멀티웨이 트리에서의 최적화된 어드레스 룩업 방법

이 강 복, 이 상 연, 이 형 섭
한국전자통신연구원 라우터기술연구부
Tel : 042-860-5142

An Optimized Address Lookup Method in the Multi-way Search Tree

Kang-Bok Lee, Sang-Yeoun Lee, Hyung-Sub Lee
Router Technology Department, Electronics and Telecommunication Research Institute
E-mail : kblee@etri.re.kr

Abstract

This paper relates to a node structure of a multiway search tree and a search method using the node structure and, more particularly, to a search method for accelerating its search speed by reducing the depth of each small tree in a multi-way search tree. The proposed idea can increase the number of keys capable of being recorded on a cache line by using one pointer at a node of the multi-way search tree so that the number of branches in a network address search is also increased and thus the tree depth is reduced. As a result, this idea can accelerate the search speed and the speed of the forwarding engine and accomplish a further speed-up by decreasing required memories and thus increasing a memory rate.

I. 서 론

인터넷의 급성장으로 인해 호스트의 수와 이들 사이의 트래픽량은 엄청난 속도로 증가하고 있다. 특히 멀티미디어 서비스에 대한 요구가 커짐에 따라 인터넷 트래픽의 증가는 한층 더 빠른 속도로 진행될 예정이다. 따라서 인터넷 데이터의 전송 시 가장 큰 병목요소로 되어 있는 라우터의 성능은 점점 더 중요해지고 있다. 특히 데이터내의 목적지 주소를 이용해 목적지를 알아내고 데이터를 최종적으로 목적지로 포워딩하는 어드레스 룩업의 역할은 라우터의 기능 중 가장 중요한 요소로 부각되고 있다.

이러한 어드레스 룩업 방법은 다양하게 연구되어 왔으며 연구된 알고리즘과 구현 방법들은 라우터의 용도와 사용환경에 따라 성능과 비용이 달라지는 특성이 있다[1][2][3][4]. 따라서 라우터의 특성에 따라 적절한 알고리즘과 구현 방법을 선택하여야 한다.

본 논문에서는 다중 검색 트리(Multi-way search tree)에서 한 노드에서 사용하는 키의 개수에 관계없이 노드에 기록되는 포인터는 1개만 사용하도록 하여, 키, 키 포인터 및 노드 포인터의 합이 캐시라인의 크기와 일치되도록 하는 구조를 제시하고 있다. 이러한 수정된 다중 탐색 트리 구조를 사용함으로서 트리의 분기수를 줄이고 검색 속도를 고속화함은 물론 이로 인해 어드레스 룩업을 위한 메모리의 용량을 줄일 수 있는 장점이 있다.

II. 다중 검색 트리 알고리즘

종래의 다중 검색 트리는 디스크와 메인 메모리 사이에서 메모리 계층 구조를 고려하여 고안된 방법이다. 즉 저속의 메모리와 상대적으로 고속인 메인 메모리를 고려하여 한 번에 디스크에서 메인 메모리로 읽어 들인 디스크 블록을 이용하여 추가적인 저속의 디스크 엑세스 없이 분기 수를 메인 메모리의 속도로 늘리는 방법이다. 이러한 다중 검색 트리의 대표적인 예

가 Btree와 그 변형들이다.

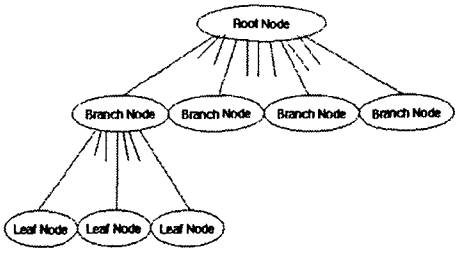

그림 1. 일반적인 Btree의 트리구조

그림 1은 일반적인 Btree의 트리 구조를 보여주고 있다. Btree가 사용된 일반적인 경우는 디스크에 저장된 데이터베이스를 엑세스하기 위해 역시 디스크에 저장된 인덱스파일을 구성하는 데 사용된다. 데이터베이스를 엑세스하기 위한 키는 문자열로 구성되어 이 노드에서 비교 결과에 따라 연결될 다음 노드를 가리키기 위한 포인터의 길이에 비해 아주 많은 메모리를 차지하게 된다. 따라서, 한 번에 엑세스 가능한 디스크 블록안에 최대한 많은 노드를 넣기 위해 키가 차지하는 메모리의 양을 줄이기 위한 방법들이 고안되었다.

