기가비트 이더넷 스위치에서 빠른 MAC 주소 테이블의 검색 방법

이 승 왕 , 박 인 철
韓國科學技術院
wang@duo.kaist.ac.kr, icpark@eekaist.kaist.ac.kr

Practical MAC address table lookup scheme for gigabit ethernet switch

S. W. Lee , I. C. Park
Korea Advanced Institute of Science and Technology

Abstract

As we know, gigabit ethernet is a new technology to be substituted for current fast ethernet used widely in local area network. The switch used in gigabit ethernet should deal with frames in giga-bps. To do such a fast switching, we need that several processes meet the budgets, such as MAC address table lookup, several giga speed path setup, fast scheduling, and etc. Especially MAC address table lookup has to be processed in the same speed with speed of incoming packets, thus the bottleneck in the process can cause packet loss by the overflow in the input buffer.

We devise new practical hardware hashing method to perform fast table lookup by minimizing the number of external memory access and accelerating with hardware.

I. 서론

기가비트 이더넷(gigabit ethernet)은 최근 몇 년 사이에 등장하여 급격하게 표준화가 이루어지며 발전하고 있는 기술인데, 이는 기존의 고속 이더넷의 100Mbps 를 훨씬 뛰어 넘는 1000Mbps 의 속도를 가지는 이더넷을 구현하는 것을 목표로 하며 일부 상용 제품이 이미 출시가 되고 있는 상황이다.

기가비트 이더넷 스위치는 초당 기가비트로 들어오는 프레임을 처리할 수 있어야 한다. 즉, 각각의 프레임에 대해 MAC 주소를 찾아서 전송할 포트를 결정하고 이와 같이 각각의 포트에 들어온 프레임들을 수 기가비트의 속도로 스위칭해서 출력 포트로 내보내게 된다.

이와 같은 과정에서 병목현상을 일으킬 수 있는 부분은 첫 단에서 MAC 주소를 검색하는 부분과 큐를 관리하여 스케줄링하는 부분, 그리고 고속의 스위칭을 하는 부분으로 간단히 생각할 수 있는데 이중 MAC 주소 검색부분을 고속으로 하여 전체 성능 향상에 기여하고자 한다.

MAC 주소 테이블은 각 주소별로 출력포트의 번호와 Virtual LAN 이나 멀티캐스트 등의 발송처리(forwarding process)를 위한 여러 가지 정보를 가지고 있는 데, 이 테이블의 검색은 주로 해싱(hashing)을 이용한다. 이는 검색 공간이 48bit 로 큰 반면 실제 존재하는 데이터는 이 보다 훨씬 적은 16bit 이기 때문이다. 해싱에 관해서는 [1]에 기본적인 해싱의 개념과 해싱 함수의 분류에 대하여 나와 있으며 [2]에서는 해싱을 하드웨어적으로 구현하여 마이크로프로세서 TLB 의 구현에 사용하였다.

II. 본론

a. 기가비트 스위치의 구조

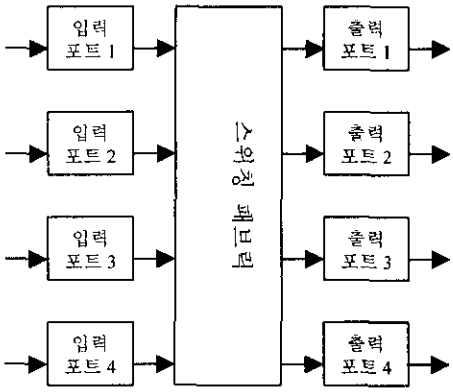

그림 1. 기가비트 이더넷 스위치의 일반적인 구조

기가비트 이더넷은 프레임이 초당 기가비트의 속도로 들어오기 때문에 트래픽이 ~2Mpacket/sec 정도로 각각의 프레임에 대해 500ns 이내에 모든 처리를 해 주어야만 최악의 경우에도 프레임의 분실 없이 처리할 수 있게 된다.

우선 기가비트 이더넷 스위치의 일반적인 구조를 살펴보면 그림 1과 같이 포트 제어기와 스위칭 패브릭(switching fabric)의 두 부분으로 크게 나눌 수 있다. 이 같은 구조는 고속으로 가면서 출력포트를 결정하는 부분과 실제 스위칭부분을 분리함으로써 성능향상을 꾀하고자 하기 위함이다.

포트 제어기의 역할은 MAC 주소 테이블의 구축과 이것의 검색 그리고 입력 또는 출력 큐의 구축과 이를 통한 스케줄링을 담당하는 부분이며, 테이블을 저장하기 위한 메모리를 가지고 있다. 이 메모리는 여러 포트 제어기간에 공유되는 구조와 각각 독립적으로 동작하는 구조의 두 가지로 크게 나뉠 수 있으나 여기서는 독립적인 메모리를 가지는 경우만 살펴보도록 한다.

