16-비트 PreCnt 변수를 갖는다. 식 (3)에서 두 번째 항은 W-h 위치에서 연결되는 깊이 h의 pBM과 cBM배열을 의미하고, 그리고 세 번째 항은 BM 배열이 m-bit 단위로 분할 될 때 PreCnt 변수에 대한 저장 공간을 나타낸다.

$$\begin{aligned}&SizeT[W,k]=min_{h\in\{k-1..W-1\}}T[W-h,k-1]\\&+nChild_{W-h}\times 2^{h}+nChild_{W-h}\times 2^{h-4}\times C\\&SizeT[W,1]=2^{W}+2^{W-4}\times C,\ C=2byte\end{aligned}\quad(3)$$

트라이 각 계층의 길이를 다르게 구성하면 필요한 메모리 공간을 더 줄일 수 있다. 그림 3A는 첫 번째 계층의 길이가 h일 때 세 개의 자식 트라이들은 서로 다른 길이를 갖고 있다. Varghese의 연구[1]에서 각 계층의 길이를 서로 다르게 할 수 있는 재귀적 수식을 가변 계층 깊이(variable stride)라는 이름으로 제공하

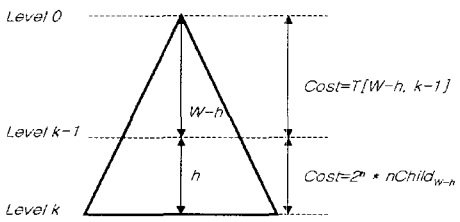


그림 2. 동적 프로그래밍에 의한 k-계층 트라이의 cost 함수: nChildW h는 W-h 깊이에서 W-h+1로 연결하는 링크를 가진 노드의 수를 의미

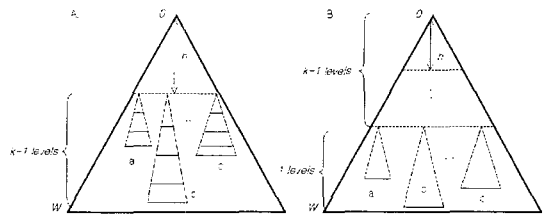


그림 3. Varghese(A)와 복합기법(B)의 가변 계층 깊이 기법의 비교

고 있지만 그림 3A의 자식 트라이를 위해서 모두 다른 계층의 깊이를 계산하는 것은 계산비용이 높은 약점이 있다. 이 연구에서는 메모리 크기를 줄이면서 동시에 복잡도를 제어하여 그림 3B처럼 마지막 계층에서만 서로 다른 깊이를 허용하고 다른 계층은 동일한 깊이를 갖도록 하였다. 참고로 모든 계층 간 깊이가 같은 경우는 고정 계층 깊이(fixed stride)라고 한다.

그림 4는 전형적인 포워딩 테이블의 프리픽스의 길이가 24비트 이하라는 것을 보여준다. 따라서 24비트 이하의 프리픽스를 32비트에 정렬하면 매우 많은 메모리를 낭비하게 된다. 예를 들어 3개 계층의 길이가 14, 22, 32로 정해졌다면 22번째 비트를 지나 24번째 비트에서 종료되는 단말노드의 비율이 99%가 됨에도 불구하고 32비트까지 확장하므로 써 많은 메모리를 낭비하게 된다. 참고로 23번째 비트에 만들어지는 노드의 최대 개수는 23부터 32번째 비트 사이에 존재하는 모든 노드의 합이다. 식 (4)는 (k-1) 계층의 길이가 식 (3)에 따라 결정될 때 가변 계층 깊이가 적용되는 k번째 계층에 필요한 메모리 크기를 나타낸다. 두 번째 항은 마지막 계층에서 끝나는 노드의 개수가 nChild이고 각 노드가 HT(R) 깊이를 가질 때 BM 배열의 크기이고, 세 번째와 네 번째 항은 PreCnt 변수와 마지막 계층의 길이를 저장하는 테이블의 크기이다.

