IP 주소 검색을 위한 최적화된 영역분할 이진검색 구조

준회원 박 경 혜*, 종신회원 임 혜 숙*

Optimized Binary-Search-on-Range Architecture for IP Address Lookup

Kyong Hye Park* *Associate Member*, Hyesook Lim* *Lifelong Member*

요 약

라우터는 입력되는 패킷을 인터넷 주소 검색을 통하여 패킷의 목적지로 향하는 포트로 포워딩하는 역할을 하는데, 입력되는 속도와 같은 속도로 패킷을 포워딩하기 위해서는 매우 빠른 검색을 제공할 필요가 있다. 본 논문에서는 이진 검색을 이용한 IP 주소 검색구조에 대해 연구하였다. 대부분의 이진 검색 알고리즘들은 균형 이진 검색을 진행하지 않아 과도한 메모리 접근을 야기함으로써 검색속도가 느린 단점이 있다. 한편 영역분할을 이용한 이진 검색 알고리즘은 매우 빠른 검색 성능을 보이지만, 메모리 요구량이 크다는 단점이 있다. 본 논문에서는 영역분할 이진 검색에서 불필요한 엔트리와 항목을 삭제함으로써 라우팅 테이블의 크기를 최적화하여 메모리 요구량을 감소시키는 방법에 대하여 연구하였다. 이러한 최적화를 통하여 프리픽스의 개수와 비슷하거나 적은 수의 엔트리를 갖는 영역분할 이진 검색 라우팅 테이블을 구성할 수 있음을 보였다. 실제 사용되는 다양한 크기의 라우팅 테이블을 이용하여 영역분할 이진 검색의 원래 구조와 최적화된 구조의 검색 성능을 비교하였으며, 다른 여러 가지 이진 검색 알고리즘과의 성능을 비교하였다.

Key Words : IP address lookup, Binary search on range, Longest prefix match

ABSTRACT

Internet routers forward an incoming packet to an output port toward its final destination through IP address lookup. Since each incoming packet should be forwarded in wire-speed, it is essential to provide the high-speed search performance. In this paper, IP address lookup algorithms using binary search are studied. Most of the binary search algorithms do not provide a balanced search, and hence the required number of memory access is excessive so that the search performance is poor. On the other hand, binary-search-on-range algorithm provides high-speed search performance, but it requires a large amount of memory. This paper shows an optimized binary-search-on-range structure which reduces the memory requirement by deleting unnecessary entries and an entry field. By this optimization, it is shown that the binary-search-on-range can be performed in a routing table with a similar or lesser number of entries than the number of prefixes. Using real backbone routing data, the optimized structure is compared with the original binary-search-on-range algorithm in terms of search performance. The performance comparison with various binary search algorithms is also provided.

※ This research was partly supported by the MIC(Ministry of Information and Communications) under a HNRC-ITRC support program supervised by IITA(Institute of Information Technology Assessment).

* 이화여자대학교 SoC Desing Lab. (hlim@ewha.ac.kr)

논문번호 : KICS2008-02-071, 접수일자 : 2008년 2월 11일, 최종논문접수일자 : 2008년 10월 24일

Ⅰ. 서 론

라우터에서 입력된 패킷을 적절한 아웃 포트로 내보내기 위해서는 들어오는 패킷의 목적지 인터넷 주소를 라우터 내의 테이블에서 검색하여 테이블 내의 프리픽스와 일치하는 엔트리를 찾아내고, 그 엔트리의 아웃 포트를 읽어내는 과정이 필요하다. 이러한 과정을 IP 주소 검색이라고 하는데, IP 주소 검색의 문제점은 인터넷 주소의 네트워크 부분인 프리픽스의 길이가 다양하며, 입력된 패킷의 목적지 주소는 다양한 길이를 갖는 여러 개의 프리픽스와 일치 가능하다는 것이다. IP 주소 검색 알고리즘은 최장길이 일치(longest prefix match, LPM) 방법을 사용하는데, 이것은 길이가 다양한 프리픽스 중 라우터에 입력된 패킷의 목적지 주소와 가장 길게 일치하는 프리픽스를 찾아내는 것을 뜻한다[1].

최장길이 일치를 찾아내는 복잡한 과정에 대하여 라우터의 성능을 좌우 하는 것 중 가장 중요한 것은 메모리 접근횟수이다. 링크 속도가 증가함에 따라 들어오는 패킷을 빨리 포워딩하는 것이 라우터의 필수 요건이므로, 들어오는 패킷의 목적지 주소와 가장 길게 매치하는 프리픽스를 빠른 시간 내에 찾아내는 것이 매우 중요하며, 검색 속도는 메모리 접근횟수와 밀접한 관련을 갖는다. 다음으로 중요한 것은 라우팅 테이블을 저장하기 위해 요구되는 메모리의 크기이다. 댁내 네트워크와 같은 다양한 네트워크의 증가에 따라 프리픽스의 수가 증가하게 되었고, 따라서 라우터의 테이블 크기도 증가하고 있다. 라우팅 테이블을 IP 주소 검색을 위해 설계된 칩 내에 효율적으로 저장 할 수 있어야 하므로 요구되는 메모리 크기는 라우터 설계시에 고려하여야 할 중요한 문제 중의 하나이다.

영역분할 이진 검색 (binary-search-on-range, BSR) 알고리즘은 IP 주소 검색을 위하여 연구되어 온 이진 검색 알고리즘 중 유일하게 균형이진 검색을 제공하는 알고리즘으로서 검색 성능에 있어서나, 요구되는 메모리의 양, 그리고 확장성에 있어 매우 뛰어난 구조이다. 하지만 프리픽스의 개수보다 라우팅 테이블의 엔트리 개수가 최대 2배가 될 수 있어 엔트리의 개수나 요구되는 메모리의 양에 있어 최적화의 여지가 있다고 하겠다.

