멀티웨이 트리에서의 IP Prefix 업데이트 방안

이 상 연, 이 강 복, 이 형 섭
한국전자통신연구원 라우터기술연구부

IP Prefix Update of Routing Protocol in the Multi-way Search Tree

Sang Yeoun Lee, Kang Bok Lee, Heyung Sub Lee
Department of Router Technology, ETRI

Abstract

Since Multi-way Search Tree reduces the number of the memory access by increasing the branch factor, it is considered a method to archive the high-speed IP address lookup. Using the combination of initial 16 bit array and Multi-way Search Tree, it also reduces the search time of IP address lookup. Multi-way Search Tree consists of K keys and K+1 key pointers. This paper shows how the IP update of Multi-way Search Tree which consists of the one pointer within a node can be performed. Using the one pointer within a node, it increases the number of keys within a node and reduces the search time of IP lookup. We also describes IP updating methods such as modification, Insertion and Deletion of address entries. Our update scheme performs better than the method which rebuilds the entire IP routing table when IP update processes.

I. 서 론

과거의 인터넷에서는 라우팅 테이블의 크기가 작고 고속화의 필요성이 중요하지 않았지만, 호스트 수의 폭발적인 증가와 기존의 어플리케이션에 비해 높은 대역폭을 필요로 하는 새로운 어플리케이션의 출현으로 트래픽이 증가하여 라우터에서 사용되는 라우팅 테이블의 크기가 증가되고 고속화의 필요성이 대두되고 있다. 이러한 필요성을 충족시키기 위해서는 IP 룩업 속도의 고속화가 가장 중요한 핵심으로 나타나고 있다. 이러한 IP 룩업 방법 중의 하나가 다중 탐색 트리(Multi-way search tree)이며 Lampson은 초기의 16bit 어레이와 다중 탐색 트리를 연결하여 IP 룩업을 수행하였다. 그러나, 라우팅 테이블에서 수정, 삽입 및 삭제와 같은 업데이트를 수행하는 경우에 라우팅 테이블에 해당하는 전체 IP 경로들을 다시 재계산하여 트리에 삽입해야 하므로 트리의 업데이트 시간이 많이 소요된다. 이러한 단점을 해결하기 위해서 두 개의 메모리 뱅크를 이용하여 일정한 기간동안 두 개의 뱅크사이를 스위치 하여 IP 라우팅 업데이트를 수행하는 방법은 있으나 메모리 뱅크를 업데이트하는 시간동안에 다른 메모리 뱅크를 업데이트하지 못하는 단점이 있다. 본 논문에서는 이러한 문제점을 트리 내에서 해결하여 IP 업데이트 시간을 줄이는 방안을 제안하였으며 일반적인 B-tree에서도 약간의 수정으로 적용이 가능하다.

II. 다중 탐색 트리에서의 IP Lookup

어드레스 검색은 어드레스가 사용되는 방법에 따라 Exact Matching과 Longest Prefix Matching 방법이 있다. *Ethernet MAC(Medium Access Control)* 어드레스와 같은 경우는 어드레스의 모든 비트가 일치해야 하는 Exact Matching 방법을 사용해야 하고 IP version 4의 경우에는 찾고자 하는 키와 첫 번째 비트부터 가장 길게 일치하는 자료를 찾는 방법(LPM : Longest Prefix Matching)을 사용해야 한다[1, 2].

라우팅 테이블에 있는 IP 경로는 <Route Prefix, Prefix Length>로 표현된다[3, 4]. 이러한 IP 경로를 B-tree를 이용하여 Longest Prefix Matching을 수행하려면 부가적인 기능이 필요하다. 그림 1은 IP 경로 쌍으로 이루어진 영역 내에서 Prefix Q가 새로이 추가되는

경우에 고려하여야 할 사항들을 나타내고 있다[4].