가변 계층 깊이는 마지막 k 계층을 검색할 때 각 노드의 깊이를 결정하기 위해 한 번 더 캐시를 검색해야한다. 그림 5에서 트라이의 마지막 계층이 가변 계층 깊이일 때 BM 트라이의 자료구조 예제를 볼 수 있다.

가변 계층 깊이 트라이에서 프리픽스를 검색하는 것은 기본적으로 고정 계층 깊이 트라이에서와 동일하다. 다만 우리가 제안하는 가변 계층 깊이 트라이에서는 그림 3B와 같이 마지막 계층에서만 가변 계층

$$Opt(N,r) = SizeT(j,r-1) + \sum^{nChild_j} 2^{height(R)} + 2^{height(R)-1} \times C \quad (4)$$

깊이가 적용되므로 마지막 계층 검색이 다르다. k계층의 가변 계층 깊이 트라이가 있을 때 IP 프리픽스 1010 0000을 검색하는 예를 그림 5에서 고려해보자. 참고로, 그림 5A는 k번째 계층이 2비트 길이이고, 5B는 3비트이다. 먼저 목적지 IP 프리픽스를 A지역에서 검색하면, 깊이가 2비트이므로 IP 프리픽스에서

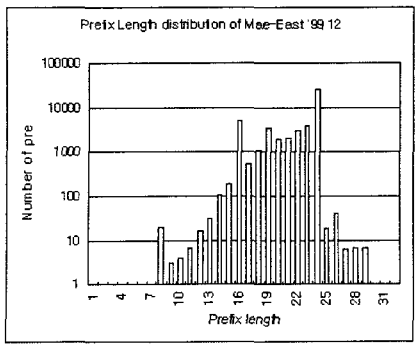


그림 4. IPMA DB의 프리픽스 길이 분포

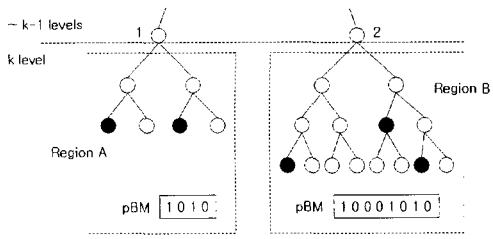


그림 5. BM 트라이에서 가변 계층 깊이 적용 방법

처음 2 비트 '10'을 오프셋 값 2로 사용하여 pBM을 검색한다. 즉, 루트부터 이 pBM 이전까지의 모든 1의 합 α와 pBM의 3번째 비트까지의 1의 합계를 더하여 색인 α+2를 구한다. 목적지 IP 프리픽스에서 남은 비트열 '10 0000'은 더 이상 검색할 트라이 노드가 없으므로 사용하지 않는다.

동일한 IP 프리픽스를 B지역에서 검색하면 B 지역의 깊이가 3비트이므로 IP 프리픽스의 첫 3비트 '101'을 offset 값 5로 사용한다. B 지역의 pBM에서 6번째 비트까지의 1의 합 2와 루트부터의 1의 합 β를 더하여 β+2 색인을 구할 수 있다. 이때 가변 계층 깊이에서는 A, B 지역의 경우와 같이 마지막 계층의 깊이가 서로 다르므로 검색과정에서 계층 깊이 정보가 필요하다. 이 정보는 그림 5의 경우 노드 1, 2에 해당하는 k 계층 트라이의 루트 노드에 각각 4비트 길이로 저장된다. 만약 마지막 계층이 16비트 이상의 길이로 만들어 지는 경우에는 4비트 이상을 할당하여야 한다.