본 논문에서는 영역분할 이진 검색 알고리즘의 최적화를 위하여 영역분할 이진 검색 알고리즘의 최대 단점인 엔트리의 개수를 줄이고, 두 개의 선계산 필드를 하나로 합쳐 메모리 요구량을 줄이는 방법에 대하여 연구하였다.

본 논문의 구조는 다음과 같다. 먼저 Ⅱ장에서는 본 논문의 최적화 대상인 영역분할 이진 검색 알고리즘을 포함하여 기존에 연구되어온 이진 검색 알고리즘을 소개한다. Ⅲ장에서는 영역분할 이진 검색 알고리즘의 최적화 방법을 설명하고, Ⅳ장에서는 최적화 방법에 의하여 개선된 성능실험 결과 및 기존의 이진 검색 알고리즘과의 성능 비교를 보인다. Ⅴ장에서는 간단히 결론을 맺는다.

Ⅱ. 기존의 이진 검색 알고리즘

2.1 이진 트라이 (Binary Trie)

이진 트라이[2]는 트라이의 경로에 따라 해당하는 노드에 프리픽스의 라우팅 정보를 저장하는 방법이다. 거쳐온 경로를 통하여 프리픽스 값을 알 수 있기 때문에, 프리픽스를 따로 저장할 필요가 없는 장점이 있지만, 모든 노드에 프리픽스가 저장되는 것이 아니므로, 비어있는 노드로 인하여 메모리 요구량이 증가될 수 있으며, 메모리 접근 횟수에 있어서 IPv4의 경우 최대 프리픽스 길이인 32까지 커질 수 있다는 단점이 있다. 또한 프리픽스의 분포에 따라 불균형 트라이가 구성되므로 검색 시간이 일정치 못하다는 단점이 있다[3][4]. 표 1에 프리픽스 셋의 예시를 보였으며, 그림 1에 표 1의 프리픽스 셋에

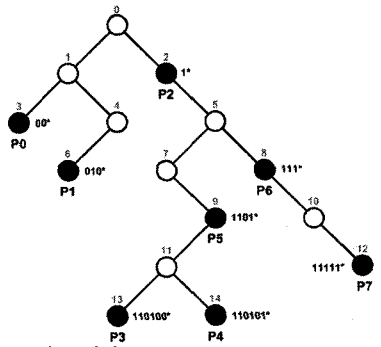


그림 1. 이진 트라이

표 1. 프리픽스 셋(set) 예시

No.	Prefix
P0	00*
P1	010*
P2	1*
P3	110100*
P4	110101*
P5	1101*
P6	111*
P7	11111*

해당하는 이진 트라이의 예를 보였다.

2.2 이진 검색 트리 (Binary Search Tree, BST)

이진 검색 트리[5]는 이진 트라이가 갖고 있는 단점인 빈 내부 노드를 없애며, 프리픽스 값에 대하여 이진 검색을 수행하는 방식이다. 이진 검색 트리 알고리즘은 다양한 길이를 갖는 프리픽스들의 크기 비교를 가능하게 하는 새로운 정의를 제안하여, 프리픽스들을 크기별로 정렬 한 후 이진 트리를 구성하였다.

이진 검색 트리 알고리즘에서 제안된 프리픽스의 크기에 관한 정의를 살펴보면, 프리픽스 길이가 같은 경우는 수학적인 값을 비교한다. 프리픽스 길이가 다른 경우에는 짧은 프리픽스 길이를 기준으로 하여 크기 비교를 하게 되는데, 예를 들어 1101*와 111*의 경우, 비교 부분은 110과 111이 되며 111이 110보다 크므로, 1101* 보다 111*이 크다고 정의 된다. 만일 비교 부분이 같은 경우, 길이가 긴 프리픽스의 다음 비트를 확인하여 그 비트가 0인 경우 작은 프리픽스로 정의 되고, 1인 경우 큰 프리픽스로 정의 된다. 예를 들어 111*와 11111*의 경우 비교 부분이 111로 같으므로, 긴 프리픽스의 다음 비트를 확인하게 되고, 그것이 1이므로 111*가 11111*보다 작은 프리픽스로 정의 된다.

하지만 이러한 정의를 사용하여 정렬된 프리픽스 리스트에 대하여 이진 검색 트리를 구성하면 이진 검색이 잘못 수행 될 수 있다. 예를 들어, P를 최장 일치 프리픽스로 갖는 어드레스가 입력 되었을 때, P를 부-스트링(sub-string)으로 갖는 다른 프리픽스가 존재하고, 이 프리픽스가 P보다 먼저 비교되는 경우에는 P가 존재하지 않는 다른 쪽 리스트로 검색이 진행될 수 있기 때문이다. 이러한 문제를 해결하기 위해 이진 검색 트리 구조에서는 프리픽스를 크기별로 정렬할 뿐만 아니라, 프리픽스의 종류를 인클로져(enclosure), 인클로즈드(enclosed), 디스조인트(disjoint)로 분류한다. 인클로져는 자신을 부-스트링으로 하는 다른 프리픽스가 존재하는 프리픽스이고, 인클로즈드는 인클로져를 부-스트링으로 갖는 프리픽스이며, 디스조인트는 부-스트링의 관계를 갖지 않는 프리픽스를 뜻한다. 예를 들어 {00*, 1*, 111*}로 이루어진 프리픽스 셋을 가정 할 때, 1*가 인클로져, 111*가 인클로즈드 프리픽스가 된다. 00*는 프리픽스 1* 나 111*와 부-스트링의 관계에 있지 않으므로, 디스조인트한 프리픽스로 정의된다.

