가변적인 복수 해싱을 이용한 글로벌 IPv6 유니캐스트 주소 검색 구조

준회원 박 현 태*, 정회원 문 병 인**, 강 성 호*

A Global IPv6 Unicast Address Lookup Scheme Using Variable Multiple Hashing

Hyuntae Park* ***Associate Member***, Byung In Moon**, Sungho Kang* ***Regular Members***

요 약

IP 주소 검색 구조는 IPv6 주소체계의 도래와 함께 더욱 고속 네트워크 기술의 중요한 이슈가 되고 있다. 본 논문에서는 차세대 인터넷 라우터를 위한 새로운 글로벌 IPv6 유니캐스트 주소 검색 구조를 제안한다. 제안하는 구조는 적절한 프리픽스 그룹화를 수행하고 각 그룹별로 가변적으로 복수 해싱을 수행한다. 이를 통해 적절한 개수의 포워딩 테이블에서 해싱의 충돌(collision)을 효율적으로 분산함으로서 오버플로우를 최소화하였으며 포워딩 테이블 구성을 위해 필요한 메모리 크기를 줄였다. 또한 단 한 번의 메모리 접근만으로 포워딩 테이블 구성 및 검색이 가능한 고속 주소 검색 구조이며 추가적 업데이트가 용이한 구조이다. 실제 6bone 테스트 라우팅 CERNET 데이터를 이용하여 균일한 복수 해싱을 이용한 구조와 제안한 구조를 비교, 실험하였으며 동일한 8개의 테이블에서 약 15%의 포워딩 테이블을 위한 메모리 절약과 약 50%의 오버플로우 감소를 확인하였다.

Key Words : IP Address Lookup, Global IPv6 Unicast Address, Variable Multiple Hashing

ABSTRACT

An IP address lookup scheme has become a critical issue increasingly for high-speed networking techniques due to the advent of IPv6 based on 128bit. In this paper, a novel global IPv6 unicast address lookup scheme is proposed for next generation internet routers. The proposed scheme perform a variable multiple hashing based on prefix grouping. Accordingly, it should not only minimize overflows with the proper number of memory modules, but also reduce a memory size required to organize forwarding tables. It has the fast building and searching mechanisms for forwarding tables during only a single memory access. Besides, it is easy to update forwarding tables incrementally. In the simulation using CERNET routing data as a 6bone test phase, we compared the proposed scheme with a similar scheme using a uniform multiple hashing. As a result, we verified that the number of overflows is reduced by 50% and the size of memory for forwarding tables is shrunken by 15% with 8 tables.

Ⅰ. 서 론

최근 유비쿼터스 시대의 도래와 함께 인터넷을 비롯한 데이터통신 중심의 네트워크는 폭발적인 호스트와 트래픽의 증가에 당면하고 있다. 이에 따라 링크 매체의 대역폭(bandwidth)과 전송속도(link speed),

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음.
* 연세대학교 전기전자공학과 컴퓨터시스템 및 관련 SoC 연구실 ({radiob, shkang}@yonsei.ac.kr)
** 경북대학교 전자전기컴퓨터학부 시스템온칩 연구실 (bihmoon@knu.ac.kr)
논문번호 : KICS2006-01-047, 접수일자 : 2006년 1월 25일, 최종논문접수일자 : 2006년 4월 26일

라우터의 데이터효율(throughput)에 비해 네트워크 성능을 저하시키는 주요 병목 현상의 요인인 패킷 전달율을 향상시키기 위한 IP 주소 검색 구조가 많이 연구되고 있다.

IP 주소 검색의 성능 향상을 저해하는 요인은 크게 두 가지를 들 수 있다. 첫째는 LPM(Longest Prefix Matching) 검색이 필요하기 때문이다. 호스트 수의 증가와 함께 도래한 IPv4 주소의 고갈을 해소하기 위해 임의의 길이로 프리픽스를 할당하는 CIDR(Classless Inter-Domain Routing)이 도입되었다. 이에 따라 각 패킷마다 프리픽스의 길이가 가변적이므로 IP 주소 검색 구조 역시 기존의 단순한 완전 매칭(exact matching) 방식으로는 주소 검색을 수행할 수 없게 되었고 가능한 모든 프리픽스 길이 중에서 가장 길게 매치하는 프리픽스를 선택하는 검색 기법이 필요하게 되었다. 둘째는 메모리 접근 속도의 한계 때문이다. 방대한 포워딩 테이블은 온칩 메모리 또는 외부 메모리에 저장된다. 패킷의 최소 단위인 40byte의 패킷이 OC768의 라인 속도 40Gbps로 라우터에 입력된다고 가정하면 초당 125,000,000개 패킷을 주소 검색해야 한다. 따라서 다른 기능의 처리 시간을 고려할 때 가정한 라인 속도로 패킷을 포워딩하기 위해서는 주소 검색 시간이 한 패킷당 8ns 이하여야 한다. 그러나 온칩(on-chip) 메모리의 경우 SRAM은 1~5ns, DRAM은 약 10ns의 접근 시간이 필요하며 외부(off-chip) 메모리의 경우에는 SRAM은 10~20ns, DRAM은 60~100ns의 접근 시간이 필요하다. 그러므로 주소 검색 과정 중의 메모리 접근 횟수를 줄이는 것이 검색 속도 향상의 주요한 이슈가 된다[1].

메모리 접근 횟수는 검색 알고리즘의 복잡도(complexity)와 연관된다. 따라서 O(1)의 해싱 알고리즘이 완전 매칭이 필요한 클래스 기반의 IP 주소 검색과 MAC 주소 검색에서 유용하게 사용되어 왔다. 그러나 LPM이 필요한 IP 주소 검색 구조에 해싱 적용은 쉽지 않다. 해싱은 완전 매칭을 위한 검색 알고리즘이기 때문에 각 길이별로 매치된 모든 결과를 비교하여 가장 길게 매치된 결과를 선택하는 과정이 필요하다. 즉 해싱 기반 IP 주소 검색 구조는 LPM을 완전 매칭 문제로 전환하여 해결할 수 있는 구조를 가져야 한다. 더욱이 서로 다른 프리픽스 입력에 대하여 동일한 해쉬 인덱스(index) 값이 발생하는 충돌(collision)을 해결할 수 있어야 한다. 충돌을 줄이는 방법은 크게 완전 해쉬 함수(perfect hash function)를 사용하여 원천적으로 발생하지 않게 하는 방법과 동일 인덱스에 대해 여러 엔트리 저장 공간을 확보하는 방법이 있다. 그러나 첫 번째 방법은 완전 해쉬 함수를 구하는 데는 긴 시간이 필요하고 두 번째 방법은 메모리 공간이 많이 필요하므로 좋은 해결책이 아니다. 따라서 많은 충돌이 발생하는 해쉬 함수를 이용한 검색구조는 메모리 효율이 좋지 않게 된다[2]. 특히 충돌의 중첩에 의해 발생하는 오버플로우 처리는 별도의 보조 테이블을 이용하여 처리해야 한다. 보조 테이블은 TCAM를 사용하거나 별도의 메모리 모듈에서 또 다른 검색 단계가 필요하게 된다. 따라서 TCAM에 의한 단가 상승이 발생하거나 부가적 검색에 따른 추가 지연이 발생하게 된다[3, 4].

더군다나 32bit 기반 IPv4 주소의 고갈문제를 근본적으로 해결하기 위해 IETF(Internet Engineering Task Force)에 의해 128bit 기반 IPv6 주소 체계가 제안되었다[5]. IPv6 주소 체계 역시 패킷에 프리픽스 길이 정보가 없고 임의의 프리픽스 길이를 가지므로 LPM기반의 검색 구조가 필요하지만 IPv4 기반 기존 연구들은 확장성 문제로 인해 그대로 적용하는데 어려움이 있다. 본 논문에서는 메모리 효율적인 고속 글로벌 IPv6 유니캐스트 주소 검색을 위해 프리픽스 그룹화 기반의 가변적인 복수 해싱 방식을 제안한다.

