RLC 압축을 이용한 고속 라우팅 검색기법

오승현°
동국대학교 컴퓨터학과
shoh°@dongguk.ac.kr

A Fast Routing Lookup Technique using RLC Compression

Seunghyun Oh°
Dept. of Computer Science, Dongguk University

요 약

라우터의 IP 주소검색은 라우터에 도착한 IP 패킷의 목적지 주소를 이용하여 적절한 다음-홉 주소를 검색하고 결정하는 것이다. IP 주소검색은 라우터 성능의 병목지점으로써 고속 백본망에 필요한 초고속 라우터 개발에 필수적인 부분이다. 현대적인 고속 라우터는 초당 수백 만개 이상의 IP 패킷을 처리할 수 있는 능력이 필요하다. 본 논문은 보통의 Pentium CPU에서 OC-48 링크 수준의 IP 주소검색을 실시할 수 있는 빠른 검색방법을 소개한다. 본 논문에서는 데이터 압축기법으로 사용되는 RLC(Run Length Coding)와 유사한 방법을 사용하였다.

1. 서 론

근래의 인터넷은 초고속 엑세스망의 보급과 인터넷 접속자의 급격한 증가 때문에 발생한 트래픽의 증가가 백본망의 확장을 요구하는 시대이다. 백본망의 확장은 네트워크 케이블의 증설과 라우터 처리속도 증가로 대표된다. 네트워크 케이블의 용량은 지속적으로 증가하여 테라비트로 전이되고있는 상태이다. 그러나 라우터의 처리속도는 그에 미치지 못하며 기가비트급 처리속도를 보이고 있다. 라우터는 세 가지 기능을 가지고 있다. 첫째, 라우터에 도착한 패킷의 처리순서를 결정하는 scheduling 기능이다. 둘째, 패킷을 출력단자(output port)로 전달하는 스위칭(Switching)이며, 마지막으로 세 번째 기능은 패킷의 목적지 IP 주소를 바탕으로 다음-홉(Next-hop) 주소를 결정하는 IP 주소 검색기능이다. IP 주소검색 기능은 CIDR[1] 주소체계를 지원하는 BGP-4[2] 라우팅 프로토콜의 보급에 따라 supernet 개념이 도입됨으로써 최장 일치 검색(LPM: Longest prefix match)을 지원해야 함에 따라 O(1)의 속도를 달성하기 어려운 문제가 되었다.

몇 년 사이에 IP 주소검색의 고속화를 위한 다양한 연구가 진행되어왔다. 이러한 연구는 크게 하드웨어를 이용하는 방법[3,4]와 소프트웨어를 기반으로 하는 방법[5,6]으로 구분할 수 있다. 하드웨어를 이용하는 방법은 CAM(Content Access Memory)을 이용[3]하거나 주소검색용 로직을 구성하는 방법[4]등 이다. 소프트웨어를 이용하는 방법은 라우팅 테이블이나 포워딩 테이블의 구조를 고속검색이 가능하도록 변경함으로써 IP 주소검색의 성능을 향상시키려는 방법이다. 소프트웨어 기반의 자료구조로는 해시 테이블과 트라이가 일반적으로 사용되었다.

소프트웨어 기반 연구들은 비트맵의 사용여부에 관계없이 메모리 검색횟수를 감소시킴으로써 IP 주소검색 속도를 단축하려고 한다. 메모리 검색횟수를 감소시키는 방법은 크게 두 가지 방법으로 구분할 수 있다. 첫째, 메모리 소요량을 크게 하더라도 최소 횟수로 검색을 할 수 있도록 한다[5]. 둘째, 메모리 소요량을 매우 작게 즉, 프로세서의 캐쉬에 저장할 수 있는 정도의 작은 크기로 만들어 메모리 검색을 캐쉬 검색으로 변경함으로써 IP 주소검색 속도를 빠르게 한다[6]. 두 가지 방법을 비트맵에 적용하면, 첫 번째 방법의 경우 트라이의 리프노드에서 프리픽스 정보를 수집하여 메모리에 저장하면 2^{32}개의 정보를 메모리에 저장하는 반면 1회의 메모리 검색으로 IP 주소검색을 완료할 수 있다. 그러나 이때 필요한 메모리의 크기가 약 4GB로 현실적으로 적용하기 어려우므로 메모리의 크기를 적당량 줄이면서 검색횟수를 감소시키는 방법을 찾게된다.