이진 검색 트리는 먼저 디스조인트 프리픽스와

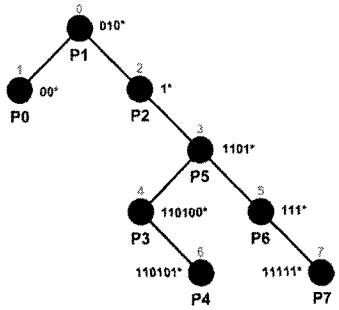


그림 2. 이진 검색 트리

인클로져 프리픽스만으로 이루어진 리스트를 정렬하여 중간에 위치한 프리픽스를 선택하여 이진 검색 트리의 노드로 결정하며, 인클로져 프리픽스가 선택이 된 경우, 해당 인클로즈드 프리픽스들이 리스트에 추가되는 방식을 재귀적으로 반복함에 의하여 이진 검색 트리를 구성하게 된다. 그림 2에 표 1의 프리픽스 셋에 대하여 구성된 이진 검색 트리를 보였다. 이진 검색 트리는 이진 트라이와 달리 빈 내부 노드가 없이 프리픽스들로만 이루어진 이진 검색 트리를 구성하는 장점이 있으나, 인클로져에 속하는 인클로즈드들의 수에 따라 트리가 급격하게 불균형이 될 수 있다는 단점이 있다. 이러한 단점을 개선하기 위해 인클로져가 갖는 인클로즈드 프리픽스들의 개수까지를 미리 고려하여 트리를 구성하는 방법이 제안되었으나, 이 알고리즘에서도 여전히 불균형이 존재한다[6].

2.3 영역분할 이진 검색 (Binary-Search-on Range)

본 논문의 최적화 대상인 영역분할 이진 검색 알고리즘[7]에서는 모든 프리픽스는 크기가 $[0,2^{32}-1]$인 직선상의 영역으로 표현될 수 있으며, 프리픽스의 길이가 길수로 작은 영역을, 짧을수록 큰 영역을 차지함을 이용한다. 먼저 프리픽스를 영역으로 표현하는데 있어, 다양한 길이의 프리픽스를 모두 최대 프리픽스 길이로 확장하되, 0으로 패딩한 값을 영역의 시작점(start point)으로, 1로 패딩한 값을 영역의 끝점(end point)으로 정한다. 그림 3에 표 1의 각 프리픽스가 나타내는 영역을 보였다. 영역분할 이진 검색 알고리즘에서는 영역의 시작점과 끝점이 모두

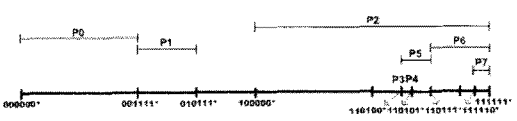


그림 3. 영역으로 표현된 프리픽스

표 2. 영역분할 이진 검색 알고리즘의 시작점과 끝점

Prefix Name	Prefix	Length	Start/End	Routing Entry
P0	00*	2	0	000000
			1	001111
P1	010*	3	0	010000
			1	010111
P2	1*	1	0	100000
			1	111111
P3	110100*	6	-	110100
P4	110101*	6	-	110101
P5	1101*	4	0	110100
			1	110111
P6	111*	3	0	111000
			1	111111
P7	11111*	5	0	111110
			1	111111

표 3. 크기순으로 정렬된 후 동일 프리픽스가 제거된 테이블

Prefix Name	Start/End	Routing Entry
P0	0	000000
P0	1	001111
P1	0	010000
P1	1	010111
P2	0	100000
P3, P5	0	110100
P4	-	110101
P5	1	110111
P6	0	111000
P7	0	111110
P2, P6, P7	1	111111

라우팅 테이블의 엔트리가 되는데, 각 엔트리에는 해당 엔트리의 값과 일치하였을 때의 포워딩 정보와, 해당 엔트리의 값보다 큰 경우에 해당하는 포워딩 정보를 선계산(pre-computation)하여 저장한다. 이러한 선계산을 갖는 엔트리로 이루어진 리스트를 크기별로 정렬 한 후, 이진 검색을 수행한다. 예를 들어, 표 1의 프리픽스에서, 가장 길이가 긴 프리픽스 길이인 6에 맞추어서 프리픽스 뒤에 0을 붙여 시작점(start point)을 만들고, 1을 붙여 끝점(end point)을 만든다. 각 프리픽스가 표현하는 영역의 시작점과 끝점을 표 2에 보였는데, 시작점은 0으로, 끝점은 1로 표현하여 시작점과 끝점을 구분하도록 하였다. 다음은 확장된 프리픽스를 크기별로 작은 것부터 큰 것 순으로 정렬 한 후, 확장된 프리픽스 중 동일한 프리픽스를 제거한다. 표 3에 정렬된 후 동일 프리픽스를 제거한 테이블을 보였다. 이 테이블에 대하여 선계산이 수행된다.