현대적인 프로세서 구조들이 일반적으로 취하는 계층적 메모리 구조는 심화되는 메모리 속도와 프로세서 속도차를 극복하기 위해 점점 대용량의 캐시가 프로세서에 탑재되고 있다. 이 캐시의 동작은 임의의 메모리 어드레스가 엑세스되면 이 메모리 어드레스를 포함하는 캐시라인 전체가 한꺼번에 프로세서의 캐시로 복사되고 캐시 내에 있는 데이터는 프로세서의 속도로 처리된다. 따라서, 다중 검색 트리의 한 노드를 한 캐시라인 크기로 만들면 추가적인 메인 메모리 엑세스 없이 분기 수를 대폭 늘일 수 있게 된다.

이러한 방법이 Lamson에 의해 제안되었고, 그림2는 6-way search tree로 구성된 다중 검색 트리의 캐시라인 구조를 보여주고 있다. 한 개의 노드는 캐시라인의 크기와 동일한 32Byte이며, 5개의 키, 6개의 포인터, 그리고 6개의 키 포인터로 구성되며 각각은 16bit의 크기를 가지게 된다[5].

그러나 한 노드에서 최대 분기 수를 결정하는 요인은 키(key) 값의 길이와 포인터(pointer)의 크기에 달려 있으므로, 키와 포인터 값을 저장하기 위한 공간을 최소화하여 한 노드안에 최대한 많은 키와 포인터를 기록하는 것이 문제이다. 또한 Lamson의 다중 검색 트리에서는 한 노드에 k개의 키 값과 k+1개의 다른 노드를 가리키기 위한 포인터(pointer)를 가지므로 노드에서 비교 결과에 따라 연결될 다음 노드를 가리키기 위한 포인터의 길이에 아주 많은 메모리를 할당 하여야 한다.

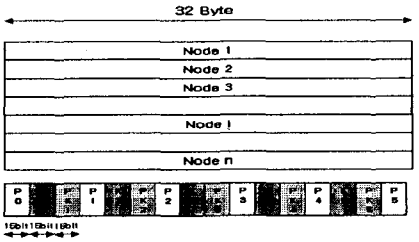

그림 2. Lamson의 다중 탐색 트리 구조

III. 제안된 IP 룩업 방법

3.1 제안된 다중 탐색 트리의 구조

본 논문에서는 다중 탐색 트리에서 한 노드에서 사용하는 키의 개수에 관계없이 노드에 기록되는 포인터는 1개만 사용하도록 하여 [키+ 키 포인터 + 노드 포인터]를 캐시라인 크기와 일치시켜 다중 탐색 트리를 고속화 하여, 네트워크 어드레스 탐색에서 분기수를 종래 방법에 비해 2배 가까이 늘리고 탐색 속도를 고속화 하며, 이에 필요한 메인 메모리의 용량을 줄이며, Longest prefix matching을 가능하게 하는 방법을 제안한다.

그림 3은 본 제안의 노드 구조를 Btree에 적용하여 8-way search tree를 구성한 예의 노드 구성을 나타내고 있다. 즉 한 개의 노드를 7개의 키와 키에 관련된 포인터로 구성하여 32Byte의 캐시라인과 일치 시키는 방법을 나타내고 있다. 그림에서와 같이 한 노드는 7개의 키(K1 ~ K7) 값과 , 1개의 포인터(Po), 그리고 1개의 키 포인터(Kp)로 구성되어 있다. 모든 키는 32비트로 구성되어 있고 포인터는 각각 16비트로 구성되어 있다.

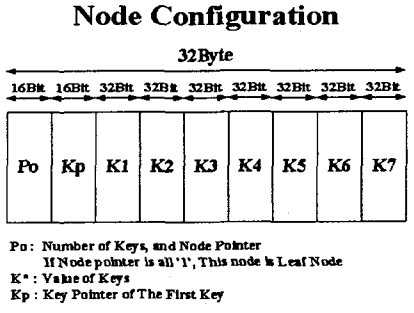

그림 3. 제안된 트리의 노드 구조

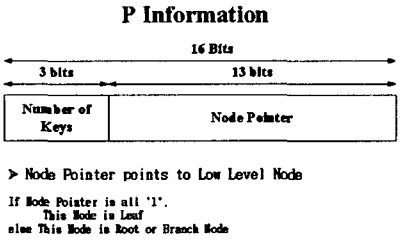

그림 4. 노드 포인터 정보

키포인터(Kp)는 노드의 첫번째 키에 해당하는 포인터를 나타내고 있는 값으로서, Branch 노드의 경우 키 값과 일치되었을때의 해당 포트 번호에 대한 정보를 나타내고, 리프 노드일 경우, Longest prefix matching을 가능하게 하는 영역정보를 나타내게 된다.

