O(1) IP 검색 방법

이 주 민[0] 안 종 석
동국 대학교 컴퓨터 공학과
{emigrant[0], jahn}@dongguk.edu

O(1) IP Lookup Scheme

Ju-Min Lee[0] Jong-Suk Ahn
Dept. of Computer Engineering, Dongguk University

요약

백본 라우터에서의 최장 길이 프리픽스 검색(LPM: Longest Prefix Matching) 속도를 향상시키기 위해 활발히 연구된 방식들은 계산 량과 사용 메모리 량을 교환하는 방식들이다. 이러한 방식들은 성능향상을 위해서 대용량의 포워딩 테이블(Forwarding Table)을 캐쉬(Cache)에 저장할 수 있는 소용량 인덱스 테이블(Index Table)로 압축함으로써 고속 캐쉬 접근 회수와 그 계산량은 증가하는 대신 저속 메모리 접근 회수를 줄이는 방식이다.[1] 본 논문에서는 저속 메모리 사용량이 증가하는 반면 저속 메모리의 접근 빈도와 계산량을 동시에 감소시키는 FPLL(Fixed Prefix Length Lookup) 방식을 소개한다. 이 방식은 포워딩 엔트리(Entry)들을 프리픽스의 상위 비트(Bit)에 의해 그룹으로 나누고, 각 그룹에 속하는 엔트리들을 같은 길이로 정렬한다. FPLL에서의 LPM검색은 목적지 주소가 속하는 그룹들의 길이를 계산하여 검색할 최장 프리픽스의 길이를 미리 결정하고, 결정된 프리픽스를 키(key)로 하여 해시 테이블(Hash Table)로 구성된 포워딩 테이블에서 완전 일치(Exact Matching) 검색을 한다. 완전 일치 검색을 위해 같은 그룹에 속한 엔트리들을 정렬할 필요가 있는데 이 정렬을 위해 여분의 포워딩 테이블 엔트리가 생성된다. 3만개 엔트리를 갖는 Mae-West[2] 경우에, FPLL방식은 12만개 정도의 여분의 엔트리가 추가로 생성되는 대신에 1번 캐쉬 접근과 O(1)의 복잡도를 갖는 해시 테이블 검색으로 LPM 검색을 수행한다.

1. 서론

최근에 인터넷의 전송 매체 속도가 증가함에 따라 패킷(Packet)을 고속으로 스위칭(Switching) 하는 라우터의 개발이 활발히 진행되고 있다. 고속 라우터의 개발을 위해서는 라우터 성능 향상에 걸림돌이라 할 수 있는 최장 길이 검색으로 명명되어 있는 목적지 주소 검색 문제를 해결하는 것이 중요하다. 따라서 이 문제를 해결하기 위한 연구가 많이 진행되고 있다[3][4][5][6]. LPM 검색이란 대용량의 포워딩 테이블에서 주어진 IP주소 값과 일치하는 엔트리 중 최장의 프리픽스를 가지는 엔트리를 검색하는 것이다.

본 논문에서 제안하는 FPLL 방식은 LPM 문제를 해결하기 위해 검색할 프리픽스를 그룹별로 같은 길이로 미리 정렬시키는 방식이다. 정렬된 각 그룹별 프리픽스 길이를 PLT(Prefix Length Table)라 명명된 배열에 저장한다. 검색 시에는 PLT를 이용하여 먼저 프리픽스 길이를 결정하고 정해진 길이 만큼의 상위 프리픽스를 이용하여 해시 테이블로 구성된 포워딩 테이블에서 O(1)의 속도로 출력 링크(Out Link)를 검색하게 된다. 본 논문에서 제안한 방식은 메모리의 사용량이 증가하는 반면 상대적으로 속도가 느린 메모리의 접근을 최소화하여 검색 속도를 향상시키는 방법이다. 즉 프리픽스 정보가 저장된 PLT를 속도가 빠른 캐시에 저장[7]하고 크기가 큰 포워딩 테이블을 해시 테이블 형태로 메모리에 저장하여 느린 메모리 접근을 최소화시킨다.[8][9]

본 논문에서 제안한 방식을 사용한 결과 3만개의 엔트리를 갖고 있는 Mae-West 포워딩 테이블의 경우 여분의 엔트리 12만개 정도가 추가로 생성되고, 이것을 해시 테이블로 구성하게 되면 약 1MB의 여분의 메모리가 필요하다. 이러한 메모리의 증가는 현재의 메모리 가격을 고려한다면 구현에 중대한 문제라고 생각되지 않는다. 이 정도의 메모리 사용량이 증가하는 것에 비해 FPLL은 하나의 LPM 검색을 위해 캐쉬에 저장된 PLT를 1회의 접근으로 검색할 프리픽스 길이를 결정하고, 결정된 길이만큼의 상위 프리픽스로 해시 테이블에서 O(1) 시간 복잡도로 검색하게 된다. 캐쉬 읽기 속도 10ns와 메인 메모리의 접근 속도 50ns를 생각한다면 FPL방식은 기가 비트급 라우터를 구성할 수 있음을 알 수 있다.