Ⅱ. 기존 연구

2.1 트라이(trie) 기반 검색 구조

IP 주소 검색의 연구 초기부터 트라이(trie) 기반의 소프트웨어 구현 방식이 가장 널리, 많이 연구되어 왔다[6]. 기본적인 1-bit 트라이 기반의 검색 구조는 엔트리 수 N, 주소길이 W에 대해서 W에 비례한 검색 시간이 필요하며 NW에 비례한 메모리 공간이 필요하다. 또한 추가 업데이트를 위해서는 전체 트라이를 완전히 재구성하는 것이 필요하였다. 따라서 128bit로 주소길이가 길어지는 IPv6 주소 체계에서의 트라이 기반의 주소 검색 구조는 트라이의 레벨이 많아짐에 따라 치명적인 구성 및 검색 속도의 저하를 초래하며 필요한 메모리 크기도 기하급수적으로 증가하게 된다. 이러한 양상은 개선된 트라이 구조를 갖는 레벨압축 검색 기법[7]과 트라이 레벨의 바이너리 검색 기법[8], 다중 영역 검색 기법[9]에서도 근본적으로 해결하지 못하고 비슷한 문제점을 갖는다.

2.2 TCAM 기반 검색 구조

TCAM(Ternary Content Addressable Memory)을 이용한 주소 검색 구조[10]은 O(1)의 검색 속도를 가지므로 현재 상용 라우터에서 가장 많이 사용되고 있다. 그러나 TCAM은 같은 크기의 RAM보다 단가가 비싸고 저장 용량에 한계가 있으며 전력 소모가 많고 자체 오버헤드가 크기 때문에 내장 메모리로 사용하기 어렵다는 단점이 있다. 따라서 전체 포워딩 테이블을 TCAM으로 구성한다면 O(N)의 저장 공간이 필요하므로 엔트리 수에 비례하여 엄청난 단가 상승을 초래할 것이다. 이러한 문제점은 128bit의 IPv6 주소 체계에서는 더욱 가중될 것이다.

2.3 해쉬 기반 검색 구조

해쉬을 이용한 검색 방식은 프리픽스 길이가 길어지거나 엔트리 수 증가에 따라 포워딩 테이블의 크기가 커지더라도 검색 속도에 큰 영향을 받지 않는 장점이 있다. 따라서 IPv6 주소 체계에서 해쉬의 적용은 더욱 유용하다. 앞 서 언급 했듯이 충돌을 최소화하고 효율적으로 처리할 수 있는 방법이 해쉬 기반 검색 구조의 핵심적인 이슈이다.

기존에 많이 이용된 방법은 XOR-folding 또는 CRC polynomial 하드웨어를 이용한 준 완전(semi-perfect) 해쉬 함수를 이용하거나 완전 해쉬 함수 대신 복수(multiple) 해쉬을 이용한 구조이다[11]. 또한 충돌 또는 오버플로우를 처리하기 위한 보조 테이블의 검색 동작을 개선하는 구조가 여럿 제안되었다[3, 4, 12].

하지만 복수 해쉬을 이용한 방식은 효과적으로 충돌을 줄였음에도 불구하고 많은 별도의 테이블을 구성함으로써 여러 독립적인 메모리 모듈이 필요한 단점이 있다. 이를 보완하기 위해 프리픽스 그룹화를 접목하여 그룹별로 해쉬을 수행한 방식이 제안되었다[13]. 이 방식은 각 프리픽스 길이마다 각각 해쉬을 수행한 것보다는 더 많은 충돌이 발생하고 필요한 메모리의 크기도 증가할 수 있으나 메모리 모듈의 수를 크게 줄일 수 있다. 또한 복수 해쉬을 이용하기 때문에 더 많은 충돌에 의한 오버플로우도 많이 발생하지 않기 때문에 효과적이었다. 그러나 각 그룹별 엔트리의 수와 분포가 크게 차이가 있기 때문에 모든 그룹에 대해 동일한 개수의 복수 해쉬을 적용하는 것은 비효율적이다.

2.4 IPv6 특성화된 검색 구조

그림 1과 같이 IPv6 주소 체계에서도 가변 길이

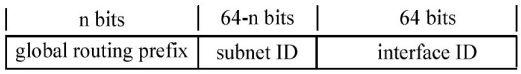

그림 4. IPv6 Global Unicast 포맷

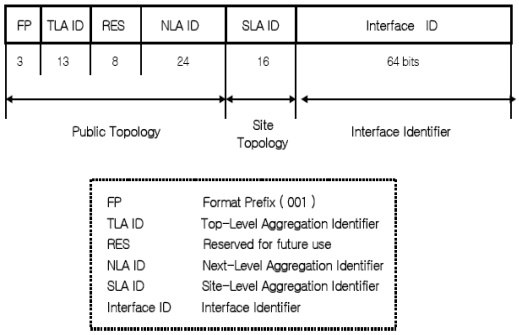

그림 5. Aggregatable Global Unicast 주소 포맷

의 프리픽스를 갖는 CIDR를 따르므로 LPM기반의 주소 검색이 필요하다. 또한 IPv6 주소는 일정한 정책에 따라 할당되고 있으며 그림 2와 같이 계층구조로 구성된다[14]. 현재는 여러 포맷 중 FP필드가 001인 Aggregatable 글로벌 유니캐스트 주소만 할당되어 있다.

기존의 IPv6 특성화된 검색 구조는 주소 체계의 계층적 구조를 응용하였다. 구체적인 방법에는 차이가 있지만 주로 계층적인 구조별로 나누어진 별도의 테이블을 연속적으로 검색하는 구조이며 테이블의 크기를 줄이기 위해 해쉬을 접목하기도 하였다[4, 15, 16]. 그러나 상위 계층 주소라고 해서 많은 엔트리를 갖는 것이 아니기 때문에 이러한 다단 검색 구조는 비효율적인 여러 번의 테이블 검색을 필요로 하게 된다.

Ⅲ. 제안하는 검색 구조

3.1 IPv6 프리픽스 분석 및 그룹화

현재 IPv6 주소는 일반 사용되고 있지 않으며 6bone 네트워크 기반에서 테스트 및 연구 목적으로 할당되고 있다[17]. 그림 3은 848개의 엔트리를 갖는 6bone의 CERNET 2006년 1월 10일 실제 할당된 프리픽스 데이터이다. IPv6 주소 할당은 정책에 따라 계층적으로 이루어진다. 그림 3의 분포에서 확인할 수 있듯이 특정 비트인 24, 28, 32, 48bit에 집중되어 있는 것을 알 수 있다. 이처럼 특정 비트에 엔트리가 집중되는 이유는 주소 할당 정책에 따르며 이를 간단히 정리하면 다음과 같다.

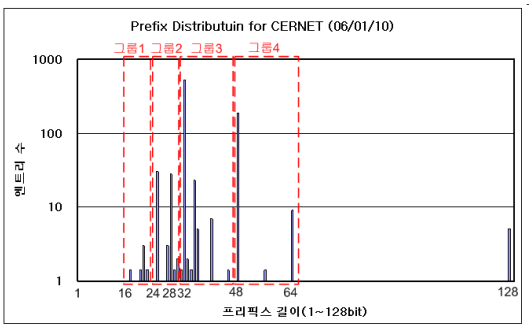

그림 6. IPv6 프리픽스 분포와 제안하는 그룹화

최상위 계층인 RIRs(Regional Internet Registeries)는 초기에 16bit가 할당되다가 RES 필드 정립과 함께 24bit가 할당되고 있다. RIRs에는 APNIC, ARIN, RIPE NCC, LACNIC가 있다. Non-RIRs의 최소 할당 비트는 초기 35bit가 할당되다가 현재는 32bit가 표준으로 할당되고 있다. 따라서 32bit 엔트리가 가장 많이 존재하고 있다. 모든 subnet는 subnet 수의 증가에 따른 유연성 확보를 위해 고정된 길이로 NLA ID 필드까지 이용한 48bit 할당을 원칙으로 하고 있다. 이에 따라 48bit는 현재 두 번째로 많은 엔트리를 갖고 있다. 특수한 경우로 64bit와 128bit가 존재한다. 64bit는 subnet이 단 하나만 존재하는 글로벌 인터넷 교환 포인터(Global Internet Exchange Pointers)에 할당된다. 128bit는 subnet이 존재하지 않는 단일 디바이스의 접속을 위해서 할당된다[16, 18].