두 번째 방법에 트라이를 적용하면 트라이의 프리픽스 정보에 다양한 자료구조를 적용하여 필요한 메모리 크기를 보통의 펜티엄 프로세서가 가진 L2 캐쉬의 크기 512KB보다 작은 크기로 만든다. 캐쉬보다 작은 크기의 포워딩 테이블을 캐쉬에 저장하면 IP 주소검색에 필요한 메모리 검색을 캐쉬검색으로 대치할 수 있으므로 약간 복잡한 자료구조를 채택하더라도 고속의 검색속도를 얻을 수 있다. 본 논문의 목표는 메모리의 크기와 메모리 검색회수 사이의 관계를 고려하여 캐쉬 크기 이하의 메모리 크기에서 최소의 메모리 검색으로 IP 주소검색을 할 수 있도록 적절한 트라이 분할 깊이를 선택하여 고속의 성능을 발휘하는데 있다.

본 논문에서 제안한 RLC-Trie는 트라이를 기반으로 매우 작은 크기의 2-level 라우팅 테이블을 사용한다. 트라이의 특정 레벨-N을 기준으로 상하위로 분리된 라우팅 테이블은 각 노드의 상태정보를 표시하는 비트열로 구성되는데 이때 많은 개수의 0 비트가 포함되는 비트열을 Run length coding 기법을 이용하여 메모리 압축을 함으로써 메모리 사용효율의 향상과 캐쉬검색 효과로 고속 IP 주소검색을 지원할 수 있다. 본 논문의 나머지는 부분은 다음과 같은 순서로 구성되어있다. 2장에서는 RLC-Trie의 구조를 소개하고, 3장에서는 RLC-Trie에 대한 실험결과를 제시한다. 마지막으로 4장에서는 결론과 향후 연구방향에 대해 기술한다.

2. RLC-trie 자료구조

RLC-trie는 트라이를 기반으로 라우팅 테이블의 크기를 압축하여 속도가 빠른 SRAM 캐쉬검색으로 고속의 IP 주소검색을

수행하는 자료구조이다. RLC-trie는 2-level 라우팅 테이블 구조로 욕적지 IP 주소의 MSB 16비트를 상위의 U-Trie[1~16]에 대한 인덱스로, 하위 16비트는 L-Trie의 비트 인덱스로 사용한다. U-Trie[17~32]의 리프노드가 하위 노드에 대한 정보가 존재할 때 즉, 16비트 보다 더 긴 길이의 프리픽스로 연결되어 있을 경우에는 2^{16} 비트의 sub L-Trie의 루트가 된다. 그러나 99% 이상의 프리픽스는 24비트 이하이므로[7] 프리픽스 정보는 트라이의 17~24 레벨에 집중되고 나머지 24~32 레벨에는 정보가 없다. 결과적으로 2^{16}비트의 대부분은 프리픽스 정보가 존재하지 않으므로 메모리만 낭비하게된다. RLC-Trie는 L-trie를 구성할 때 프리픽스 정보의 존재 유무를 확인하여 불필요한 부분의 정보는 생략함으로써 높은 메모리 사용효율을 제공한다.

RLC-Trie는 2-level 구조이다. 먼저 프리픽스 트라이를 16비트 레벨에서 자르고, 상위 16비트 레벨의 자료구조를 U-trie[1~16], 하위 16비트 레벨의 자료구조를 L-Trie[17~32]로 분리한다. IP 주소검색은 MSB 16비트로 U-Trie를 먼저 검색하고, L-trie는 더 긴 프리픽스를 검색할 때 사용된다. U-trie와 L-trie는 트라이의 각 노드에 비트 값을 할당한 비트열의 집합이다.