2장에서는 본 논문에서 제안한 FPLL의 구조 및 검색 알고리즘에 대해서 자세히 설명하고 3장에서는 알고리즘에 대한 실험결과 및 메모리 사용관계를 설명한다. 마지막으로 4장에서는 결론과 향후 연구 문제들을 나열한다.

2. FPLL 구조 및 검색 방법.

2.1 FPLL을 위한 라우팅 엔트리 확장법과 PLT 생성 방법

본 논문에서 제안한 FPLL 방식은 앞에서 언급하였듯이 상위 n비트로 그룹을 짓고, 그룹별로 프리픽스 길이를 같게 하여 검색 시 그 길이를 미리 정할 수 있게 하는 방법이다. 따라서, 같은 그룹에 속한 엔트리들을 같은 길이로 맞추어 주는 확장이 필요하다. 여기서 확장은 동일 그룹에 대해서 짧은 프리픽스를 갖는 엔트리를 긴 프리픽스 갖는 엔트리의 길이로 맞추어 주면서 라우팅 정보는 손상시키지 않는 것을 의미한다. 결과적으로 본 논문에서 제안한 FPLL은 두 가지의 테이블로 구성되는데 각 그룹별 프리픽스 길이를 저장하는 PLT와 해시 테이블 형태로 저장된 포워딩 테이블이 그것이다. 상위 n비트에 의해 그룹화를 할 경우 2^n개의 그룹이 생성되며 그룹별 프리픽스 길이를 저장한 것이 PLT가 된다.

FPLL 방식을 쉽게 설명하기 위해 IP 주소를 6비트라고 가정하고 이 IP 주소에 대해서 상위 3비트를 이용하여 그룹화하는 경우를 예로 든다. 3비트로 그룹을 생성하게 되면 000 ~ 111의 8개 그룹이 생성되고, 8개의 그룹에 대해서 각 그룹에 해당하는 엔트리들의 프리픽스를 가장 긴 프리픽스 길이를 가지는 엔트리의 길이로 맞추어 주는 확장을 하게된다. 그림 1의 (a)가 확장 이전인 처음 라우팅 엔트리를 나타내고 그림1의 (b)가 확장후의 라우팅 엔트리들을 보여준다. 여기에 나타난 엔트리들을 기준으로 설명을 하면 01*의 경우 010*와 011* 이렇게 두 그룹에 속하게 되고, 010*는 다시 010 1*와 같은 그룹에 속

하게되므로 010 1*와 길이를 맞추어 주어야한다. 따라서 01* 엔트리에 대해서 먼저 010 */3과 011 */3으로 만들어 주는 확장이 필요하고, 010 */3을 다시 010 0*/4로 확장시키는 과정이 필요하게된다. 최종적으로 01* 가 010 0*/4와 011*/3 두 개의 엔트리로 확장이 된다. 이에 반해 11*의 경우는 110과 111의 두 그룹에 속하게 되나 11*이 동일 그룹에서 최장 길이를 가지게 된다. 따라서 포워딩 테이블에서의 엔트리 확장 즉, 110*/3과 111*/3의 엔트리를 생성하는 확장은 불필요하고 단지 PLT 구성 시 110과 111 그룹에 대해서 프리픽스 길이를 2로 저장하면 된다. 이러한 방법을 이용하여 라우팅 엔트리 확장이 끝나면 확장된 포워딩 테이블을 기준으로 PLT를 구성하게 된다. 이렇게 생성된 PLT를 그림 2의 (a)에서 보여주고 있다. 또한 확장이 끝난 후의 포워딩 테이블은 해시 테이블에 저장하게 되는데 그림 2 (b)가 각 프리픽스에 대한 해시 키 값을 보여주고 있다. 즉 상위 비트에 나머지 비트는 0으로 채운 것이 해시 키 값이 되는 것이다.