이와 같이 주소 할당은 IPv6 주소 포맷과 할당정책에 많은 영향을 받는다. 따라서 향후 IPv6 주소의 프리픽스 길이별 분포도 이와 크게 달라지지 않을 것임을 예측할 수 있다. 단지 현재는 할당 초기 시점이기 때문에 주로 상위 계층에서 할당이 이루어지고 있으므로 실제 subnet에서 IPv6 주소체계를 많이 사용하게 되는 향후 시점에서는 32bit 프리픽스 수는 점차 증가폭이 작아지고 상대적으로 48, 64bit 프리픽스 수는 크게 증가할 것으로 예측된다. 또한 정책에서는 최소 길이만을 언급하였으므로 많은 호스트를 갖는 도메인에서는 더 긴 프리픽스 길이를 갖게 될 것이다. 따라서 많지는 않더라도 16~64bit 길이의 프리픽스는 어느 것이든 존재할 수 있다.

프리픽스 그룹화(prefix grouping)는 여러 프리픽스를 적당한 길이로 묶어서 하나의 테이블에 저장하는 것을 말한다. 이 방식의 가장 큰 장점은 테이블 수를 줄임으로서 필요한 메모리 모듈의 수를 줄일 수 있다는 점이다. 또한 그룹별로 다른 테이블에 프리픽스가 분산되기 때문에 그 만큼 충돌의 발생 가능성도 낮아지는 효과가 있다. 물론 프리픽스 길이 별로 테이블 수가 존재한다면 늘어난 테이블 수 만큼 충돌이 분산되는 효과도 더 커지지만 테이블 수의 증가는 곧 독립적인 메모리 모듈의 증가를 의미한다. 즉 테이블의 수와 충돌의 발생 확률은 tradeoff 관계이므로 프리픽스의 통계적 분포에 따라 효과적으로 그룹화하면 적절한 테이블 수와 충돌 발생 확률 관계의 포화점(saturation point)에 근접할 수 있다.

본 논문에서 제안하는 그룹화는 그림 3에서 함께 표시하였다. 그룹의 선택은 프리픽스 분포를 분석하여 다음 두 가지 사항을 고려하여 수행하였다. 첫째, 임의의 프리픽스 길이를 갖는 것끼리 묶는 것보다 연속된 프리픽스 길이끼리 묶는 것이 용이하다. 그 이유는 연속된 비트 입력에 의해서 인덱스 값을 생성하는 CRC-32를 해쉬 함수로 사용하기 때문이다. 또한 분포를 고려할 때 임의의 프리픽스 길이끼리 특별한 연관성이 없으며 오히려 인접한 프리픽스 길이끼리 비슷한 엔트리 수를 갖는다는 점을 고려하였다. 둘째, 특정 그룹에서 가장 많은 엔트리를 갖는 프리픽스 길이가 각 그룹의 첫 번째 프리픽스 길이가 되도록 한다. 그룹별로 해쉬 함수의 입력은 첫 번째 프리픽스 길이를 사용하기 때문에 프리픽스 분포가 많은 길이를 그룹의 시작 프리픽스로 함으로써 더 신뢰도 높은 해쉬 인덱스 값을 구할 수 있다.

이를 토대로 제안하는 그룹화는 프리픽스를 16~23, 24~31, 32~47, 48~64인 4개의 그룹으로 나누었다. 이는 향후에도 할당 정책을 고려할 때 각 그룹별로 16, 24, 32, 48bit 프리픽스가 가장 많은 엔트리를 갖게 될 것이기 때문에 일반적으로 IPv6 주소 체계에 적용 가능한 그룹화이다. 128bit 프리픽스는 예외적으로 처리한다. 65~127bit의 프리픽스가 존재하지 않음에도 불구하고 128bit를 그룹 4에 포함할 경우 그룹 4 테이블의 프리픽스 필드는 128bit까지 수용할 수 있어야 한다. 따라서 그룹 4 테이블을 위한 메모리 공간이 낭비된다. 그러므로 128bit 프리픽스는 해싱에 의한 구성 및 검색을 수행하지 않고 TCAM으로 구성된 별도의 NS Table (No-Subnet Table)을 통해 수행하도록 한다. 128bit 프리픽스에 해당되는 엔트리가 많지 않기 때문에 NS Table을 위한 TCAM은 많은 양이 필요치 않으므로 단가 상승의 큰 요인이 되지 않을 것이다.

3.2 CRC-32를 이용한 가변적인 복수 해쉉

최근의 기존 연구에서는 해쉉 하드웨어로서 선처리 시간이 없는 XOR-folding을 많이 이용하고 있다. 그러나 XOR-folding은 입력되는 프리픽스 길이와 해쉬 인덱스 길이에 따라 고정된 하드웨어로 구현되기 때문에 본 논문에서 제안하는 가변적인 복수 해쉉의 하드웨어로는 적합하지 않다. 제안하는 구조에서는 복수 해쉉의 가변 적용에 따라 그룹별로 해쉬 인덱스 길이와 적용하는 해쉬함수의 수를 적응성 있게 재설정할 수 있어야 한다. 따라서 해쉉 하드웨어도 이러한 유연성을 가지고 있어야 한다. 본 논문에서는 그림 4와 같은 CRC-32 polynomial을 해쉉 하드웨어로 이용하였다. CRC-32 polynomial은 원하는 길이만큼 복수의 인덱스를 추출하기에 용이하다. 또한 선 처리시간으로 입력된 프리픽스 길이만큼의 시스템 클럭 수가 필요하지만 충돌을 최소화할 수 있는 준 완전 해쉬 함수로의 우수한 성능을 가지고 있으며 간단한 하드웨어로 구현이 용이하다. 또한 다양한 프리픽스 길이가 입력으로 쓰이더라도 이에 크게 영향을 받지 않고 일정하고 매우 높은 엔트로피를 갖는 장점이 있다[19].

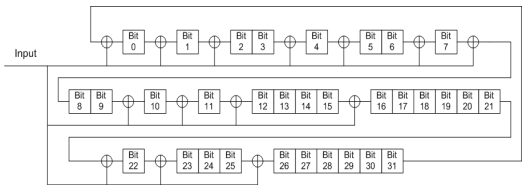

그림 7. CRC-32 polynomial 해쉉 하드웨어

근본적으로 충돌은 해쉉을 통해 큰 데이터베이스를 더 작은 데이터베이스로 줄이거나 분산시킴으로서 발생하며 이 과정에서 많은 키 값을 훨씬 작은 인덱스 값으로 바꾸기 때문에 일어난다. 즉 키 값이 많을수록 인덱스 값이 적을수록 충돌은 더 많이 발생하는 것이다. 인덱스 값의 범위는 할당하는 메모리 크기에 의해 결정된다. 따라서 일정한 메모리 효율에 대하여 충돌의 발생 확률은 해당 엔트리 개수에 비례한다고 볼 수 있다. 여기서 메모리 효율이란 각 테이블에 예상되는 엔트리 개수에 대하여 충돌에 대비한 메모리의 여유를 얼마나 되는지를 의미한다.

표 1과 같이 각 그룹별로 엔트리 수는 상당한 차이를 보인다. 그룹 1은 약 1%정도, 그룹 2는 약 7%정도인데 반해 그룹 3, 4는 대부분을 차지한다. 그러므로 모든 프리픽스 길이 또는 모든 그룹에 대하여 일정한 개수의 해쉬 함수를 할당하는 것은 비효율적이다. 각 그룹별로 엔트리 비율에 따라 적용하는 해쉬 함수의 수를 가변 적용하면 적절한 수의 테이블을 가지고도 효과적으로 충돌 발생 가능성을 줄일 수 있다.

