고속 패킷(packet) 처리를 위한 IP lookup scheme

박 우 종, 정 민 섭, 정 진 우, 강 성 봉
삼성전자 중앙연구소 NS Lab.
전화 : (031) 200-3707 / 팩스 : (031) 200-3121

IP lookup scheme for high speed packet forwarding

Jong Woo Park,
Samsung Electronics
E-mail : fssmpark@samsung.co.kr

Abstract

In this paper, we propose a new scheme which improves the IP address lookup time. The new scheme is composed of two core technologies, named the prefix alignment and the prefix distance ordering.
Now, as the Internet is being used commonly by improving the data transmission capacity, the need for enlarging the bandwidth of the Internet is on the rise. IP address lookup performance problem is an important obstacle in the router executing high speed packet forwarding. This results from the fact that the prefixes routing table is composed of and the traffic being processed in unit time are largely on the increase.
The proposed lookup scheme is divided into two parts in technology, the one is the algorithm forming a routing database(routing table), the other is the lookup procedure in the actual packet processing.

I. 서론

통신 이전에 호 설정(call setup)을 미리 수행하는 서킷(circuit)망과는 달리 IP를 기반으로 하는 패킷 망(packet network)에서는 중간 노드(Node)에 해당하는 라우터가 개별 패킷의 IP 헤더(header) 내의 주소를 보고 패킷의 경로(route)를 결정하게 된다. 개별 패킷에 대해서 어떻게 처리할 것인가를 결정하는 것을 포워딩 결정(Forwarding Decision)이라 한다. 이러한 포워딩 결정을 위해 라우터들은 정해진 약속(protocol)에 의해 라우팅 정보(routing information)를 주고, 받으며 라우팅 테이블을 구성한다.

즉 각각의 IP 패킷에 대한 처리 정보를 일정한 규칙에 의해 특정 구조로 만들어 놓은 것을 라우팅 테이블이라 한다. 이후 개별 IP 패킷의 처리를 위하여 라우팅 테이블을 참조해서 원하는 정보를 얻어내는 것을 IP address lookup이라 한다. 이와 같이 라우팅 테이블을 만들고 원하는 정보를 얻어내는 전반을 IP address lookup scheme이라 한다.

현재 사용하고 있는 IP 는 version 4(IPv4) 이다. 1993년 이후, 인터넷상에 늘어나는 네트워크와 호스트 수를 감당하기 위해 CIDR(Classless Inter-Domain Routing)에 기반 한 IP 주소 할당과 관리를 위한 제안[1],[2]이 받아들여진 후 인터넷상에서 서로 다른 관리 도메인 사이에서 패킷의 경로를 결정하기 위한 방법으로 Longest Matching Prefix(LMP; Best Matching Prefix/Longest Prefix Match)기법이 사용되고 있다.
LMP기법은 이름에서 알 수 있듯이 32 bit의 IP 주소를 비교하는 것이 아니라 8 ~ 32 bit에 다양한 길이의 네트

워크 식별자(network ID) 혹은 개별 IP 주소를 비교해서 테이블 상의 데이터 중에 가장 맞는 정보를 찾아 해당 정보를 취하는 방식이다. 즉 다양한 길이의 데이터를 다루어야 한다는 것이 LMP lookup의 핵심이다.

현재까지 개발된 대부분의 IP LMP(Longest Matching Prefix) 알고리즘들은 크게 다음과 같이 분류할 수 있다. 하나는 기존의 CPU(Processor)와 고속의 memory에 의한 lookup 구조이고, 다른 하나는 CAM(Content Addressable Memory)와 같은 새로운 lookup구조이다.
기존의 IP lookup구조는 크게 trie based 구조와 hash based 구조가 있다.