그림 4는 노드 포인터 Po의 정보를 보여주고 있다. 포인터 Po는 노드 포인터로서, 키의 개수와 노드 포인터에 대한 정보를 동시에 가지고 있다. 즉 MSB 3비트는 노드에 포함된 키의 개수를 나타내며, LSB 13비트는 트리의 하위레벨의 위치를 알려주는 포인터의 역할을 하게 된다. 이때 포인터의 값이 모두 '1'이면 해당 노드가 리프 노드임을 나타낸다. 만약 포인터의 값이 모두 '1'이 아니라면 루트 노드이거나 분기 노드로서 다음 노드를 찾아가는 포인터로 사용한다. 모든 키는 32비트로 구성되어 있고 포인터는 각각 16비트로 구성되어 있다.

3.2 제안된 다중 검색 트리의 검색 방법

제안된 다중 검색 트리에서 검색은 노드 리드, 주소 비교 및 주소지 이동의 순서로 진행된다. 그림 5는 생성된 트리에서 특정한 키를 가진 항목을 찾는 방법을 보여주고 있다. 먼저 8-way 노드를 읽은 다음 7개의 키 값을 검색할 IP 주소와 비교한다. 이때 IP 주소와 키 값이 일치할 경우에는 해당하는 키를 검색 완료한 것이므로, 검색 결과에 따른 키 포인터를 이용하여 목표 주소지로 찾아가면 된다.

그러나 비교 결과 일치하는 키가 없으면 먼저 노드 포인터의 값을 읽어 리프 노드인지 분기 노드인지 판별하게 된다. 만약 판별한 결과가 리프 노드에 해당한다면 키의 범위를 알아내고, 키 포인터를 계산하고 ,그리고 계산된 키 포인터를 이용하여 목표주소지로 찾아가게 된다. 노드 포인터 검사결과 분기 노드로 판명되면 다음 노드로 이동하기 위해 해당하는 포인터를 이용해서 다음 노드 포인터를 계산하고, 다음 노드로 이동한다. 앞과 동일한 방법으로 키 값을 읽고 키 값들

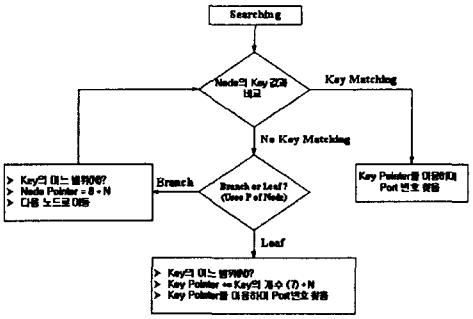

그림 5. 어드레스 탐색 순서도

을 비교한다. 최종으로 리프 노드에서 일치하는 키값이 나오거나 리프 노드에 도달할 때까지 반복한다.

3.3 제안된 다중 탐색 트리의 예

그림 6은 8-way search tree의 구성시 리프노드의 구성 예를 보여주고 있다. 1*, 1001*, 10100*과 같이 3개의 prefix로 구성되어 라우팅 테이블에 대한 트리를 구성 할 경우, 3개의 prefix를 기준으로 해당하는 범위의 키를 6개 생성한다. 생성된 6개의 키는 다음과 같다. 1* 으로부터 100000 과 111111, 1001* 로부터 100100, 100111, 그리고 10100* 로부터 101000, 101111 이다.

제안된 검색 트리는 Longest prefix matching을 수행하므로, 각 영역별로 포트 번호가 할당되어 있다. 루트 노드에부터 leaf 노드까지 검색하여 일치하는 키를 찾으면 포트에 대한 정보를 구할 수 있으며, leaf node에서 일치하는 키가 없으면 해당하는 범위를 통하여 포트에 대한 정보를 찾을 수 있다.

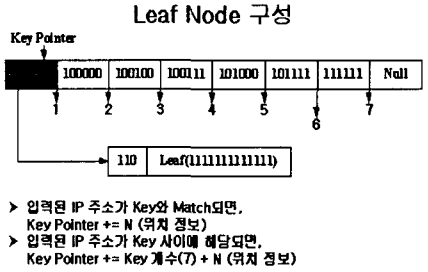

그림 6. 리프 노드의 구조

그림7은 6개의 키로 구성된 다중 검색 트리에서 Longest Prefix Matching을 수행하는 방법을 보여주고 잇다. 예를 들어, IP 주소가 110000인 패킷이 도착하면 생성된 tree에서 101111과 111111사이의 6번째 키 포인터에 해당되므로 키 포인터 = Kp + 키 개수 + 6의 메모리 위치의 식에서 해당하는 IP의 포트번호를 알 수 있다. 반면에, 입력된 IP 주소가 키와 일치되면 키 포인터는 Kp와 위치정보(N)에서 바로 구할 수 있다.