본 연구에서 제안하는 복합 검색기법은 구현시 3단계로 구성된다. 첫 단계는 라우팅 테이블의 프리픽스 정보를 완전이진 트라이로 구축한다. 두 번째 단계는 동적 프로그래밍을 사용하는 controlled-prefix 기법을 적용하여 주어진 계층 개수에 대해 최적의

트라이 계층간 높이를 산출한다. 마지막으로 세 번째 단계에서는 bit-map 압축 단계로 트라이 정보를 bit-map으로 구성하여 포워딩 테이블을 구성한다. 동적 프로그래밍을 이용하여 계층 개수에 따른 계층별 높이 계산은 라우팅 테이블의 상태에 따라 변화될 수 있기는 하지만 자주 변동되어 즉시 반영해야할 정도의 즉성은 높지 않으므로 주간 단위 또는 월간 단위로 수행되어도 충분하므로 off-line 형태로 구현되어도 충분하다. 첫 번째와 세 번째 단계는 라우팅 테이블의 변경주기에 맞추어 즉시 동작하여야 하는데 실험결과 평균적으로 0.5GHz CPU에서 100ms 정도의 시간을 소모한다. 이 속도는 라우팅 테이블이 변경되는 주기를 1초 정도로 가정할 때 충분히 작은 시간이다.

4. 실 험

본 연구에서 제안하는 하이브리드 프리픽스 검색 기법의 성능을 측정하기 위해서 계층 개수의 변화에 따른 메모리 크기와 검색성능의 변동 상태, 프로세서의 성능변화에 따른 검색성능의 변화를 측정했다. 첫째 bit-map 트라이를 다양한 계층으로 구성하여 각각의 메모리 크기와 검색성능을 측정하였다. 둘째, 트라이의 마지막 계층을 대상으로 분기도를 고정한 고정 계층 깊이 기법과 마지막 계층의 분기도가 고정되지 않은 가변 계층 깊이 기법의 성능을 비교하였다. 마지막으로 프로세서의 진화가 매우 빠른 상황을 반영하여 다양한 성능의 프로세서 환경에서 성능을 측정했다.

실험에 사용된 라우팅 테이블은 지금은 종료된 IPMA[6] 프로젝트의 백본 라우팅 테이블이고, 검색 대상 IP 프리픽스는 랜덤으로 만들어진 주소이다. 검색속도는 백만 개의 IP프리픽스를 투입하여 얻어진 총 검색시간을 IP 프리픽스의 개수로 나눈 평균값이며, 실험결과의 검색속도는 모든 라우팅 테이블에 대해 측정된 평균값을 사용했다. 참고로, 표 1에 실험에 사용된 IPMA 백본 라우팅 테이블의 정보를 정리하였다. 자료는 99년도에 수집된 것으로 Aads와 Paix를 제외한 Mae-East 등은 백본 테이블의 크기이다. 원래의 프리픽스 개수와 Bit-map 트라이에 저장된 프리픽스의 개수가 틀린 것은 라우팅 테이블을 트라이로 변경할 때 완전이진 트라이[3]로 변경되기 때문으로 중복 프리픽스가 생성된 것이다.

4.1 Bit-map 트라이의 메모리 크기

트라이의 계층 수 변화에 따르는 압축효과를 측정하기위해 bit-map 트라이의 계층 개수를 2부터 6까지 변화하면서 메모리 크기를 측정했다. 계층 위치는 동적 프로그래밍 기법을 사용하여 메모리 크기가 최소가 되는 계층의 위치를 계산하고, 그림 6에 계층의 개수가 2~6사이에서 변화할 때 메모리 크기와 검색속도의 변화를 표시했다. 메모리 크기는 각 계층의 크기를 모두 합산하여 하나의 트라이 크기가 도출됐으며, 검색속도는 Intel 551.2MHz CPU, 512KB L2 캐시 사양의 리눅스 서버에서 IPMA의 다섯 가지 라우팅 테이블에 대해 실험한 평균값이다.

그림 6은 메모리 크기와 검색속도 사이에 유의미한 관계가 있음을 보여준다. 계층의 개수가 많아질수록 메모리 크기는 줄어드는데 이것은 계층이 세분화될수록 프리픽스 노드가 없는 널(null) 노드가 배제되어 bit-map 배열의 개수가 감소되기 때문이다. 3~6 계층의 메모리 크기는 그다지 많은 변화를 보이지 않고 조금씩 줄어드는 효과를 보이는 반면 검색속도는 미세하게 증가하고 있다. 계층 3~6에서 메모리 크기의 감소에도 불구하고 검색속도가 미세하게 증기로 DRAM 메모리 접근을 캐시 접근으로 대치하는 이유는 모든 트라이가 L2 캐시보다 작은 크 효과를 얻은 후에 계층이 증가한 만큼 계층 검색 횟수가 증가했기 때문이다. 결과적으로 높은 메모리 압축효과를 얻기 위해 트라이 계층을 지나치게 세분화할 필요는 없다고 판단된다. 즉, L2 캐시 크기보다 작은 3~6개 계층의 트라이에서 비슷한 검색성능을 얻을 수 있으므로 목표 검색성능과 L2 캐시의 크기를 감안하여 포워딩 테이블 구축이 더 용이한 계층 개수를 결정하는 것이 좋다고 판단된다.