표 1. 그룹별 엔트리 분포

그룹 / 데이터	그룹1 (16~23)	그룹2 (24~31)	그룹3 (32~47)	그룹4 (48~64)	NS (128)
엔트리수	6	64	572	201	5
비 율	0.70%	7.55%	67.45%	23.70%	0.60%

제안하는 구조에서는 표 1의 엔트리 비율 분석에 따라 그룹 1, 2에는 1개의 해쉬 함수를, 그룹 3, 4에는 3개의 해쉬 함수를 가변적으로 적용한다. 이에 따라 포워딩 테이블은 4개의 그룹에 대해 2개의 해쉬 함수를 적용한 경우와 같은 8개의 테이블로 구현한다. 해쉬 함수의 키 값은 각 그룹의 가장 짧은 프리픽스를 사용하므로 각 그룹별로 16, 24, 32, 48bit 가 된다. 따라서 IPv6 주소라고 해서 128bit를 모두 해쉬 함수의 입력으로 사용할 필요가 없으므로 해쉉 하드웨어로 CRC-32를 사용하여도 큰 무리가 없다.

하지만 이렇게 할 경우 그룹 4는 48bit의 입력이 필요하다. 그러나 주소 할당 정책에 따라 48bit 이상이 되는 프리픽스 길이의 앞부분 16bit은 2001(16)과 3ffe(16)로만 구성이 되어 있다. 즉 RFC 2450[14]에 따라 상위 16bit는 일정한 값으로 할당되어 있다. 2001(16)은 Sub-TLA로 할당되어 거의 모든 프리픽스의 상위 16bit가 동일하게 이 값을 갖는다. 3ffe(16)은 6bone 테스트 도메인을 위해 할당되어 있지만 2001(16)에 비해 그 수는 상대적으로 적으며 RFC3701[17]에 언급된 바와 같이 2006년 6월 6일 소멸될 예정이다. 따라서 모든 프리픽스가 동일한 상위 16bit를 가지므로 이 부분은 해쉬 키 값으로 적당치 않다. 그러므로 제안하는 구조에서는 그룹 4의 해쉬 키 값으로 전체 48bit 중 상위 16bit를 제외한 32bit를 해쉬 키로 이용하였다. 따라서 네 그룹 모두 CRC-32 해쉉 하드웨어로 충분히 효과적인 인덱스 값을 얻을 수 있다.

3.3 포워딩 테이블 구성을 위한 메모리 구조

구성 및 검색을 위한 메모리 접근은 프리픽스 또는 목적지 주소를 입력으로 한 해쉉의 결과인 인덱스 값을 메모리 주소로 일정하게 맵핑하여 이루어

진다. 이러한 접근 단위 블록을 버켓(bucket)이라 한다. 각 버켓은 한 개 이상의 로드(load)로 구성된다. 로드는 포워딩을 위한 프리픽스 정보를 저장하는 기본 단위이다.

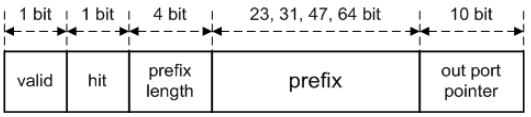

그림 8. 제안한 구조의 한 로드 구조

하나의 로드는 그림 5와 같이 구성된다. valid, hit 필드는 엔트리의 추가, 삭제를 통해 테이블의 효과적인 관리를 위해서 쓰인다. 또한 out port pointer 필드는 각 프리픽스에 해당하는 아웃 포트 정보를 담고 있다. 각 그룹별로 표현해야 할 모든 프리픽스 수는 현재 엔트리를 고려할 때 4, 5, 7, 3개이다. 이 수는 할당이 정책적으로 이루어진다는 점을 고려할 때 크게 변하지 않을 것이다. 만약 모든 할당되지 않은 프리픽스까지를 고려한다 할지라도 8, 8, 16, 17개이다. 따라서 prefix length 필드는 그룹 4의 하나 정도를 무시하고 4bit로 표현한다. 프리픽스 자체를 저장하는 prefix 필드는 가장 긴 프리픽스를 고려해야 하므로 각 그룹별로 23, 31, 47, 64bit로 표현한다. 이렇게 구성된 로드의 폭(width)은 이상적인 수치이다. 각 그룹별 테이블은 별도의 메모리 모듈에 구성되며 메모리 모듈의 폭은 byte단위이다. 제안하는 구조에서는 한 번의 메모리 접근으로 구성을 하기 위해 버켓 단위가 아닌 로드 단위를 byte단위로 확장한다. 결과적으로 각 그룹별로 한 로드의 폭은 5, 6, 8, 10byte가 된다.

버켓 당 로드 수가 아주 많다면 오버플로우 없이 충돌을 충분히 지원할 수 있다. 하지만 필요한 메모리가 많아지며 메모리 효율이 떨어진다. 만일 똑같은 메모리 효율일 때 로드 수의 증가는 결국 해쉬 인덱스 길이가 줄어드는 것을 의미하므로 오히려 충돌을 많이 발생하게 될 것이다. 반면에 버켓 당 로드 수가 아주 적다면 각 버켓 당 충돌의 영향이 민감해져서 버켓이 가득 차는 오버플로우가 많이 일어날 것이다. 본 논문에서는 해싱을 가변 적용한 것과 마찬가지로 그룹별로 로드 수도 가변 적용하였다. 이를 통해 모든 테이블은 1/2로 동일한 메모리 효율을 유지한다. 동일 메모리 효율을 가정할 때, 그룹 1, 2는 전체 엔트리 수가 적기 때문에 해쉬 인덱스 비트 수도 작게 하고 로드 수를 많이 하는 것이 더 효율적이고 그룹 3, 4는 전체 엔트리 수가 많기 때문에 다수의 해싱에 의해 충분히 엔트리가 분산될 수 있으므로 그에 비해 로드 수를 더 작게 하는 것이 효율적이다. 즉 적용하는 해쉬 함수가 많으면 그만큼 로드 수는 적게 할당하여 전체적인 메모리 효율에는 변화가 없도록 하였다. 결과적으로 버켓 당 그룹 1, 2는 2개, 그룹 3, 4는 1개의 로드를 할당하였다.

해쉬 인덱스 비트 수는 다음과 같이 결정하였다. 해쉬 인덱스 비트 수는 다음과 같이 결정하였다. 각 그룹 x별로 저장해야 할 전체 프리픽스 개수가 N_x, 테이블의 수를 T_x, 테이블 당 버켓의 수를 B_x, 버켓 당 로드의 수를 L_x이라 하면, 메모리 효율은

$$\text{메모리 효율} = \frac{N_x}{B_x \cdot L_x \cdot T_x} \tag{1}$$

로 나타낼 수 있다.

앞서 언급한 바와 같이 엔트리 분석에 따라 가변 적용한 그룹별 해쉬함수 개수와 로드 수를 결정하였으므로 T_1=1, T_2=1, T_3=3, T_4=3 이며 L_1=2, L_2=2, L_3=1, L_4=1 이다. 각 그룹별 테이블 당 버켓의 수, B_x는 메모리 효율을 1/2를 유지하기 위해서 $B_1=N_1$, $B_2=N_2$, $B_3=(2/3)N_3$, $B_4=(2/3)N_4$가 된다. 해쉬 인덱스 비트 수, H_x는 2의 제곱으로 버켓의 수를 나타내므로,

$$H_x = \lceil \log_2 B_x \rceil \tag{2}$$

로 결정된다.

충돌이 중첩되어 각 테이블의 참조된 버켓이 모두 가득 차서 저장할 여유가 없을 때 오버플로우가 발생되며 이 경우에는 Overflow Table에 저장한다. Overflow Table은 TCAM으로 구성하여 한 번의 접근으로 구성 및 검색이 가능하도록 한다. TCAM은 다른 RAM 메모리에 비해 단가가 높지만 본 논문에서 제안하는 복수 해싱의 가변 적용을 통해 오버플로우를 줄임으로써 적은 엔트리만을 수용하면 되므로 큰 부담 없이 구현할 수 있다. 또한 128bit 프리픽스를 위한 NS Table도 TCAM으로 구성하였다. NS Table 역시 엔트리 수가 적기 때문에 큰 부담 없이 구현할 수 있다. 이와 같이 구성된 포워딩 테이블의 메모리 구조는 그림 6과 같으며 한 블록이 하나의 로드를 의미하며 각 그룹별 포워딩 테이블의 버켓 수는 2^{Hx}로 표현할 수 있다.