영역분할 이진 검색(BSR)에서는 각 라우팅 엔트리에 대하여 입력된 패킷의 목적지 IP 주소가 엔트리의 값보다 큰 경우의 최장일치 프리픽스에 해당하는 포워딩 정보 및 엔트리와 정확하게 일치하였을 때의 포워딩 정보를 선계산 하여 저장한다. 예를 들어, 그림 3에서 엔트리 111110의 경우, 프리픽스 P2, P6, P7에 공통으로 속하게 되나, 프리픽스 P7이 가장 긴 프리픽스이므로, "엔트리보다 큰 경우" 필드와 "엔트리와 같은 경우" 필드 모두에 프리픽스 P7에 해당하는 포워딩 정보를 저장하여야 한다. 엔트리 110101의 경우 프리픽스 P4와 프리픽스 P5에 공통으로 속하고, P4가 긴 프리픽스이므로 프리픽스 P4에 해당하는 포워딩 정보를 저장하여야 한다. 그러나 P4는 최대길이의 프리픽스로서, 입력된 패킷의 목적지 주소가 110101보다 큰 경우에는 해당하지 않는다. 그러므로 "엔트리와 같은 경우" 필드에는 프리픽스 P4에 해당하는 포워딩 정보를, "엔트리보다 큰 경우" 필드에는 프리픽스 P5에 해당하는 포워딩 정보를 저장하게 된다. 본 논문에서는 설명의 편의를 위하여 포워딩 정보대신 프리픽스의 이름을 사용하여 선계산을 수행하였으며, 그 결과를 표 4에 보였다.

표 4. 영역분할 이진 검색 알고리즘의 라우팅 테이블

Value	BMP 엔트리와 같은 경우	BMP 엔트리보다 큰 경우
000000	P0	P0
001111	P0	-
010000	P1	P1
010111	P1	-
100000	P2	P2
110100	P3	P5
110101	P4	P5
110111	P5	-
111000	P6	P6
111110	P7	P7
111111	P7	-

이 라우팅 테이블을 이용하여 이진 검색을 진행하는데, 영역분할 이진 검색 알고리즘은 엔트리 값과 정확하게 같을 때와 엔트리 값보다 클 때에 대해 포워딩 정보를 저장하였기 때문에, 검색이 아래로 진행 될 때 "엔트리보다 큰 경우" 필드의 값을 기억한다. 인풋 값이 비교되는 엔트리의 값보다 작아 검색이 위로 진행되는 경우에는 새로운 정보를 기억하지 않는다. 만일 인풋과 엔트리의 값이 정확히 일치하면 "엔트리와 같은 경우" 필드의 값을 포워딩 정보로 기억하고 검색을 종료한다.

예를 들어, 표 4에 보인 라우팅 테이블에 대하여 인풋이 111110이 들어왔음을 가정하면, 테이블의

중간 엔트리인 110100과 인풋 111110을 비교한다. 인풋이 크므로 검색은 아래로 진행되는데, 이 과정에서 110100의 "엔트리보다 큰 경우" 필드의 포워딩 정보인 P5를 기억한다. 다음은 아래쪽 엔트리의 중간 엔트리인 111000과 비교하며 인풋이 크므로, "엔트리보다 큰 경우" 필드의 포워딩 정보인 P6를 기억하고 다시 아래로 진행한다. 아래쪽 리스트의 중간인 111110과 비교되고, 인풋과 정확히 일치하므로 검색은 여기서 종료되며, 최종적인 포워딩 정보는 111110 의 "엔트리와 같은 경우" 필드의 값인 P7로 업데이트 된다. 또 다른 예로 110111이라는 인풋이 들어왔음을 가정하면, 중간 엔트리인 110100과 비교하여 인풋이 크므로 P5를 기억하고 아래쪽 리스트로 진행한다. 아래쪽 리스트의 중간 엔트리인 111000가 비교하여 인풋이 작으므로, 현재까지의 기억정보를 변경함 없이 위쪽 리스트로 진행한다. 위쪽 리스트의 중간 엔트리인 110111과 엔트리의 값이 일치하므로 P5로 현재까지의 기억정보를 변경하고, 검색을 종료한다.

영역분할 이진 검색은 다른 트리 구조와 달리 완벽하게 균형된 이진 검색(balanced binary search)을 이용하기 때문에 메모리 접근 횟수가 매우 작다는 장점이 있다. 일반적인 트리 구조의 경우에는, 노드가 많이 달린 쪽으로 검색이 진행 될 경우 검색 영역이 크게 줄어들지 않는다는 단점이 있으나, 영역분할 이진 검색은 메모리 접근 후 그 엔트리보다 작거나 큰 부분 중 한 부분은 검색영역에서 제외되므로, 한번의 비교로 검색해야 할 엔트리의 1/2이 검색 대상에서 제외되며, 이 과정의 반복으로 메모리 접근 횟수에 대한 검색성능이 매우 좋아지게 된다. 하지만 영역분할 이진 검색 알고리즘의 엔트리 개수는 프리픽스 개수의 최대 2배가 될 수 있는 단점이 있다. 영역분할 이진 검색 알고리즘의 라우팅 테이블의 엔트리 개수를 최적화 시킨다면, 라우팅 테이블을 저장하는데 요구되는 메모리 요구량을 감소시킬 수 있고, 더불어 검색속도의 향상을 기대할 수 있다. 또한 "엔트리보다 큰 경우" 필드와 "엔트리와 같은 경우" 필드로 나누어 선계산 된 필드를 하나로 합쳐 저장한다면, 메모리 요구량이 한층 줄어드는 효과를 기대할 수 있다.