Prefix Q의 추가로 이 범위 안에 있는 Prefix들은 Prefix Q의 업데이트에 영향을 받게 된다. Unique Prefix는 Prefix의 길이가 32 비트인 경우로 동시에 Start와 End로 표시되는 점이며 어떠한 영역도 갖지 않는 IP 경로를 의미한다. Prefix Q의 추가로 Q Start와 Start(1)사이의 영역, End(1)과 Start(3) 사이의 영역, End(3)과 Unique Prefix 사이의 영역 및, Unique Prefix와 Q End 영역사이가 Prefix Q의 추가로 동시에 업데이트되어야 할 영역들이다. 다중 탐색 트리의 경우에는 각 키가 Exact matching에 사용되는 포트 번호와 자기보다 큰 영역을 나타내는 영역 포트 번호를 갖고 있으므로 이 예의 경우에서 수정되어야 할 키들은 Q Start와 Q End의 포트와 영역 포트 그리고, End(1), End(3), 및 Unique Prefix의 영역 포트가 업데이트되어져야 한다.

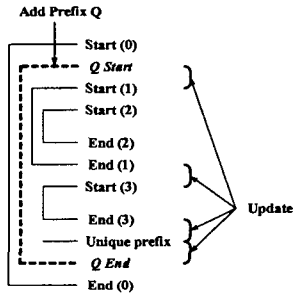

그림 1. 새로운 IP 경로의 추가

III. 다중 탐색 트리에서의 IP 업데이트

본 논문은 초기의 16비트 어레이와 다중 탐색 트리가 결합된 방법에서 한 캐시라인 안에 들어가는 노드 크기 내에서 분기 수를 최대화하여 검색을 고속화하고, B-tree의 문제점인 Longest Prefix Matching을 가능하게 하는 노드 구조와 다중 탐색 트리 내에서 IP 영역을 고려한 업데이트를 수행하는 방법을 제안한다. 다중 탐색 트리의 한 노드에서 사용하는 키의 개수에 관계없이 노드에 기록되는 포인터는 한 개만 사용하도록 하여 IP 어드레스 검색에서 노드 내에 저장 가능한 키의 개수를 늘임으로써 검색 속도를 고속화하고, 다중 탐색 트리의 노드 내에 한 개의 포인터 사용으로 인하여 하위 노드들이 연속적인 메모리에 할당된다는 이점을 활용하여 키의 삭제, 수정 및 추가와 같은 업데이트의 수행 시에 트리 내의 노드 사이를 이동하면서 스택을 이용하여 해당 범위의 키를 업데이트하는 방법을 제안하였으며 이 제안은 일반적인 B-tree에서도 적용이 가능하다.

그림 2는 초기의 16 비트 어레이와 8-way 검색 트리로 구성된 다중 탐색 트리를 구성한 예를 나타내고 있다. 동작 원리는 라우터에 입력된 목적지 IP 주소는 상위 16비트 세그먼트를 이용하여 초기의 16비트 어레이를 검색한다. 첫 번째 I 플래그가 'FALSE'로 설정되어 있으면 노드 포인터에 대한 정보가 없는 경우이므로 해당하는 포트 번호를 반환하고 룩업을 종료한다. I 플래그가 'TRUE'라면 노드 포인터가 지시하는 다중 탐색 트리의 루트 노드 주소로 이동하고 IP 주소의 하위 Offset를 이용하여 다중 탐색 트리의 검색을 시작하여 트리에서 포트 번호가 반환되면 IP 룩업을 종료한다. 이러한 초기의 16비트 어레이를 사용하는 IP 확장 방법은 Lampson에서 사용하는 방법과 동일하지만 다중 탐색 트리에서의 노드 구성과 업데이트 면에서 그 차이점이 크다. 초기의 16비트 어레이의 P, U, S 플래그는 16비트 어레이 테이블의 업데이트 시에 사용된다. P 플래그는 라우팅 테이블에서 해당 엔트리가 입력된 경우에 'TRUE'로 설정되며, U 플래그는 해당 엔트리가 라우팅 테이블에서 입력될 때, Prefix 길이가 16인 경우(Unique Prefix)에만 'TRUE'로 설정된다. S 플래그는 해당 엔트리가 IP 주소 범위에서 시작점인지 아니면 종료점인지를 나타낸다.