그림 3.(a)는 U-Trie의 자료구조를 보여준다. 트라이의 레벨 16에 위치한 노드와 레벨이 16보다 작지만 리프노드인 노드에 대해 프리픽스 정보를 가졌거나 차일드노드 정보 즉, 16비트 보다 더 긴 프리픽스로 연결되는 노드에 대해서는 1을 할당하고 나머지 노드에는 0을 할당한다. 비트 값들은 좌측에서 우측으로 차례대로 수집되며, 수집된 비트열의 길이는 2^{16}이다. 이 비트열은 *bitvec* segment와 bitvec에 포함된 비트 1의 누적 개수를 표시하는 offset 값의 쌍으로 저장된다. bitvec segment의 길이가 16비트이므로 U-trie의 테이블은 4,096개의 엔트리로 구성된다. offset 필드는 segment에 있는 비트 1의 누적 개수를 미리 계산하여 저장하고 있음으로써 비트 1의 개수를 계산할 때 시간을 단축할 수 있다. 최대 2^{16}개의 비트 1이 있을 수 있으므로 16비트 길이가 필요하다.

U-Trie에 포함된 비트 1은 두 가지 의미를 가질 수 있다. 어떤 비트 1은 프리픽스를 의미하고, 어떤 비트 1은 차일드 노드의 존재를 의미한다. 하지만 U-Trie의 bitvec에서는 그 의미를 구분할 수 없다. 따라서 비트 1의 의미를 구분하기 위해서는 다음-홉 정보와 차일드노드 정보를 저장하는 별도의 자료구조 *U-Trie Port(UTP)*가 필요하다. UTP의 구조는 그림 3.(b)에서 확인할 수 있다. UTP는 0~2^{16}개의 정보를 단지 비트열로 저장함으로써 메모리를 절약한 U-Trie에 대해 실제 저장된 비트 1의 개수와 동일한 개수의 엔트리만을 수용함으로써 메모리 낭비를 방지한다. UTP의 Max_depth와 Min_depth는 16비트 이상의 프리픽스 중에서 가장 긴 프리픽스와 가장 짧은 프리픽스의 길이정보를 담는다. 그림 1의 예를 보면 레벨 16에 있는 노드 e는 16비트 이상의 프리픽스로 연결되는 노드이므로 노드 e는 하위 16비트 레벨을 대표하는 노드이며 동시에 L-Trie의 루트노드이다. 그림 1에서 p1~3 노드는 노드 e에서 연결되는 프리픽스이므로 노드 e의 Max_depth는 p3노드의 level 3, Min_depth는 p1의 level 1이다. L-Trie에 대해 우리는 세 가지 압축방법을 적용하여 크기를 획기적으로 줄일 수 있다.

L-trie는 세 가지 형태의 비트열을 갖고 있다. 즉, 그림 2의 노드 f와 같이 Max_depth가 24이하인 프리픽스만 있을 때는 2^8 비트열을 갖고 U-Trie의 Ptr_L2 필드는 해당 비트열이 L-Trie에 저장된 위치를 가르치는 포인터가 된다. 두 번째 비트열의 형태는 Min_depth가 24보다 큰 경우의 비트열이다. 이것은 모든 프리픽스가 그림 2와 같이 17~24 사이에는 프리픽스가 없고 25~32 깊이에만 프리픽스가 있다. 그러므로 노드 j를 기준으로 2^{16}비트의 비트열이 필요하지만 모든 프리픽스가 24보다 크기 때문에 17~24 사이의 비트열은 생략한다. 25~32 사이의 프리픽스를 저장하기 위해서는 트라이 레벨 24에 존재하는 노드를 루트로 2^8 비트열을 저장한다. 그림 2의 p1과 p2가 각각 2^8 비트열로 차례대로 L-Trie에 저장되며 이때 노드 j의 비트열 블록에 대한 포인터는 L-Trie Short.ptr_L2에, 비트열 블록의 offset 정보는 L-Trie.offset에 기록한다.