Trie based lookup

trie 구조는 시작점(root)를 기점으로 끝점(leaf)까지 중간 노드(intermediate node)로 연결되어있는 구조를 의미한다. trie구조에 있어서의 성능은 원하는 노드까지 찾아가는데 걸리는 중간노드의 수를 최소화하는데 있다.
trie 구조의 대표적인 알고리즘으로는 patricia trie 알고리즘이 있다. 현재 인터넷 백본(Net/3, FreeBSD등)상의 많은 라우터들이 patricia trie 알고리즘을 채택하고 있으며 이 구조는 radix trie의 특정한 형태이다. patricia trie는 어떠한 길이의 prefix에 대한 LMP도 처리가능하며 구현이 쉽다는 장점이 있다. patricia trie구조를 기본으로 수많은 trie구조가 제안되고 사용되고 있다. 이러한 대표적인 알고리즘으로는 Level Compressed(LC) trie[3] 및 prefix trie[4] 가 있다.

Hash based lookup

hash란 말 그대로 덩어리란 얘기이다. hash based lookup이라 함은 따라서 비교의 대상이 되는 데이터(prefix)들을 특정한 기준으로 쪼개서 처리하는 방법을 의미한다. hash 기능의 성능은 대상 주소들을 얼마나 균등하게 나눌 수 있는 가에 달려있다. 대표적인 hash 알고리즘에는 1. CRC를 이용하는 방법(비교대상이 되는 비트열의 CRC 값을 계산해서 이를 hash key로 이용하는 방법), 2. Checksum을 이용하는 방법(비트열의 checksum 값을 계산해서 이를 hash key로 사용하는 방법), 3. XOR값을 이용하는 방법(비트열을 XOR시킨 값을 hash key로 이용하는 방법), 4. 비트열의 일부를 이용하는 방법등이 있다. 비교하고자 하는 대상 비트열의 특성에 따라서 다양한 hash key를 구할 수 있고, 이에 따라서 hash based lookup scheme의 성능이 달라지게 된다. 다양한 크기의 cache 메모리를 사용하는 것은 traffic의 locality라는 특징을 이용하여 packet의 관계를 효과적으로 이용하기 위한 방법이 되며, 이도 일종의 hash를 이용하는 것이라 할 수 있으며, Large-Memory Archicture에서 제안하는 기법도 비트열의 일부를 이용하는 hash based lookup이 된다. 이밖에도 기본적인 trie 구조나 hashing방식을 변형한 많은 알고리즘이 제안되었으며 trie와 hashing을 병합하는 방식들도 제안되고 있다.

Hardware based lookup

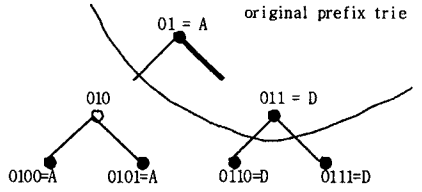
고속의 프로세서와 메모리에 의한 기존의 lookup구조와는 달리 hardware에 기반한 lookup구조가 제안되고 있다. 이러한 방식에는 lookup전용 processor와 CAM을 기반으로 하는 구조 등이 대표적이다.
Lookup전용 프로세서라함은 lookup기능을 수행하기 위해 고안된 프로세서를 의미한다. 범용 프로세서를 이용하여 독특한 알고리즘으로 lookup기능을 수행하는 것과는 별도로 네트워크상에서 issue가 되는 기능을 수행하도록 특정 프로세서를 개발하는 것이 현재 추세이다. 이중에 IP lookup을 수행하는 lookup전용 프로세서들이 개발되고 있다. 대표적인 프로세서로는 alliance사의 IPRPv4와 NetLogic System사의 CIDR 프로세서등이 있다. 이러한 프로세서들은 CAM기술에 기반한 것도 있고 혹은 다른 독특한 기술에 기반한 것도 있다. hardware를 기반으로 하는 구조는 기존의 구조가 갖는 성능상의 문제를 해결하기 위한 시도이며, 현재는 기술적인 결합으로 서로의 장점을 살리는 방안이 활발히 연구되고 있다.