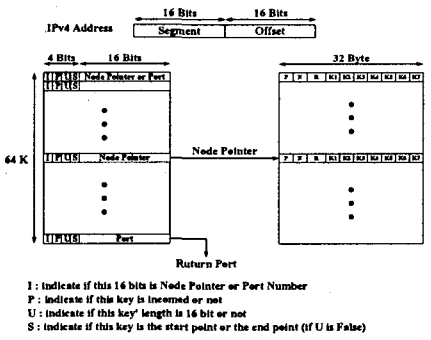

그림 2. 16 비트 어레이와 8-way Tree 구조

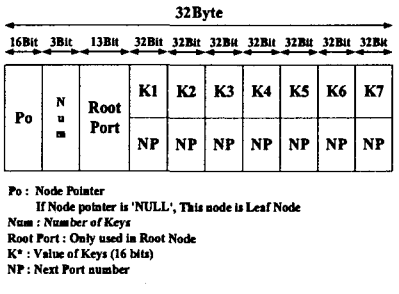

그림 3. 노드 구조

그림 3은 각 노드의 구조를 나타내고 있다. 포인터 Po는 노드 포인터로서, 하위 노드의 주소를 표시하며 leaf 노드인 경우에는' NULL'를 나타낸다. 3 비트 NUM은 노드내의 키의 개수에 대한 정보를 표시하고, Root Port는 이 16비트 어레이의 엔트리에 해당하는 포트 번호를 표시하며 루트 노드에서만 사용되고 branch 노드나 leaf 노드에서는 'NULL'로 설정된다. Root Port는 하위 노드 내에 있는 키가 16비트 어레이의 엔트리와 같은 범위에 속해 있는 경우에 루트 노드에 있는 Root Port를 참조한다. 그러므로, 16비트 어레이의 엔트리의 업데이트로 인하여 Root Port가 변경된 경우에 하위 노드에서 Root Port를 참조하는 키들이 같은 업데이트 과정을 수행해야 할 필요성을 방지하기 위해서 사용된다. 나머지 비트는 키 값과 키 값에 해당하는 포트 번호를 저장하기 위해 사용된다.

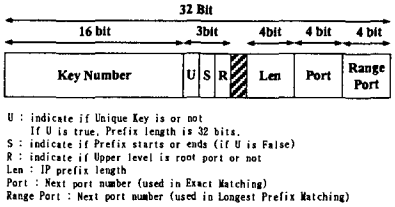

그림 4. 키의 구조

그림 4는 각 32 비트 키의 구성을 상세히 나타낸 것으로 검색을 요청한 IP의 하위 Offset과 노드 내의 16비트 키 값과 비교하며, 키와 동일한 값인 Exact Matching이 되는 경우에 사용되는 포트(Port = 4 비트)와 이 키 값보다 큰 경우에 사용되는 영역 포트(Range Port = 4 비트)가 저장되어 있다. Len은 IP Prefix의 하위 Prefix의 길이를 표시하며, 세 비트의 플래그 비트들은 업데이트 시에만 사용된다. U 플래그는 키의 Prefix 길이가 32비트인 경우에 설정되고(Unique Prefix), S 플래그는 해당 엔트리가 IP 주소 범위에서 시작점인지 아니면 종료점인지를 나타낸다. 그리고, R 플래그는 상위 IP 범위가 루트 노드인 경우에만 설정되어 IP 업데이트 시에 상위 IP 범위의 검색 시간을 줄이기 위해 사용된다.