3.4 제안하는 구성 및 검색 유닛

제안하는 방식을 통해 구성 및 검색 동작을 할 수 있는 주 처리 유닛을 VILU6(Variable IP address Lookup Unit for IPv6)로 명명한다. 하드웨어 도식도는 그림 6과 같다. 그림 6에서의 데이터 폭(width)은 최대 크기를 의미하며 동작 모드와 프리픽스 길이에 따라 그 이하가 된다. VILU6은 시스템 제어신호에 의해 구성동작과 검색동작을 수행하며 크게 입력 신호 및 해쉬 처리부, NS Manager, OF Manager, Priority Encoder 및 출력부와 여러 PEG로 구성된다.

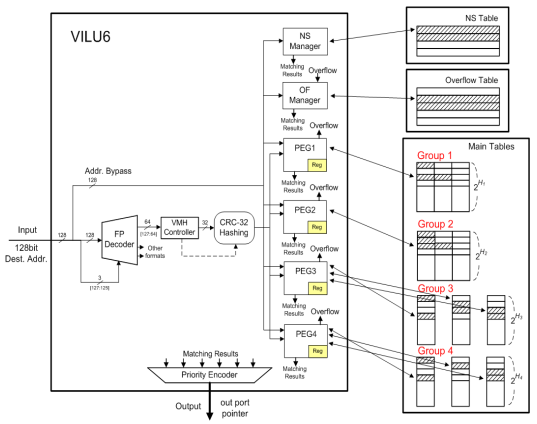

그림 9. 제안하는 주소 검색 구조와 포워딩 테이블의 메모리 구조

3.4.1 구성 동작

입력신호로는 각 패킷의 헤더 정보 중 목적지 주소의 프리픽스를 받는다. VILU6 내부로 입력된 프리픽스는 두 패스로 나누어진다. 하나의 패스로는 프리픽스 값 전체와 관련 포워딩 정보를 bypass하여 모든 PEG와 NS Manager, OF Manager로 보내져 구성할 원본 데이터로 쓰인다. 각 PEG 및 OF Manager에서는 각각 해쉬 인덱스 값 또는 PEG의 overflow 응답이 있을 때 까지 대기한다. 반면 만약 128bit 프리픽스라면 대기 없이 NS Manager에 의해 NS Table에 저장하고 구성 동작을 종료한다.

다른 패스로는 FP Decoder로 입력한다. FP Decoder는 상위 3bit를 제어신호로 사용하며 추후 패킷 포맷에 따른 별도의 구성 및 검색동작을 지원하기 위한 유닛이다. 현재는 FP필드 값이 001인 Aggregatable 글로벌 유니캐스트 포맷만을 지원하며 향후 IPv6의 새로운 포맷에 대해 지원할 예정이다. Aggregatable 글로벌 유니캐스트 포맷의 프리픽스는 VMH Controller로 보내진다. VMH Controller는 각 그룹별로 인덱스 값 추출을 위한 CRC 레지스터의 선택과 CRC-32 해쉬 유닛에서 PEG로 보내는 해쉬 인덱스의 비트 수와 개수를 저장하며 제어할 수 있다. 이 3가지 설정치를 통해 복수 해쉬의 가변적용을 엔트리 변화에 따라 적응성 있게 가능토록 한다. 또한 해당 네트워크 환경 및 트래픽에 따라 더 효율적이도록 가변적일 수도 있다. 가변적으로 설정치를 조정할 수 있도록 수동 모드로 관리자에 의해 세팅 가능토록 하였다. 필요에 따라 라우터 또는 VILU6 내부에서 추가 모니터링 유닛을 통해 입력되는 라우팅 데이터의 엔트리를 분석함으로서 좀 더 적응성 있게 자동적으로 설정치를 최적화할 수도 있다. 그룹 4에 해당하는 33~64bit 프리픽스의 경우에는 상위 16bit를 제외한 나머지 프리픽스 값을 CRC-32로 보낸다. 그 외 32bit 이하 길이의 프리픽스는 해당 그룹의 키 값 길이만큼 CRC-32로 보낸다. 즉 각 그룹별로 16, 24, 32, 32bit의 상위비트를 보낸다.

버퍼링을 통해 한 사이클 당 상위 한 비트 씩 CRC-32 해쉬 유닛에 입력된다. CRC-32 해쉬 유닛에서는 프리픽스의 해당 그룹에서 원하는 사이클만큼 경과한 후 하나 또는 복수의 해쉬 인덱스 값을 CRC 레지스터에서 선택하여 해당 PEG로 보낸다. 해당 PEG에는 이렇게 구해진 복수 인덱스 값이 참조하는 버켓의 빈 로드 수를 레지스터에 저장하고 있다. 이를 통해 참조된 버켓 중 더 빈 로드가 많은 버켓을 선택하여 bypass하여 대기 중이던 프리픽스 원본 데이터와 관련 포워딩 정보를 해당 버켓의 빈 로드에 저장한다. 이렇게 구성 동작이 종료되면 OF Manager에 오버플로우가 발생하지 않았다는 응답을 한다. 만약 저장할 빈 로드가 없다면 OF Manager에 오버플로우 발생 응답을 한다. 오버플로우가 발생하면 OF Manager는 대기 중이던 원본 프리픽스 데이터를 Overflow Table에 저장하고 종료한다. 이와 같이 구성 동작은 로드 단위의 메모리 접근 덕분에 한 번의 메모리 접근으로 완료할 수 있다.

예를 들어 2001::/16, 2001:250:202::/48, 2001:250:20b::/48, 2001:250:202:1::1/128 프리픽스를 순서대로 구성한다고 가정하자. 먼저 2002::/16 프리픽스가 입력되면 일단 2002(16) 데이터 부분은 bypass되어 각 PEG에서 대기한다. FP Decoder를 거쳐 VMH Controller에서는 입력된 프리픽스가 그룹 1에 속하므로 16bit 데이터 전체를 해쉬 유닛으로 입력하며 제어 신호로써 16 사이클 후에 지정한 해쉬 인덱스 길이만큼을 CRC 레지스터에서 뽑도록

한다. 여기서는 3bit 길이의 4(100(2))로 가장한다. 그룹 1에 해당하는 PEG1에서는 주소 4에 해당하는 버켓의 로드 수를 확인한다. 여유 로드가 있으면 해당 주소의 포워딩 테이블에 저장하고 꽉 차 있으면 Overflow Table에 저장한다. 2001:250:202::/48와 2001:250:20b::/48의 경우도 유사하게 진행되며 해쉬 유닛에는 프리픽스의 앞부분 16 bit를 제외한 32 bit만을 해쉬 키로 사용하고 32 사이클 후에 각각 다른 위치의 레지스터를 선택하여 3개씩 해쉬 인덱스 값을 뽑는다. 추출된 해쉬 인덱스 값이 5 bit 길이의 5, 8, 12와 3, 8, 10라 가정하고 첫 번째 테이블의 주소 3, 5가 차 있다고 가정하자. 그러면 2001:250:202::/48은 두 번째 테이블의 주소 8에 저장된다. 다음에 들어 온 2001:250:20b::/48은 첫 번째, 두 번째 테이블의 해쉬 인덱스가 가리키는 버켓이 모두 차 있으므로 세 번째 테이블의 주소 12에 저장된다. 다음으로 2001:250:0:1::1/128이 입력된다면 앞 선 프리픽스들과 달리 NS Manager에 의해 NS Table에 바로 저장하고 구성 동작을 완료한다.

3.4.2 검색 동작

데이터 흐름은 구성 동작과 거의 일치 한다. 입력신호로는 128bit 목적지 주소 전체를 받는다. 입력된 주소는 역시 두 패스로 나누어진다. 하나 패스로는 주소 값 전체를 bypass하여 모든 PEG와 NS Manager, OF Manager로 보내지며 역시 대기한다. 다른 패스로는 구성동작과 마찬가지로 FP Decoder로 보내지며 다양한 패킷 포맷을 지원하기 위해 1차적인 디코딩을 한다. 이 후 예상 프리픽스 길이는 16~64bit 이므로 전체 128bit 주소 중 상위 64bit만을 VMH Controller로 보낸다. VMH Controller는 설정치에 따라 CRC-32 해쉬 유닛을 제어한다. CRC-32 해쉬 유닛에는 버퍼링을 통해 한 사이클 당 상위 한 비트 씩 입력된다. 각 그룹별로 필요한 16, 24, 32 사이클 후에 PEG1, PEG2, PEG3이 필요로 하는 복수 인덱스 값을 CRC 레지스터에서 추출한다. PEG4에서는 17~48bit의 32bit를 해쉬 키로 사용한 인덱스 값을 32 사이클 후에 추출한다.