Ⅲ. 영역분할 이진 검색 알고리즘의 최적화 구조

본 논문에서는 영역분할 이진 검색의 단점인 라우팅 테이블의 엔트리 수가 프리픽스 개수의 최대 2배가 될 수 있어 메모리 요구량(memory requirement)이 커지는 단점을 보완하기 위하여, 어떤 프리픽스의 끝점과 다른 프리픽스의 시작점이 연속되는 경우, 끝점을 제거하여 엔트리 개수를 줄이는 방법과, "엔트리와 같은 경우" 필드와 "엔트리보다 큰 경우" 필드를 하나로 합치는 방법을 통해 메모리 요구량을 줄이는 최적화 방법에 대하여 연구하였다. 이러한 최적화 구조는 [8]의 논문에서 기본 아이디어가 제안된 바 있으며, 본 논문에서는 구체적인 예시를 통하여, 최적화 구조의 원리를 설명하였으며, 기존의 영역분할 이진 검색 알고리즘과의 성능 비교뿐 아니라, 기존에 나와 있는 다양한 이진 검색 알고리즘과의 성능 비교를 통하여 최적화 구조의 장점을 보였다.

영역분할 이진 검색 알고리즘의 최적화 방법을 설명하면 다음과 같다. 영역분할 이진 검색 알고리즘에서 두 개의 필드가 필요했던 까닭은 프리픽스를 영역으로 표현하는데 있어 시작점과 끝점을 사용하였기 때문이다. 시작점만 가지고 프리픽스를 표현한다면, "엔트리와 같은 경우" 필드와 "엔트리보다 큰 경우" 필드를 분리할 필요가 없으며, 하나의 필드인 "엔트리 보다 크거나 같은 경우" 필드로 생각할 수 있을 것이다. 따라서 끝점 엔트리에 1을 더한 값이 다음 영역의 시작점이 될 것이며, 이 엔트리에 "엔트리 보다 크거나 같은 경우"의 정보를 저장한다.

이 과정에서 기존에 존재하는 엔트리와 끝점에 1을 더한 엔트리가 같은 경우, 끝점 엔트리를 지울 수 있으므로 테이블 엔트리 수의 감소를 가져온다. 여기서 고려하여야 할 사항은 최대 길이 프리픽스의 정보가 포함된 경우인데, 이 경우에는 두 개의 엔트리를 사용하여 표현하여야 한다는 점이다. 그 이유는 최대 길이의 프리픽스는 영역에서 한 점으로 표현되기 때문에, 기존의 엔트리에는 최대 길이의 포워딩 정보를 저장하여야 하며, 기존의 엔트리에 1을 더한 새로운 엔트리는 다음 영역의 시작점으로서, 해당 영역의 포워딩 정보를 저장하여야 한다.

표 4의 예로 엔트리 개수 및 필드 수에 있어 최적화된 라우팅 테이블을 표 5에 보였다. 끝점 엔트리인 001111에 1을 더한 값인 010000은 이미 엔트리로서 존재하므로, 0011111 엔트리를 제거할 수 있으며, 마찬가지로 끝점 엔트리 110111에 1을 더한 값인 111000 또한 이미 존재하므로, 110111 엔트리가 제거 될 수 있다. 그러므로, 두 개의 엔트리가 제거되었다. 끝점 엔트리 010111의 경우는 1을 더하여 엔트리 011000으로 변경되었다. 엔트리 110100는 최대 길이

표 5. 영역분할 이진 검색 알고리즘의 최적화된 테이블

Value	BMP 엔트리보다 크거나 같은 경우
000000	P0
010000	P1
011000	-
100000	P2
110100	P3
110101	P4
110110	P5
111000	P6
111110	P7
111111	-

프리픽스 엔트리이므로 P3의 포워딩 정보가 저장되고, 이 값에 1을 더한 새로운 엔트리가 생성되어야 하나, 이미 그러한 엔트리가 존재하므로 새로운 엔트리가 추가되지 않는다. 110101 엔트리의 경우 최대 길이 프리픽스 엔트리이므로, 새로운 엔트리인 110110을 생성하여 프리픽스 P5의 포워딩 정보를 저장하였다. 단, 라우팅 테이블의 마지막 엔트리는 검색 영역에 있어 가장 끝 점이 될 것이므로, 포워딩 정보는 "엔트리보다 크거나 같은 경우"가 아닌, "엔트리와 같은 경우"에 해당하는 정보가 저장된다.

최적화 구조에서의 검색과정은 일반적인 이진 검색 과정과 동일하다. 최적화된 테이블에 대하여 입력 주소가 접근된 엔트리의 값보다 큰 경우에는 미리 계산되어 저장된 포트 넘버를 기억하며 라우팅 테이블의 아래쪽으로 검색을 계속 진행한다. 입력 주소가 접근된 엔트리의 값보다 작은 경우에는 기억 정보의 변경 없이 라우팅 테이블의 위쪽으로 검색을 계속 진행한다. 입력 주소가 접근된 엔트리의 값과 같거나 더 이상 진행할 엔트리가 없을 때 검색이 종료되는데, 접근된 엔트리의 값과 같은 경우에는 저장된 포트 넘버를 출력하고 검색을 종료하고, 더 이상 진행할 엔트리가 없어 검색을 종료할 때에는 현재까지 기억된 포트넘버를 출력하고 검색을 종료한다. 검색과정에 대한 pseudo-code는 다음과 같다.

```
Search(input)
{
 low = 0; //low 주소는 라우팅 엔트리의 시작 주소를 default 로 가짐
 high = M-1;
 //M은 라우팅 엔트리의 개수로서
 // high는 라우팅 엔트리의 마지막 주소를 default 로 가짐
 While (low <= high) {
     mid = (low+high)/2;
     if (input < mid.value) { //입력된 주소가 중간 엔트리의 값보다 작은 경우
         high = mid-1;
     }
     else if (input > mid.value) { //입력된 주소가 중간 엔트리의 값보다 큰 경우
         low = mid+1;
         BMP_portnum = mid.portnum;
     }
     else if (input == mid.value) { //입력된 주소가 중간 엔트리의 값과 같은 경우
         BMP_portnum = mid.portnum;
         break;
     }
 }
 return BMP_portnum;
}
```

