비트-맵 트라이를 이용한 빠른 라우팅 검색

오승현° 나승구 안종석
동국대학교 컴퓨터공학과
{shoh, sgna, jahn}@dgu.ac.kr

Bit-Map Trie for Fast Routing Lookups

Seung-Hyun Oh° Seung-Gu Na Jong-Suk Ahn
Dept. of Computer Engineering, Dongguk University

요 약

기가비트 속도를 지원하는 고속 라우터의 IP 주소 검색은 소프트웨어로 구현할 수 없다는 일부의 믿음과는 달리 소프트웨어만으로도 고속 IP 주소 검색의 구현이 가능하다. 기가비트 라우터의 IP 주소 검색은 최장프리픽스일치 기법을 사용하여 라우팅 엔트리를 검색하는데, 5Gbps 속도를 지원하기위해서는 평균 512byte의 패킷을 800 nsec 이하의 속도로 처리하여야 한다.

본 논문에서는 범용 펜티엄 프로세서의 캐쉬 크기에 적합한 고속 라우팅을 위한 포워딩 테이블 구조를 제안하였으며, 400 MHz의 펜티엄 II 프로세서를 이용한 실험에서 초당 수백만개의 IP 주소 검색을 실현하였다. 제안된 포워딩 테이블은 약 48,000여개의 실제 라우팅 엔트리에 대해 284Kbyte의 매우 작은 크기로 작성되었는데, 이 크기는 펜티엄 프로세서의 L2 캐쉬에 저장될 수 있는 작은 크기이다. 제안된 포워딩 테이블을 이용한 평균 검색 시간은 라우팅 테이블 별로 320~530 nsec가 소요되었다.

1. 서론

인터넷이 예상보다 빠르게 보급 되고, WWW 중심으로 인터넷이 사용됨으로써 구내에서 처리될 수 있는 트래픽 보다 광대역(Wide-area) 백본을 경유하는 트래픽의 비율이 압도적인 상태가 되었다. 이러한 트래픽 분포로 대량의 백본 트래픽을 고속으로 처리하기 위한 초고속 네트워크 수요가 확산되고, 기가비트(Gigabit)급 고속 라우터에 대한 연구가 빠르게 진행되고 있다. 기가비트급 고속 라우터는 초당 백만개 이상의 패킷을 처리할 수 있는 속도를 지원하여야 하는데, 일반적으로 이러한 속도의 라우터는 하드웨어를 기반으로 구현되어야하며, 소프트웨어로 구현하는 것은 불가능한 것으로 여겨져왔다[1].

IP 네트워크에서 라우터는 도착한 패킷의 목적지 주소를 이용하여 다음 홉 주소와 출력 링크를 결정하고, IP 헤더(Header)의 몇가지 정보를 처리한후 출력 링크로 보내는 작업을 시행한다. IP 헤더의 정보 처리는 TTL 값을 차감하는 것과 같은 몇가지로 라우터의 패킷 처리 성능에 별다른 영향을 미치지 않는다. 또 수신된 패킷을 출력 링크로 이동시키는 일은 고속 하드웨어의 도움으로 빠르게 처리할 수 있다. 마지막으로 IP 라우터는 라우팅 테이블에서 라우팅 검색(Routing Lookup), 즉 IP 주소 검색(IP Address Lookup)을 실시하여 IP 데이터그램(Datagram)을 포워딩할 곳을 결정한다. 라우팅 검색은 최장프리픽스일치(Longest Prefix Matching) 개념으로 목적지 주소와 일치하는 엔트리(Entry)를 검색하여야 한다.

본 연구는 '99 대학기초연구사업의 지원을 받았습니다.

검색의 결과는 목적지로 향하는 다음 홉(Next-Hop)이며, 라우팅 테이블은 개념적으로 다음 홉과 연관된 임의의 길이의 프리픽스(Prefix)로 구성된다. 결국 라우터의 IP 패킷 처리에서 IP 주소 검색은 가장 많은 처리시간을 소요하는 결정적인 요소이다.