그림 5는 라우팅 프로토콜에 의해서 IP Prefix가 업데이트를 수행하는 과정을 순서도로 나타낸 것이다. 업데이트는 크게 세 가지로 수정, 삽입 및 삭제로 분류할 수 있으며 각 분류에 따라 약간 다른 업데이트가 수행된다. 업데이트 시에 가장 먼저 수행되는 것은 해당 엔트리의 Prefix의 길이가 16보다 작은 지의 여부이다. Prefix의 길이가 16비트보다 작은 경우에는 초기의 16비트 어레이에서 P, U, S 플래그를 이용하여 간단히 업데이트를 수행한다. 이 16비트 어레이 테이블을 업데이트하는 방법은 Gupta에 의해 제안된 네 번째 방법과 동일하다[3]. 해당 엔트리의 Prefix의 길이가 16비트보다 긴 경우에는 해당 엔트리의 다중 탐색 트리에서 업데이트를 수행해야 한다. 우선, 해당 엔트리의 I 플래그가 설정되어 있지 않는 경우(노드 포인터가 널인 경우)에 업데이트 방법이 삭제가 아니라면 새로운 루트 노드를 만들고 새로운 키 값을 삽입하고 업데이트를 종료하지만, 삭제인 경우에는 해당키가 없는 경우이므로 에러를 반환한다. I 플래그가 설정되어 있으면(노드 포인터가 루트 노드를 지시하는 경우) 해당 엔트리의 루트 노드로 이동하여 해당 키 값이 존재하는지 여부를 판별한다. 키가 존재하지 않고 업데이트가 삭제가 아니라면 새로운 키의 쌍을 삽입하고 입력된 IP 경로보다 더 큰 범위의 IP Prefix의 포트 번호를 찾고 해당하는 IP 경로 범위 내에서 트리의 업데이트를 수행한다. 만약, 키가 존재하지 않고 업데이트가 삭제인 경우에는 에러를 반환한다. 키 값이 존재하면 업데이트 방법이 수정이나 삭제인 경우에 해당한다. 수정인 경우에는 검색한 키 값에서부터 해당하는 키 쌍이 발견될 때까지 트리 업데이트를 수행하고, 삭제인 경우에는 더 큰 범위의 IP Prefix의 포트를 찾고 해당하는 키 쌍까지 트리 업데이트를 수행한 이후에 키 쌍을 트리에서 삭제한다.

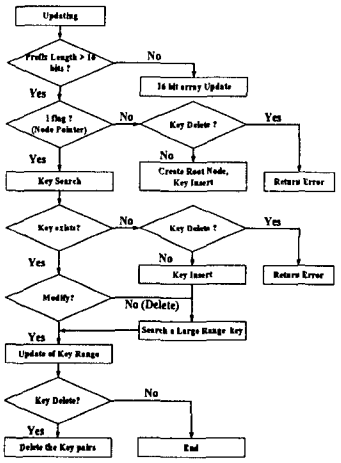

그림 5. IP Prefix 업데이트 순서도

그림 6은 K1과 K7, K2와 K6, K3과 K5가 키의 한 쌍을 이루고 있으며, K2와 K6의 IP 쌍이 수정되는 경우이다. K4는 독립된 키(Unique Prefix)이며 스택 수행 과정에 영향을 주지 않는다. 우선, K2를 검색하여 검색 과정에서 지나온 경로를 레벨마다 존재하는 레지

스터에 해당 노드 주소와 통과한 위치 정보를 저장한다. 예에서 K2가 하위 첫 번째 노드에 두 번째 위치에서 발견되었으므로 K2의 포트(Port, Range Port)를 수정하고 스택에 저장, 오른쪽 키(K3)로 이동한다. 이동할 때마다 각 키에 저장된 U, S 플래그를 조사하여 스택의 저장 여부를 판별한다. S 플래그가 설정되어 있으면(시작점) 스택에 저장하고 다음 키로 이동하며, S 플래그가 설정되어 있지 않으면(종료점) 스택에서 해당 키를 제거하면서 K6을 만날 때까지 각 키의 Range 포트를 업데이트한다. 만약, 검색 된 키가 branch 노드에 존재한다면 가장 근접한 오른쪽 하위 leaf 노드의 첫 번째 키로 이동하여 위의 업데이트 과정을 수행한다. 이 예에서는 K2의 포트와 Range 포트, K5의 Range 포트와 K6의 포트가 수정된다.