각 PEG는 복수의 해쉬 인덱스 값을 입력 받아 각 인덱스가 참조하는 각 테이블의 버켓의 모든 데이터를 읽어 들인다. PEG 내부에서 읽어 들인 버켓의 모든 데이터를 각 로드 단위로 나누어 bypass되어 대기 중인 원본 데이터와 해당 길이만큼 매치 작업을 수행한다. 매치되어 검색된 결과는 다수의 PEG에서 있을 수 있다. 이 모든 결과는 Priority Encoder로 보낸다. 아울러 NS Manager와 OF Manager를 통해 NS Table과 Overflow Table도 검색한다. 이 테이블은 TCAM으로 구성되어 있기 때문에 PEG와 병렬적으로 검색할 수 있다. 이렇게 매치된 결과도 Priority Encoder로 보낸다.

Priority Encoder는 각 그룹별 PEG와 NS Manager, OF Manager에서 보내 온 모든 매치 결과 중에 가장 긴 프리픽스 길이를 갖는 결과 값을 선택한다. 즉 LPM를 수행한다. 그 결과 해당하는 out port pointer를 최종 출력한다. 만약 매치된 결과가 전혀 없다면 기본 설정된 out port pointer를 출력한다. 이상에서와 같이 검색 동작도 구성 동작과 마찬가지로 단 한 번의 메모리 접근으로 완료할 수 있다.

앞 서 예로 들은 2001::/16, 2001:250:202::/48, 2001:250:20b::/48, 2001:250:202:1::1/128 프리픽스가 포워딩 테이블에 구성되어 있고 목적지 주소가 2001:250:202:0:0:0:0:0인 패킷이 들어왔다고 가정하자. VMH Controller를 거쳐 상위 64bit인 2001:250:202:0 만이 한 사이클 마다 한 비트 씩 해쉬 유닛에 입력이 되면서 16 사이클 후에는 PEG1에 대한 인덱스가, 23 사이틀 후에는 PEG2, 32 사이클 후에는 PEG3에 대한 인덱스가 출력되며 추가로 16 사이클 후에 PEG4에 대한 인덱스가 출력된다. 이를 각 PEG에 보내어 해당 인덱스가 가리키는 모든 로드의 저장된 엔트리와 비교한다. 이와 함께 NS Table과 OF Table에서도 매칭이 이루어진다. 가정한 예에서는 2001::/16, 2001:250:202::/48이 매칭이 되므로 Priority Encoder에서는 2001:250:202::/48를 최종적인 LPM 결과로 out port pointer를 출력하고 검색동작을 완료한다.

3.5 추가 업데이트의 용이성

제안한 구조는 새로운 라우팅 정보가 들어 왔을 때 추가 업데이트 작업이 용이하다. 추가 업데이트 동작은 구성동작과 동일한 과정에 따라 동일한 구조에서 가능하기 때문이다. 따라서 기존의 방식에서 문제가 되었던 전체 테이블을 재구성하거나 이중의 포워딩 테이블을 구성할 필요가 없다. 추가 업데이트 작업은 자주 발생하는 작업이 아니고 검색동작에 비해 중요한 동작은 아니지만 만약 검색과정과 독립적, 병렬적으로 추가 업데이트 작업을 수행하기 원한다면 구조의 변경 없이 CRC-32 해쉬 유닛을 하나 더 추가해 주고 PEG의 구성과 검색 동작을 병렬로 구현하면 간단히 해결된다. 따라서 이에 따

른 하드웨어의 구조도 크게 변경하지 않고 구현할 수 있다. 또한 추가 업데이트해야 하는 엔트리의 양이 시스템을 처음 구현했을 때 고려했던 양보다 많다고 할지라도 각 그룹의 해쉬 인덱스 길이, 각 그룹별 해싱 개수 등을 VMH Controller의 엔트리 분석에 따른 재설정이 가능하므로 항상 최적화된 성능을 유지할 수 있다. 효과적인 포워딩 테이블의 유지를 위한 삭제 작업도 valid, hit필드의 값을 통해 쉽게 수행할 수 있으며 PEG 내부 레지스터의 값을 조정하는 작업만이 필요하다. 따라서 제안하는 구조는 포워딩 테이블의 구성 및 검색뿐만 아니라 유지 및 추가 업데이트에서도 용이성을 가지고 있으며 특히 엔트리 변화가 심한 BGP(Border Gateway Protocol)을 사용하는 에지(edge) 라우터에서도 가변적인 네트워크 환경에 적응성 있게 높은 성능을 유지할 수 있다.

Ⅳ. 실험 결과

본 논문의 실험에서 사용한 데이터는 실제 6bone에서 제공하는 2006년 1월 10일 CERNET 라우팅 데이터와 non-random하게 생성된 데이터를 이용하였다. CERNET 라우팅 데이터는 전체 엔트리 수가 848개이며 Aggregatable 글로벌 유니캐스트 주소 포맷으로만 할당되어 있다[20]. 실험은 기본 메모리 조건을 같게 하기 위해 메모리 효율을 1/2에 근사하게 하였으며 수식 (2)에 따라 해쉬 인덱스 비트 수를 결정하였다. 또한 메모리 모듈 수에 따른 오버플로우 발생의 전제조건을 갖게 하기 위해 모든 경우에 포워딩 테이블은 8개로 동일하게 하였다. 표 2는 균일하게 해싱 적용된 구조와 제안하는 구조를 비교실험한 결과이다. 향후 IPv6 주소는 계층적 구조를 기반으로 예외적인 할당이 있을 수도 있으나 분포 비율은 크게 변하지 않을 것으로 예상된다. 즉 32, 48 bit 프리픽스의 엔트리가 상대적으로 매우 많으므로 이에 따라 그룹 3, 4의 엔트리가 그룹 1, 2 보다 상대적으로 많다.

따라서 제안하는 분석을 근거로 그룹 3, 4에 해싱을 가변 적용하였으며 이를 기존에 제시된 균일하게 해싱 적용하는 방식과 비교실험하였다. 비교실험의 첫 번째 경우는 그룹 3만을 가변 적용한 것이며 두 번째 경우는 그룹 3, 4 모두 가변 적용한 결과이다. 표 2의 결과에서 알 수 있듯이 균일하게 복수 해싱을 적용한 것보다는 해싱을 가변 적용한 것이 더 좋은 성능을 보였으며 그룹 3만을 가변 적용한 경우 보다 그룹 3, 4를 모두 가변 적용한 경우에 더 좋은 성능을 보이는 것을 알 수 있었다. 이와 같은 성능의 향상은 가변 적용에 따라 해싱이 늘어난 그룹의 오버플로우 수는 감소된 반면에 해싱이 감소한 그룹은 오버플로우 수의 증가가 없었기 때문이다.