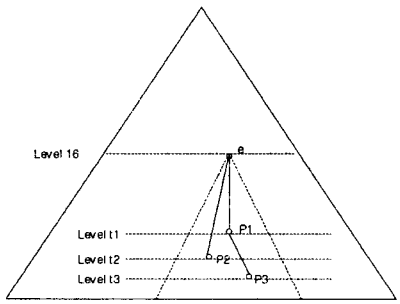

그림 1 U-Trie Port의 Max_depth와 Min_depth

세 번째는 비트열의 형태는 Min_depth는 24보다 작고 Max_depth는 24보다 큰 경우이다. 그림 2의 노드 j와 같이 프리픽스들이 17~32 사이에 고르게 분포된 경우이다. 이때는 17~24 사이의 p3 영역을 먼저 하나의 2^8 비트열에 저장하고, 25~32 사이의 프리픽스 p4와 p5는 두 번째 형태의 p1과 p2 비트열을 만드는 것과 같은 방법으로 비트열을 생성한다. 이때 프리픽스 p3은 L-Trie에 저장하지 않고 비트열과 다음-홉 정보를 별도의 그림 3 (f) *L-Trie Long*에 저장한다. 그 이유는 p4, p5 비트열과 함께 L-Trie에 저장될 경우 L-Trie의 인덱스로 사용될 IP 주소의 길이를 결정할 때 혼동이 초래되기 때문이다. 그림 3 (f)에서 L-Trie Long의 구조를 볼 수 있다.

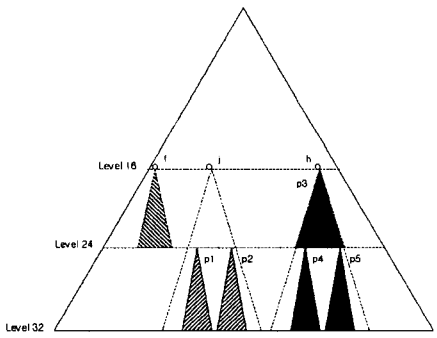

그림 2 L-trie의 세 가지 비트열 분포

대폭적인 L-Trie의 압축에도 불구하고 우리가 제안하는 RLC-Trie의 장점은 U-Trie와 L-Trie의 기본구조가 여전히 동

일하다는 것이다. 즉, <segment+offset>의 구조를 유지하면서 크기만을 압축하여 단순한 검색 알고리즘 구성이 가능하다. 그러나 약 1.2MB의 L-Trie 크기는 펜티엄 프로세서의 L2 캐쉬크기보다 더 크므로 속도가 빠른 SRAM 캐쉬의 이점을 활용하기는 어렵다. 본 논문의 두 번째 메모리 압축방법은 데이터 압축방법으로 사용되는 런-길이 코딩(RLC:Run length coding)과 유사한 방법이다. RLC는 반복적으로 나타나는 데이터를 생략하고 반복회수를 기입하는 방법이다. 예를 들면, 'ABBBCC'는 'A3B2C'로 압축한다. 17~32비트 길이의 프리픽스가 L-Trie 영역에서 희박하게 흩어져있는 이유로 L-Trie의 비트열에는 많은 비트 0의 반복이 존재한다. 누적된 비트 1의 개수가 다음-홉 주소에 대한 인덱스가 되는 트라이 기반의 알고리즘에서 비트 0은 트라이의 구조와 프리픽스의 위치를 표시하지만 동시에 불필요한 부담이 된다. 우리는 RLC 압축방법을 이용하여 반복되는 비트 0을 생략함으로써 약 369KB의 L-Trie를 더 압축할 수 있다.