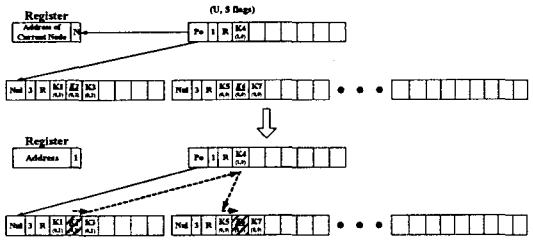

그림 6. 키 수정의 예

Ⅳ. 시뮬레이션

시뮬레이션 환경은 CPU 800MHz, RAM 256MB이며 리눅스 6.2에서 이루어졌다. 라우팅 테이블은 mae-east의 2001년 4월 11일의 라우팅 테이블을 참조하였으며 엔트리의 수는 약 19,630개에 해당한다.[1] 모의 실험은 IP Prefix의 수정, 삽입 및 삭제로 나누었으나 삽입과 삭제는 그 결과가 동일하여 하나의 결과로 표시하였다.

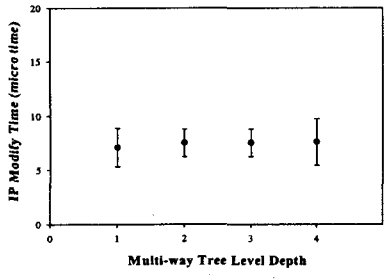

그림 7. IP Prefix 수정 시간

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

그림 7은 IP Prefix가 수정인 경우에 Multi-way의 level depth에 따른 IP 업데이트 시간을 평균과 표준편차로 표시한 것이다. level에 따른 수정 시간의 차이가 적은 이유는 IP Prefix 쌍이 이웃한 노드 내에 존재하는 경우의 수가 크기 때문이다. 그림 8은 삽입 및 삭제인 경우에 Multi-way의 level depth 별로 업데이트 시간을 측정한 것으로 레벨의 수가 증가할수록 메모리 엑세스의 수가 증가하므로 이로 인해 업데이트 시간이 증가함을 나타내고 있다.

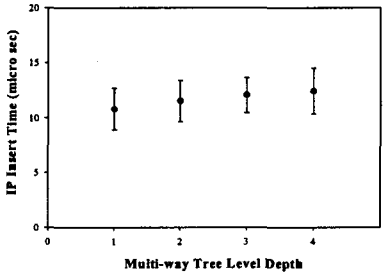

그림 7. IP Prefix 삽입 및 삭제 시간

Ⅴ. 결 론

Lampson의 IP 업데이트는 라우팅 테이블에 해당하는 전체 IP 경로들을 다시 재계산하여 트리에 삽입해야 하므로 트리의 업데이트 시간이 많이 소요되는 단점이 있다. 본 논문에서는 이러한 단점을 해결하기 위해서 다중 탐색 트리에서의 IP Prefix가 시작점과 종료점으로 표시되는 키의 쌍으로 구성된다는 것을 이용하여 수정, 삽입 및 삭제와 같은 IP 업데이트 시에 트리 내에서 이러한 키의 쌍을 이용하여 업데이트를 수행함으로써 IP 업데이트 시간을 줄이는 방안을 제안하였다.

[참 고 문 헌]

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

[2] A. McAuley and P. Francis, "Fast routing table lookup using CAM's," in Proc. IEEE Infocom 93.

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

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