표 1. IPMA DB('99.12) 의 프리픽스 정보

	No. of Original Prefix	No. of Prefix in the Bit-Map 트라이
Mae-East	48290	89621
Mae-West	32462	56329
PacBell	26897	44136
Paix	18093	16768
Aads	10434	28361

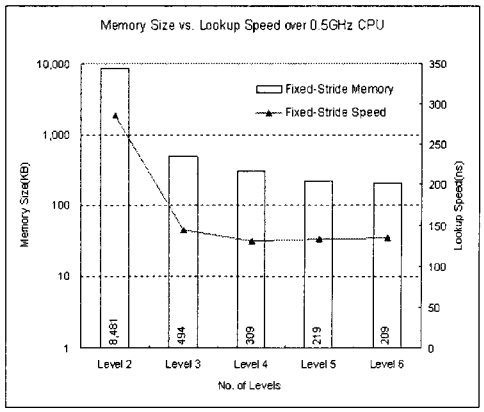


그림 6. 고정 계층 깊이 기법에서 메모리 크기와 검색속도 비교

4.2 Bit-map 트라이의 메모리 최적화

앞 절에서 살펴본 바와 같이 Bit-map 트라이의 메모리 크기는 프리픽스 검색성능과 직접적으로 연관이 있다. 즉, 포워딩 테이블을 구현하는 bit-map 트라이 크기가 작아질수록 검색속도는 개선된다. 물론 그림 6에서 트라이 검색속도가 극적으로 개선되는 포워딩 테이블 크기의 경계값(threshold)은 시스템의 L2 캐시 크기이다. 우리는 앞에서 언급한 바와 같이 제한된 계층 개수 범위에서 트라이를 최대한 압축함으로써 더 나은 성능을 얻을 수 있으므로 고정 계층 깊이 트라이를 가변 계층 깊이 기법으로 변경하고 메모리 압축률을 측정하였다.

첫째, 고정 계층 깊이 기법은 k개 계층의 깊이가 계층간에 동일하게 설정된다. 두 번째, 가변 계층 깊이 기법은 동적 구조로 k-1개 계층은 고정된 깊이를 갖고, k번째 계층은 다양한 깊이를 갖는다. 가변 계층 깊이 기법은 마지막 k번째 계층의 깊이를 라우팅 테이블의 상태에 맞추어 동적으로 조절함으로써 불필요한 트라이 정보를 배제하는 방법이다. 특히 가변 계층 깊이 기법은 고정 계층 깊이 기법의 계층별 분기도를 그대로 적용하고, 오직 k번째 계층에서만 라우팅 테이블의 상태에 따라 결정된 값을 갖는 기법-1과 전체 계층의 분기도를 다르게 가질 수 있는 기법-2 두 가지로 구분하여 비교했다. 이렇게 두 가지 기법으로 구분하는 이유는 가변 계층 깊이 기법과 고정 계층 기법의 효과를 비교하는데 효과적이기 때문이다.