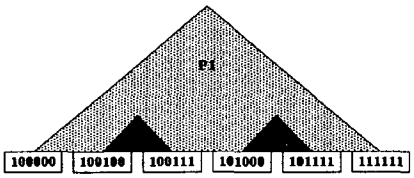

그림 7. 다중 검색 트리에서의 Longest Prefix Matching 방법

Ⅳ. 시뮬레이션 결과 및 평가

시뮬레이션 환경은 Sun의 Ultra60 및 솔라리스2.6의 환경에서 이루어졌다. 라우팅 테이블은 mae-east 의 2001년 3월 27일의 라우팅 테이블을 참조하여 IP Lookup 시간을 95% 신뢰도 구간으로 표현한 것입니다.[1)]

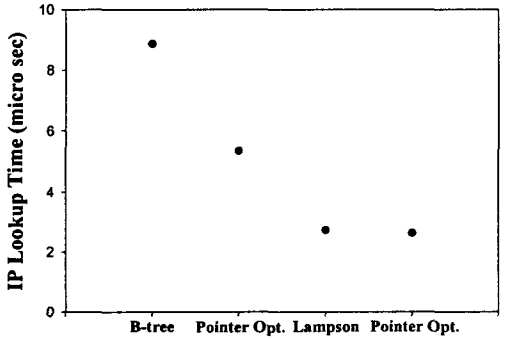

그림 8. 어드레스 탐색 시간

그림 8은 지금까지 알려진 다양한 룩업 알고리즘과 제안된 포인터 최적화(Pointer Optimization) 방법을 비교한 결과로서 전체 엔트리에 대한 평균 검색 시간을 나타내고 있다. 다중 검색 트리의 대표적인 방법인 Btree와 비교 할 경우 2배 정도의 빠른 검색 속도를 보여주고 있으며, Lamson의 방법과 비교 할 경우 전체 엔트리에 대한 평균 검색 시간은 거의 동일한 결과를 보여주고 있지만, 사용되는 메모리의 양은 감소하게 된다.

1)ftp://ftp.merit.edu/statistics/ipma/cooked/routing_table/table/mae-east

Ⅴ. 결 론

룩업 성능을 판별하는 기준은 검색 속도, 메모리 용량, 업데이트 시간 및 다양한 형태로의 응용 가능성등이 있으나, 이들 모든 요건들을 충족 시키는 알고리즘은 많지 않다. 뿐 만 아니라 동일한 알고리즘일 경우에도 하드웨어로 구현 하는 방법에 따라 성능 차이는 크게 달라질 수 있다. 따라서 적용하고자 하는 라우터 시스템의 용도에 따라 적절한 알고리즘의 선택 및 하드웨어 구현 방법을 선택하는 것이 중요하다.

본 논문은 다중 검색 트리에 있어서 분기수를 최소화 함으로서 메모리 용량을 최소화 하고 검색 속도를 증가 시킬 수 있는 어드레스 룩업 알고리즘을 제안하고 있다. 제안된 다중 검색 트리 알고리즘은 현재 알려져 있는 다양한 룩업 알고리즘 및 자료 구조들과 복합적으로 적용이 가능하다. 따라서 하드웨어로 구현 시 table-expansion, pipelined memory access등의 방법을 병행함으로서 더 빠른 룩업 성능을 얻을 수 있다.

[참 고 문 헌]

[1] R. Perlman, *Interconnections, Bridges and Routers.* Reading, MA: Addison-Wesley, 1992.

[2] V. Srinivasan and G. Varghese, "Fast address look-ups using controlled prefix expansion," *ACM TOCS*, vol. 17, pp. 1-40, Feb. 1999

[3] Stefan Nilsson and Gunnar karlsson, "IP-Address Lookup Using LC-Tries," *IEEE JSAC*, Vol.17, No.6, June 1999.

[4] N. McKeown, P. Gupta, and S. Lin, "Routing lookups *in hardware at memory access speeds*," *in* Proc. IEEE Infocom'98, SanFrancisco, CA.

[5] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," in Proc. IEEE Infocom'98, SanFrancisco, CA.