그러나 최근의 몇가지 주목할 만한 연구[6,7,8,9]에서 고가의 하드웨어 지원 없이 소프트웨어만으로 기가비트 라우터를 구현할 수 있는 포워딩(Forwarding) 테이블 구조를 제안하고 있다. 본 연구에서는 기존의 연구에서 제안하고 있는 지나치게 복잡한 자료구조 대신 간단한 자료구조를 사용하여 소프트웨어만으로 빠른 IP 주소 검색(Address Lookup)을 가능하게하는 포워딩 테이블을 제안한다. 실험 결과에 의하면 펜티엄 II 400 Mhz에서 초당 수백만개의 IP 주소 검색 성능을 보였다.

본 논문의 나머지 부분은 다음과 같이 구성되어있다. 먼저 2장과 3장에서 관련연구 및 라우팅과 포워딩의 차이를 기술한다. 3, 4장에서는 포워딩 테이블의 자료구조에 대해 설명하고, 실험결과에 대해 논의한다.

2. 관련연구

IP 주소 검색이 라우터의 병목(Bottleneck)으로 작용하여 고속 라우터의 개발이 어렵게되자 IP 주소 검색을 회피하는 기술이 소개되었다. 즉, IP 계층의 하부 계층인 링크 계층에서 스위칭을 하거나, IP 계층을 우회[2,3,4]하고, 또는 ATM과 같은 새로운 네트워크 계층을 사용하는 연구가 진행되었다.

전통적인 라우팅 테이블 구현은 최장프리픽스일치를 고려하여 수정된 패트리샤 트리(Patricia tree)[5]를 이용하는 것이다. 각각의 비트를 검색하는 NetBSD가 대표적인 구현 예이며, 40,000개의 엔트리를 갖는 라우팅 테이블이 약 2MB의 크기를 갖는다. 최근에는 패트리샤 트리를 개량하여 검색 속도를 증진시키는 방법이 많이 연구되고 있다.

고속 라우터를 구현하기 위한 최근의 기법은 크게 세가지로 구분할 수 있다. 첫째, 소프트웨어를 기반으로 하는 방법은 더 효율적인 데이터 구조와 검색 기법을 적용하는 방법으로, 정확한 일치 기법(Exact matching scheme)[6]을 적용하기 위한 연구와 트라이(trie)[7] 구조를 변경하는 연구의 두 가지로 분류할 수 있다. 둘째, 하드웨어를 사용하는 기법은 CAM (Content-Addressable Memory)[8], 병렬처리 및 캐쉬를 적용하는 연구로 분류할 수 있다. 마지막으로 패킷 헤더에 라벨이나 추가적인 정보[9]를 기록하는 등의 프로토콜을 변경하여 빠른 포워딩을 구현하는 기법이 있다. 본 연구는 소프트웨어를 기반으로 트라이 구조를 변경하는 연구로 분류할 수 있다.

3. 비트-맵 자료구조

32 비트 길이를 갖는 IP 주소 검색을 위한 포워딩 테이블은 모든 엔트리의 프리픽스(Prefix)를 확장하면 그림 1.(a)와 같이 깊이(Depth) 32의 이진 트리로 표현할 수 있다. 이때 트리의 넓이 즉, IP 주소의 가지수는 최대 2^{32}개 이다. 하나의 프리픽스 비트 값은 트리에서 하나의 노드와 대응되고, 트리 탐색은 프리픽스 검색으로 대응된다.

포워딩 테이블의 프리픽스 엔트리가 모든 비트 조합을 포함하는 것은 아니며, 모두 32 비트 길이를 갖는 것도 아니다. 그림 1.(b)에서 점으로 표현되는 부분들이 프리픽스가 존재하는 곳이 된다. 프리픽스 점들은 루트노드로부터 비트 값(1 or 0)에 따라 확장된 결과 이며, 특정 프리픽스는 완전 이진 트리로 표현할 수 있는 범위 r 만큼의 IP 주소에 대해 대표성 즉, 최장프리픽스일치 개념에 의해 다음 홉을 갖는 엔트리로 선택된다. 또한 그림 1.(b)에서 보는 바와 같이 특정 프리픽스의 범위에 포함되는 더 긴 길이의 프리픽스는 자신의 범위만큼 상위 프리픽스를 대체하게된다.