그림 7은 2개의 가변 계층 깊이 기법과 하나의 고정 계층 깊이 기법에서 bit-map 트라이의 크기와 검색속도를 함께 표시한 것이다. 메모리 크기를 의미하는 막대그래프가 계층별로 3개씩 표시하였는데, 좌측 막대는 고정 계층, 중간 막대는 가변 계층 기법-1이고, 우측 막대는 가변 계층 기법-2의 메모리 크기이다. 고정 계층 깊이 기법의 크기는 계층 값 k가 커질수록 작아진다. 이것은 계층 구분이 세밀해질수록 프리픽스 정보나 링크정보가 없는 불필요한 노드정보가 제외되기 때문이다. 가변 계층 깊이 기법-1, 2는 2~4개 계층까지는 크기가 감소하지만 5 계층 이상에서는 조금씩 증가한다. 이것은 불필요한 노드가 제외되는 효과보다 bit-map 트라이에 추가되는 계층 k의 깊이 정보 즉, 분기도 정보의 크기가 더 커지기 때문이다.

고정 계층 깊이 기법과 가변 계층 깊이 기법-1은 크기와 검색속도가 거의 비슷한 값을 기록하고 있다. 이것은 가변 계층 깊이 기법-1이 k번째 계층에서만 동적으로 최대 깊이를 측정하여 bit-map 트라이를 구축하는데 반영함으로써 메모리 감축효과가 그다지 크게 나타나지 않았기 때문이다. 반면에 가변 계층 깊이 기법-2는 모든 계층의 분기도를 결정할 때 최대 깊이를 감안하므로 성능이 향상된다. 그러나 계층 4 이상에서는 모든 압축방법이 비슷한 크기와 검색속도를 보이고 있다. 이것은 계층 개수의 증가가 bit-map 트라이 크기 압축에 의한 효과를 상쇄하고, 메모리 검색횟수의 증가로 연결되기 때문이다.

표 2는 프리픽스 검색이 완료되는 계층의 분포를 관찰함으로써 계층 개수 증가가 메모리 검색횟수 증가로 연결되어 검색속도가 늦어지는 이유를 보여주고 있다. 즉, 계층 개수 k가 증가함에 따라 트라이의 2~6번째 계층을 검색하는 비율이 증가하는 만큼

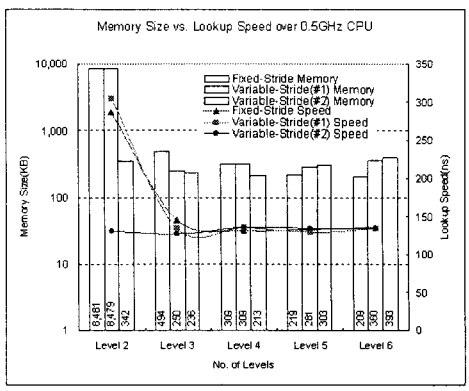


그림 7. 세 가지 bit-map 트라이 기법에서 메모리 크기와 검색속도: 500MHz 펜티엄 2 CPU

메모리 검색횟수가 증가하여 검색속도가 느려진다. 2개의 계층을 가진 고정 계층 깊이 기법은 모든 프리픽스 검색이 계층 1에서 100% 완료되어 한 번의 메모리 검색이 발생했다. k가 3일 경우 약 4%의 프리픽스 검색이 2번째 계층에서 검색 완료되어 메모리 검색횟수가 4% 증가하고 속도가 느려진다.

표 3은 IPMA의 Mae-East DB를 대상으로 동적 프로그래밍을 적용하여 k의 변화에 따라 계산된 계층의 위치이다. 이 표의 값은 하나의 예제이며, 다양한 DB에 따라 다른 위치가 계산될 것이다. 고정 계층 깊이와 가변 계층 깊이 기법-1은 모든 k에 대해 k번째 계층이 24~32의 8-bit 깊이로 구성된다. 이것은 매우 작은 개수의 프리픽스들이 32-bit 길이를 갖고 있기 때문이며, 가변 계층 깊이-2의 경우에는 16, 19, 22 등 다양한 깊이를 갖고 있다. 참고로 표 3의 위치 값의 차가 분기도이다.