prefix	out link
01*	e0
010 1*	e1
100 01*	s0
11*	s1

(a) 확장 이전의 엔트리

prefix	out link
010 0*	e0
011 *	e0
010 1*	e1
100 01*	s0
11*	s1

(b) 확장 후의 엔트리

그림 1. 확장 이전의 엔트리와 확장후의 엔트리

2.2 목적지 주소에 대한 출력링크 검색 방법

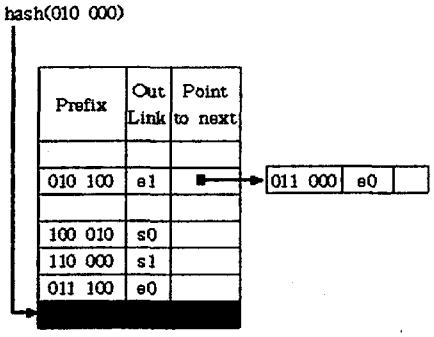
이번 절에서는 본 논문에서 제안한 방식에 대해 IP 검색이 이루어지는 방법을 설명한다. 검색 시에는 먼저 상위 3비트를 가지고 PLT에서 포워딩 테이블에 사용될 프리픽스 길이를 정하게 된다. 그 후 정해진 길이 만큼의 상위 비트를 이용하여 포워딩 테이블에서 O(1)의 속도로 출력 링크를 검색하게 된다. 그림 2에 나타난 PLT와 포워딩 테이블을 이용하여 목적지 주소가 검색되는 과정을 설명하도록 한다. "010 010"의 목적지 주소를 가지는 패킷의 경우를 예로 들면, 상위 3비트인 "010"을 이용하여 그림 2. (a)의 PLT에서 검색을 하고 그 결과로 포워딩 테이블 검색에 사용될 프리픽스 길이가 4로 정해지게된다. 다음은 정해진 길이 만큼의 프리픽스의 상위 비트를 이용하여 해시 테이블에서 검색을 하는 것이다. 즉, 정해진 길이가 4이므로 프리픽스의 상위 4비트 부분은 "010 0"이 되고 해시 테이블 검색에서는 "010 000"을 키 값으로 사용하여 포워딩 테이블을 검색하게된다. 그 결과로 출력 링크 e0을 검색하게 되는데 그림 2의 (c)가 포워딩 테이블 및 포워딩 테이블에서 검색된 엔트리를 보여준다. 검색에 사용되는 키 값은 PLT에서 정해진 만큼의 상위 프리픽스를 사용하되 전체 IP 주소 길이만큼의 비트를 사용하고 나머지 하위비트를 0으로 채워 사용하게 된다.

group	prefix 길이
000	0
001	0
011	3
100	5
101	0
110	2
111	2

(a) 각 그룹의 프리픽스 길이가 저장된 PLT의 모습

prefix	hash key
010 0*	010 000
011 *	011 000
010 1*	010 100
100 01*	100 010
11*	110 000

(b) 확장이 끝난 엔트리와 그에 해당하는 해시 키 값.



(c) 해시 테이블로 구성된 포워딩 테이블

그림 2. 상위비트에 의한 그룹화 방식에 대한 PLT와 포워딩 테이블 구성.

3. 실험

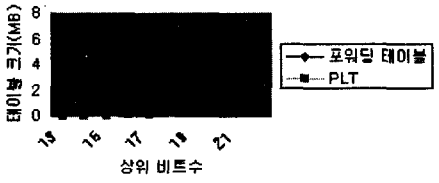
이번 장에서는 실제 백본용 라우팅 테이블을 대상으로 본 논문에서 제안한 알고리즘을 적용하여 실험한 결과에 대해서 설명한다. PLT의 크기는 상위 비트를 몇 비트로 하느냐에 따라 쉽게 계산이 되는 반면, 포워딩 테이블은 라우팅 엔트리 특성에 따라 그 확장되는 정도가 달라지게 되고 이에 따라 포워딩 테이블에 사용되는 메모리의 양이 결정되게 된다. 따라서 본 실험에서는 MAE-West 및 MAE-East 두 가지 백본 라우팅 테이블에 대해서 실험을 하였으며, 각 라우팅 테이블별로 그룹화를 위해 사용되는 상위 비트수를 달리 하면서 확장 시 생성되는 라우팅 엔트리 개수를 측정해 보았다. MAE-West는 2002년 4월 데이터이며 총 27,609개의 라우팅 엔트리로 구성되어 있고 MAE-East는 1999년의 데이터이고 총 54,249개의 라우팅 엔트리로 구성되어 있다.[9]