II. 본론

제안하는 lookup 알고리즘 및 구조는 hardware를 기반으로 개별 패킷에 대한 lookup을 수행하되 데이터 구조는 hash 및 trie구조를 모두 갖고 있는 구조이다. ip address prefix는 그 정의에 따라서 길이 8 ~ 32의 가변길이를 갖는다. 이러한 prefix들을 기준 길이(=16)를 기준으로 prefix expansion을 이용하여 변형된 hash table을 구성한다. 이렇게 구성된 hash table을 prefix alignment table이라 명명한다. prefix alignment table은 기본적으로 길이 16이하의 prefix들에 대한 정보를 포함한다. 길이가 16보다 큰 프리픽스들에 대해서는 상위 16 bit를 공유하는 prefix들을 모아 prefix alignment table 이후에 다시 여러 개의 sub-prefix table을 이루는데, 이를 prefix distance ordering table이라 명명한다. prefix alignment table의 각 엔트리는 이후에 연결되는 prefix distance ordering table의 시작점(root node)에 대한 pointer정보를 포함하고 있다. prefix distance ordering table은 기본적으로 LC trie 구성 알고리즘을 응용한다. 본 제안은 이렇게 prefix alignment table과 prefix distance ordering table로 이루어 진 데이터 베이스와 데이터 베이스로부터 원하는 정보를 취하는 lookup 구조로 이루어진다.

prefix alignment table

다양한 길이의 prefix들을 기준 길이로 확장함으로써 메모리상의 엔트리를 소모하는 대신 억세스 싸이클(access cycle)을 줄이는 기법이다. Large memory scheme으로 알려진 바 있는 이 기법은 one-level prefix expansion이라는 이름으로 소개된 바 있다. 이후 controlled prefix expansion이라는 이름으로 대상 데이터의 특징에 따라 기준 길이를 변화시키는 기법이 소개되었다.



a. prefix expansion example(trie)

prefix	information
01	A
011	D

original table

expanded prefix	information
0100	A
0101	A
0110	D
0111	D

expanded table

b. prefix expansion example(table)

그림 1. prefix alignment table structure example

그림 1.에서 보는 바와 같이 길이 2, 3인 prefix 01, 011을 기준길이 4로 확장시키면 0100, 0101, 0110, 0111로 된다. 우선 01의 확장에서 0100 ~ 0111 모두 A의 정보를 갖는다. 이후 011의 확장에 의해 0110과 0111은 D의 정보를 갖게 된다. 이러한 방식으로 길이 8 ~ 16의 prefix들을 기준길이 16에 확장을 하여 alignment table을 얻게 된다. 이는 2^{16} 개의 엔트리를 갖는 변형된 hash table이 된다. prefix의 길이가 16보다 큰 경우는 상위 16 bit를 공유하는 prefix들을 모아 alignment table에 포함된 16 bit를 제외한 나머지 sub-prefix들로 prefix distance ordering table을 구성한다.

prefix distance ordering table

prefix distance ordering table이란 alignment table내에서 처리되지 않는 sub-prefix들로 이루어지는데, trie의 level(height)을 줄여서 leaf node까지의 lookup time을 줄인 LC-trie 알고리즘을 응용한 데이터 구조이다. 이 구조는 branch, skip, pointer의 세 변수로 이루어진다. sub-trie가 필요한 경우, alignment table의 해당 엔트리는 포인터로 sub-trie의 시작점을 연결하게 된다. 그림 2.는 prefix distance ordering table의 형성 알고리즘과 구조를 설명하기 위한 예이다.