다음으로 검색 과정의 예를 통해 앞서 설명한 기존의 테이블과 같은 결과를 갖는지 설명한다. 예를 들어, 표 5에 보인 최적화된 라우팅 테이블에 대하여 인풋이 111110이 들어왔음을 가정하면, 테이블의 중간 엔트리인 110100과 비교한다. 인풋이 크므로 검색은 아래로 진행되는데, 이 과정에서 110100의 포워딩 정보인 P3를 기억한다. 다음은 아래쪽 리스트의 중간 엔트리인 111000과 비교하며 인풋이 크므로, 포워딩 정보인 P6를 기억하고 다시 아래로 진행한다. 아래쪽 엔트리인 111110과 비교되고, 인풋과 정확하기 일치하므로 검색은 여기서 종료되며, 최종적인 포워딩 정보는 111110 엔트리의 포워딩 값인 P7로 업데이트 된다. 또 다른 예로 110111이라는 인풋이 들어왔음을 가정하면, 중간 엔트리인 110100과 비교하여 인풋이 크므로 P3를 기억하고 아래쪽 리스트로 진행한다. 아래쪽 리스트의 중간 엔트리인 111000가 비교하여 인풋이 작으므로, 현재까지의 검색 정보를 업데이트 함 없이 위쪽 리스트로 진행한다. 위쪽 리스트의 중간 엔트리인 110110과 비교하여 인풋이 크므로 P5로 현재까지의 기억정보를 업데이트하고, 아래쪽으로 진행하여야 하나 더 이상 진행할 곳이 없으므로, 검색을 종료한다.

표 4와 표 5를 비교하여 보면 엔트리의 개수는 11개에서 10개로, 필드의 개수는 두 개에서 한 개로 줄어들었음을 볼 수 있으며, 프리픽스의 개수가 많아질수록 줄어드는 엔트리의 개수가 많아져 최적화 구조의 성능이 더 좋아질 것으로 생각된다. Ⅳ장의 성능 평가를 통하여 이를 확인하였다.

Ⅳ. 영역분할 이진 검색 알고리즘의 최적화 구조의 성능 평가

본 논문에서는 실제 백 본 라우터에서 사용된 다양한 크기의 라우팅 데이터[9]를 가지고 성능을 실험하였다. IPv4에서 사용하는 데이터의 경우 프리픽스의 최대 길이가 32비트 이며, CIDR(Classless Inter-Domain Routing) 주소 체계로 인해 프리픽스의 길이가 8에서 32비트의 다양한 분포를 보인다. [8]의 경우, 끝점을 제거하고, 두 개의 필드를 하나로 통합하는 실험을 하였으며 생성된 라우팅 테이블의 엔트리 수와 필드의 수를 기존의 영역분할 이진 검색(BSR)과 비교하였다. 본 논문에서는 끝점을 제거하고 두 개의 필드를 통합한 라우팅 테이블에서 검색 속도와 메모리 요구량을 원래의 영역분할

표 6. 기존의 이진 검색 구조와 본 논문의 최적화 구조의 메모리 접근 횟수 비교

검색 구조	메모리 접근 횟수
B-trie[2]	O(W)
Multi-bit trie	O(W/k), k는 stride의 크기
P-trie	O(W)
BST[5]	O(log$N$)
WBST[6]	O(log$N$)
Waldvogel BSL[10]	O(log$W$)
Leaf-push BSL[11]	O(log$W$)
BSR[7]	O(log$N$)
OBSR	O(log$N$)

표 6 영역분할 이진 검색 알고리즘과 최적화 구조의 엔트리 수 비교

Prefix Set	# of Prefix	# of Entry	
		BSR[7]	OBSR
MAE-WEST-1	14553	29095	25706
Aads	20204	40389	34680
MAE-WEST-2	29584	57793	45816
MAE-EAST	39464	78872	65391
PORT80	112310	165520	115859
Grouptlcom	170601	248318	164799
Telstra	227223	322476	235499

이진 검색 알고리즘과 비교하였으며, 다양한 기존의 이진 검색 알고리즘들과 비교하였다. 룰의 개수를 N, 프리픽스의 최대 길이를 W라 하였을 때, 기존의 이진 검색 구조와 본 논문의 최적화 구조의 메모리 접근 횟수를 나타내면 표 6과 같다.

영역분할 이진 검색 알고리즘과 최적화 구조의 경우 모두 O(N)의 메모리 요구량과 O(log$N$)의 메모리 접근 횟수로 표현 할 수 있으나, 실제 성능은 점근적으로 표현되기 어려운 구조이므로 실제 라우터에서 사용하는 데이터를 사용하여 성능을 측정하였다. 메모리 접근은 IP 주소 검색과정 중 시간을 가장 많이 소요하는 작업으로서, 메모리 접근 회수는 성능비교의 척도로 사용되어 왔으므로, 본 논문에서도 각 알고리즘의 메모리 접근 회수를 사용하여 성능을 비교하였다. 라우팅 데이터를 사용하여 시뮬레이션을 한 결과 기존의 영역분할 이진 검색(BSR) 알고리즘과 최적화 구조(optimized binary-search-on-range, OBSR) 모두 바른 검색 결과를 보임을 확인하였으며, 두 구조의 엔트리 수를 비교한 결과를 표 7에 보였다.