표 2. 실제 라우팅 데이터를 이용한 비교실험 결과

구조 / 비교내용	기존 균일한 해싱 적용[14] (Double Hashing)	제안하는 검색구조 (그룹 3 가변적용)	제안하는 검색구조 (그룹 3,4 가변적용)
포워딩테이블 수	8		
그룹별 해싱 수	2 / 2 / 2 / 2	1 / 2 / 3 / 2	1 / 1 / 3 / 3
그룹별 로드 수	1 / 1 / 1 / 1	2 / 1 / 1 / 1	2 / 2 / 1 / 1
그룹별 해쉬 인덱스 비트 수	3 / 6 / 10 / 7	3 / 6 / 9 / 7	3 / 6 / 9 / 7
메모리크기(KB)	19.33	15.33	16.58
그룹별 오버플로우 수	1 / 24 / 15 / 35	1 / 24 / 8 / 35	1 / 23 / 8 / 5
전체 오버플로우 수	75	68	37
오버플로우 비율	8.8 %	8 %	4.3 %

그런데 그룹 1, 3, 4에 비해 상대적으로 그룹 2의 오버플로우가 많은 것을 볼 수 있다. 이는 그룹 2의 해싱 수를 증가하거나 메모리를 많이 할당하여도 크게 감소되지 않았다. 그 이유는 그룹 2의 해쉬 키 값으로 사용되는 프리픽스의 상위 비트가 앞 서 설명한 바와 같이 동일하기 때문이다. 즉 그룹 2의 24bit 키 값 중에서 16bit 이상의 상위 비트가 일정하기 때문에 충돌의 분산에 전혀 도움이 되지 않았으며 실제로 하위 8bit만이 해쉬 키 값으로의 효용가치가 있었기 때문이다. 이러한 현상은 IPv4 주소 체계와는 다른 IPv6 주소 체계의 특징적인 점이며, 또 다른 문제점이라 할 수 있다. 따라서 IPv6 주소 검색 구조에서는 그룹별로 고정된 해쉬 키 값보다는 프리픽스 길이 별로 해쉬 키 값을 입력해 주는 것이 더 유용할 것이다. 별도의 실험을 통해 그룹별 포워딩 테이블을 구성하고 각 프리픽스 길이별로 해싱을 한 결과, 세 실험 모두에서 그룹 2의 오버플로우는 발생하지 않았다. 따라서 제안하는 구조는 IPv6 주소 체계에 대하여 세 번째 실험과 같이 해쉬 함수의 개수는 각 그룹별로 1, 1, 3, 3개씩 할당하고 이에 따라 각 그룹별 테이블의 로드 수는 2, 2, 1, 1개씩 하는 것이 유용하다.

앞 서 언급한 바와 같이 하드웨어 기반의 IP 주소 검색 구조는 메모리 접근 회수에 의해 검색속도 성능이 좌우된다. 따라서 기존의 연구와 메모리 접근 회수를 비교해 봄으로써 제안하는 구조의 검색 시간을 비교, 분석할 수 있으며 표 3과 같다. 여기서 최대 메모리 접근 회수가 worst case complexity가 된다. 비교 결과에서 알 수 있듯이 제안하는 구조는 TCAM기반 구조와 균일하게 복수해쉬을 적용한 구조와 함께 최소, 최대 경우 모두 단 한 번의 메모리 접근으로 검색 동작을 수행하는 고속 처리 능력을 보여 주었다. 같은 검색 성능을 갖는 TCAM 기반 구조와 균일하게 복수해쉬을 적용한 구조에 비해서도 필요한 SRAM과 TCAM의 크기가 더 작으므로 메모리 효율 측면에서도 제안하는 구조의 성능이 더 뛰어나다고 할 수 있다.

표 4는 기존의 해쉬 함수 기반의 주소 검색 구조와 제안하는 구조의 구성 동작 시 메모리 접근 회수를 비교한 것이다. 기존의 검색 구조는 버켓 단위로 접근이 이루어지므로 구성 동작 시 해당 버켓의 데이터를 일단 읽은 다음 처리하고 다시 테이블에 쓰는 동작이 필요하였다. 이에 비해 제안하는 구조는 각 버켓의 유효 로드 수를 레지스터에 저장하고 있고 로드 단위로 메모리에 접근이 가능하기 때문에 바로 쓰기 동작을 통해 해당 로드에 구성할 수 있다. 따라서 제안하는 구조는 최소, 최대 모든 경우에 대하여 한 번의 메모리 접근으로 테이블을 구성할 수 있다.

표 3. 검색동작 시 메모리 접근 회수 비교

주소 검색 구조	검색동작 시 메모리 접근 회수 (최소 / 최대)
Range search[9]	1 / 16
TCAM-based[10]	1 / 1
단일 해쉬 적용[12]	1 / 5
복수 해쉬 및 그룹화[13]	1 / 1
Partitioning[16]	1 / 3
제안하는 구조	1 / 1

표 4. 구성동작 시 메모리 접근 회수 비교

주소 검색 구조	구성 동작 시 메모리 접근 회수 (최소 / 최대)
단일 해쉬 적용[12]	2 / 2
복수 해쉬 및 그룹화[13]	2 / 2
제안하는 구조	1 / 1

CERNET 라우팅 데이터는 실제 할당된 IPv6 주소로 구성된 포워딩 테이블 중에 가장 많은 엔트리 수를 가지고 있다. 그럼에도 불구하고 아직 IPv6 주소 체계가 널리 할당되어 사용되고 있지 않기 때문에 약 800개 정도 밖에 되지 않는다. 따라서 이 데이터는 향후 IPv6 주소 체계의 폭발적인 엔트리 수 증가를 반영하지는 못한다. 공인된 testbench는 없으나 현재 IPv6 주소 분포와 할당 정책을 반영하여 향후 데이터 분포를 예측한 testbench의 생성이 필요하다. 기존 연구[21]에서 각 프리픽스 길이별로 random하게 생성된 데이터는 실제 분포 특성이 전혀 없기 때문에 신뢰성이 떨어지므로 여러 non-random 생성 방식이 제안되었다. 이 방식들은 독창적이긴 하지만 IPv4 또는 AS Number에 기반하므로 오히려 작의적인 면이 있다. 본 논문에서는 기존 방식 중에서 가장 신뢰성이 높다고 판단되는 특정 프리픽스 길이에 가중치를 주어 random하게 생성함으로서 전체적으로 non-random하게 구성하는 방식[1]을 사용하였으며 기존 방식에서 48, 64 bit에 지나치게 치우치는 단점을 보완하기 위해 32 bit에 대해서도 5% 정도 가중치를 주었다. 이 가중치는 결국 전체 엔트리 중 해당 프리픽스의 비율이 된다. 이러한 비율에 따라 10k, 20k, 30k, 40k, 50k 엔트리를 생성하였다. 이를 기반으로 제안하는 구조의 엔트리 수 증가에 따라 필요한 메모리 크기를 비교하였다. 필요한 메모리는 포워딩 테이블 구성을 위한 SRAM과 오버플로우 처리를 위한 TCAM으로 구분할 수 있으며 그 결과는 그림 7, 8과 같다. 그림 7에서 알 수 있듯이 엔트리 수가 증가함에 따라 기존의 균일하게 이중 해쉬을 적용한 구조에 비해서 제안한 가변적으로 적용한 구조의 오버플로우 수가 덜 발생하는 것을 알 수 있다. 따라서 오버플로우 처리를 위해 필요한 TCAM의 크기도 훨씬 덜 필요한 것을 알 수 있다. 그림 8에서는 엔트리 수 증가에 따라 포워딩 테이블을 위한 메모리 크기를 비교하였다. 역시 제안하는 구조의 필요한 메모리 크기가 더 적은 것을 확인 할 수 있었다.

이상에서 비교, 실험한 결과와 같이 본 논문에서 제안하는 복수 해쉬의 가변 적용에 의한 IPv6 구성 및 검색 구조는 기존의 균일한 복수 해쉬을 적용한 방법에 비해 메모리 효율이 우수하며 적절한 개수의 메모리 모듈을 사용하여 더 효율적인 충돌의 분산을 이룰 수 있었고 이에 따라 오버플로우를 줄일 수 있었다. 더욱이 제안하는 구조는 향후 IPv6주소 체계가 널리 사용되는 시점에서 엔트리 수 증가에

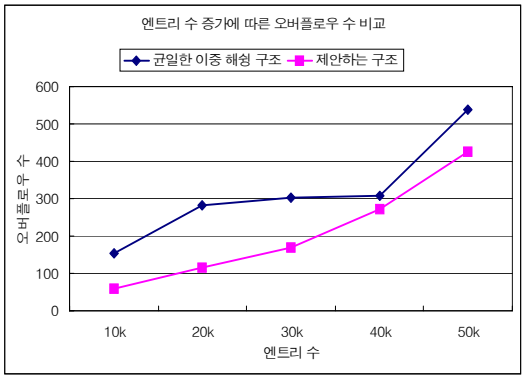

그림 10. 엔트리 수 증가에 따른 오버플로우 수 비교

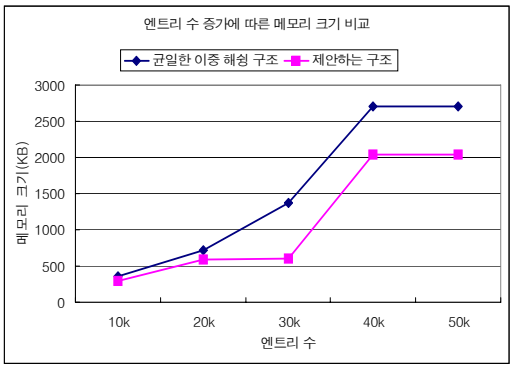

그림 11. 엔트리 수 증가에 따른 메모리 크기 비교

따라 더 효율적인 메모리 공간으로 포워딩 테이블을 구성할 수 있을 것이다. 뿐만 아니라 한 번의 메모리 접근으로 빠르게 포워딩 테이블의 구성 및 검색이 가능하기 때문에 차세대 라우터에서 유용하게 사용될 수 있을 것이다.