prefix	information
00	①
001	②
010	③
010011	④
100	⑤
110	⑥
111	⑦

a. prefix table

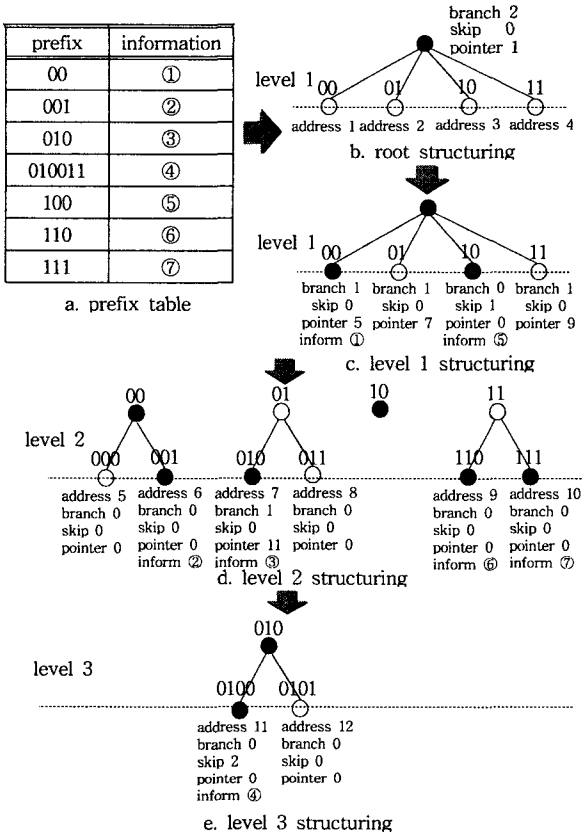


그림 2. prefix distance ordering table structure example

a. 의 prefix table에서 가장 짧은 길이는 2이다. 따라서 길이 2를 기준으로 prefix를 나열하면, b.를 얻을 수 있다. 시발점(root)에 branch값 2, skip 값 0, pointer 값 1을 할당한다. 여기서 branch는 기준에서부터 확인할 비트의 수가 되고, skip값은 기준에서 branch를 적용하기 전에 건너뛸 비트 수, pointer는 해당 노드에서 다음 단계의 노드를 찾아가기 위한 연결정보가 된다. c.에서는 level 1에 존재하는 4개의 노드에 알맞은 정보를 채워 넣는다. 우선 prefix "00", "01", "10", "11"에 각각 address 1, 2, 3, 4를 할당하고, prefix "00"은 그 자체로 의미 있는 prefix이므로 a.의 prefix table에서 해당하는 노드 값 ①을 넣는다. prefix "00"이후에 존재하는 prefix "001"을 표현하기 위해 "001"에서 "00"을 뺀 값 1의 길이를 branch 값으로 하여 level 2에 해당하는 "000", "001"의 노드를 구성한다. prefix "000", "001"은 각각 주소 5, 6을 할당받으며 prefix "00"은 branch 1, skip 0, pointer 5를 갖게 된다. 여기서 pointer 5는 "00" 이후의 노드 중 가장 왼쪽에 위치하는 노드("000")의 주소를 가리킨다. prefix "01"은 그 자체로 의미 있는 노드가 아니므로 노드 값은 포함하지 않고 "00"과 마찬가지로 "010"을 표현하기 위해 d. 와 같이 분기한다. 즉 prefix "01"에 해당하는 주소 2에 branch, skip, pointer를 c. 와 같이 표기한다. prefix "10"은 그 자체로는 의미 없는 노드

이고 이후 prefix "100"만이 존재한다. 이렇게 의미 없는 중간 노드 이후에 존재하는 prefix들이 다음 branch에 해당하는 비트 열을 공유하는 경우나 단 하나의 prefix 만이 존재하는 경우에 skip을 사용하게 된다. 위 경우는 "10" 이후의 "0"에 해당하는 1을 skip하고 주소 3에 prefix "100"에 해당하는 노드 값 ⑤를 넣는다. prefix "11"도 같은 방법으로 구성된다. prefix "010"에 해당하는 주소 7의 엔트리는 이후에 존재하는 "0100"을 표현하기 위해 역시 "0100"에서 "010"을 뺀 "0"의 길이 1을 branch 값으로 분기하여 e.의 결과를 얻는다. 여기서 prefix "10"과의 차이는 현재 노드에서 다음에 존재하는 노드가 단 하나일 경우 현 노드가 의미 있는 노드인가 아닌가에 따라 다르게 분기하게 됨을 알 수 있다.

address	branch	skip	pointer	prefix	information
0	2	0	1	root	-
1	1	0	5	00	①
2	1	0	7	-	-
3	0	1	0	100	⑤
4	1	0	9	-	-
5	0	0	0	-	-
6	0	0	0	001	②
7	1	0	11	010	③
8	0	0	0	-	-
9	0	0	0	110	⑥
10	0	0	0	111	⑦
11	0	0	0	0100	④
12	0	0	0	-	-

표 1. distance ordering table example

표 1.는 그림 2.의 a. prefix table에 대한 distance ordering table의 구성 예이다. 표 1.에 대해 개별 비트 열에 대한 lookup 예를 들어보자.