표 7의 결과에서 볼 수 있듯이, 최적화 구조는 두 개의 필드를 하나로 합치는 과정에서, "엔트리와 같은 경우"에 저장된 값이 최대 길이의 프리픽스인 경우 새로운 엔트리가 생성됨에도 불구하고, 한 영역의 끝점과 다른 영역의 시작점이 연속된 경우 끝점(end point)을 제거함으로써, 엔트리 수에 있어 기존의 영역분할 이진 검색(BSR) 보다 최대 34% 감소한다는 것을 알 수 있으며, 라우팅 테이블의 크기가 클수록, 그 효과가 뛰어남을 알 수 있다. 특히 Grouptlcom의 경우, 원래 라우팅 테이블의 프리픽스의 개수보다도 적은 수의 엔트리를 갖는 라우팅 테이블의 구성이 가능함을 볼 수 있었으며, 이는 라우팅 테이블 자체가 불필요한 프리픽스를 많이 포함하고 있는 것으로 생각될 수 있다.

표 8에는 메모리 접근회수에 따르는 검색 성능 및 요구되는 메모리 사용량에 있어 최적화된 영역분할 이진 검색의 성능을 보였다. 표 8에서 보여주는 바와 같이, 프리픽스의 개수가 220,000이 넘는 라우팅 데이터에 대하여 최대 18번, 평균 17.4번의 메모리 접근으로 매우 빠른 IP주소 검색이 가능함을 알 수 있으며, 라우팅 데이터의 크기가 커져도 그 성능이 크게 나빠지지 않는 것을 볼 수 있다. 요구되는 메모리의 양에 있어서는 프리픽스당 5 byte에서 9 byte의 적은 메모리를 요구하며, 특히 라우팅 데이터의 크기가 커질수록 프리픽스당 요구되는 메모리의 양이 적어져, 확장성이 매우 우수함을 볼 수 있다. 표 9와 그림 4에서 이진 검색(binary search)을 이용하는 다른 알고리즘과 평균 메모리 접근 횟수(Tavg)에 대해 비교하였다.

표 8. 최적화된 BSR 구조의 성능실험 결과

	no. of prefix	no. of entry	T_{max}	T_{avg}	Mem(MB)	Mem/prefix (byte)
MAE-WEST-1	14553	25706	15	14.39	0.12	8.83
Aads	20204	34680	16	14.78	0.17	8.58
MAE-WEST-2	29584	45816	16	15.24	0.22	7.74
MAE-EAST	39464	65391	16	15.66	0.31	8.28
PORT80	112310	115859	17	16.53	0.55	5.16
Grouptlcom	170601	164799	18	17.08	0.79	4.83
Telstra	227223	235499	18	17.44	1.12	5.18

표 9. 기존의 이진 검색 알고리즘과의 평균 메모리 접근 회수 비교

	no. of prefix	B - Trie[2]	BST [5]	BSR [7]	OBSR
MAE-WEST-1	14553	22.9	14.1	14.21	14.39
MAE-WEST-2	29584	23.1	15.6	15.20	15.24
PORT80	112310	22.2	26.0	16.75	16.53
Grouptlcom	170601	22.3	27.0	17.28	17.08
Telstra	227223	24.6	30.8	17.64	17.44

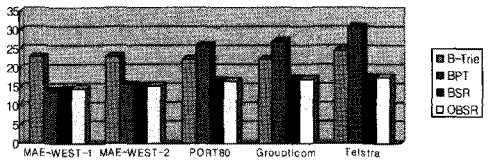


그림 4. 평균 메모리 접근 회수 비교

영역분할 이진 검색 알고리즘은 기존의 다른 이진 검색 알고리즘에 비하여 월등히 좋은 검색 성능을 보이며, 최적화된 구조는 영역분할 이진 검색 알고리즘의 평균 검색 성능을 조금 더 향상 시킨 것을 볼 수 있다. 특별히 영역분할 이진 검색 알고리즘 및 최적화 구조의 성능은 라우팅 테이블의 크기가 커져도 그 성능이 크게 나빠지지 않으므로, 확장성이 매우 뛰어난 구조임을 알 수 있다. 최적화된 영역분할 이진 검색 알고리즘과 원래의 영역분할 이진 검색[7] 알고리즘의 평균 검색 성능을 비교할 때, 라우팅 테이블의 크기가 작은 경우, 최적화 구조의 엔트리 개수가 원래의 영역분할 이진 검색의 엔트리 개수보다 적음에도 불구하고, 최적화 구조의 평균 검색 성능이 원래의 영역분할 이진 검색 알고리즘의 검색성능보다 조금 나쁜 것을 볼 수 있다. 이는 기존의 이진 검색 알고리즘에서 특정 엔트리와 정확하게 일치하여 검색을 즉시 종료시킬 수 있는 인풋들에 대하여 최적화 방법으로 엔트리 개수를 줄였을 때, 정확하게 일치하는 경우에 해당되지 못하고 검색을 끝까지 진행하여야 하는 경우가 발생하기 때문인 것으로 판단된다.