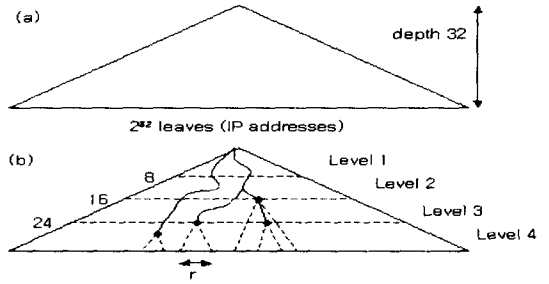

그림 1 프리픽스 이진 트리 표현 (a) 이진 트리 (b) 4 단계의 프리픽스 트리 구조

프리픽스를 트리로 표현하기 위해서는 그림 2와 같이 트리를 확장하여야 한다. 즉, 모든 노드는 2개의 자식 노드를 갖거나 자식 노드가 없어야 하는 트라이가 되어야 한다. 이렇게 표현된 프리픽스 트라이는 결국 깊이 32의 리프(Leaves) 노드의 관점에서 보면 모든 표현 가능한 프리픽스 비트를 순서대로 배열한 것과 같으며, 프리픽스의 길이가 n일 때에는 $r=2^{(32-n)}$의 범위를 갖는 것으로 볼 수 있다. 그림 2에서 검게 표시된 노드는 프리픽스가 있는 노드이고, 흰 노드는 빈(Blank) 노드, 사선 노드는 자식 노드 또는 하위 프리픽스가 있는 노드이다. 이러한 개념을 적용하여 트라이를 3 계층(16 비트, 8 비트, 8 비트)로 구성하고, 프리픽스 유무를 표현하는 비트-벡터(Bit-vector)를 사용하는 방법에 대한 연구[11]가 발표된바 있다. [11]에서는 비트-벡터를 이용하여 포워딩 테이블에서 프리픽스의 출현 순위를 표현하는 배열(Array)로 포워딩 테이블을 구성하였다.

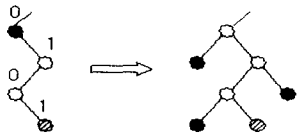

그림 2 프리픽스 트리의 확장

본 논문에서는 그림 1.(b)와 같이 프리픽스 트라이를 8 비트 깊이의 4 계층으로 구성하였다. 전체 프리픽스 트라이를 8 비트 깊이의 부(sub) 트라이로 분할하면 깊이 8에서 리프 노드의 개수는 최대 2^8-1이 된다. 그림 3과 같이 깊이 8의 부 트라이의 각 리프 노드 256 비트는 프리픽스 비트 정보와 자식 노드 정보를 두개의 비트로 표시한다. 하나의 노드는 IP 주소 검색 때 필요한 시프트(Shift) 연산의 회수를 줄이기 위해 모든 선행 노드의 프리픽스 비트(tp)와 자식 노드 비트(tc) 개수를 누적한 필드값과 현재 노드의 8 바이트 마다 비트 개수를 기록한 3개의 필드(p1,p2,p3/prefix, c1,c2,c3/child)를 갖는다. 전체 프리픽스 트라이를 깊이 8의 부 트라이로 넓이우선 탐색 순서로 구성하고, 각각의 부 트라이를 하나의 노드로 갖는 비트-벡터(Bit-vector)테이블로 구성한다. 비트-벡터 테이블의 노드 개수는 약 54,000개의 라우팅 테이블에 대해 약 3,700개가 되었으며 284Kbyte의 크기로 구성되었다. 이러한 크기는 펜티엄 II 400MHz 프로세서의 L2 캐쉬에 저장되는 충분히 작은 크기이며, 라우팅 테이블의 크기에 따라 평균 320~530 nsec의 검색 속도를 기록하였다.