비트 열 "01001000"에 대해서 원하는 정보를 구해보면, 우선 루트에서 skip = 0, branch = 2의 정보를 이용해 비트 열 "01001000"에서 처음 2비트인 "01"을 얻을 수 있다. 여기에 루트가 가리키는 포인터 값을 더하면 01 + 1 = 2 임으로 주소 2를 참조한다. 주소 2는 그 자체로 prefix 정보를 갖지 않기 때문에 주소 2의 엔트리 내용 중 skip = 0, branch = 1의 값을 이용해 위와 같이 0 + 7 = 7로 다음에 참조할 주소 7을 얻는다. 주소 7은 자체로 prefix "010"에 대한 정보를 갖는다. 따라서 prefix "010"에 해당하는 노드 정보 ③을 0우선 취한다. branch 값이 0이 아니기 때문에 다음 레벨이 존재함을 알 수 있다. 따라서 다음 참조할 주소를 갖은 방법으로 구하면, 주소 11을 얻고 이를 참조해서 해당 prefix "010011"과 해당 비트 열을 비교해 일치하지 않음을 알 수 있다. 이러한 경우 LMP의 정의에 의해 우리는 비트 열 "01001000"에 가장 잘 맞는 정보로 ③을 취하게 된다.

combining two concepts and structuring database

표 2.는 prefix alignment table과 distance ordering table을 연결하는 전체 데이터 베이스를 형성한 예이다.

prefix	information
"0011"	①
"01"	②
"011"	③
"1"	④
"100"	⑤
"110"	⑥

a. prefix table

aligned prefix	prefix	inform	pointer
00	-	-	100
01	"01"	②	200
10	"1"	④	300
11	"1"	④	400

b. alignment table

address	branch	skip	pointer	prefix	information
100	0	2	0	"11"	①

address	branch	skip	pointer	prefix	information
200	0	1	0	"1"	③

address	branch	skip	pointer	prefix	information
300	0	1	0	"0"	⑤

address	branch	skip	pointer	prefix	information
400	0	1	0	"0"	⑥

c. distance ordering table

표 2. structuring a complete database example

III. 결론

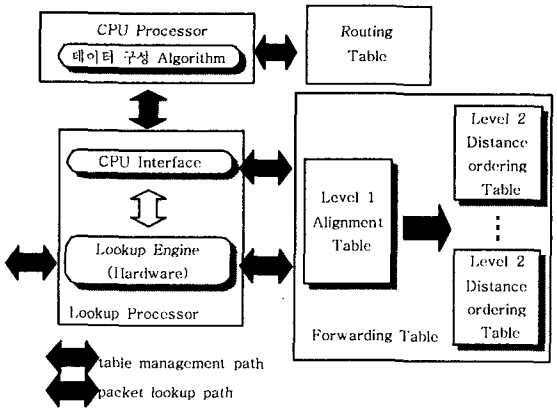


그림 3. overall lookup scheme

그림 3.에서 보듯이 데이터 베이스의 형성 및 관리는 CPU processor가 수행하고 개별 패킷에 대한 lookup은 순수 하드웨어로 이루어진 lookup engine이 수행한다. 우리는 IP 포워딩 엔진(forwarding engine)을 개발하면서 본 lookup scheme을 채택하였고, 이를 통해 본 제안이 합리적이고 뛰어남을 확인하였다.

reference

[1] Y. Rekhter, T. Li, "RFC 1518" september 1993.
[2] V. Fuller, T. Li, J. Yu, K. Varadhan, "RFC 1519" september 1993.
[3] Stefan Nilsson and Gunnar Karlsson, "IP-Address Lookup Using LC-Tries", IEEE Journal, June 1999.
[4] Henry Hong-Yi Tzeng, "On Fast Address-Lookup Akgorithms" IEEE Journal, June 1999.
[5] V. Srinivasan and G. Varghese, "Fast address Lookups using controlled prefix expansion", Proc. ACM, February 1999.