V. 결 론

복수 해쉬 함수 기반의 IP 주소 검색 구조는 적용하는 해쉬 함수의 개수에 따른 테이블 수와 메모리의 크기가 많아질수록 충돌을 줄일 수 있으나 그만큼 메모리 효율은 낮아진다. 따라서 최대한 테이블 수와 메모리 용량을 줄이면서도 충돌을 최소화할 수 있어야 한다. 본 논문에서는 빠른 포워딩 테이블 구성 및 검색이 가능하고 메모리 효율이 높은 IPv6 주소 검색 구조를 제안하였다. IPv6 주소 할당 정책을 고려하여 프리픽스를 길이별로 그룹화하고 그룹별로 해쉬을 가변 적용하여 해쉬함수 기반의 주소 검색에서 문제되는 충돌의 발생 가능성을 적절한 개수의 메모리 모듈에서도 충분히 감소시켰다. 이를 통해 오버플로우 처리를 위한 TCAM의 크기를 줄이고 전체 포워딩 테이블 구성을 위한 메모리 크기를 줄일 수 있었다. 또한 메모리의 버켓 및 로드 구조를 개선하여 단 한 번의 메모리 접근으로 구성 및 검색이 가능하며 추가 업데이트도 용이하였다. 이렇게 제안된 방식을 바탕으로 완전한 하드웨어 구조로서 VILU6을 제안하고 구성 및 검색동작을 설명하였다. 더욱이 제안된 VILU6은 엔트리 분포의 변화에 따라 적응적으로 그룹별 해쉬함수의 개수를 조절하고 해쉬 인덱스 길이를 조절할 수 있도록 재설정 가능하기 때문에 엔트리 변화가 심한 에지(edge) 라우터의 최고 성능을 유지할 수 있게 한다. 그러므로 제안하는 글로벌 IPv6 유니캐스트 주소 검색 구조는 미래의 다이나믹한 네트워크 환경에서 초고속 차세대 라우터의 검색 구조로 유용하다.

참 고 문 헌

[1] X. Sun, and Y. Q. Zhao, "An On-Chip IP Address Lookup Algorithm," *IEEE Transactions on Computers*, Vol.54, No.7, pp.873-885, July 2005.

[2] D. Bemmann, "IP Lookup on a Platform FPGA: a Comparative Study," *Proc. of IEEE International Parallel and Distributed Processing Symposium*, p.166, April 2005.

[3] H. Lim, J. Seo, and Y. Jung, "High Speed IP Address Lookup Architecture Using Hashing," *IEEE Communications Letters*, Vol.7, No.10, pp.502-504, Oct. 2003.

[4] Y. Chu, P. Lin, J. Lin, H. Su, and M. Chen, "ASIC Design of Fast IP Lookup for Next Generation IP Router," *Proc. of IEEE International Symposium on Circuits and Systems*, pp.3825-3828, May 2005.

[5] S. Deering, and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *RFC 2460*, Dec. 1998.

[6] H. J. Chao, "Next generation routers," *Proc. of the IEEE*, Vol.90, pp.1518-1558, Sept. 2002.

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

[8] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, "Scalable high speed IP routing lookups," *Proc. of ACM Special Interest*

Group on Data Communications, pp.25-36, Sept. 1997.
〔9〕 B. Lampson, V. Srinivasan, and G. Varghese, “IP Lookups Using Multiway and Multicolumn Search,” *Proc. of IEEE International Conference on Computer Communications*, pp.1248-1256, April 1998.
〔10〕 A. McAuley and P. Francis, “Fast Routing Table Lookup Using CAMs,” *Proc. of IEEE International Conference on Computer Communications*, pp.1382-1391, March 1993.
〔11〕 A. Broder and M. Mitzenmacher, “Using multiple hash functions to improve IP lookups,” *Proc. of IEEE International Conference on Computer Communications*, pp.1454-1463, 2001.
〔12〕 H. Lim and Y. Jung, “A Parallel Multiple Hashing Architecture for IP Address Lookup,” *Proc. of IEEE Workshop on High Performance Switching and Routing*, pp. 91-95, 2004.
〔13〕 김혜란, 정여진, 임창훈, 임혜숙, “프리픽스 그룹화를 이용한 병렬 복수해싱 IP 주소검색구조,” *한국통신학회 논문지*, Vol.30, No.3B, pp.65-72, March 2005.
〔14〕 R. Hinden, “Proposed TLA and NLA Assignment Rules,” *RFC 2450*, Dec. 1998.
〔15〕 B. Al-Khaffaf, E. Karuppiah, and R. Abdulah, “Efficient partition based IPv6 lookup algorithm for packet forwarding,” *Proc. of Asia-Pacific Conference on Communications*, Vol.1, pp.238-242, Sept. 2003.
〔16〕 S. Yong, and H. Ewe, “Robust routing table design for IPv6 lookup,” *Proc. of International Conference on Information Technology and Applications*, Vol.1, pp.531-536, July 2005.
〔17〕 R. Fink and R. Hinden, “6bone (IPv6 Testing Address Allocation) Phaseout,” *RFC 3701*, Mar. 2004.
〔18〕 Regional Internet Registeries(APNIC, ARIN, RIPE NCC), “IPv6 Address Allocation and Assignment Policy,” *RIPE 267*, http://www.ripe.net/ipv6/ipv6allocs.html, January 2003.
〔19〕 R. Jain, “A Comparison of Hashing Schemes for Address Lookup in Computer Networks,” *IEEE Transactions on Communications*, Vol.40, No.3, pp.1570-1573, Oct. 1992.
〔20〕 Internet Engineering Task Force(IETF), “6bone: testbed for deployment of IPv6,” http://www.6bone.net/
〔21〕 M. Wang, S. Deering, T. Hain, and L. Dunn, “Non-random generator for IPv6 tables,” *Proc. of IEEE Symposium on High Performance Interconnects*, pp.35-40, Aug. 2004.

박 현 태 (Hyuntae Park) 준회원

2006년 2월 연세대학교 전기전자공학과 학사
2006년 2월 연세대학교 전기전자공학과 석사
2006년 3월~현재 연세대학교 전기전자공학과 박사과정
<관심분야> 고속네트워크 관련 SoC설계 및 네트워크 프로세서 설계

문 병 인 (Byung In Moon) 정회원

1995년 2월 연세대학교 전자공학과 학사
1997년 2월 연세대학교 전자공학과 석사
2002년 2월 연세대학교 전기전자공학과 박사
2002년~2004년 하이닉스반도체 선임연구원
2004년~2005년 연세대학교 연구교수
2005년~현재 경북대학교 전자전기컴퓨터학부 전임강사
<관심분야> 디지털 집적회로 설계, SoC 설계, 마이크로프로세서 구조 설계

강 성 호 (Sungho Kang) 정회원

1986년 2월 서울대학교 제어계측공학과 학사
1988년 6월 The University of Texas, Austin 전기 및 컴퓨터공학과 석사
1992년 6월 The University of Texas, Austin 전기 및 컴퓨터공학과 박사
1989년~1992년 미국 Schlumberger Inc. 연구원
1992년 The Univ. of Texas at Austin, Post Doctoral Fellow
1992년~1994년 Motorola Inc. 선임 연구원
1994년~현재 연세대학교 전기전자공학과 교수
<관심분야> SoC 설계 및 응용, DFT, SoC 테스트