표 10과 그림 5에서 프리픽스 당 메모리 요구량을 타 구조와 비교하여 보았다. 영역분할 이진 검색 알고리즘의 최적화 구조는 라우팅 데이터의 크기가 커질수록 불필요한 엔트리를 삭제하고 "엔트리보다 큰 경우" 필드와 "엔트리와 같은 경우" 필드를 "엔트리보다 크거나 같은 경우" 필드로 합쳐 메모리 요구량을 감소시킴으로써, 타 구조와 비교하였을 때

표 10. 기존의 이진 검색 알고리즘과의 프리픽스당 메모리 요구량 비교 (byte)

	no. of prefix	B - Trie[2]	BST [5]	BSR [7]	OBSR
MAE-WEST-1	14553	31	10	12.00	8.83
MAE-WEST-2	29584	22	10	11.72	7.74
PORT80	112310	12	10	8.84	5.16
Grouptlcom	170601	11	10	8.73	4.83
Telstra	227223	12	10	8.52	5.18

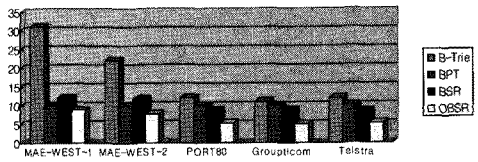


그림 5. 프리픽스 당 메모리 요구량 비교(byte)

메모리 요구량에서 매우 효율적인 구조라는 것을 알 수 있다. 또한 기존의 영역분할 이진검색 구조와 비슷한 메모리 요구량을 보이는 이진 검색 트리 구조 보다 최적화된 구조의 메모리 요구량이 현저히 적다. 본 논문에서 연구한 최적화 구조는 영역분할 이진 검색의 메모리 요구량에 관한 성능을 최대 45%까지 향상시킨 것을 볼 수 있다.

V. 결 론

패킷이 입력되는 속도와 같은 속도(link-speed)로 포워딩되도록 하는 것이 라우터의 궁극적인 목적이며, 이를 위해서는 고속의 IP 주소 검색을 통하여 입력되는 패킷을 패킷의 목적지로 향하는 포트로 포워딩하여야 한다. 영역분할 이진 검색 알고리즘은 IP 주소 검색을 위하여 연구되어온 이진 검색 알고리즘 중 유일하게 균형이진 검색을 제공하는 알고리즘으로서 검색 성능에 있어서나, 요구되는 메모리의 양, 그리고 확장성에 있어 매우 뛰어난 구조이다. 하지만 불필요한 엔트리가 라우팅 테이블에 존재하여 필요 이상의 메모리가 요구된다는 단점이 있다. 따라서 본 논문에서는 영역분할 이진 검색(BSR) 알고리즘의 성능을 최적화하는 방안을 연구하였다. 엔트리 개수의 최적화를 위하여, 연속된 엔트리의 끝 점을 제거하는 방법으로 엔트리의 개수를 최대 34%까지 감소시킬 수 있음을 보였으며, 두 개의 선 계산 필드를 하나로 합치는 방법으로 요구하는 메모리의 양을 최대 45%까지 줄일 수 있음을

보였다. 또한 큰 프리픽스 셋으로 갈수록 패킷을 포워딩하는 속도가 상대적으로 좋아져 확장성이 우수한 구조임을 보았다. 기존에 연구되어온 다른 이진 검색 알고리즘과의 성능 비교를 통하여 영역분할 이진 검색 알고리즘의 최적화 구조가 검색 속도나 메모리 사용량에 있어 월등히 우수함을 보였다.

참 고 문 헌

[1] M.A. Ruiz-Sanchex, E.W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", IEEE Network, pp.8-23, March/April 2001.

[2] H. J. Chao, "Next generation routers," Proceedings of the IEEE, Vol.90, No.9, pp.1518-1558, Sep. 2002.

[3] S. Nilsson and G. Karlsson, "IP-Address Lookup using LC-tries," IEEE Journal on Selected Area in Communication, Vol.17, pp.1083-1092, June 1999.

[4] HyeSook Lim, JuHyoung Mun, "An Efficient IP Address Lookup Algorithm Using a Priority Trie", GLOBECOM 2006.

[5] N. Yazdani and P. S. Min, "Fast and scalable schemes for the IP address lookup problem," Proc. IEEE HPSR2000, pp. 83-92, 2000.

[6] Changhoon Yim, Bomi Lee, and Hyesook Lim, "Efficient Binary Search for IP Address Lookup", IEEE Communications Letters, Vol. 9, No. 7, pp.652-654, Jul. 2005.

[7] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," IEEE/ACM Trans. Networking, Vol.7, No.3, pp.324-334, Jun. 1999.

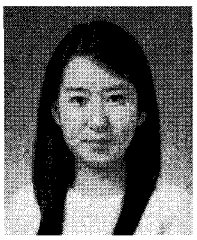
[8] Hyun-Sic Kim, Hyuntae Park, Daein Kang and Sungho Kang, "A New Efficient Binary Search on Range for IP Address Lookup", 2006 International SoC Design Conference, pp.39-42, 2006.

[9] http://www.potaroo.net

[10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in Proc. ACM SIGCOMM Conf., Cannes, France, pp.25-35, 1997.

[11] J. Mun, H. Lim, and C. Yim, "Binary Search on Prefix Lengths for IP Address Lookup,"IEEE Communications Letters., Vol.10, No.6, pp.492-494, June. 2006.

박 경 혜 (Kyong Hye Park) 준회원



2007년 2월 이화여자대학교 정보통신학과 학사

2007년 3월~현재 이화여자대학교 전자정보통신학과 석사과정

<관심분야> Router나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계

임 혜 숙 (HyeSook Lim) 종신회원

1986년 2월 서울대학교 제어계측공학과 학사

1986년 8월~1989년 2월 삼성휴렛팩커드 연구원

1991년 2월 서울대학교 제어계측공학과 석사

1996년 12월 The University of Texas at Austin, Electrical and Computer Engi-neering, Ph.D.

1996년 11월~2000년 7월 Lucent Technologies, Bell Labs, Member of Technical Staff

2000년 7월~2002년 2월 Cisco Systems, Hardware Engineer

2002년 3월~현재 이화여자대학교 공과대학 전자공학과 부교수

<관심분야> Rourer나 switch 등의 Network 관련 SoC 설계, TCP/IP 관련 하드웨어 설계