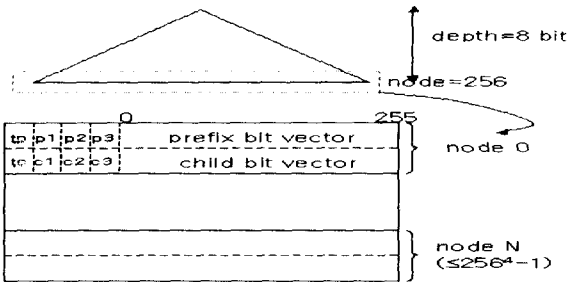

그림 3 비트-벡터(Bit-Vector) 테이블

비트-벡터 테이블을 이용한 IP 주소 검색은 표 1의 알고리즘과 같다. 줄 2에서 IP 주소를 8 비트 단위로 나누어서 사용하는 것은 프리픽스 트라이를 8 비트 단위로 계층화 했기 때문이다. 구해진 8 비트의 값은 부 트라이에서 대응되는 리프 노드를 뜻하며, 비트-벡터 노드의 prefixBit 필드와 childBit 필드

의 위치를 의미한다. 줄 3에서와 자식 노드가 있음을 알았을때는 현재 위치까지의 자식 노드의 개수와 프리픽스 비트의 개수를 구하고, 줄 5와 같이 이것을 비트-벡터 테이블의 인덱스로 사용하여 다음 8 비트에 대한 검색 노드로 사용한다. 마지막으로 자식 노드가 더 이상 없고, 프리픽스 비트가 있으면 다시 프리픽스 비트의 개수를 계산한다.

```
1. Let bit-vector table K(idx:=0), and
   routing entry Z:=default
2. value:=get 8 bit of dest. IP address from MSB
3. if K(idx).childBit(value)==1
        idx:=number of all child bit and
        if K(idx).prefixBit(value)==1
                Z:=number of all prefix bit-1
4. else if K(idx).childBit(value)==0
        if K(idx).prefixBit(value)==1
                return pointer(Z)=number of all
                                  prefix bit-1
        else    return pointer(Z)
5. repeat 2 to 4 for remaind three 8 bit of address
```

알고리즘 1 검색 알고리즘

검색의 결과로 구해진 프리픽스 비트의 개수는 포워딩 테이블 엔트리의 인덱스가 된다. 그러나 비트-벡터 테이블의 프리픽스 비트 개수는 포워딩 테이블의 엔트리 개수보다 많으므로 구해진 인덱스로 직접 포워딩 테이블을 접근할 수는 없다. 비트-벡터 테이블의 프리픽스 비트 개수가 많은 이유는 그림 2에서와 같이 이진 트리(Binary tree)를 트라이로 확장했기 때문이다. 따라서 비트-벡터 테이블의 프리픽스 비트는 확장된 트라이의 인덱스이며 이진 트리의 인덱스와 트라이의 인덱스를 대응시키는 별도의 자료구조-pointer-를 사용하여야 한다. 필요한 자료구조의 개수는 확장된 트라이의 프리픽스 비트의 개수와 같으며 검색 완료후 1회의 메모리 읽기를 추가로 발생시킨다.

4. 실험 및 토론

비트-벡터 테이블을 이용한 포워딩 실험은 표 1과 같이 미국의 실제 라우팅 테이블[12]을 이용하여 실시하였다. 표 1에서 보는바와 같이 라우팅 테이블의 크기는 4만8천여개 이상의 백본급 라우터부터 수천개의 엔트리를 가진 기업(Enterprise)급 라우터까지로 다양하게 구성되어 있다. 포워딩 테이블의 구성(Build) 소요시간은 메모리에 있는 프리픽스 트라이로부터 비트-벡터 테이블을 구성하는 시간으로 약 90 msec부터 280 ms가 소요되었다. [10]에 의하면 백본급 라우터의 포워딩 테이블이 초당 1회 정도의 수정 빈도임을 감안할 때 충분히 작은 시간이다.