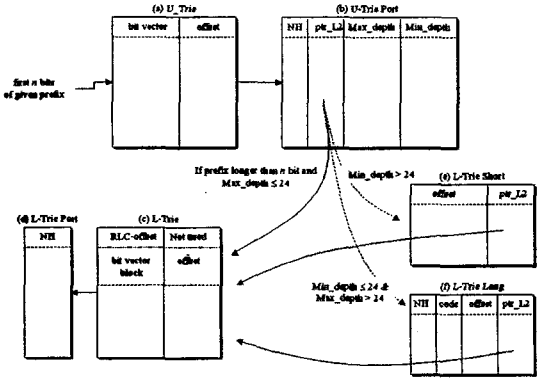

그림 3 RLC-Trie 구조

본 연구에서 RLC 압축방법을 적용하는 방법은 각 비트열 segment가 모든 비트 0으로 구성되어있을 때는 해당 segment를 L-Trie에 저장하지 않는 것이다. 이때 저장하지 않은 segment가 트라이의 구조를 왜곡하는 것을 막기 위해 추가적인 비트열(RLC-offset)이 사용된다. RLC-offset은 16개의 segment에 대해 비트 1이 포함되어 L-Trie에 segment가 저장될 때는 비트 1을, segment가 저장되지 않을 때는 비트 0을 할당한다.

3. 실 험

RLC-Trie 기반의 IP 주소검색은 IP_MSB[0:15]를 인덱스로 U-Trie에서 검색이 종료되는 경우 U-Trie와 UTP에 대한 두 번의 메모리 검색이 필요하므로 DRAM 메모리의 검색속도를 50ns로 가정할 때 100ns가 소요된다. 그러나 U-Trie의 한 segment에 들어있는 비트 1의 개수가 2 이하일 경우에는 U-Trie의 offset에 다음-홉 정보가 저장되어 있으므로 한번의 메모리 검색으로 완료할 수 있다. 메모리 검색회수를 기준으로 볼 때 최대검색 시간은 5회의 메모리 검색으로 250ns이다. 이 경우는 L-Trie Short이나 L-trie Long을 접근하는 경우로 U-trie와 UTP에 대한 메모리 검색 2회와 L-trie Short 또는 L-Trie Long에 대한 검색 1회, L-trie와 L-Trie Port에 대한 검색 2회로 총 5회의 검색이 필요하다. 그러나 실제 라우팅 테이블에서 99%의 프리픽스 길이가 24비트 이하이고 대부분의 패킷이 24비트 이하의 프리픽스에 매칭되며, 우리의 실험에서는 77%의 패킷이 U-Trie에 대한 검색으로 종료되었다. 따라서 RLC-Trie의 평균검색 속도는 2×0.77+5×0.23회의 메모리 검색을 할 것이므로 DRAM 검색속도 50ns를 적용하면 134ns의 성능을 발휘할 수 있다.

4. 결 론

본 연구에서 사용한 RLC-Trie는 작고 효율적인 라우팅 테이블 구조를 도입함으로써 고속의 IP 주소검색을 수행할 수 있다. 특히 프리픽스 트라이를 비트열로 전환할 때 효율적인 세 가지 압축방법을 도입함으로써 작은 메모리 크기를 구현하였다. 세 가지 압축방법들은 다른 비트열 기반의 연구들에도 손쉽게 적용할 수 있다. 계산에 의하면 패킷당 평균 134ns의 검색속도를 지원할 수 있으므로 비교적 저성능의 PC환경에서 OC-48 링크를 지원할 수 있다. 본 연구의 향후과제는 실제 라우팅 테이블에 RLC-Trie를 적용하여 성능을 측정하고 기존연구와 비교하며, 라우팅 프리픽스의 분포를 상세히 조사하여 L-Trie의 구조를 최적화하는 것이다.

참고문헌

[1] V. Fuller, T. Li, J. Yu and K. Varadhan, "Classless Inter-Domain Routing(CIDR): an Address Assignment and Aggregation Strategy", RFC 1519, IETF, Sept. 1993

[2] Y. Rekhter and T. Li, "A Border Gateway Protocol 4(BGP-4)", RFC 1771, IETF, Mar. 1996

[3] A.J. McAuley and P. Francis, Fast routing table lookup using CAMs, Proceedings IEEE Infocom93, San Francisco, 1993

[4] N. Huang and S. Zhao, "A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers", IEEE JSAC, vol. 17, no. 6, June 1999

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

[6] M. Degermark, A. Brodnik, S. Carlsson and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," Proceedings of ACM SIGCOMM'97, Oct. 1997

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