앞에서 각 프레임당 최대 500ns 가 주어져 있다고 하나 각 단계에서 소모하는 시간이 많으므로 실제로 테이블 검색에 할당되는 부분은 많지 않으므로 이것을 최소로 줄여야 전체 시간을 만족시킬 수 있다.

새로운 해싱 방법을 제시하기에 앞서 기존의 해싱에 대해 간단히 살펴보면 그림 2와 같이 해시 키를 만드는 부분과 이를 이용하여 연결 리스트를 탐색하는

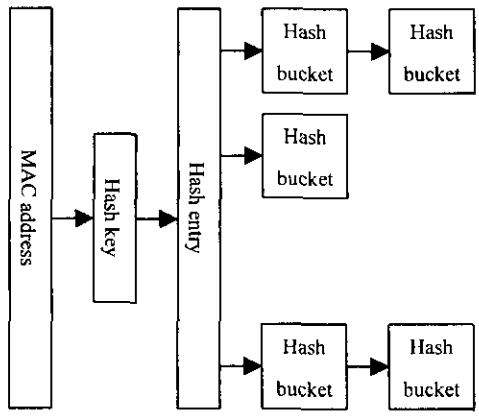

그림 2. 기존 해싱 방법

부분으로 나눌 수 있다. 이 같은 해싱을 실제 스위치에서 구현하려면 해시 엔트리, 해시 버킷(hash bucket)을 읽어 들일 때에 모두 외부 메모리 억세스를 수행해야 하고 외부 메모리에서 데이터를 읽어 들이는 시간은 내부적인 동작보다 오래 걸리므로 결과적으로 많은 시간을 요하게 된다. 그래서 이 논문에서는 이 메모리 접근 회수를 줄이는 데 역점을 두었다.

새로운 방법은 파셜 매칭을 통한 미스의 조기 발견과 연결 리스트를 단축하여 탐색 속도를 빠르게 하는 데 역점을 두었다.

b. 파셜 매칭(partial matching)

파셜 매칭이란 해싱을 할 때에 비교할 대상을 전부 비교하지 않고 그 일부만을 비교하여 가부를 판단한 다음 마지막으로 실제 값으로 비교를 하여 최종적인 판단을 하는 방식이다. 이 방식의 장점은 적은 비교만으로 미스를 조기에 발견할 수 있다는 장점이 있으나 최종적인 비교를 한번 더 해야 한다는 과부하가 존재

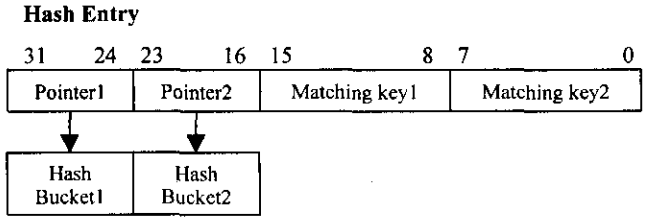

그림 3. 제안된 해싱에서 엔트리의 구조

하는 단점도 있기 때문에 조심스럽게 사용되어야 한다.

본 논문에서는 16bit 의 해시 키(hash key)로 엔트리를 구한다음 8bit 로 파셜 매칭을 수행하는 방식을 사용한다. 이것은 그림 3과 같이 해시 엔트리에 파셜 매칭할 키를 넣어 두고 엔트리를 읽어 올 때 같이 읽히도록 만들어서 메모리 접근 수를 줄이도록 하였다.

이 방법은 또한 연결 리스트를 사용하지 않고 해싱을 수행할 수 있게 하는 장점이 있다. 즉, 그림 3과 같이 해시 엔트리에 파셜 키(parial key)와 포인터를 동시에 둠으로써 매칭 시 두 키를 동시에 비교하고 그 중 일치하는 것의 포인터를 반환하면 해싱이 완료되므로 단 한번의 메모리 접근만으로도 해싱이 가능해진다. 그리고 두 해시 버킷 모두에 존재하지 않는 경우도 기존 방법에서 모든 연결 리스트를 탐색하지 않고 해시 엔트리만을 읽어옴으로써 구현할 수 있다. 단, 차후의 MAC 주소의 비교를 위해 두 번의 메모리 읽기는 필요하다.

그러나 새로운 방법에서는 파셜 키와 포인터를 각각 두개씩 한 워드에 포함시키기 위해서 포인터의 크기를 한 바이트로 제한하였다. 이렇게 하고 전체 해시 버킷을 256 개씩 다발을 지음으로써 해시 버킷의 어드레싱을 해결하였으나 이로 인해 약간의 성능저하가 발생한다.