4.3 Bit-Map 트라이와 프로세서의 성능

알고리즘을 설계할 때 흔히 선택할 수 있는 교환 가능한 요소로 메모리 크기와 프로세서의 연산능력을 생각할 수 있다. 알고리즘을 위해 필요한 메모리의 크기가 클 때 계산량의 증가를 감수하고 자료구조를 압축하여 메모리 크기를 줄일 수 있다. 물론 그 반대의 경우도 선택할 수 있다. Bit-map 트라이도 이러한 개념을 적용할 수 있다. 즉, 종래의 트라이가 과다한 메모리를 요구하는 단점을 가지므로 bit-map 압축을 통해 필요한 메모리를 줄일 수 있다. 가변 계층 깊이 기법도 이러한 목적을 만족시키는 방법이다. 이미 언급한 바와 같이 이러한 메모리 압축은 필연적으로 계산량의 증가를 불러일으킨다. 그러나 근래의 프로세서 처리속도 발전은 매우 고무적인 해결방법을 제시해주고 있다. 즉, 메모리 압축에 따라 늘어난 계산양이 프로세서 처리 속도의 증가로 상쇄될 수 있다.

그림 8은 프로세서 속도와 L2 캐시의 크기가 다른 3개의 시스템에서 프리픽스 검색속도를 비교한 것이다. 실험은 고정 계층 깊이 기법과 2개의 가변 계층 깊이 기법을 5가지 종류의 IPMA DB에 적용한 결과이다. 0.5GHz 프로세서의 평균 검색속도는 130ns, 3GHz 시스템은 평균 25ns 정도를 보여주는데 이 수치를 단순 비교하면 약 5.2배 정도로 프로세서 클록

표 2. 다양한 계층수와 세 가지 검색기법에 대한 프리픽스 검색 완료 계층의 분포

	No. of Bit-Map 계층	Hit on the 계층 1(%)	계층 2(%)	계층 3(%)	계층 4(%)	계층 5(%)	계층 6(%)
Fixed-stride	2 계층 트라이	100.00	0.00	-	-	-	-
	3	96.32	3.68	0.00	-	-	-
	4	93.66	5.47	0.88	0.00	-	-
	5	73.58	21.59	4.05	0.78	0.00	-
	6	71.14	22.79	3.79	1.85	0.44	0.00
Variable-stride scheme-1	2 계층 트라이	100.00	0.00	-	-	-	-
	3	98.45	1.55	0.00	-	-	-
	4	93.67	5.44	0.89	0.00	-	-
	5	73.64	21.59	4.00	0.78	0.00	-
	6	71.14	22.81	3.92	1.94	0.20	0.00
Variable-stride scheme-2	2 계층 트라이	93.84	6.16	-	-	-	-
	3	93.83	3.91	2.27	-	-	-
	4	74.75	19.98	3.29	1.99	-	-
	5	74.82	19.86	3.31	1.63	0.38	-
	6	74.81	19.89	2.16	1.92	0.75	0.48

표 3. Mae-East DB에 동적 프로그래밍을 적용하여 산출한 계층별 깊이

	No. of Bit-Map 계층	Depth of 계층 1	계층 2	계층 3	계층 4	계층 5	계층 6
Fixed-stride & Variable-stride scheme-1	2 계층 트라이	24	32	-	-	-	-
	3	18	24	32	-	-	-
	4	16	21	24	-	-	-
	5	12	17	21	24	32	-
	6	11	16	19	22	24	32
Variable-stride scheme-2	2 계층 트라이	16	32	-	-	-	-
	3	16	19	32	-	-	-
	4	11	16	19	32	-	-
	5	11	16	19	22	32	-
	6	11	16	18	20	22	32