그림 3의 (a)와 (b)에 MAE-West와 MAE-East 라우팅 테이블에 대한 실험결과를 각각 도시하였다. 포워딩 테이블의 크기는 생성된 라우팅 엔트리들에 대해서 각 9바이트가 사용될 때를 기준으로 계산하였다(프리픽스를 표시하기 위해 4바이트, 아웃 링크 정보 저장용으로 1바이트, 해시 테이블에서 같은 버킷에 대해서 2개 이상의 엔트리가 들어갈 수 있으므로 포인터로 4바이트). PLT는 각 그룹별 프리픽스 길이만 저장하면 되므로 한 그룹에 대해 한 바이트면 충분하다. 따라서 n비트를 상위 비트로 사용할 경우 PLT를 위해 2^n 바이트가 필요하게 된다. 실험결과로 상위 13~22비트에 대해서만 도시한 이유는 13비트보다 적은 비트를 사용할 경우 라우팅 테이블 크기는 더 늘어나며 19비트 이상을 사용할 경우 PLT가 캐쉬에 저장될 수 없으므로 이들에 대한 실험 결과는 무의미하다고 할 수 있다. 실제로는 이들에 대해서도 실험을 하였으나 그 의미가 없다고 사려되어 실험결과에서 제외하였다.

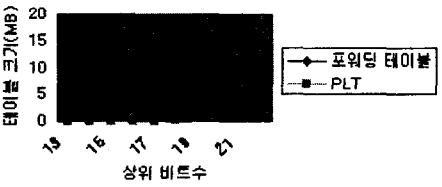
그림 3에 나타난 실험 결과에서 볼 수 있듯이 상위 비트를 많이 사용하면 할수록 포워딩 테이블 크기는 작아지는 반면 PLT의 크기가 커짐을 알 수 있다. 상위 비트를 많이 사용할수록 포워딩 엔트리가 줄어드는 이유는 상위 비트를 많이 사용함

에 따라 그만큼 많은 그룹으로 나뉘게 되고 프리픽스 길이가 다른 엔트리들이 동일 그룹에 속하게되는 경우가 줄어들기 때문이다. 즉, 사용하는 상위 비트수가 32비트에 가까워질수록 추가로 생성되는 라우팅 엔트리는 0에 가까워지는 반면 PLT가 커지게 된다. 본 논문에서는 PLT를 캐쉬에 저장하여 검색 속도를 높이는 방법을 선택하였고 현재 펜티엄4의 경우 512KB의 캐쉬를 사용하므로 캐쉬에 PLT를 저장하기 위해서는 상위 비트를 19비트 이하로 사용하여야한다. 상위 비트를 18비트로 한다면 PLT를 위해서 256KB가 사용되고 포워딩 테이블을 저장하기 위해서 MAE-West, MAE-East 각각 1.3MB, 2.6MB가 필요하다. 이 외의 경우에도 사용되는 메모리가 수MB 또는 수십 MB 이내로 이는 최근 메모리 가격을 고려했을 때 라우터 구성에 큰 문제가 되지 않는다고 사려된다.

검색에 소요되는 시간을 계산해 보면 PLT 검색을 위해 1번의 캐쉬 접근이 필요하고 포워딩 테이블이 해시 테이블로 구성되어 있으므로 포워딩 테이블 검색에 O(1)의 시간이 필요하게 된다. 포워딩 검색에 해시 함수를 이용하므로 사용되는 해시 함수에 따라 총 검색 시간이 달라질 수 있다. 하지만 해시 버킷 당 라우팅 엔트리가 평균 2개정도 들어가는 해시 함수는 쉽게 만들 수 있다고 사려된다. 예를 들어 큰 소수를 이용하여 모듈러 연산을 사용하여 해시 테이블 구성하는 것을 하나의 방법으로 생각할 수 있다. 현재 사용되는 캐시와 메모리의 속도가 각각 10ns와 50ns임을 생각하고 버킷 검색에 평균 2회의 메모리 접근이 필요하다고 하더라도 한 패킷을 검색하는데 소요되는 시간이 110ns 정도임을 쉽게 계산할 수 있다. 물론 위의 시간에 CPU의 처리 시간이 추가적으로 필요하겠지만 CPU처리 속도가 메모리 속도에 비해 월등히 빠르므로 라우팅 속도 계산에 큰 영향을 미치지 않으리라 판단된다. 일단 본 논문에서는 적절한 해시 함수를 찾는 것과 실제 검색속도 실험에 대해서는 향후 과제로 남겨둔다.