Site	Date	Routing entries	Leaves	Bit-vector size	Build time (ms)	Seek time (ns)
Mae-East	99.12	48,290	52,858	284K	290	530
Mae-West	99.12	29,588	32,462	247K	210	475
PacBell	99.12	25,275	26,897	226K	190	395
AADS	99.12	16,864	18,093	190K	140	380
Paix	99.12	9,618	10,434	141K	80	320

표 1 포워딩 테이블 생성 데이터

비트-벡터 포워딩 테이블을 이용한 검색은 리눅스 커널 2.2.12에서 펜티엄 II 400MHz 프로세서(L1 cache 32Kbyte, L2 cache 512Kbyte)를 이용하였다. Paix 라우터의 경우 평균 검색 시간은 320 nsec를 기록하였고 가장 큰 Mae-East 라우터는 530 nsec를 기록하였다. 검색에 사용한 IP 주소값은 랜덤(Random) 함수로 생성 하였고, 백만번의 검색을 반복하여 성능을 측정하였다. 현재의 일반적인 프로세서는 우리의 이해 범위내에서는 특정 데이터 블록을 L2 캐쉬에 상주시킬 수 있는 특별한 방법이 없으므로, 우리의 실험 결과는 포워딩 테이블이 L2 캐쉬와 메모리에 혼재하고 있는 상태의 결과치로 판단된다.

5. 결론

본 논문은 기가비트 라우터를 지원할 수 있는 포워딩 테이블을 구현하기 위한 새로운 자료구조를 제안 하였다. 실험을 통해 구현된 자료구조는 평균 320~530 nsec의 검색 속도로 초당 수백만개의 IP 주소 검색 성능을 보였다. 또한 구현된 포워딩 테이블 자료구조는 약 48,000여개의 엔트리를 가진 실제 라우팅 테이블에 대해서 약 280 Kbyte의 크기로 일반적인 펜티엄 프로세서의 L2 캐쉬에 저장될 수 있는 크기이다.

향후의 연구는 포워딩 테이블이 프로세서의 L2 캐쉬에 상주하도록 구성된 환경에서 속도를 측정하고, 기존에 제안된 다양한 자료구조와 알고리즘을 하나의 플랫폼에서 구현하여, 본 논문에서 제안된 자료구조 및 상호간의 성능을 비교하여 분석하는 것이다.

6. 참고 문헌

[1]. Washington University Workshop on Integration of IP and ATM, November 1996, *http://www.arl.wustl.edu/arl/workshop/atmip*

[2]. Peter Newman, Tom Lyon, and Greg Minshall, "Flow labeled IP: a connectionless approach to ATM", *IEEE Infocom*, San Francisco, California, March 1996

[3] Guru Parulkar, Douglas C. Schmit, and Jonathan Turner, "IP/ATM: A strategy for integrating IP with ATM", *Computer Comminication Review*, 25(4):49-58, October 1995. Proceedings ACM SIGCOMM '95 Conference

[4]Gurudatta Parulkar, Douglas C. Schmidt, and Janthan S. Turner, "GIPR: a gigabit IP router", In *proc. of Gigabit Networking Workshop*, Boston, MA, April 1995

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

[7] Tong-Bi Pei and Charles Zukowski, "Putting Routing Tables in Sillicon", *IEEE network Magazine*, January 1992

[8] A.J. McAuley and P. Francis, "Fast routing table lookup using CAMs", IEEE Infocom, v3, 1382-1391, San Francisco, 1993

[9] A. Bremler-Barr, Y. Afek, and S. Har-Peled, "Routing with Clue", *Proceedings ACM SIGCOMM 99*, Cambridge, September 1999

[10] Stanford University Workshop on Fast Routing and Switching, December 1996, *http://tiny-tera.stanford.edu/Workshop_Dec96/*

[11] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen pink, "Small Forwarding Tables for Fast Routing Lookups", *Proceedings ACM SIGCOMM* 97, October 1997

[12] Michigan University and merit Network, Internet Performance Management and Analysis (IPMA) project, *http://nic.merit.edu/~ipma*