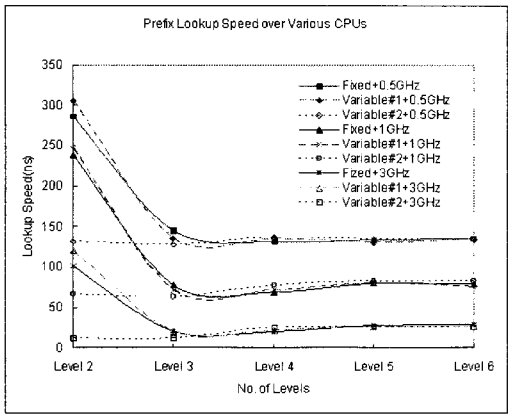


그림 8. 다양한 프로세서에서의 프리픽스 검색속도 측정: 0.5GHz+ 512KB L2 캐시, 1GHz+256KB, 3GHz+512KB

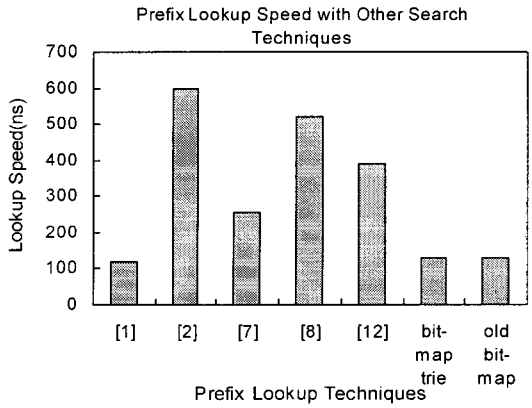


그림 9. Bit-map 기반 복합기법과 기존 연구의 검색속도 비교

차이 6배에 근접하고 있다. Bit-map 트라이 검색방법은 계산양의 증가를 CPU 처리속도의 증가로 대치할 때 매우 좋은 효율을 보여준다.

4.4 기존연구와 비교

그림 9는 기존의 몇 가지 검색 기법들과 bit-map 트라이의 성능을 비교한 것이다. 기존 연구와 성능을 비교하기에 앞서서 비교대상 연구들의 특징을 살펴보면 다음과 같다. Nilsson의 연구[2]는 구축된 트라이가 최대 32개 링크로 연결되어 최악의 경우 32번의 메모리 접근이 발생하는 문제를 해결하기 위해 트라이의 레벨을 압축하는 기법이다. [7]은 메모리 접근 1회에 두 가지 경로만 선택 가능한 구조인 이진검색을 캐시 워드 크기가 허용하는 한 m 다중검색으로 전환하여 $\log_m N$의 빠른 속도를 얻는 방법이다. Degermark의 연구[8]는 라우팅 테이블의 자료구조를 비트 벡터로 압축하여 캐시 접근을 통해 성능을 개선하는 점이 우리 연구와 유사하다. [12]는 동일한 길이의 프리픽스로 그룹을 구성하고 각 그룹을 해시 테이블에 저장한다. 가장 긴 프리픽스의 그룹부터 해시 검색을 하고 차례로 짧은 길이의 프리픽스 그룹으로 검색한다.

[1]은 앞에서 많은 설명이 있었으므로 생략한다. 다만 이 방법은 완전 해시를 위한 해시함수 구성에 필요한 시간이 13분 정도로 매우 길다는 약점이 있다. 또한 이 연구에서는 4-단계 트라이 분할이 가장 좋은 성능을 기록하였지만 본 연구의 bit-map 트라이는 3~4 단계의 구조가 더 좋은 성능을 기록하였다. 참고로 bit-map 트라이를 제외한 나머지 연구의 성능은 [12]에서 인용하거나 500MHz 펜티엄 2 CPU로 속도를 환산한 것이다.

5. 결 론

대용량 라우팅 테이블의 고속 검색을 위해 본 연구에서 bit-map 압축기법과 controlled-prefix 기법이 복합된 새로운 고속 IP 검색기법을 제안하였다. 제안된 기법은 라우팅 테이블의 프리픽스로 완전이진 트라이를 구성하고, 트라이의 노드와 링크정보를 한 비트로 표시함으로써 매우 높은 메모리 압축률을 제공한다. 높은 압축률이 적용된 트라이 포워딩 테이블은 고속의 SRAM 캐시에 저장할 수 있으며 결과적으로 저속의 DRAM 메모리 접근을 최소화 할 수 있다. Controlled-prefix 기법은 라우팅 테이블의 상태를 고려하여 트라이 크기가 최소로 압축될 수 있는 트라이의 계층 구조를 결정한다. 결과적으로 압축된 트라이 포워딩 테이블은 압축해제에 필요한 연산이 증가하지만 저속 메모리 접근이 통제됨으로써 고속 IP 검색이 가능해진다.