(a) MAE-West 실험 결과



(b) MAE-East 실험 결과

그림 3. 확장후의 포워딩 테이블과 PLT의 크기

다음으로 메모리 사용량과 검색 속도의 관계에 대해서 설명하기로 한다. IP 검색을 가장 빨리 할 수 있는 방법은 가능한 모든 IP주소에 대해서 배열을 만들어 아웃 링크를 저장하는 것이다. 즉, 2^{32}개의 배열을 사용하여 출력 링크를 저장한다면 한 번의 메모리 접근으로 검색을 완료 할 수 있다. 이 경우는 PLT가 따로 필요하지 않고 포워딩 테이블만 하나가 필요하며 목적지 주소가 바로 해시 테이블 검색을 위한 키가 된다. 하지만 이 경우 출력 링크 저장용으로 한 바이트만을 사용한다 할지라도 2^{32}바이트 즉 4기가바이트가 필요하게 되므로 라우터 구성에는 적합하지 않다고 할 수 있다. 더욱이 이러한 방식을 IPv6에 적용하기에는 더욱 부적합하다고 할 수 있다.

4. 결론 및 향후 연구

본 논문에서는 사용하는 메모리의 크기와 검색 속도를 효율적으로 교환할 수 있는 FPLL알고리즘을 소개하였다. 실험에 의하면 18비트를 상위비트로 하여 그룹화 할 경우 256KB의 캐쉬가 필요하며, MAE-West와 MAE-East에 대해서 각각 1.3MB와 2.6MB의 메모리가 필요하다. 하지만 이는 최근 메모리에 가격을 고려하였을 때 라우터 구성에 큰 문제가 되지 않는다고 사려되며 이에 반해 한번의 캐쉬 접근과 O(1)의 메모리 접근으로 검색을 완료할 수 있다.

검색 속도는 상대적으로 느린 메모리에 저장된 해시 테이블 검색에 얼마만큼의 시간이 소요되느냐에 따라 결정된다고 볼 수 있다. 따라서 해시 함수에 따라 그 결과가 달라질 수 있으며, 적절한 해시 함수를 찾는 것은 향후 과제로 남겨두었다. 향후에는 본 논문에서 제안한 방법을 사용하여 MAE-West와 MAE-East 라우팅 테이블에 대해서 실제 라우팅 검색 속도에 대한 실험을 해 볼 예정이다.

또 다른 향후 과제로는 IPv6에 적용할 수 있는 FPLL을 개발하는 것이다. IPv6의 경우 네트워크 주소로 64비트를 사용하고 있다[10]. 본 논문에서 제안한 상위 1단계에 의한 그룹화만을 사용할 경우 상위비트가 19비트 이하가 되게 되고 나머지 비트는 45비트 이상이 된다. 따라서 확장되는 라우팅 엔트리 수는 적어도 2^{45}정도가 될 수 있으며, 이를 위해 사용되는 메모리 양을 계산한다면 라우터 구성에 적합하지 않음을 알 수 있다. 따라서 이를 보완하기 위한 다단계 그룹화 방식을 고안중에 있으며, 적절한 다단계 그룹화를 찾아내는 것이 다음 목표이다.

참고 자료

[1] Donald R. Morrison, "PATRICIA - Practical Algorithm to Retrieve Information Coded In Alfanumeric," *journal of the ACM*, 15(4):514-534, October 1968

[2] Michigan University and merit Network, Internet Performance Management and Analysis (IPMA) project, http://www.merit.edu/ipma

[3] B. Lampson, IP Lookups using Multiway and Multicolumn Search. IEEE\slash ACM Transactions on Networking 1997

[4] Mikael Degermark, Small Forwarding Tables for Fast Routing Lookups. Proc. ACM SIGCOMM Conference pages 3-14, October 1997

[5] Seunghyun Oh, Bit map Trie : A Data Structure for Fast Forwarding Lookups, Globecom2001 part2, Globecom, Nov, 2001

[6] S. Venkatachary and G. Varghese, "Faster IP Lookups using Controlled Prefix Expansion," *In Proceedings of ACM Sigmetrics'98*, June 1998

[7] David C. Feldmeier, "Improving gateway performance with a routing-table cache," *In proceedings of IEEE Infocom'98*, New Orleans, Louisiana, March 1998

[8] R. Jain, A Comparison of Hashing Schemes for Address Lookup in Computer Networks, IEEE Transactions on Communications, vol. 40, no. 10, pp. 1570-1573, 1992.

[9] Andrei Broder, Using multiple Hash Functions to Improve IP Lookups. IEEE INFOCOM. pp. 1454-1463. 2001.

[10] IPv6 포럼 코리아, http://www.ipv6.or.kr/