또한 이런 방식으로 두개의 키를 동시에 비교하기 위해서는 하드웨어로 구현하는 것이 간단하면서도 전체 시스템의 성능향상에 도움이 된다.

c. 시뮬레이션 결과

시뮬레이션은 64k 개의 엔트리와 64k 개의 해시 버킷을 가지는 경우에 대하여 MAC 주소의 수가 증가함에 따라 평균 검색 속도를 비교하였다. 그리고 비교의 공정성을 위하여 새로운 방식과 같이 기존 방식의 연결 리스트길이를 2 로 제한하고 그에 따라 해싱 함수의 충돌로 저장되지 못한 주소의 수도 같이 표시하였다.

위 그림 4에서 보듯이 주소의 수가 많아질수록 기존 방법들은 평균 검색 시간이 점차 증가하는 추세를

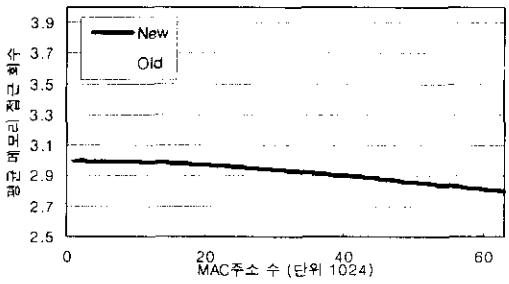

그림 4. 평균 검색 시간

보이고 있는 반면 새로운 방법은 조금씩 감소하고 있는 것을 볼 수 있다 그 이유는 기존 방법은 주소 수가 많아지면 연결 리스트에서 두번째 항목에서 히트가 나는 경우가 많아지므로 점점 시간이 오래 걸리고 또 충돌로 해시 엔트리에 들어가지 못한 주소에 대해서도 연결 리스트를 탐색하므로 시간이 오래 걸리게 된다.

그러나 새로운 방법은 두번째 항목에서 히트가 나더라도 시간이 추가적으로 더 필요하지 않으며 또한 충돌로 해시 엔트리에 들어가지 못한 주소는 한번만 메모리를 읽으면 알 수 있기 때문에 충돌하는 양이 많아 질수록 평균 검색 시간이 짧아지게 된다.

그림 5를 보면 해시 함수의 충돌로 해시 엔트리에 들어갈 수 없는 주소의 수를 전체에 대한 백분율로 나타내었는데 해시 버킷을 조각으로 나누어서 각각 저장할 수 있는 위치를 제한 하였기 때문에 그로인해 일부에서 해시 버킷 수가 모자라 해시 엔트리에 들어가지 못하는 주소가 발생하였다. 이 값은 주소의 수가 적을 때는 거의 없다가 주소 수가 증가하면 최대 30%정도까지 추가적인 손실이 발생한다.

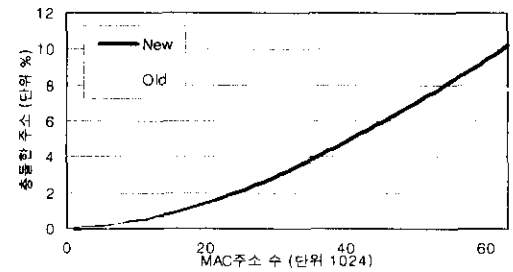

그림 5. 평균 충돌 회수

본 논문에서 제시한 알고리즘은 해시 함수의 충돌과 해시 버킷의 이용율은 떨어지지만 일반적인 경우 64k 개의 주소를 모두 쓰는 경우는 거의 없으며 보통 4k 에서 10k 정도만 사용하는 경우가 대부분이다. 이런 경우에서 볼 때 충돌로 인한 손실은 거의 없으면서 검색시간의 이득은 크다는 것을 알 수 있는데 그림 4,그림 5를 보면 20k 개의 주소를 저장할 때에 오버헤드는 거의 없으면서 검색속도는 약 9%정도 향상되는 것을 볼 수 있다.

d. 결론

본 논문에서 사용한 파셜 매칭을 사용하고 연결 리스트를 한 단으로 줄이는 구조는 적은 주소값을 가지는 테이블을 검색하는 데는 우수한 성능을 나타내는 것을 알 수 있으나 주소의 수가 해시 버킷의 수에 근접하면 제안된 방법은 기존 방법에 비해 버킷을 비효율적으로 사용하여 성능이 떨어지는 단점이 있어서 검색 시간이 짧다고 하더라도 사용하는데 문제가 있다. 그러나 이런 단점은 해시 함수를 MAC 주소에 맞게 효율적으로 선택하면 어느 정도 극복이 가능하리라고 보며 앞으로 연구할 가치가 있으리라고 본다.

III. 참고 문헌

[1] D. Knuth, *The Art of Computer Programming*, vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass, 1973

[2] M.V. Ramakrishna, E. Fu, E.Bahcekapili, *Efficient Hardware Hashing Design for High Performance Computers*, Technical Report Series, Dept. of CS, RMIT, April 1996