대용량 데이터의 교환이 일반적 상황이 되고 다양한 종류의 데이터가 융합되는 환경에서 고속 프리픽스 검색은 필수적이며, 다양한 기기가 통합되는 추세에서는 네트워크 보안을 위한 준비도 필수적인 요소이다. 프리픽스 검색은 이러한 요구사항에 대해 꼭 필요한 기초기술이며 고가의 하드웨어나 전용 하드웨어의 지원 없이 저가의 일반 시스템에서 고속 검색

서비스를 제공할 수 있는 소프트웨어 중심의 기법이라는 점과 신뢰성과 상업성을 위해 필요할 경우 하드웨어 기반 장비의 SoC로 전환 가능한 방법으로써 의미가 있다.

향후의 연구는 복합 검색기법의 SoC 전환에 필요한 준비 작업이 될 것이다. 라우팅 테이블이나 보안 조건은 실시간 적으로 변화가 발생하므로 IP 프리픽스 검색기법은 실시간으로 데이터 변화를 반영할 수 있어야한다. 우리는 bit-map 기반의 복합 검색기법에 대해 실시간 데이터 변화 방법을 모색할 것이며, 더 나아가 SoC 기반의 하드웨어 설계로 전환하는 부분에 대한 연구가 필요한 상태이다.

참 고 문 헌

[1] S. Venkatachary and G. Varghese, "Faster IP Lookups using Controlled Prefix Expansion," *Proc. of ACM Sigmetrics*, Vol. 26, No. 1, pp. 1-10, 1998.

[2] S. Nilsson and G. Karlsson, "Fast Address Lookup for Internet Routers," *Proc. of the IFIP TC6/WG6.2 Fourth International Conference on Broadband Communications*, pp. 11-22, 1998.

[3] 오승현, "능동적 트라이 압축을 이용한 고속 IP 검색," *정보처리학회 논문지*, 제10-A권, 제5호, pp. 453-462, 2003.

[4] P. Gupta, et al., "Routing Lookups in Hardware at Memory Access Speeds," *Proc. of IEEE Infocom '98*, San Francisco, Vol. 3, pp. 1240-1248, 1998.

[5] A. J. McAuley and P. Francis, "Fast routing table lookup using CAMs," *Proc. IEEE Infocom '93*, San Francisco, Vol. 3, pp. 1382-1391, 1993.

[6] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," *RFC 3031*, IETF, Jan., 2001.

[7] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookups using Multiway and Multicolumn Search," *Proc. of INFOCOM '98*, San Francisco, CA, Vol.3, pp. 1248-1256, 1998.

[8] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *Proc. of ACM SIGCOMM '97*, Cannes, France, pp. 3-14, 1997.

[9] T. Cormen, C. Leiserson, and R. Riverst, *Introduction to Algorithms*, The MIT Press, 1990.

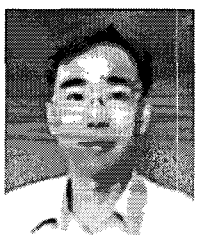
[10] G. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, 2nd Ed., Addison Wesley, 1991.

[11] K. Sklower, "A Tree-Based Routing Table for Berkeley Unix", *Proc. of the Winter Usenix Conference*, 1991.

[12] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookups," *Proc. of ACM SIGCOMM '97*, pp. 25-36, 1997.

[13] Torrent Networking Technologies Corporation, "High-Speed Routing table Search Algorithms," *A technical paper*, http://www.torrentnet.com.

[14] P. Newman, G. Minshall, and L. Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, Vol. 35, Issue 1, pp. 64-69, Jan. 1997.



오 승 현

1988년 동국대학교 전자계산학과(학사).

1998년 동국대학교 컴퓨터공학과(석사).

2001년 동국대학교 컴퓨터공학과(박사).

1987년~1996년 (주)대우엔지니어링.

2002년~현재 동국대학교 컴퓨터멀티미디어학부 조교수

관심분야 : 실시간 프로토콜, 차세대 네트워크, 센서 네트워킹